

Diploma  
in  
Computer Applications, Business  
Accounting and Multilingual DTP

# Artificial Intelligence and Machine Learning using Python



राष्ट्रीय उर्दू भाषा विकास परिषद्

قومی کونسل برائے فروغ اردو زبان

**NATIONAL COUNCIL FOR PROMOTION OF URDU LANGUAGE (NCPUL)**

Ministry of Education, Department of Higher Education, Govt. of India  
Farogh-e-Urdu Bhawan, FC-33/9, Institutional Area, Jasola, New De lhi-110 025

Tel.No.: 011-49539000 Fax : 49539099

Website: [www.urducouncil.nic.in](http://www.urducouncil.nic.in)

E-mail: [urducouncil@gmail.com](mailto:urducouncil@gmail.com)

Year of Publication (3rd Edition) : 2024

Quantity : 15000

Printed By : Makoff Printers  
Paper Used : BGPPL Super Nova Maplitho  
Thickness : 70GSM  
Title : 300 GSM BGPPL Royal Art Gold

## Preface

National Council for Promotion of Urdu Language (NCPUL) is mandated to take action for making available in Urdu language the knowledge of scientific and technological development as well as knowledge of ideas evolved in the modern context. In the emerging information technological scenario, it was necessary that this technology is made available to the Urdu speaking population of the country with a view to transform Urdu speaking population into employable technical workforce.

It was in 1999 when a humble beginning was made and some Multilingual DTP Computer Centres were set up at select locations. From the beginning, attempt was made to provide standard course contents and ensure quality in computer education at par with the quality of agencies dedicated to computer awareness and education in the country. In this context, NIELIT, which is an approved Government of India agency for imparting computer education in non-formal sector, was engaged for conducting examination and certification. There has been a demand that course contents need to be upgraded. Therefore, the NCPUL has entered into a MoU with NIELIT and delegated the powers of regulating academic standards and examinations to it. This course is now **Computer Applications, Business Accounting & Multilingual DTP (CABA-MDTP)** and it is hoped that course will enable the students pursuing this course to get CCC, CABA-MDTP Diploma and 'O' Level Certification from NCPUL and NIELIT. The important change in the eligibility will enable students with +2 qualification to pursue 'O' Level Diploma and those who are with Matric or equivalent qualification to pursue CCC and CABA-MDTP.

I am sure that the new courseware of NIELIT will meet the requirements of the students and NCPUL will be in a position to discharge its mandate of linking Urdu to the contemporary requirements particularly, in the context of ever increasing use of information technology in our life.

**Prof. Dhananjay Singh**  
(Director)



## Contents

<b>Chapter 1</b> .....	8
<b>Introduction to Programming</b> .....	8
<b>1.1 The Basic Model of Computation</b> .....	8
1.1.1 How to Write Algorithms? .....	9
1.1.2 Advantages of an algorithm .....	9
<b>1.2 Flowchart</b> .....	9
1.2.1 Features of the flowchart .....	9
1.2.2 Advantages of flowchart .....	9
1.2.3 Constraints of flowchart .....	10
<b>1.3 Programming Languages</b> .....	10
1.3.1 Machine Language .....	10
1.3.2 Assembly Language .....	11
1.3.3 High Level Language .....	11
<b>1.4 Compilation</b> .....	11
<b>1.5 Testing &amp; Debugging and documentation</b> .....	12
<b>Exercises</b> .....	14
Multiple Choice Questions .....	14
<b>Chapter 2</b> .....	15
<b>Algorithms and Flowcharts</b> .....	15
<b>2.1 Flowchart symbols</b> .....	15
<b>2.2 Basics of flowcharts /algorithm for sequential processing</b> .....	16
<b>2.3 Decision based processing and iterative processing</b> .....	16
2.3.1 Decision-based Processing .....	16
2.3.2 Iterative (Looping) Processing .....	17
<b>Exercises</b> .....	28
Multiple Choice Questions .....	28
State whether statement is true or false .....	28
Practice Questions .....	28
<b>Chapter 3</b> .....	29
<b>Programming with Python</b> .....	29
<b>3.1 Python Introduction</b> .....	29
3.1.1 Technical strength of Python .....	29
3.1.2 Introduction to Python Interpreter and program execution .....	29

3.1.3	Using comments.....	30
3.1.4	Literals .....	31
3.1.5	Constants .....	31
3.1.6	Python Built in Data Types .....	32
3.1.7	Python constructs.....	36
3.1.8	Expressions .....	36
3.1.9	Arithmetic Operator .....	37
3.1.10.	Relational Operator .....	40
3.1.11.	Logical Operators .....	43
3.1.12.	bitwise operators .....	45
3.1.13	Conditional Statements.....	50
	Multiple Choice Questions .....	70
	State whether statement is true or false .....	71
	Fill in the blanks.....	71
	Lab Exercises.....	72
Chapter 4	.....	73
String Handling and Sequence Types	.....	73
4.1	String Handling.....	73
4.1.1.	Creating a string.....	73
4.1.2.	String indexing .....	73
4.1.3.	string slicing.....	75
4.1.4.	traversing a string .....	76
4.1.5.	Concatenation of string .....	77
4.1.6.	Other operations on strings.....	78
4.1.7	accepting input from console .....	80
4.1.8	print statements.....	81
4.1.9	simple programs on strings .....	82
4.2	Sequence Data Types .....	83
4.2.1.	list .....	83
4.2.2.	tuple.....	83
4.2.3.	Dictionary .....	84
4.2.4.	Indexing and accessing elements of lists, tuples and dictionaries .....	85
4.2.5.	slicing in list, tuple.....	88
4.2.6.	Concatenation on list, tuple and dictionary .....	89
4.2.7.	Concept of mutubility .....	91

Exercises.....	97
Multiple Choice Questions .....	97
Chapter 5 .....	100
Functions.....	100
5.1 Top-down Approach of Problem Solving .....	100
5.2 Modular Programming and Functions .....	101
5.2.1. Modular Programming.....	101
5.2.2. Module .....	101
5.2.3. Advantages of Modular Design.....	101
5.3 Function and function parameters .....	102
5.3.1. HOW to Define a Function in Python .....	102
5.3.2. How to Define and Call a Basic Function in Python .....	103
5.3.3. It just prints hello world whenever it is called.....	103
5.3.4. How to Define and Call Functions with Parameters.....	103
5.4 Local Variables.....	104
5.4.1. Syntax of Local Variable in Python.....	104
5.4.2 How Local Variable Works in python? .....	105
5.5 The Return Statement .....	105
5.6 Default argument values.....	105
5.7 keyword arguments .....	106
5.8 VArArgs parameters. ....	107
5.9 Library function: .....	107
5.9.1. input().....	107
5.9.2. eval() .....	108
5.9.3. print() function .....	108
5.9.4. print() Parameters.....	109
5.9.5 String Functions:.....	109
5.9.6 rfind() function .....	111
5.9.6 Date and time functions .....	114
5.9.7 recursion .....	115
5.9.8 Packages and modules .....	116
Exercise .....	121
Multiple choice questions .....	121
State whether statement is true or false .....	122
Fill in the blanks.....	122

<b>LAB exercise</b> .....	123
<b>Chapter 6</b> .....	124
<b>File Processing</b> .....	124
<b>6.1 Concept of Files</b> .....	124
<b>6.2 File opening in various modes and closing of a file</b> .....	124
<b>6.3 Reading from a file</b> .....	124
<b>6.4 Writing onto a file, File functions - open()</b> .....	125
<b>6.5 close()</b> .....	125
<b>6.6 read()</b> .....	126
<b>6.7 readline()</b> .....	126
<b>6.8 readlines()</b> .....	127
<b>6.9 write()</b> .....	127
<b>6.10 writelines()</b> .....	128
<b>6.11 tell()</b> .....	129
<b>6.12 seek()</b> .....	129
<b>6.13 Command Line arguments</b> .....	130
<b>Exercise</b> .....	135
<b>Multiple choice Question</b> .....	135
<b>Chapter 7</b> .....	138
<b>Machine Learning and AI</b> .....	138
<b>7.1. Types of Machine Learning Algorithms (supervised, unsupervised)</b> .....	138
<b>7.1.1 Supervised Machine Learning</b> .....	139
<b>7.1.2. Unsupervised Machine Learning</b> .....	140
<b>7.1.3</b> .....	140
<b>7.2. Feature engineering</b> .....	141
<b>7.3. Preparing Data</b> .....	141
<b>7.3.1 Training Data, Test data</b> .....	142
<b>7.3.1.1 What is Training Data?</b> .....	142
<b>7.3.1.2 Data Validation</b> .....	142
<b>7.4. Introduction to different Machine Learning Algorithms</b> .....	143
<b>7.5. Training the Machine learning model and predicting the results</b> .....	144
<b>7.6. Applications of Machine Learning. Introduction to Artificial Intelligence</b> .....	148
<b>7.6.1 Applications of Machine Learning:</b> .....	148
<b>1. Image Recognition:</b> .....	148
<b>2. Speech Recognition:</b> .....	149



3. Traffic prediction:.....	149
4. Product recommendations: .....	149
5. Self-driving cars: .....	150
6. Email Spam and Malware Filtering:.....	150
7. Virtual Personal Assistant: .....	151
8. Online Fraud Detection:.....	151
9. Stock Market trading: .....	151
10. Medical Diagnosis: .....	151
11. Automatic Language Translation:.....	152
7.6.2 Introduction to Artificial Intelligence: .....	152
7.7. Common Applications of AI: .....	154
7.7.1. AI Application in E-Commerce: .....	154
7.7.2. Applications of Artificial Intelligence in Education: .....	154
7.7.3. Applications of Artificial Intelligence in Lifestyle: .....	155
7.7.4. Applications of Artificial intelligence in Navigation:.....	156
7.7.5. Applications of Artificial Intelligence in Robotics: .....	156
7.7.6 Applications of Artificial Intelligence in Human Resource.....	156
7.7.7. Applications of Artificial Intelligence in Healthcare .....	156
7.7.8. Applications of Artificial Intelligence in Agriculture .....	156
7.7.9. Applications of Artificial Intelligence in Gaming .....	157
7.8. Advantages and Disadvantages of AI.....	157
7.9. Common examples of AI using python .....	158
Exercise .....	161
Objective Type Questions.....	161
Subjective Type Questions .....	163
Chapter 8 .....	164
Data Science and Analytics Concepts.....	164
8.1. What is Data Science and Analytics? The Data Science Process.....	164
8.2. Framing the problem .....	166
8.3. Collecting .....	167
8.4. Processing .....	168
8.4.1 Six stages of data processing .....	168
8.5. Cleaning and Munging Data .....	169
8.6. Exploratory Data Analysis .....	170
8.7. Visualizing results .....	172

<b>Exercise</b> .....	173
<b>OBJECTIVE TYPE QUESTIONS</b> .....	173
<b>SUBJECTIVE TYPE QUESTIONS</b> .....	175
<b>Chapter 9</b> .....	176
<b>Introduction to NumPy (7 Hrs.)</b> .....	176
<b>9.1. Array Processing Package</b> .....	176
<b>9.2. Array types</b> .....	177
<b>9.3. Array slicing</b> .....	177
<b>9.3.1 Negative Slicing</b> .....	178
<b>9.4. Computation on NumPy Arrays – Universal functions</b> .....	180
<b>9.4.1 Array arithmetic</b> .....	181
<b>9.5. Aggregations: Min, Max, etc.</b> .....	182
<b>9.5.1. Python numpy sum:</b> .....	182
<b>9.5.2. Python numpy average:</b> .....	183
<b>9.5.3. Python numpy min :</b> .....	183
<b>9.5.4. Python numpy max</b> .....	184
<b>9.6. N-Dimensional arrays</b> .....	185
<b>9.7. Broadcasting</b> .....	187
<b>9.8. Fancy indexing</b> .....	191
<b>9.9. Sorting Arrays</b> .....	192
<b>Exercise</b> .....	194
<b>Objective Type Question</b> .....	194
<b>Subjective Type Questions</b> .....	196
<b>Chapter 10. Data Analysis Tool: Pandas</b> .....	197
<b>10.1. Introduction to the Data Analysis Library Pandas</b> .....	197
<b>10.2. Pandas objects – Series and Data frames</b> .....	197
<b>10.2.1 Pandas Series</b> .....	198
<b>10.2.2 Pandas Dataframe</b> .....	199
<b>10.3 Data indexing and selection</b> .....	200
<b>10.4 Nan objects</b> .....	206
<b>10.5 Manipulating Data Frames</b> .....	207
<b>10.6 Grouping</b> .....	212
<b>10.7 Filtering</b> .....	214
<b>10.8 Slicing</b> .....	217
<b>10.9 Sorting</b> .....	220

<b>10.10 Ufunc .....</b>	<b>222</b>
<b>Exercise .....</b>	<b>223</b>
<b>OBJECTIVE TYPE QUESTIONS.....</b>	<b>223</b>
<b>SUBJECTIVE TYPE QUESTIONS.....</b>	<b>226</b>

# Chapter 1

## Introduction to Programming

### 1.1 The Basic Model of Computation

To complete a task there arises a need to write a set of instructions and these set of instructions are called program and to solve problems on a computer, one must first write a step-by-step solution using simple instructions for operations and then run the programme to obtain the results. There are numerous ways to solve a given problem, and thus solutions may differ depending on the thought process of an individual.

Nevertheless, some fundamental steps in problem solving remains unchanged. These fundamental steps are:

- a. Defining the problem and choosing the data types.
- b. Pinpointing the computation steps and their sequence required to obtain the solution.
- c. Choosing decision points, i.e. selecting the appropriate operations at specific state.
- d. Knowing what to expect and comparing it to the actual values.

#### Step 1. Understanding the Problem

First, understand the problem by reading it carefully and try to figure out what is the expected output. Do not start drawing flowcharts right away. Take some test data and solve the problem manually on paper. Let us take an example and understand how to solve the problem systematically:

Software engineer before writing the actual program writes algorithm or flowchart.

#### Step 2. Generate potential solutions

The next step is to create a list of possible solutions to the problem you've discovered. There are many ways to generate solutions. This can be done individually or in a group setting.

#### Step 3. Choose one solution

Once a list of possible solutions has been made, it's time to put your to the test. In order to find the best solution for the problem, analyse every possible resolution and decide which is best for the situation you are in. One might want to consider many elements before choosing one solution. These elements include efficacy, practicality, timeliness, resources, and cost. This is also where risk management will be used to help make a decision. Like brainstorming, choosing a solution doesn't have to be done alone.

### ALGORITHM

What is an Algorithm and its features?

Software engineers typically use algorithms to plan and solve problems. An algorithm is a series of steps to solve a particular problem. Alternatively, an algorithm is an ordered, unique set of steps that produces a result and finishes in a finite amount of time.

Algorithm has the following characteristics:

1. Can be written in any language suitable to developer.
2. Easy to understand.
3. Very simple, complete, clear and systematic program flow.
4. No standard format and have no defined rules of writing.

### 1.1.1 How to Write Algorithms?

Let us write an algorithm to find the volume of a cuboid.

**Step 1 - Define algorithm input:** Many algorithms capture the data to be processed. To calculate the volume of the cuboid, you have to enter the length, width and height of the cuboid.

**Step 2 - Define Variables:** Algorithm variables can be used in multiple places. Three variables, the length, width and height of the cuboid, can be defined as LENGTH, WIDTH and HEIGHT.

**Step 3 - Outline the algorithm's operations:** Use input variable for computation purpose, e.g. to find area of cuboid multiply the LENGTH, WIDTH and HEIGHT variable and store the value in new variable (say) VOLUME. An algorithm's operations can take the form of multiple steps and even branch, depending on the value of the input variables.

**Step 4 – Algorithm output:** Use the input variable for computational purposes. To find the area of the box, multiply the LENGTH, WIDTH, and HEIGHT variables and store their values in a new variable (e.g. VOLUME). The operation of the algorithm can take the form of multiple sets and can also branch according to the value of the input variable.

### 1.1.2 Advantages of an algorithm

- ❖ It is a step-by-step solution to a given problem and is much easier to understand.
- ❖ The algorithm uses a precise system.
- ❖ It is programming language independent, so anyone can easily understand it without any programming knowledge.
- ❖ Each step of the algorithm has its own logical sequence, which is easy to debug.

## 1.2 Flowchart

Flowchart is another method widely used for planning and solving the programming problems. Flowchart is a combination of two words 'Flow' and 'Chart'. 'Flow' means direction/sequence and 'Chart' means diagram i.e. unlike algorithm, flowchart explains the sequence of steps involved diagrammatically i.e. in picture form. This is another commonly used programming tool. Observing at the flow chart, you can easily understand the operations performed by the system and the sequence of operations. Flowcharts are often regarded as blueprints for designs used to solve specific problems.

### 1.2.1 Features of the flowchart

- ❖ Uses fixed figures to define the steps involved i.e. have rules.
- ❖ Depicts Logical flow very clearly.
- ❖ Being graphical representation it is simple and very easy to understand.
- ❖ No programming required
- ❖ Easy to debug

### 1.2.2 Advantages of flowchart

- ❖ Suitable for large systems: While developing large systems numerous programmers are involved and develop different modules. Under such circumstances, division of flowchart becomes easy and programmers draw flowchart of his/her assigned module.
- ❖ Effective communication and easy to understand: Since, it is a pictorial representation of program so it is quite simple and easy to understand.

- ❖ Logics can be easily interpreted: Logics defined in the flowchart can be easily depicted as conditions/ decision-making figures can distinguished from others and the directions can be identified.
- ❖ Testing becomes easier: Being a graphical representation the testing in case of flowchart becomes very easy and the program flow in the wrong direction can be easily identified.

### 1.2.3 Constraints of flowchart

- ❖ Flowchart require more time to draw thus it is a laborious work to draw a flowchart.
- ❖ Modifications are carried out with great difficulty in case of the flowchart.

## 1.3 Programming Languages

A computer can only do what a person convey to do through set of instructions, which is a program. Program need to be written in a language, which a computer can understood and perform the task desired.

Computer programming languages has been broadly classified into the three categories:

- a) Machine Language
- b) Assembly Language
- c) High Level Language

### 1.3.1 Machine Language

Machine language, additionally called machine code, is the primary language of computers. It is read by the computer's imperative processing unit (CPU), includes binary numbers, and seems like a totally lengthy collection of 0s and 1s. Since binary code is the best language that computer hardware can understand, in the long run the source code of a human-readable programming language should be translated into machine language through a compiler or interpreter.

Each CPU has its own machine language. The processor reads and processes instructions that instruct the CPU to perform simple tasks. An instruction consists of a specific number of bits.

An instruction written in machine language consists of two parts. The first part is the operation to perform and the second part is the location or address where the data is stored, on which the operation is to be performed.

OPCODE	OPERAND
(Operation Code)	(Location/ Address)

### Advantages and Disadvantages of Machine Language

- ❖ Machine language allows you to use your computer quickly and efficiently. You don't need a translator to translate the code. The computer understands this directly.
- ❖ Machine dependent i.e. every machine has its own unique code. Program executed on one machine will not run in another.
- ❖ All operation codes have been remembered.

- ❖ All memory addresses have to memorize.
- ❖ Errors are difficult to correct or find in programs written in machine language.
- ❖ Very complex and difficult to understand.

### 1.3.2 Assembly Language

Set of instructions or a program can be easily written in alphanumeric character instead of 0's and 1's. Expressive and easily memorable characters are nominated for this purpose. For instance, MUL is used for multiplication and DIV for division, SUB for subtraction etc. Programs in which such characters are used such programs are said to be written in assembly language.

#### Advantages and Disadvantages of Assembly Language

- ❖ It is fast in speed and execution time is very less.
- ❖ Machine or hardware dependent. Program executed on one machine will not run in another.
- ❖ Writing large programs in assembly language is a tedious task.
- ❖ It is very complex and difficult to understand.
- ❖ More lines of code to be written even for a small program

### 1.3.3 High Level Language

High-level language was introduced to abolish the difficulties faced by the programmers in writing programs in machine language and assembly language. High-level language eliminated the major problem of the programmers by making program machine independent i.e. a program written on one machine could be executed on another machine without any difficulty.

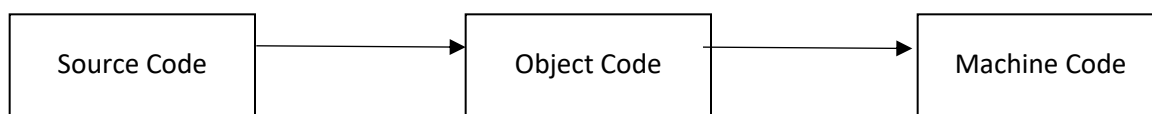
#### Advantages and Disadvantages of High-Level Language

- ❖ It is a convenient language because it is written in simple English words that can be easily understood and learned.
- ❖ No machine dependency and a program developed on one machine can be easily executed on another.
- ❖ Modification or editing is quite easy and simple.
- ❖ Debugging and Testing is simple.
- ❖ It has well defined syntax which need to be followed

## 1.4 Compilation

A programmer writes code in a language which is called a source language and the program is known as source program/ source code. The computer understands only machine code or object code. So there arise a need of translator that converts source language into machine code or the object code and this translator is known as compiler.

Compilation is a technique for converting source code to machine code. The compiler is used to assist in carrying out this task of conversion. The compiler checks the source code for any error whether it is related to syntax, ranges, limits before generating the object code. If the source code is free of such errors only than object code is generated otherwise the developer has to remove all such errors before repeating the process again.



The compilation process consists of four stages:

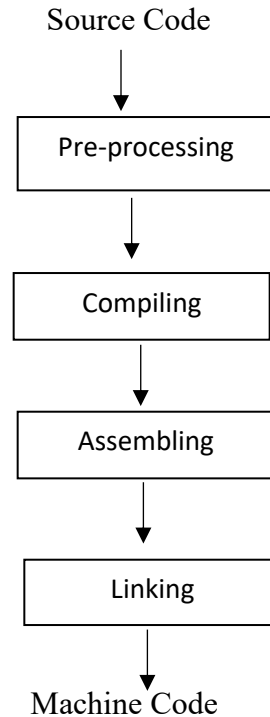
- a. Pre-processing
- b. Compilation
- c. Assembling
- d. Linking

**Pre-Processing:** In this stage, the developer written code is first passed to the pre-processor, which expands source code. After expanding the code, it is forwarded to compiler.

**Compiling:** At the stage of compiling, the source code is converted into assembly code using compiler.

**Assembling:** During assembling code in assembly language is transformed into object code with the support an assembler.

**Linking:** In this final stage, the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file.



## 1.5 Testing & Debugging and documentation

Testing and Debugging, these two tasks in software development have their own importance and these tasks are being carried out to ensure that the final software is error free and working as expected. Most of the time these two tasks are thought as same, however they are very different and have their own functionality.

The difference between the testing and debugging processes is that testing identifies software flaws but does not correct them. Debugging, as opposed to testing, is a more structured procedure where issues are not only found but also separated and fixed. Whereas, testing is the process of identifying flaws or mistakes in a piece of software; it can be carried out manually by a tester or automatically. Debugging is the process of fixing bugs that were discovered during the testing stage. Debugging is the responsibility of developers and programmers, and it cannot be automated.

Debugging is actually the process of correcting a software bug. It is the process of locating, analyzing, and eliminating errors. This activity starts when the programme doesn't work as it should and ends when the issue has been fixed and the software has been tested successfully. Errors must be fixed at every stage of debugging, which makes it a very difficult and time-consuming operation.

Testing on the other hand is the process of confirming and validating that the software or an application is bug free and satisfies the technical specifications established by its design and



development. Moreover, effectively and effectively fulfils the needs the user while managing all the exceptional and boundary values.

Documentation in software engineering is the canopy term that includes all written documents and materials dealing with software product development. All software development products, whether created by a small team or a large corporation, require some related documentation. Different types of documents are created through the whole (SDLC). Documentation exists to explain product functionality, unify project-related information, and allow for discussing all significant questions arising between stakeholders and developers.

**Documentation varies at different stages as under:**

Types of Software Documentation:

**a) Requirement Documentation:**

This document contains the requirements on the basis of which the software was developed and contains the description of how the software shall perform and which environment setup would be appropriate to have the best out of it. These are generated while the software is under development and is supplied to the tester groups too.

**b) Architectural Documentation:**

Architecture documentation is a special type of documentation that concerns the design. It contains very little code and is more focused on the components of the system, their roles and working. It also shows the data flows throughout the system.

**c) Technical Documentation:**

These contain the technical aspects of the software like API, algorithms etc. It is prepared mostly for the software developers. This form of documentation helps in choosing the technical manpower requirement when the maintenance and support is to be provided for the software.

**d) End-user Documentation**

As the name suggests these are made for the end user. It contains support resources for the end users. This kind of documentation in actual helps end user to understand the functionality of the software and its features so that the end user could use the available features as per need.

## Exercises

### Multiple Choice Questions

1. Which one of the following is a step for starting start development?
  - a. Understanding problem / requirement
  - b. Coding
  - c. Testing
  - d. None of the above
2. Which of the method is used by software development organization for program execution and flow before starting actual coding?
  - a. Algorithm
  - b. Flowchart
  - c. Any of the above
  - d. None of the above
3. An algorithm has a feature that:
  - a. Algorithm can be written in any language as per user choice
  - b. Algorithm has defined symbols
  - c. Algorithm uses specific language
  - d. None of the above
4. A flowchart has following feature that:
  - a. Flowchart explains program in a particular language and uses no symbols
  - b. Flowchart uses special symbols for program flow
  - c. Flowcharts can be drawn using any symbol as per user choice
  - d. None of the above
5. Machine language is written in:
  - a. C language
  - b. English Language
  - c. 0's and 1's
  - d. None of the above

### STATE WHETHER STATEMENT IS TRUE OR FALSE

1. Computer without the need of translator can understand high-level language. (T/F)
2. Machine language instruction has two parts OPCODE and OPERAND. (T/F)
3. Machine language is very easy to understand. (T/F)
4. Compiler is used to translate High-level language to Machine Language. (T/F)
5. Testing and Debugging are the same processes. (T/F)

### FILL IN THE BLANKS

1. Compilation is a technique to translate source code into \_\_\_\_\_.
2. Debugging is a process to identify the \_\_\_\_\_ and fix them.
3. High-level language is machine \_\_\_\_\_.
4. Assembly language uses \_\_\_\_\_ instead of 0's and 1's.
5. Written record of the software development is called \_\_\_\_\_.

## Chapter 2

### Algorithms and Flowcharts

#### 2.1 Flowchart symbols

Flowchart use different shapes to expression flow and steps in process involved. These shapes are call flowchart symbols.

The symbols used and their description:

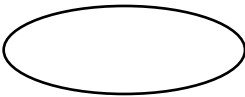



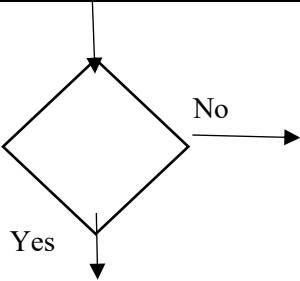
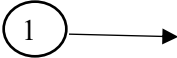
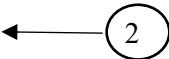
Symbol	Shape	Symbol Usage
	Oval	Used for starting and ending the flowchart.
	Parallelogram	Used for taking input from the users or other external sources and displaying/sending output to the terminal.
	Rectangle	Used for representing the processing being carried out.
	Arrow	Used for expressing the flow/direction/sequence of execution.
	Diamond	Used for making decision and to choose path accordingly.
	Circle	Used for Entry Connector
	Circle	Used for Transfer Connector

Table 1: Flowchart Symbols

## 2.2 Basics of flowcharts /algorithm for sequential processing

Writing a logical step-by-step method to solve the problem is called the . In other words, an algorithm is a procedure for solving problems. In order to solve a mathematical or computer problem, this is the first step in the process. An algorithm includes calculations, reasoning, and data processing. Algorithms can be presented by natural languages, pseudocode, and flowcharts, etc.

A is the graphical or pictorial representation of an algorithm with the help of different symbols, shapes, and arrows to demonstrate a process or a program. With algorithms, we can easily understand a program. The main purpose of using a flowchart is to analyse different methods.

Algorithm	Flowchart
It is a procedure for solving problems.	It is a graphic representation of a process.
The process is shown in step-by-step instruction.	The process is shown in block-by-block information diagram.
It is complex and difficult to understand.	It is intuitive and easy to understand.
It is convenient to debug errors.	It is hard to debug errors.
The solution is showcased in natural language.	The solution is showcased in pictorial format.
It is somewhat easier to solve complex problem.	It is hard to solve complex problem.
It costs more time to create an algorithm.	It costs less time to create a flowchart.

Table 2: Difference Between Algorithm and Flowchart

## 2.3 Decision based processing and iterative processing

### 2.3.1 Decision-based Processing

Conditional/Selection logic, also known as decision logic, is used for making decisions. It is used for selecting the appropriate path out of the two or more alternative paths in the program logic. It is depicted as either an IF...THEN...ELSE or IF.....THEN structure. The flowchart shown below illustrate the logic of these structures.

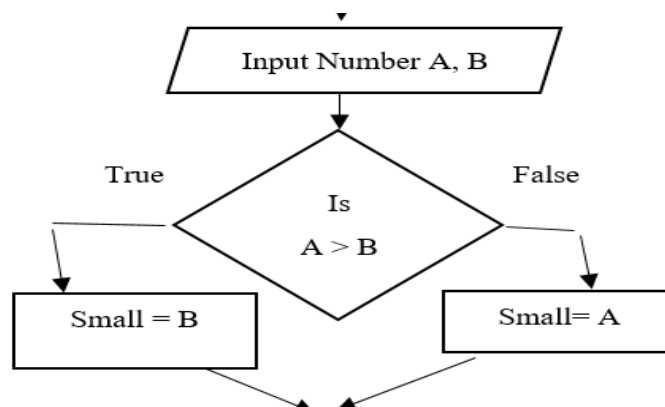


Figure 1: Decision-Based Processing

Depending on the result received after the comparison of A and B the next step is executed. This is an example of decision based processing.

### 2.3.2 Iterative (Looping) Processing

Many jobs that are required to be done with the help of a computer are repetitive in nature. For example, the calculation of the salary of different workers in a factory is done by the (No. of hours worked) x (wage rate). This calculation will be performed by an accountant for each worker every month. Such types of repetitive calculations can easily be done using a program that has a loop built into the solution of the problem. The loop is defined as a block of processing steps repeated a certain number of times. An endless loop repeats infinitely and is always the result of an error. Figure below illustrates a flowchart depicting the concept of looping. It shows the flowchart for printing values 1, 2, 3..., 20.

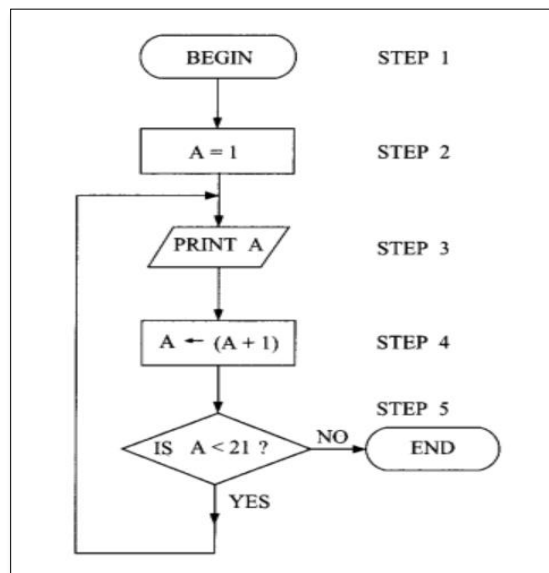


Figure 2: Iterative Processing

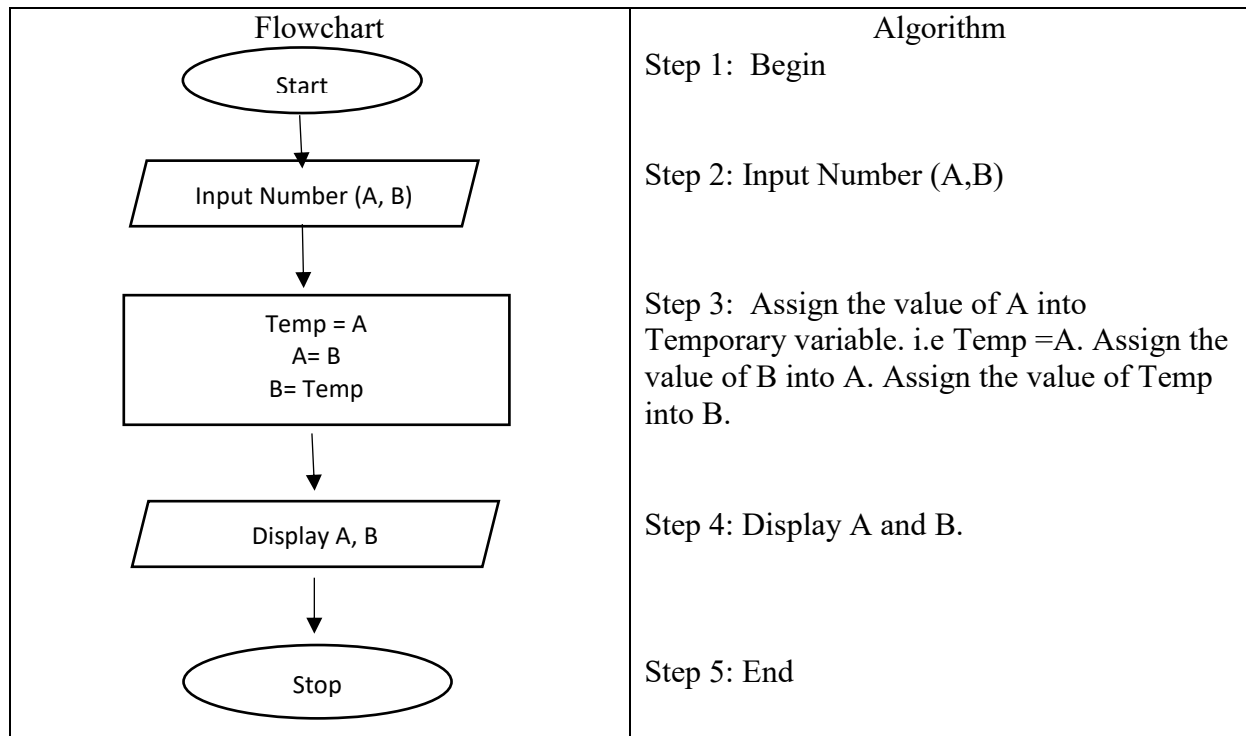
In above figure the current value of A is compared with 21. If the current value of A is less than 21, steps 3 and 4 are repeated. As soon as the current value of A becomes more than 21, the path corresponding to “NO” is followed, and the repetition process stops.

### Exchanging values of two variables using flowchart and algorithm.

Exchanging values of two variable means that our aim is to interchange the values of the two given variable.

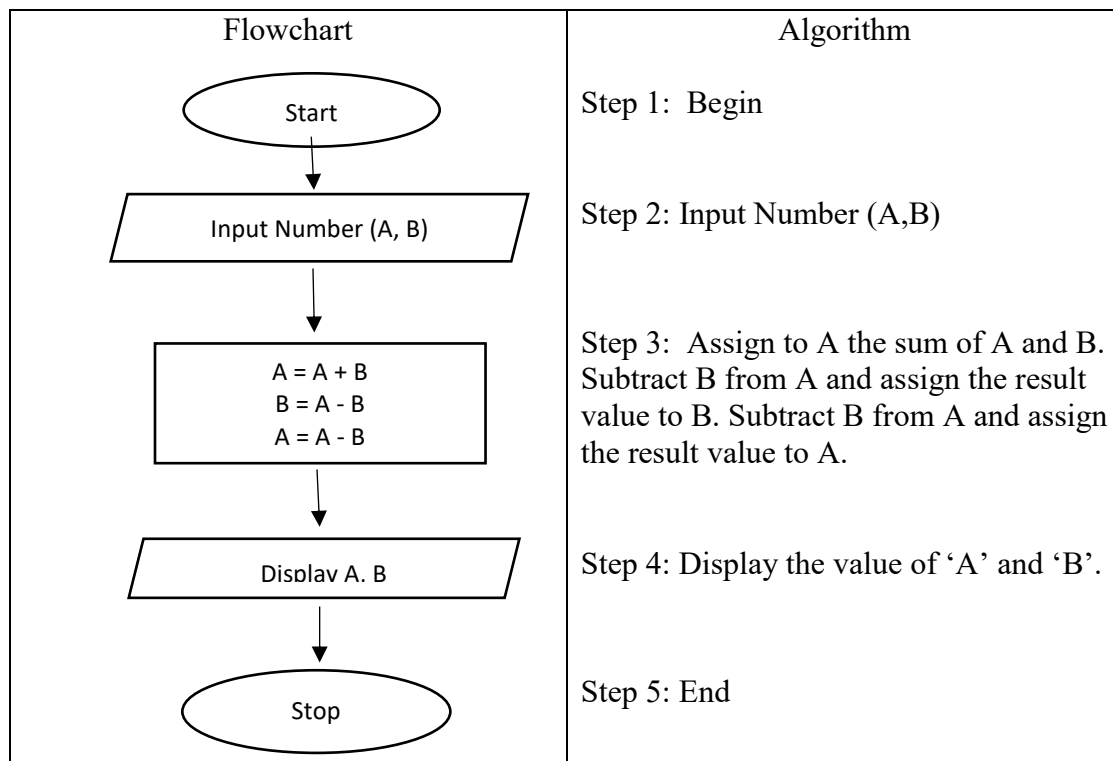
Suppose we have two variable A and B and having original values of 10 and 15 respectively. So, after interchange the values of A should be 15 and B should be 10.

## Solution1: Exchanging values using a temporary variable



Example 1: Exchanging values of two variables

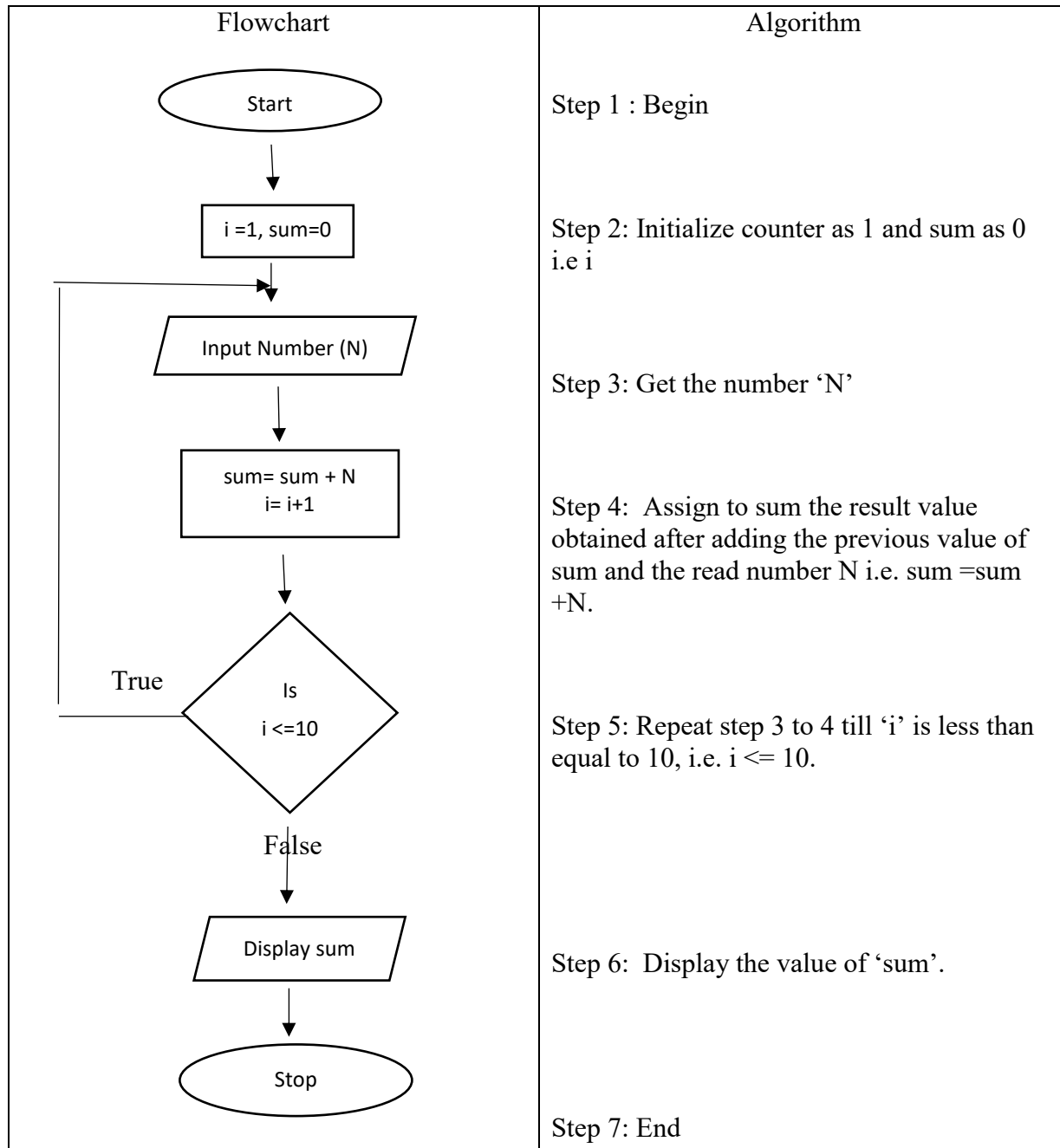
## Solution 2: Exchanging values without using a temporary variable



Example 2: Exchanging values without variables

**Summation of a set of numbers using flowchart and algorithm.**

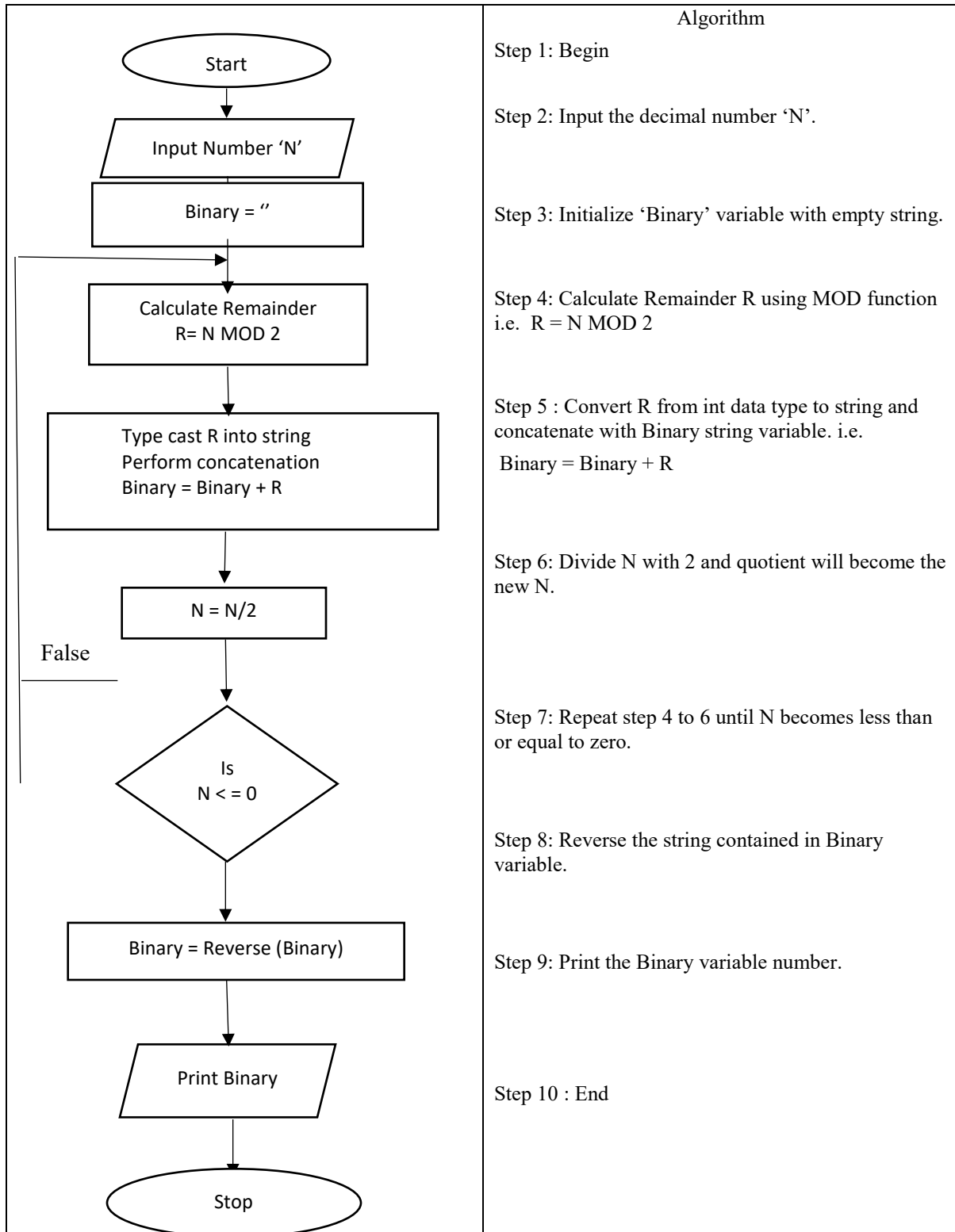
Let us take an example that we have to take ten numbers from the user and display sum of these ten numbers



Example 3: Summation of a set of numbers

**Decimal to Binary Base Conversion using flowchart and algorithm.**

Decimal number is converted into binary by dividing the number successively by 2 and printing the remainder in reverse order.

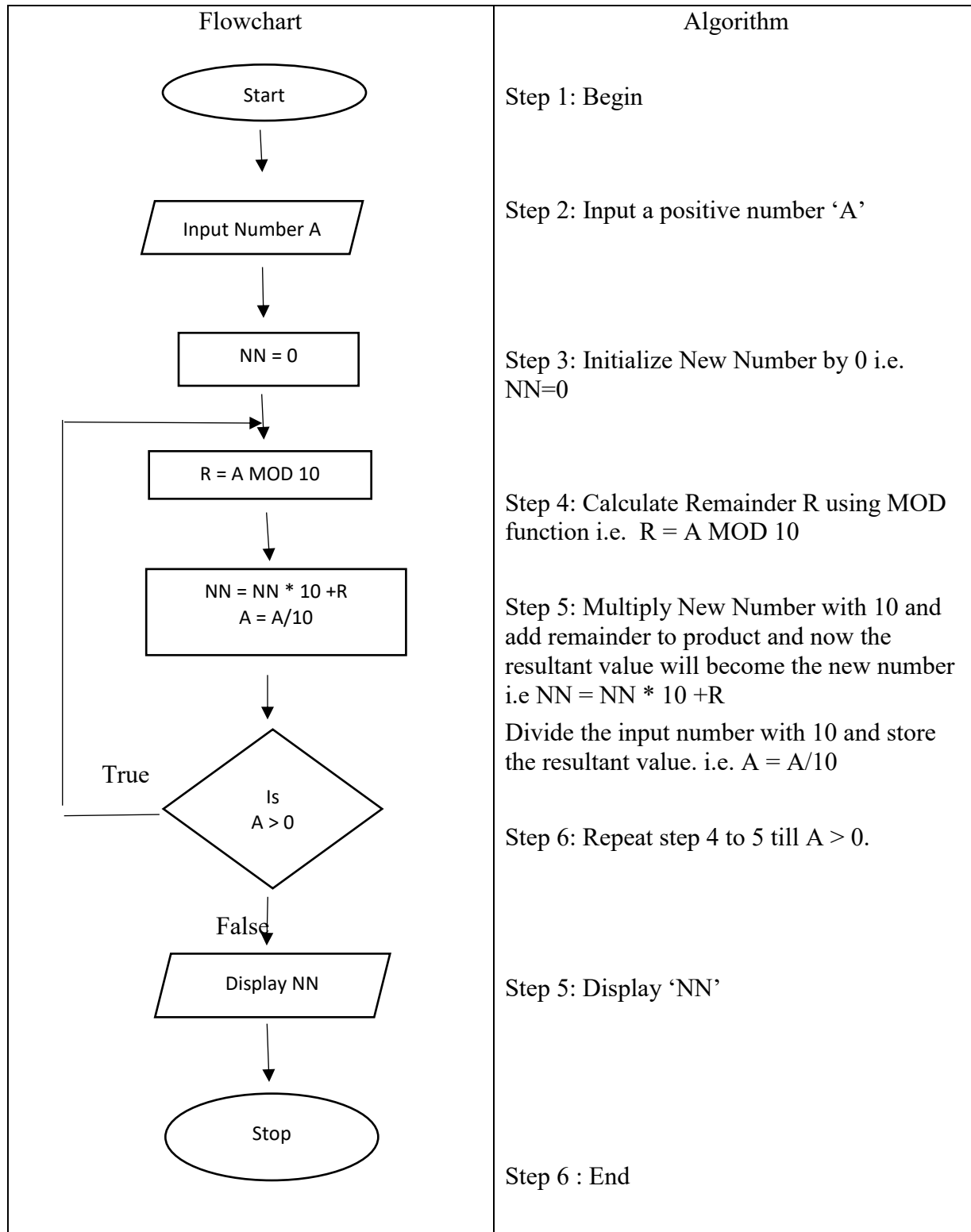


Example 4: Decimal Base to Binary Base conversion



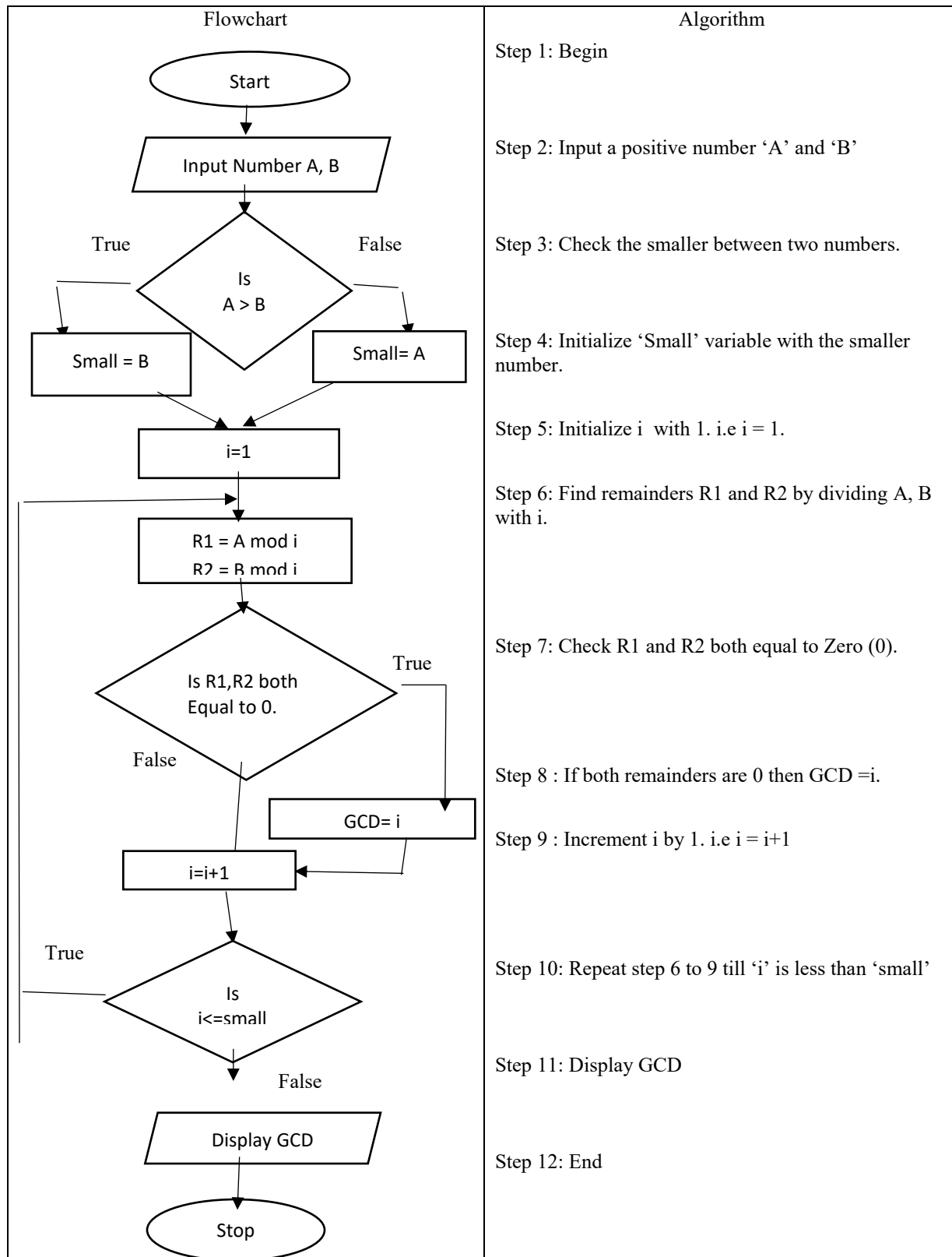
**To perform Reversing Digits of Number using flowchart and algorithm.**

Target is take an input number like 1234 and generate the reverse as 4321.



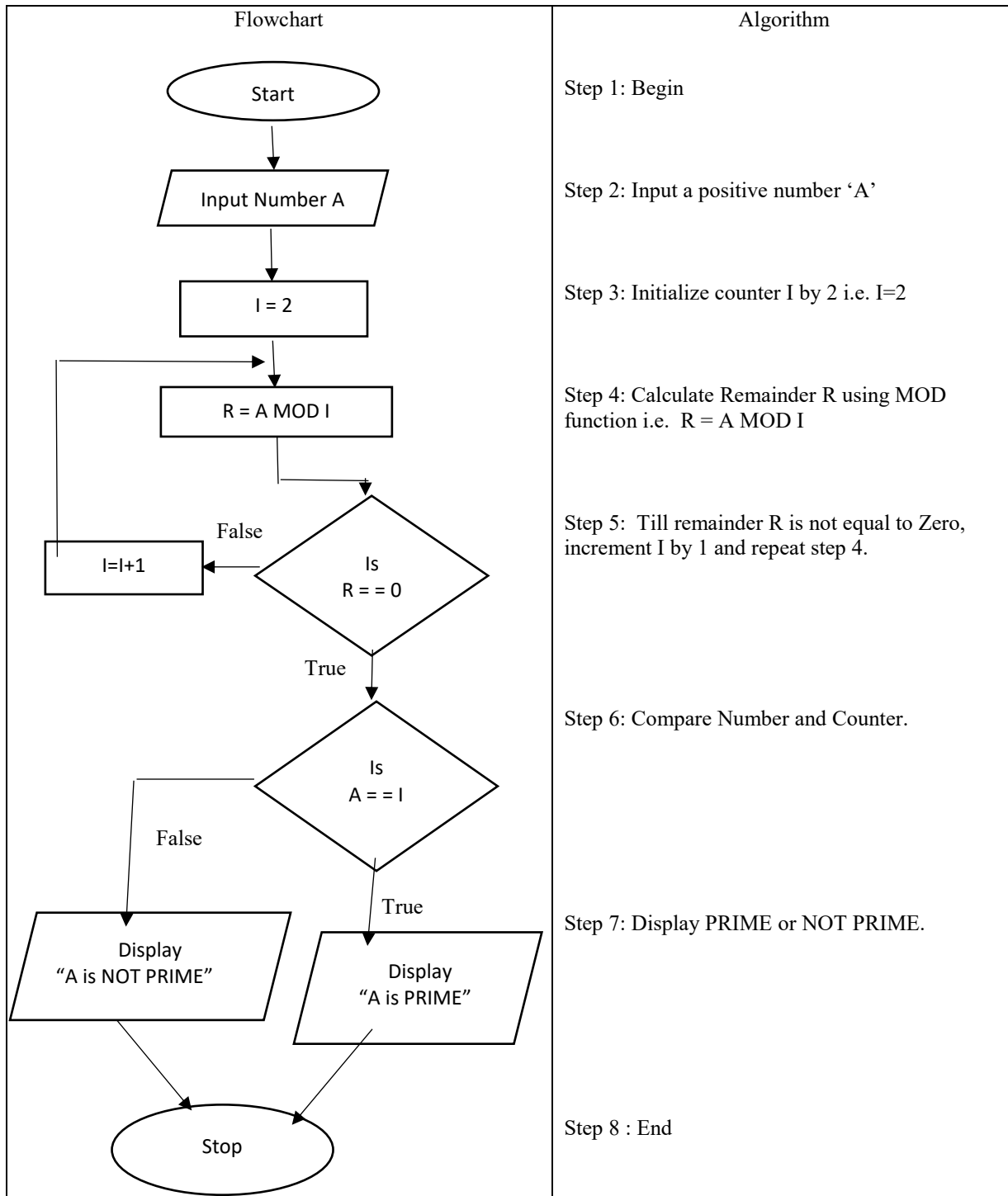
Example 5: Reverse digits of an integer

## To find Greatest Common Divisor using flowchart and algorithm



Example 6: GCD (Greatest Common Divisor) of two numbers

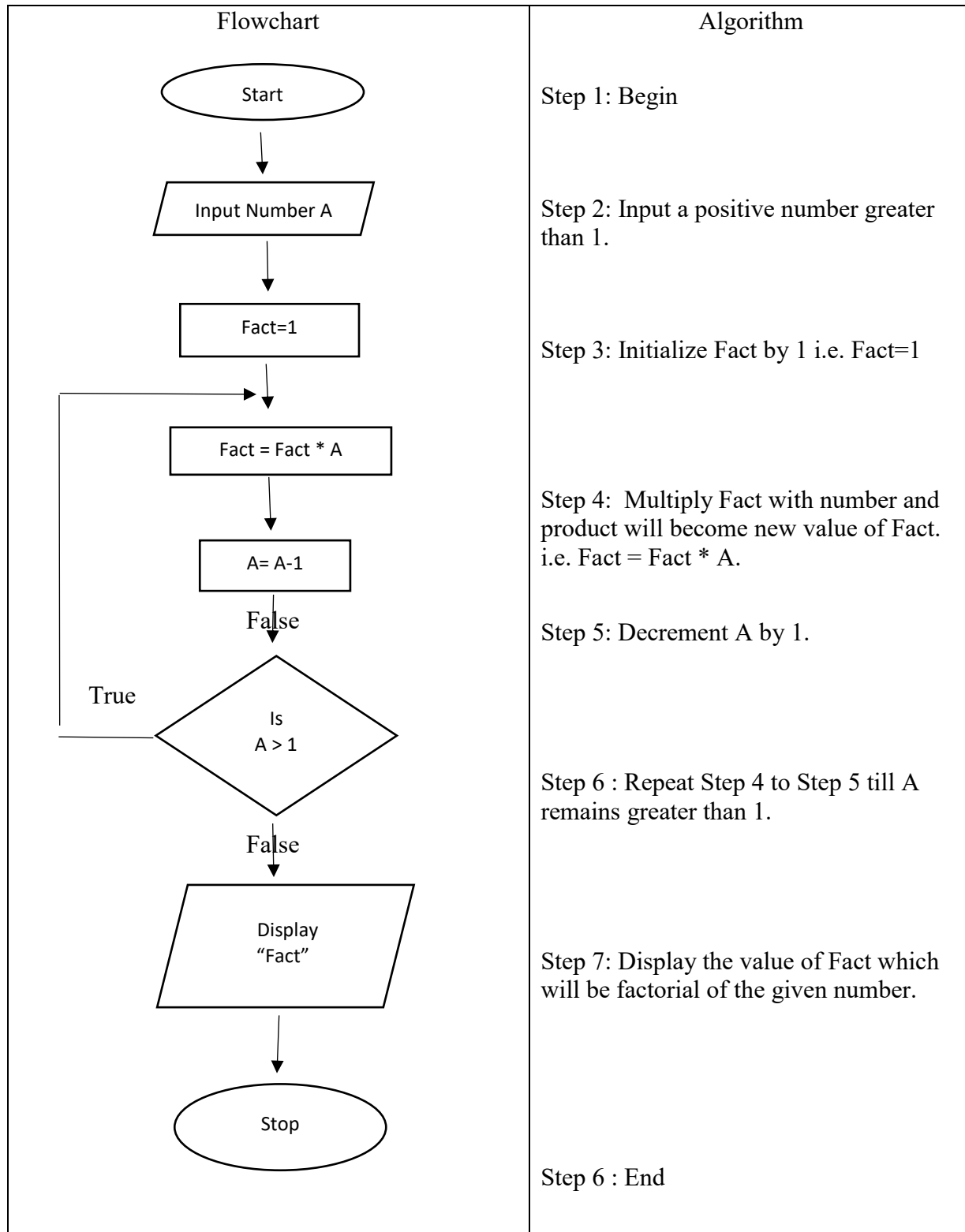
Flowchart and algorithm to test whether a number is a Prime Number.



Example 7: Test whether a number is prime

**Flowchart and algorithm to calculate Factorial of Number:**

Suppose factorial of 4 is to be calculated then it will be  $24 = 4 \times 3 \times 2 \times 1$ .

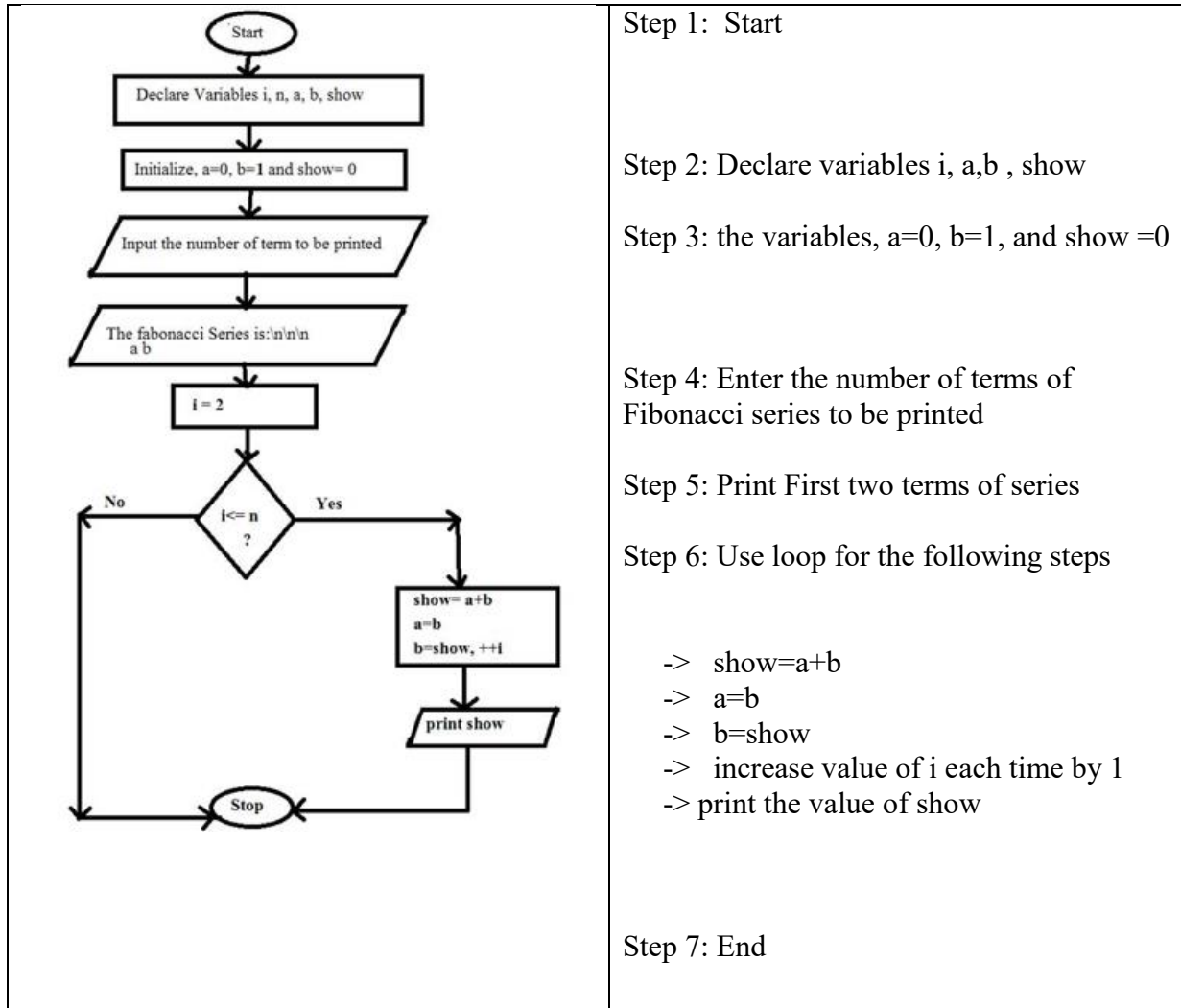


Example 8: Factorial Computation

**Flowchart to find Fibonacci Series:**

A series written as:

0,1,1,2,3,5,8.... is called Fibonacci series. In this series current number is the sum of previous two numbers.



Example 9: Fibonacci sequence

**Algorithm for reversing an array.**

Suppose we are having an array with 5 elements as below:

Arr[5] = 

1	3	5	7	9
---	---	---	---	---

Target is to reverse the contents of array as under:

Arr[5] = 

9	7	5	3	1
---	---	---	---	---

The algorithm to complete the desired task is as given below:

<p>Step 1: Begin</p> <p>Step 2: Input an array.</p> <p>Step 3: Initialize <math>i=0</math> , <math>L = \text{Length of array}</math></p> <p>Step 4: Repeat steps 5 to till <math>i &lt; L</math></p> <p>Step 5: <math>\text{temp} = \text{array's } L-1 \text{ element}</math></p> <p>Step 6: <math>\text{array's } L-1 \text{ element} = \text{array}[i] \text{ element}</math></p> <p>Step 7: <math>\text{array}[i] \text{ element} = \text{temp}</math></p> <p>Step 8: Increment 'i' and decrement L</p> <p>Step 9: Display reverse array</p> <p>Step 10 : End</p>
---

**Algorithm for finding largest of in a given array.**

The below given algorithm will find the largest element and smallest element in given array.

Step 1: Input the array elements.

Step 2: Initialize small = large = arr[0]

Step 3: Repeat from i = 2 to n

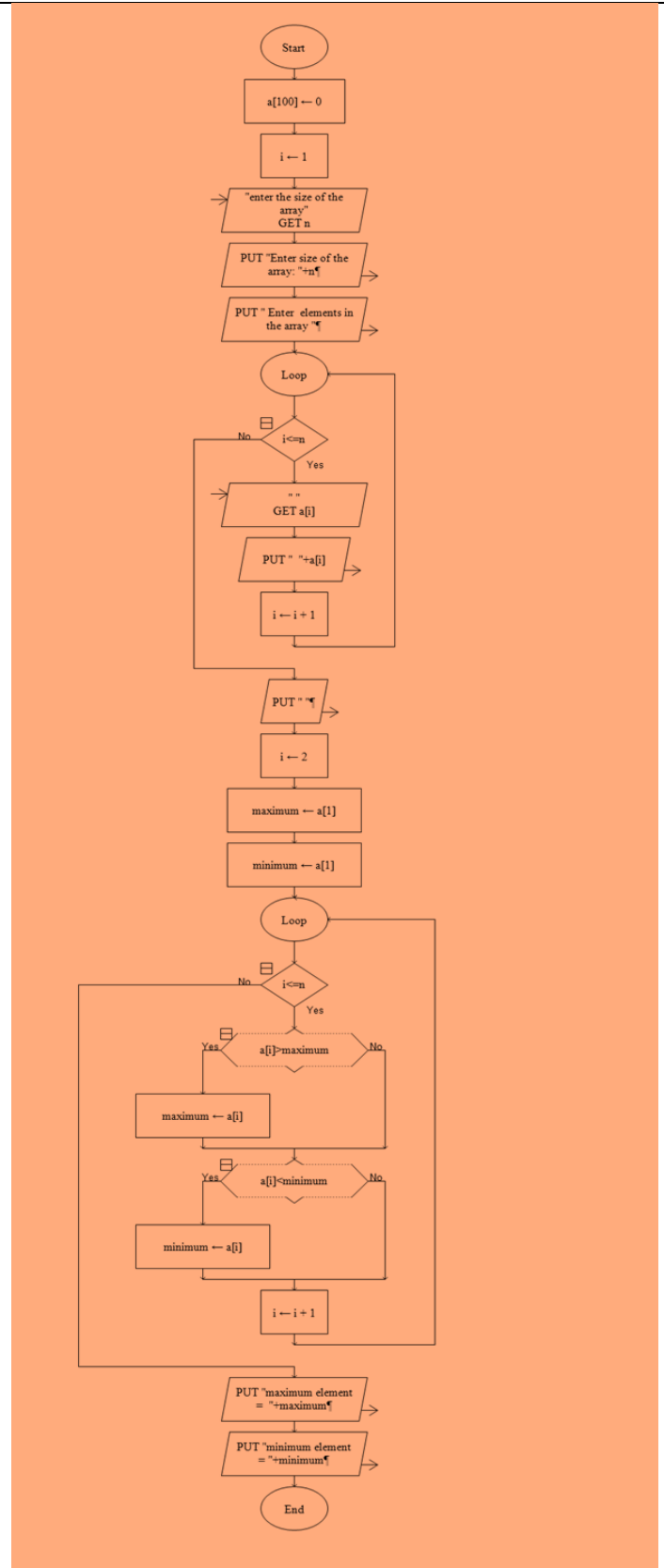
Step 4: if(arr[i] > large)

Step 5: large = arr[i]

Step 6: if(arr[i] < small)

Step 7: small = arr[i]

Step 8: Print small and large.



Example 10: Find largest number in an array

## Exercises

### Multiple Choice Questions

- 1) Flowcharts uses \_\_\_\_\_ symbol for input.
  - a. Oval
  - b. Square
  - c. Triangle
  - d. None of the above
- 2) Flowcharts uses \_\_\_\_\_ symbol for decision making.
  - a. Oval
  - b. Rectangle
  - c. Diamond
  - d. None of the above
- 3) Algorithm uses which symbol for processing.
  - a. Rectangle
  - b. Oval
  - c. Triangle
  - d. None of the above
- 4) To repeat some fixed steps until a condition is met is called \_\_\_\_\_.
  - a. Decision based processing
  - b. Iterative processing
  - c. Sequential processing
  - d. None of the above
- 5) Iteration is also called \_\_\_\_\_.
  - a. Decision based processing
  - b. Looping
  - c. Any of the above
  - d. None of the above

### State whether statement is true or false

- 1) Flowcharts cannot be used for demonstrate iterative processing. (T/F)
- 2) Algorithm are only used for decision bases processing demonstration. (T/F)
- 3) Iterative processing means to repeat a fixed steps until a condition is met. (T/F)
- 4) Sequential processing and Decision based processing are same. (T/F)
- 5) Sequential processing and Iterative based processing are same.

### Practice Questions

- 1) Draw a flowchart to write table of 2.
- 2) Draw a flowchart to find area and perimeter of a rectangle.
- 3) Draw a flowchart to find maximum and minimum number from given three numbers.
- 4) Write an algorithm to find cube and square of number.
- 5) Write an algorithm to find even and odd number.



## Chapter 3

### Programming with Python

#### 3.1 Python Introduction

##### 3.1.1 Technical strength of Python

In today's computer world Python is gaining popularity day by day and big companies including Google, Yahoo, Intel, IBM etc. are widely using Python. So many reasons exist for the popularity Python from its availability to ease of use.

Python has the following technical characteristics:

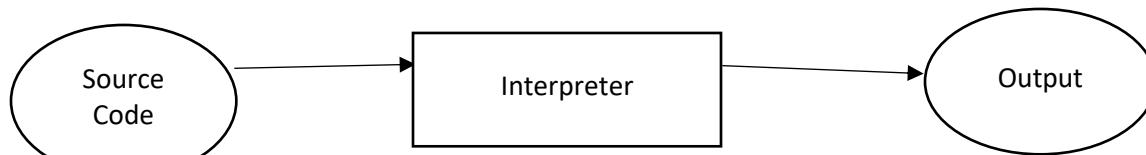
- a. **Free and Open Source:** Python is free to use and available to download from its official website
- b. **Easy-to-learn:** Python is comparatively very easy to learn and use than many other computer languages. The syntax, structures, keywords etc. used in Python are very simple and easy to understand.
- c. **Extensive Libraries:** Library is the strength of Python. When we download Python it comes with the huge library having immense inbuilt modules which makes coding easier and saves valuable time.
- d. **Portable:** Key strength of python is its portability. Users can run python programs on various platforms. Suppose you wrote a program in windows and now you want to run this program on Linux or Mac Operating system, You can easily run your programs on (Windows, Mac, Linux, Raspberry Pi, etc). You can say Python is a platform-independent programming language.
- e. **Interpreted:** Python is interpreted language, which means it does not require any kind of compiler to run the program. Python converts its code into bytecode, which gives instant results. Python is interpreted means that its code is executed line by line, which makes it easier to debug.
- f. **Object-Oriented:** Python can be used as an object-oriented language in which data structure and functions are combined in a single unit. Python supports both object-oriented and procedure-oriented approach in the development. The object-oriented approach deals with the interaction between the objects on the other hand procedure-oriented approach deals with functions only.
- g. **GUI Programming:** Python provides many solutions to develop a Graphical User Interface (GUI) very fast and easily.
- h. **Database Connectivity:** Python supports all the database required for the development of various projects. Programmers can pick the best suitable database for their projects. Few databases which are supported by Python are MySQL, PostgreSQL, Microsoft SQL Server etc.

##### 3.1.2 Introduction to Python Interpreter and program execution

An interpreter is a kind of program that executes other programs. When you write Python programs, it converts source code written by the developer into intermediate language which is again translated into the native language / machine language that is executed.

The python code you write is compiled into python bytecode (0's and 1's), which creates file with extension “.py”. The bytecode compilation happened internally, and almost completely hidden from developer. Compilation is simply a translation step, and byte code is a lower-level, and platform-independent, representation of your source code. Roughly, each of your source statements is translated into a group of byte code instructions. This byte code translation is performed to speed execution byte code can be run much quicker than the original source code statements.

Python is an interpreted language since the programs written in Python are executed using an interpreter not by a compiler. In case of the languages like C, C++ the program written are compiled first and the source code is converted into byte code i.e. (0's and 1's).



Python, doesn't need to be converted into binary. You just run the program directly from the source code.

### 3.1.3 Using comments

Commenting your code helps explain your thought process, and helps you and others to understand later about your code and flow of program. This allows you to more easily find errors, to fix them, to improve the code later on, and to reuse it in other applications as well.

Commenting is important to all kinds of projects, no matter whether they are small, medium, or large. It is an essential part of your workflow, and is seen as good practice for developers. Without comments, things can get confusing, real fast.

**3.1.3.1 Single-Line Comments: Such comment starts with a hash character (#), and is followed by text that contains further explanations.**

```
# Program to add two numbers
a = 5
b = 7
c = a + b
print(c)
```

In above example, single-line comment has been written using (#) statement in the beginning. By writing comments the user could easily understand that the program has been developed for adding two numbers.

### 3.1.3.2 Multiple-Line Comments: We can add a multiline string (triple quotes) in your code, and place your comment inside it as explained below:

```
"""
Program will do :
1. Addition
2. Subtraction
"""
a = 5
b = 7
c = a + b
```

### 3.1.4 Literals

Literal is actually a data value assigned to a variable or given in a constant. Like 29, 1, “Python”, ‘Yes’ etc. Literals supported by Python:

#### 3.1.4.1 String Literals

Strings literals are formed by surrounding text between single or double quotes. Example, ‘Python’, “Hello World”, ‘We are learning Python’ etc. Strings are sequence of characters and even numeric digits are treated as characters once enclosed in quotes. Multiline string literals are also allowed like

```
“This is world of programming
Python is a best to learn
We are working”
```

#### 3.1.4.2 Numeric Literals

Python supports the following types of numeric literals:

```
A = 99          # Integer literal
B = 21.98       # Float literal
C = 5.13j       # Complex literal
```

#### 3.1.4.3 Boolean Literals

Booleans literals are also supported Python in which have the values in True or False. Like

```
X = True
Y = False
```

### 3.1.5 Constants

Constants are those items which holds the values directly and these values can’t be changed during the execution of the program thus they are called as constants. For example,

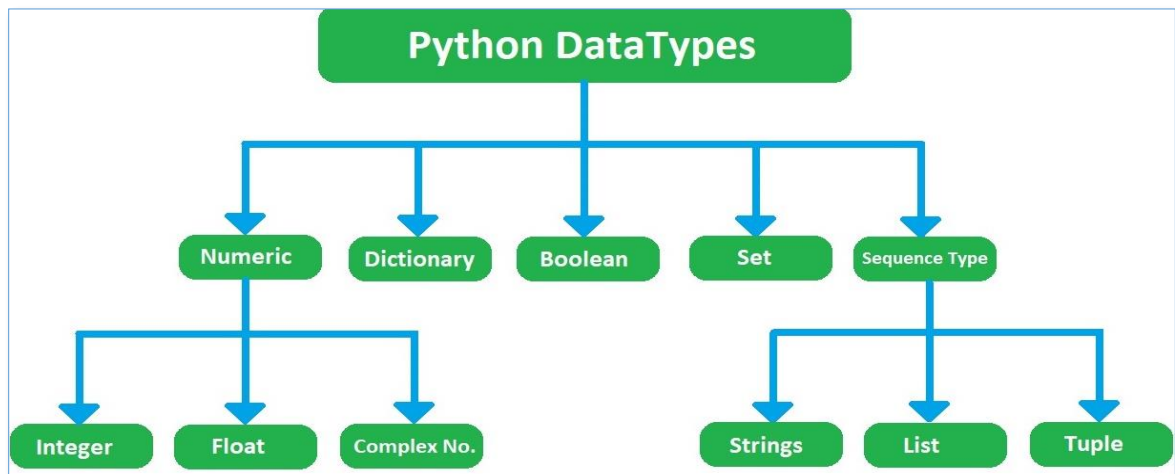
```
>>> print(123)
>>> print(23.56)
>>> print(“Python World”)
```

Now, during the execution of the program the value of 123 or 23.56 or “Python World” cannot be modified. When you run the program the output will be:

Program	Output	Explanation
>>> print(123)	123	Numeric constant digits are directly printed as it is on the monitor as output.
>>> print(23.56)	23.56	Float constant digits are directly printed as it is on the monitor as output along with the decimals.
>>> print(“Python World”)	Python World	In case of string literals, the string or characters enclosed in quotes are printed only i.e. output is printed on without quotation marks on the screen.

### 3.1.6 Python Built in Data Types

Variables can store data of different types, and different types can do different things. Python supports the following built in data types:



#### 3.1.6.1 Numeric

Numeric data types are used to store the numeric values in a variable. For example,

A = 99

B = 21.5

Numeric data is of various types so accordingly the numeric data types are of the following types

- int
- float
- complex

**int(integer):** Whole numbers are stored in variables as integer data type. For example,

```
a = 12
b = 16
```

**float:** The numbers having decimal are stored as float data types. For example,

```
a = 12.6
b = 17.9
```

**Complex:** Complex data type is used to contain complex numbers, which have real and imaginary part. For example,

```
a = 5 + 6j
b = 8 + 9j
```

```
A = 12
B = float(15.6)
C = 5 + 7j
print(A)
print(B)
print(C.real, C.imag)
```

Output:

```
12
15.6
5.0 7.0
```

### 3.1.6.2 Dictionary

Python provides another composite called a **dictionary**, which is similar to a list in that it is a collection of objects.

Dictionaries are Python's implementation of a data structure that is more generally known as an associative array. A dictionary consists of a collection of key-value pairs. Each key-value pair maps the key to its associated value.

You can define a dictionary by enclosing a comma-separated list of key-value pairs in curly braces (`{}`). A colon (`:`) separates each key from its associated value:

```
# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
```

```
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}

# using dict()
my_dict = dict({1:'apple', 2:'ball'})

# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
```

As you can see from above, we can also create a dictionary using the built-in dict() function.

### 3.1.6.3 Boolean

**Python Boolean** type is one of the built-in data types provided by Python, which represents one of the two values i.e. True or False. Generally, it is used to represent the truth values of the expressions. For example,  $1==1$  is True whereas  $2<1$  is False. For Example:

```
a = 10
b = 20
print(a == b)
```

**Output:**

**False**

### 3.1.6.4 Set

Set is another data type in Python and it works as mathematical sets. Very much similar to lists and sets are denoted as curly brackets. **Set** is an unordered collection of data types that is iterable, mutable and has no duplicate elements. For example:

```
result = set([1, 2, 4, 4, 3, 3, 3, 6, 5])
print(result)
```

**Output:**

{1, 2, 3, 4, 5, 6}

### 3.1.6.5 Sequences

Sequences are containers with items stored in a deterministic ordering. Each sequence data type comes with its unique capabilities.

There are many types of sequences in Python.

#### Types of Sequence in Python:

##### a) Strings:

In python, the string is a sequence of Unicode characters written inside a single or double-quote. Python does not have any char type as in other languages (C, C++), therefore, a single character inside the quotes will be of type str only.

1. To declare an empty string, use `str()` or it can be defined using empty string inside quotes.

Example of String in Python:

```
name = "LearnPython"
print(name)
```

**Output:**

```
LearnPython
```

2. Strings are immutable data types, therefore once declared, we can't alter the string. Though, we can reassign it to a new string.

**b) Lists:**

Lists are a single storage unit to store multiple data items together. It's a mutable data structure, therefore, once declared, it can still be altered.

A list can hold strings, numbers, lists, tuples, dictionaries, etc.

1. To declare a list, either use `list()` or square brackets `[]`, containing comma-separated values.

Example of Lists in Python:

```
list_1 = ["LearnPython ", "Sequences", "Tutorial"] # [all string list]
print(f'List 1: {list_1}')
```

```
list_2 = list() # [empty list]
print(f'List 2: {list_2}')
```

```
list_3 = [2021, ['hello', 2020], 2.0] # [integer, list, float]
print(f'List 3: {list_3}')
```

```
list_4 = [{'language': 'Python'}, (1,2)] # [dictionary, tuple]
print(f'List 4: {list_4}')
```

**Output:**

```
List 1: ['LearnPython', 'Sequences', 'Tutorial']
```

```
List 2: []
```

```
List 3: [2021, ['hello', 2020], 2.0]
```

```
List 4: [{'language': 'Python'}, (1, 2)]
```

**c) Tuples:**

Just like Lists, Tuples can store multiple data items of different data types. The only difference is that they are immutable and are stored inside the parenthesis `()`.

To declare a tuple, either use `tuple()` or parenthesis, containing comma-separated values.

Example of Tuple in Python:

```
tuple_1 = ("LearnPython ", "Sequences", "Tutorial") # [all string tuple]
print(f'tuple 1: {tuple_1}')
```

```
tuple_2 = tuple() # [empty tuple]
print(f'tuple 2: {tuple_2}')
```

```
tuple_3 = [2021, ('hello', 2020), 2.0] # [integer, tuple, float]
print(f'tuple 3: {tuple_3}')
```

```
tuple_4 = [{'language': 'Python'}, [1,2]] # [dictionary, list]
print(f'tuple 4: {tuple_4}')
```

**Output:**

```

tuple 1: ('PythonGeeks', 'Sequences', 'Tutorial')
tuple 2: ()
tuple 3: [2021, ('hello', 2020), 2.0]
tuple 4: [{'language': 'Python'}, [1, 2]]

```

**3.1.7 Python constructs****3.1.7.1 Assignment Statement**

Assignment statement serves various purpose and used for creating a variable, initializing a variable or modifying the value of an existing variable.

The operator used for assignment is “=” also known as assignment operator.

Variable placed on the Left Hand Side (L.H.S) of the assignment operator is set with the constant value or value of another variable present on the Right Hand Side (R.H.S) of the assignment operator.

**Example 1. Program to explain the assignment statement and initialization.**

Consider a variable A whose value is to be initialized with 10 then the syntax would be:

```

A = 10
print (A)
Output:
10

```

In above example, the constant value 10 is assigned to the variable A and it can be verified with the print statement that A is holding the value 10.

**Example 2. Program to explain the assignment statement.**

```

A = 10
B = A
print(B)
Output:
10

```

In above example, A is assigned the value of 10 and thereafter B is assigned with the value of A i.e. so in result B will be holding the value i.e. 10 as shown in the above code.

**3.1.8 Expressions**

Expressions are used to obtain the desired intermediate or final results. Expressions means combination of values, which can be constants, strings, variables and operators. Few examples of expressions are as follows:

```

12 + 3
12 / 3 * (1+2)
12 / a
a * b * c

```



With the above example, it is clear that expressions are combination of operands and operators and produce desired final or intermediate results. In examples given 12, 3, 1, 2, a, b, c are operands and '+', '/', '\*\*' are operators.

Expressions are written on the RHS of the assignment operator and their result value is stored in a variable for future reference.

**Example 3. Program for explaining the working of expressions.**

```
A = 2 + 3
B = A * 5
C = B / A
D = (A + B) - (C + A)
print(A)
print(B)
print(C)
print(D)
```

Output:

```
5
25
5
20
```

In above example, the value of A is printed as 5 after evaluating the expression '2+3', value of B is 25 and assigned after evaluating the expression 'A \* 5' since the value of A is '5'. Value of C is evaluated using an expression where both operands are variables and D is evaluated using more than one expression.

### 3.1.9 Arithmetic Operator

Operators are applied on the operand to obtain the desired results and there are different types of operators. The arithmetic operators are the most basic operators used for in general calculations or arithmetic operations. The arithmetic operators include +, -, /, \*, % (modulus), \*\* (exponent) etc.

**Example 4. Explaining Binary Operators with Program**

```
A = 2 + 2
B = 5 * 2
C = 10 / 2
D = 10 % 2
print(A)
print(B)
print(C)
print(D)
```

Output:

```
4
10
5
0
```

In above example, all operators are binary operators i.e. operators are applied on two operands to obtain the desired output. The modulus (%) operator returns the remainder value of the division process when 10 is divided by 2 and store the remainder value in the variable D.

**Example 5. Explaining Binary Operators with Program**

```
#Explaining Binary Operators with Program
```

```
A = 6+2  # Addition Operator
B = 6-2  # Subtraction Operator
C = 6 * 2 # Multiplication Operator
D = 6/2  # Division Operator
E = 6%2  # Modulus Operator
F = 6**2 # Exponential Operator
```

```
print("Sum of 6+2 is: ", A)
print("Subtraction of 6-2 is: ", B)
print("Multiplication of 6*2 is: ", C)
print("Division of 6/2 is: ", D)
print("Modulus of 6%2 is: ", E)
print("Exponential of 6**2 is: ", F)
```

```
#End of Program
```

Output:

```
Sum of 6+2 is: 8
Subtraction of 6-2 is: 4
Multiplication of 6*2 is: 12
Division of 6/2 is: 3.0
Modulus of 6%2 is: 0
Exponential of 6**2 is: 36
```

**Example 6. Program to calculate the profit of a businessperson.**

```
# Program to calculate the profit of a businessperson.
```

```
CP = float(input("Enter the Cost Price of the Item: "))
SP = float(input("Enter the Selling Price of the Item: "))
```

```
P = SP - CP
```

```
print("Profit earned is: ", P)
```

```
#End of Program
```

Output:  
Enter the Cost Price of the Item: 100  
Enter the Selling Price of the Item: 150  
Profit earned is: 50.0

**Example 7. Program to calculate area of rectangle.**

```
#Program to calculate area of rectangle

length = int(input("Enter the length: "))
width = int(input("Enter the width: "))
area = length * width
print("Area of Rectangle is : ",area)

#End of Program
Output:
Enter the length: 5
Enter the width: 4
Area of Rectangle is: 20
```

**Example 8. Program to explain the functionality of Modulus Operator (%).**

```
#Program to explain the functionality of Modulus Operator (%)

num1 = int(input("Enter the first number : "))
num2 = int(input("Enter the second number : "))

quotient = num1/num2
remainder = num1%num2

# Division operator (/) returns the quotient after dividing num1 by num2
# Modulus operator (%) returns the remainder after dividing num1 by num2

print("Quotient is : ",quotient)
print("Remainder is: ",remainder)
#End of Program

Output:
Enter the first number : 15
Enter the second number : 3
Quotient is : 5.0
Remainder is: 0
```

**Example 9. Program to explain the operator precedence.**

```
#Program to explain the operator precedence.
```

```
A = (4+5)/3*2-1
```

```
B = 6/3+2*(3+2)
```

```
print("Value of A is: ",A)
```

```
print("Value of B is: ",B)
```

```
#End of Program
```

Output:

Value of A is: 5.0

Value of B is: 12.0

The operator precedence from highest to lowest is as under:

- i. () (Parenthesis)
- ii. \*\* (Exponential)
- iii. – (Negation)
- iv. / (Division) \* (Multiplication), % (Modulus)
- v. + (Addition) - (Subtraction)

According to operator precedence the value of A and B is calculated after evaluating the expressions as under:

Value of A is calculated as :	Value of B is calculated as :
Step 1: (4+5) is evaluated and result is 9.	Step 1: (3+2) is evaluated and result is 5.
Step 2: 9/3 is evaluated and result is 3.	Step 2: 6/3 is evaluated and result is 2.
Step 3: 3 * 2 is calculated and result is 6.	Step 3: 2*5 is calculated i.e. result is 10.
Step 4: 6-1 is calculated and result is 5.	Step 4: 2 + 10 is calculated and result is 12.

**3.1.10. Relational Operator**

The purpose of relational operator is to compare the values of the operands and find the relation among the operands. The relational operators are binary operators since the comparison can be performed between two operands. Upon comparing the operands, the relational operators return Boolean value which is either True or False.

The relational operators are as follows:

Symbol	Function Perform	Format	Output
>	Greater than	$x > y$	Returns True if x is greater than y; else False
<	Less than	$x < y$	Returns True if x is less than y; else False
==	Equal	$x == y$	Returns True if x is equal to y; else False
!=	Not Equal	$x != y$	Returns True if x is not equal to y; else False
>=	Greater than equal to	$x >= y$	Returns True if x is greater than or equal to y; else False
<=	Less than equal to	$x <= y$	Returns True if x is less than or equal to y; else False

**Example 10. Program to explain the relational operators >, <.**

```
#Program to explain the relational operators.

x = int(input("Enter the value of x: "))
y = int(input("Enter the value of y: "))

print("The Value returned by the x>y expression is")
print(x>y)
print("\n") # For inserting one extra line

print("The Value returned by the x<y expression is")
print(x<y)
print("\n")
```

Output:

```
Enter the value of x: 12
Enter the value of y: 15
The Value returned by the x>y expression is
False

The Value returned by the x<y expression is
True
```

In the above example, the value given to x is 12 and y is 15. Since x is less than y thus  $x > y$  returns False whereas  $x < y$  returns True.

**Example 11. Program to explain the relational operators  $==$ ,  $!=$ .**

```
x = int(input("Enter the value of x: "))
y = int(input("Enter the value of y: "))

print("The Value returned by the  $x==y$  expression is")
print(x==y)
print("\n")

print("The Value returned by the  $x!=y$  expression is")
print(x!=y)
print("\n")
#End of Program
```

Output:

```
Enter the value of x: 24
Enter the value of y: 24
The Value returned by the  $x==y$  expression is
True

The Value returned by the  $x!=y$  expression is
False
```

In the above example, the value given to x is 24 and y is 24. Since x and y are equal thus  $x==y$  returns True whereas  $x < y$  returns False.

**Example 12. Program to explain the relational operators  $>=$ ,  $<=$ .**

```
x = int(input("Enter the value of x: "))
y = int(input("Enter the value of y: "))

print("The Value returned by the  $x>=y$  expression is")
print(x>=y)
print("\n")

print("The Value returned by the  $x<=y$  expression is")
print(x<=y)
print("\n")

#End of Program
```

Output:

```
Enter the value of x: 12
```

```

Enter the value of y: 23
The Value returned by the x>=y expression is
False

The Value returned by the x<=y expression is
True

```

### 3.1.11. Logical Operators

The purpose of logical operator is to combine two or more conditional statements. Logical operators are very useful in many situations where the result is dependent on more than one conditions.

The logical operators are as follows:

Operator	Format	Output
or	$x > y$ or $a > b$	Returns True; if one of the statement is True.
and	$x > y$ and $a > b$	Returns True; if both statements are True.
not	not ( $x > y$ or $a > b$ )	Opposite the result, returns False if the result is true

**Example 13. Program to explain the ‘or’ logical operator.**

```

x = 5
y = 10
a = 7
b = 9

C = (x>y) or (a<b)

print("Result upon applying 'or' operator : ",C)

#End of Program

Output:

Result upon applying 'or' operator :  True

```

In example above, variable x, y, a, b are initialized with values 5,10,7 and 9 respectively. Output of (x>y) is False since 5 is smaller than 10 and the output of (a<b) is True since 7 is smaller than 9.

Logical operator ‘or’ is applied on the two statements (x>y), (a>b) having output False and True respectively. Since ‘or’ logical operator produces True output when any of the statements is True and in this case one statement is True (a<b). Thus, the output is True.

**Example 14. Program to explain the ‘and’ logical operator.**

```
x = 5
y = 10
a = 7
b = 9
C = (x>y) and (a<b)

print("Result upon applying 'and' operator : ",C)

#End of Program
Output:

Result upon applying 'and' operator :  False
```

In example above, logical operator ‘and’ is applied on the two statements (x>y), (a>b) having output False and True respectively. Since ‘and’ logical operator produces True output only when both statements are True and in this case only one statement is True (a<b). Thus, the output is False.

**Example 15. Program to explain the ‘not’ logical operator.**

```
x = 5
y = 10
a = 7
b = 9

C = not((x>y) and (a<b))

print("Result upon applying 'not' operator : ",C)

#End of Program

Output:

Result upon applying 'not' operator :  True
```

In example above, logical operator ‘not’ is applied on the result of the statement ((x>y) and (a>b)) which is False. Since ‘not’ logical operator reverses, the input provided. Thus, the final output becomes True.



**Example 16. Program to explain the precedence of the logical operator.**

```

x = 5
y = 10
a = 7
b = 9

C = not(a<b) or (x>y) and (a<b)

print("Result upon applying 'not' operator : ",C)

Output:

Result: False

```

Precedence order of the logical operators from highest to lowest is ‘not’, ‘and’ then ‘or’.

Precedence order could be understood from the above example. According to the precedence first of all the ‘not(a<b)’ is evaluated and output of the statement is False. In second step, (x>y) and (a<b) is evaluated which produces output False since (x>y) is False. Finally, the preference is given to ‘or’ and the output of step1, step2 is combined using ‘or’ operator. The result produced is False since output of both steps 1 and step 2 is False.

**3.1.12. bitwise operators**

Bitwise operators are similar to other operators but they operate on bits instead of integers or characters etc. The smallest unit of data storage is bit which is represented in 0 and 1. Bitwise operators works on the bits i.e. 0 and 1.

The functionality of bitwise operators is:

Symbol	Function
>>	Right shift
<<	Left shift
&	AND
	OR
^	XOR
~	One’s Compliment

**Example 17. Program to explain the working of Right Shift Operator (>>)**

```
num1 = 9
num2 = 10

new_num1 = num1 >> 1
new_num2 = num2 >> 3

print("Value of num1 after Right Shift is : ",new_num1)
print("Value of num2 after Right Shift is : ",new_num2)
```

Output:

```
Value of num1 after Right Shift is : 4
Value of num2 after Right Shift is : 1
```

In above example, variable 'num' has been initialized to '9'. Assume computer is using eight digits to represent a binary number then number '9' is represented as 0000 1001. After applying right shift operator on digit '9' the number in binary form becomes 0000 0100 i.e. 4 which is the new\_num1. Understand that the digits are shifted to right by 1 position i.e. 1 bit is lost and the empty position created on the left is filled by '0' digit.

Similarly, variable 'num2' has been initialized to '10' and using eight digits to represent a binary number the number '10' is represented as 0000 1010. After applying right shift operator on digit '10' the number in binary form becomes 0000 0001 i.e. 1 which is the new\_num2. Understand that the digits are shifted to right by 3 positions i.e. 3 bits are lost and the empty position created on the left is filled by '0' digit.

**Example 18. Program to explain the working of Left Shift Operator (<<)**

```
num1 = 10
num2 = 5

new_num1 = num1 << 1
new_num2 = num2 << 2

print("Value of num1 after Left Shift is : ",new_num1)
print("Value of num2 after Left Shift is : ",new_num2)
```

Output:

```
Value of num1 after Left Shift is : 20
Value of num2 after Left Shift is : 20
```

In above example, variable 'num1' has been initialized to '10'. Assume computer is using eight digits to represent a binary number then number '10' is represented as 0000 1010. After applying right shift operator on digit '10' the number in binary form becomes 0001 0100 i.e. 20 which new\_num1. Understand that the digits are shifted to left by 1 position i.e. 1 bit is lost and the empty position created on the right is filled by '0' digit.

Similarly, variable 'num2' has been initialized to '5' and using eight digits to represent a binary number then number '5' is represented as 0000 0101. After applying right shift operator on digit '5' the number in binary form becomes 0001 0100 i.e. 20 which new\_num2. Understand that the digits are shifted to left by 2 positions i.e. 2 bits are lost and the empty position created on the right is filled by '0' digit.

### Example 19. Program to explain the working of Bitwise AND Operator (&)

```
num1 = 8
num2 = 7
num3 = 10
res1 = num1 & num2
res2 = num1 & num3
print("Result1 : ",res1)
print("Result2 : ",res2)
```

Output:

Result1 : 0

Result2 : 8

In above example, variables 'num1', 'num2', 'num3' has been initialized to '8', '7' and '10' respectively.

Now in binary format:

num1 = 8 = 0000 1000

num2 = 7 = 0000 0111

res1 = 0 = 0000 0000 (num1 & num2)

Bitwise AND (&) Operator gives output 1 if both the corresponding bits are 1, otherwise 0. So, we can notice that in variable 'res1' all bits are 0 since none of the corresponding bits are 1 in variable 'num1' and 'num2'.

num1 = 8 = 0000 1000

num3 = 10 = 0000 1010

res2 = 8 = 0000 1000 (num1 & num3)

Bitwise AND (&) Operator gives output 1 if both the corresponding bits are 1, otherwise 0. So, we can notice that in variable 'res2' 1 is present where the corresponding bit is also 1 in variable 'num1' and 'num3'.

**Example 20. Program to explain the working of Bitwise OR Operator ( | ).**

```

num1 = 8
num2 = 7
num3 = 10
res1 = num1 | num2
res2 = num1 | num3
print("Result1 : ",res1)

print("Result2 : ",res2)

```

Output:

```

Result1 : 15
Result2 : 10

```

In above example, variables 'num1', 'num2', 'num3' has been initialized to '8', '7' and '10' respectively.

Now in binary format:

num1 = 8 = 0000 1000

num2 = 7 = 0000 0111

res1 = 15 = 0000 1111 (num1 & num2)

Bitwise OR ( | ) Operator gives output 1 if any of the corresponding bits are 1, otherwise 0. So, we can notice that in variable 'res1', 1 is present where any of the corresponding bit is 1 in variable 'num1' and 'num2'.

num1 = 8 = 0000 1000

num3 = 10 = 0000 1010

res2 = 10 = 0000 1010 (num1 & num3)

Bitwise OR ( | ) Operator gives output 1 if any of the corresponding bits are 1, otherwise 0. So, we can notice that in variable 'res2', 1 is present where any of the corresponding bit is 1 in variable 'num1' and 'num2'.

**Example 21. Program to explain the working of Bitwise XOR Operator ( ^ ).**

```

num1 = 8
num2 = 7
num3 = 10

res1 = num1 ^ num2

res2 = num1 ^ num3

print("Result1 : ",res1)

print("Result2 : ",res2)

```

Output:

Result1 : 15

Result2 : 2

In above example, variables 'num1', 'num2', 'num3' has been initialized to '8', '7' and '10' respectively.

Now in binary format:

num1 = 8 = 0000 1000

num2 = 7 = 0000 0111

res1 = 15 = 0000 1111 (num1 & num2)

Bitwise XOR ( ^ ) Operator gives output 1 if one of the corresponding bits is 1 and other is 0, otherwise it gives output as 0. So, we can notice that in variable 'res1', 1 is present where any of the corresponding bit is 1 in variable 'num1' and 'num2'.

num1 = 8 = 0000 1000

num3 = 10 = 0000 1010

res2 = 2 = 0000 0010 (num1 & num3)

Bitwise XOR ( ^ ) Operator gives output 1 if one of the corresponding bits is 1 and other is 0, otherwise it gives output as 0. So, we can notice that in variable 'res2', 1 is present where any of the corresponding bit is 1 in variable 'num1' and 'num2'.

**Example 22. Program to explain the working of Bitwise One's complement Operator ( ~ ).**

```
num1 = 8
```

```
num2 = 7
```

```
res1 = ~num1
```

```
res2 = ~num2
```

```
print("Result1 : ",res1)
```

```
print("Result2 : ",res2)
```

Output:

Result1 : -9

Result2 : -8

In above example, variables 'num1', 'num2', 'num3' has been initialized to '8', '7' and '10' respectively. Bitwise One's complement Operator ( ~ ) is a unary operator.

Now in binary format:

num1 = 8 = 0000 1000

res1 = -9 = - 0000 1001 (~num1)

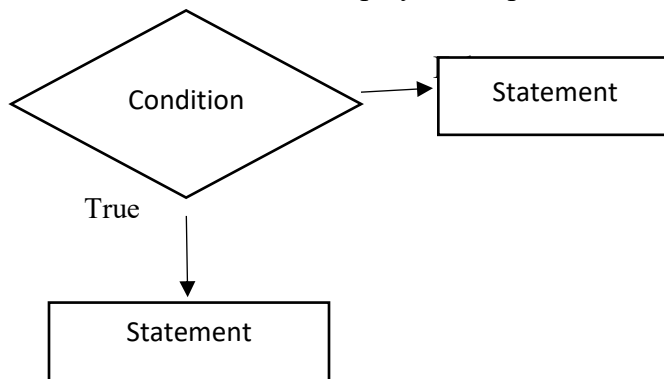
num2 = 7 = 0000 0111

res1 = -8 = - 0000 1000 (~num2)

### 3.1.13 Conditional Statements

In our life, many times we encounter situations where we have to make a decision be it your game, favourite food, movie, or the cloth. Similarly, in programming conditional statements help us to make a decision based on certain conditions. These conditions are specified by a set of conditional statements having boolean expressions which are evaluated to a boolean value of true or false.

We have seen in the flowcharts that flow of the program changes based on the conditions. Thus the condition based on decisions plays an important role in programming.



The various types of conditional statements are:

- i. if statement
- ii. if-else statement
- iii. if-elif-else statement

#### 3.1.8.1 if STATEMENT

The if statement test a condition and when the condition is 'true' a statement or a set of statements are executed and the actions are performed as per the instructions given in the statements otherwise the statements attached with the if statement are not executed.

Syntax of the if statement:

```

if (expression) :
    statement
  
```

**Example 23. Program to explain the working of *if* statement.**

```
#Program to explain the working of if statement.
```

```
marks = int(input("Enter the marks : "))
```

```
if(marks > 80):
    print("Grade A")
```

```
name = input("Enter the name :")
```

```
if(name == "Kapil"):
    print("You entered name Kapil")
```

```
#End of Program
```

Output:

```
Enter the marks : 85
```

```
Grade A
```

```
Enter the name :Prashant
```

In the above example, user entered marks as 85 and the *if* condition is checked since marks are greater than 80 so the condition becomes ‘true’ and the print statement is executed and output ‘Grade A’ is printed on the screen.

Then, the second input was asked and user entered name as Prashant and the *if* condition is checked since name entered is not ‘Kapil’ so the condition becomes ‘false’ and the print statement is not executed.

So, from the execution of the program we can conclude that the statement attached with *if* statement is executed only when *if* condition is ‘true’ otherwise they are not executed and behave like a comment.

**Example 24. Program to check the correct input entered by the user.**

```
num1 = int(input("Enter a number greater than 1 : "))
```

```
if(num1>1):
    print("Correct Number")
```

```
# End of Program
```

Output:

```
Enter a number greater than 1 : 5
```

```
Correct Number
```

Since the user entered number value as 5, so the *if* statement becomes true and the statement attached to it gets printed.

### 3.1.8.2 If-else STATEMENT

The if-else statement test a condition and when the condition is ‘true’ a statement or a set of statements attached with if block are executed and otherwise the statements attached with else block is executed. In if-else statement the programmer get an option to write a set of code that will be executed when the test condition is even ‘false’ in comparison to if statement.

Syntax of the *if-else* statement:

```
if (expression) :
    block1 statements
else:
    block2 statements
```

**Example 25. Program to explain the working of *if-else* statement.**

```
marks = int(input("Enter the marks : "))

if (marks >=50):
    print("Student is Pass")
else :
    print("Student is Fail")

# End of Program
```

Output:

Enter the marks : 47

Student is Fail

In above example, user has been asked for the input and user entered marks as 47 since the marks are less than 50 so the if condition becomes false. Consequently, the block attached with if is not executed and the control is transferred to statements attached to the else block and are executed.

**Example 26. Program to find bigger number between two numbers.**

```
#Program to find bigger number between two numbers.

num1 = int(input("Enter the FIRST number : "))
num2 = int(input("Enter the SECOND number : "))

if(num1>num2):
    print("FIRST number is Bigger")
else:
    print("SECOND number is Bigger")

# End of Program
```

Output:

Enter the FIRST number : 25  
Enter the SECOND number : 52  
SECOND number is Bigger



**Example 27. Program to find a number is even or odd.**

```
num1 = int(input("Enter the number : "))  
  
if(num1%2==0):  
    print("Number is EVEN")  
else:  
    print("Number is ODD")
```

# End of Program

Output:

```
Enter the number : 22  
Number is EVEN
```

**Example 28. Program to display a menu and calculate area of a square and volume of a cube.**

```
print("1. Calculate area of Square")  
print("2. Calculate volume of a Cube")  
choice = int(input("Enter your choice (1 or 2): "))  
side = int(input("Enter the side length: "))  
if(choice==1):  
    print("Area of Square is : ", side * side)  
else:  
    print("Volume of Cube is : ",side * side * side)
```

# End of Program

Output:

```
1. Calculate area of Square  
2. Calculate volume of a Cube
```

```
Enter your choice (1 or 2): 2
```

```
Enter the side length: 10  
Volume of Cube is : 1000
```

### 3.1.8.3 if-elif-else STATEMENT

In case of if or if-else statement only a single condition is tested and the statements attached are execute. However, many times situation is seen where more than one statement is to be tested before reaching to the conclusion.

Syntax of the if-elif-else statement:

```
if (expression):
    block1 statements
elif (expression):
    block2 statements
else:
    block3 statements
```

**Example 29. Program to explain the working of *if-elif-else* statement.**

```
marks = int(input("Enter the marks : "))

if(marks>=75):
    print("Student got DISTINCTION")
elif(marks <75 and marks >=50):
    print("Student is PASS")
else:
    print("Student is FAIL")
```

#End of Program

Output:

```
Enter the marks: 67
Student is PASS
```

In above example, user entered marks as 67. The first if condition becomes false since marks are less than 75 so the print statement in block1 is not executed and the control is transferred to elif condition which become true since the marks lies in the range and the block2 print statement is executed.

**Example 30. Program to salary deduction according to leaves in a month.**

```
leaves = int(input("Enter the number of Leaves: "))

BP = int(input("Enter the Basic Salary: "))
OA= 10000 #Other Allowance

if(leaves < 5 ):
    sal= BP + OA
```

```
elif(leaves > 5 and leaves < 20):
    sal= (BP/2) + OA

else:
    sal = 0

print("The Salary of the official for the month is: ",sal)

# End of Program

Output:
```

```
Enter the number of Leaves: 20
Enter the Basic Salary: 10000
The Salary of the official for the month is: 0
```

**Example 31. Program to Grade allocation according to marks.**

```
marks = int(input("Enter the marks : "))

if (marks >= 90):
    print("Grade A")

elif (marks < 90 and marks >= 75):
    print("Grade B")

elif (marks < 75 and marks >= 60):
    print("Grade C")

elif (marks < 60 and marks >= 50):
    print("Grade D")

else:
    print("Fail")

# End of Program
```

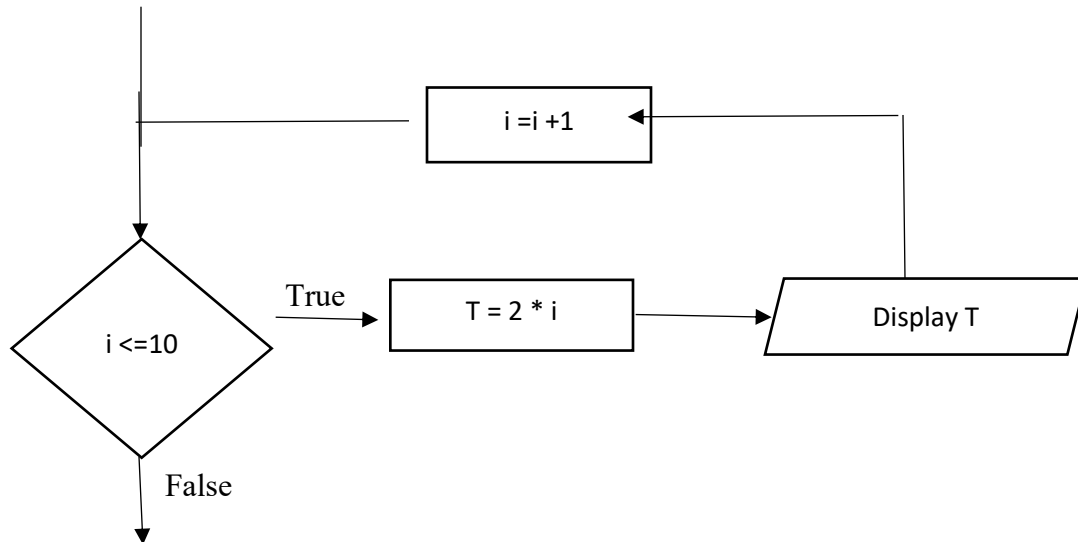
```
Output:

Enter the marks : 60
Grade C
```

### 3.1.9 Notion of iterative computation and control flow

We have noticed in flowcharts that some statements or steps are repeated again and again until a particular condition is achieved. In daily routine activities also we notice that we have to do continuously the same tasks till the target is achieved or we obtain the desired result. Such concept is called iteration or looping where steps are repeated to achieve the set target.

Statements of a program are executed in a sequential manner by default until a condition is being introduced and flow of program gets modified depending upon the condition. That means based on the conditions the sequential flow of the program is controlled or decided i.e. flow of control or control flow of a program depends on the set conditions.



Flowchart for the printing of table of 2 where a set of statements are repeated again and again until the value of  $i$  is greater than 10. We can notice that the flow of program is based on the result of the condition.

### 3.1.10 Range function

The `range()` function returns a sequence of numbers, starting from 0 by default, and increments by 1 (by default), and stops before a specified number.

Syntax:

`range(start, stop, step)`

### 3.1.11 while STATEMENT

The while loop is used to repeat a set of statements until a condition is reached. The syntax for the while loop is:

```
while (condition):
    block of statements
```

Condition is an expression whose result will be either 'true' or 'false'. The block of statements will be executed till the condition remains 'true' and when the condition becomes 'false' loop terminates.

**Example 32. Program to explain the working of *while* statement.**

```
# Print table of 2
i=1
while(i<=10):
    T= 2*i
    print("2 *",i," = ",T)
    i=i+1
#End of Program
Output:
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
2 * 10 = 20
```

In above program, value of loop index ‘i’ in this case is tested and statements attached to the while loop are executed till the value of i remains less than equal to 10 i.e. till condition remains ‘true’. When the value of ‘i’ becomes greater than 10 the loop terminates and program ends. *while* loop is an entry controlled loop i.e. entry into the loop for executing the statements are allowed only when entry condition is *true*.

**3.1.12 for STATEMENT**

*for* statement is another way in Python for iteration or looping through which a set of statements are repeatedly executed till the condition remains ‘true’.

Syntax of *for* statement is:

```
for <variable> in [sequence]:
    block of statements
```

In the case of *for* loop the variable i.e. loop index takes the value of the elements present in a list one by one and executes the statements attached with the loop till the last element and loop terminates after completing the process for the last element in the list.

**Example 33. Program to explain the working of *for* statement.**

```
#Program to explain the working of for statement.
```

```
for i in [5,7,9,11]:
```

```
    print("The value in the sequence is: ",i)
```

```
#End of Program
```

Output:

The value in the sequence is: 5

The value in the sequence is: 7

The value in the sequence is: 9

The value in the sequence is: 11

In the above example, the loop index 'i' took the value of the elements present in the list one by one in sequence and the statement attached to the *for* loop are executed number of times equal to the number of elements present in the list. In above case, since the number of elements in the list are four i.e. 5, 7, 9, and 11. So the statement attached is executed four times.

**Example 34. Program to explain the working of *for* statement.**

```
for i in ['cat','dog','horse','lion','tiger']:
```

```
    print("Hello World")
```

```
#End of Program
```

Output:

Hello World

Hello World

Hello World

Hello World

Hello World

In this case, loop index 'i' will take values of the elements of list which are string values and the statement attached with the loop is executed 5 times since the number of elements in the list are 5. With this example, it is clear that the index value can be either string or integer and loop iteration depends on the number of elements in the list.

**Example 35. Program to explain the working of *for* statement using range () function.**

```
for i in range (5):  
    print ("value of i: ", i)  
#End of Program  
Output:  
value of i: 0  
value of i: 1  
value of i: 2  
value of i: 3  
value of i: 4
```

Using range(n) function the *for* loop can be implemented and index variable 'i' automatically initialized by 0 and takes value ranging from 0,1,2,3,4, ..., n-1 where n is the upper limit. Index variable is incremented by 1 till it reached the upper limit decremented by 1. In this case the n is 5 so 'i' is initialized to 0 and upper limit is 5 so the loop is executed 5 times for the value 0,1,2,3,4 and same is printed as output.

**Example 36. Program to explain the working of *for* statement using range() function.**

```
for i in range(3,7):  
    print("value of i: ", i)  
#End of Program  
Output:  
value of i: 3  
value of i: 4  
value of i: 5  
value of i: 6
```

In this case, the range(a,n) function takes two parameters, with first value of parameter index variable 'i' is initialized and the second parameter is the upper limit. Index variable is incremented by 1 till it reach the upper limit decremented by 1. In this example, 'i' is initialized to 3 and 'i' takes the values from 3,4,5,6 i.e. since 7 is the upper limit and the statements attached with the loop are executed 4 times.

**Example 37. Program to explain the working of *for* statement using range() function.**

```
for i in range(3,10,2):  
    print("value of i: ", i)  
#End of Program
```

Output:

```
value of i: 3
value of i: 5
value of i: 7
value of i: 9
```

In this case, the `range(a,n,b)` function takes three parameters, with first value of parameter index variable 'i' of the *for* loop is initialized, the second parameter is the upper limit and third parameter is the incremental value. Index variable is incremented by the value 'b' till it reach the upper limit decremented by 1. In this example, 'i' is initialized to 3 and 'i' takes the values from 3,5,7,9 since index value 'i' is incremented by 2 and the statements attached with the loop are executed 4 times.

**Example 38. Program to count numbers from 1 to 5 using *while* and *for* statements.**

<pre># using while statement  i=1 while(i&lt;=5):     print(i)     i=i+1  #End of Program  Output : 1 2 3 4 5</pre>	<pre># using for statement  for i in range(1,6):     print(i)  #End of Program  Output : 1 2 3 4 5</pre>
---	--

**Example 39. Program to generate multiplication table of the number entered.**

<pre># using while statement  num = int(input("Enter the number: "))  i=1 while(i&lt;=10):     T = num *i     print(num,"*",i,"=",T)     i=i+1 #End of Program  Output :  Enter the number: 5</pre>	<pre># using for statement  num = int(input("Enter the number: "))  for i in range(1,11):     T = num*i     print(num,"*",i,"=",T)  #End of Program  Output :  Enter the number: 5</pre>
---	--



5 * 1 = 5	5 * 1 = 5
5 * 2 = 10	5 * 2 = 10
5 * 3 = 15	5 * 3 = 15
5 * 4 = 20	5 * 4 = 20
5 * 5 = 25	5 * 5 = 25
5 * 6 = 30	5 * 6 = 30
5 * 7 = 35	5 * 7 = 35
5 * 8 = 40	5 * 8 = 40
5 * 9 = 45	5 * 9 = 45
5 * 10 = 50	5 * 10 = 50

**Example 40. Program to print alphabets of a word.**

<pre>#using while statement word = input("Enter the word: ") i=0 # while(i&lt;len(word)):     print(word[i])     i=i+1 #End of Program Output :</pre> <p>Enter the word: CRICKET</p> <p>C R I C K E T</p>	<pre>#using for statement word = input("Enter the word ") for w in word:     print(w) #End of Program  Output :</pre> <p>Enter the word: CRICKET</p> <p>C R I C K E T</p>
---	---

len() is function that returns number of characters in a string

**Example 41. Write a program to read 5 numbers from keyboard and find their sum.**

<pre>#using while statement  i=1 total=0  while(i&lt;=5):     num = int(input("Enter the number: "))     total= total+num     i=i+1  print("Sum is : ", total) #End of Program Output :</pre> <p>Enter the number: 1 Enter the number: 2 Enter the number: 3 Enter the number: 4 Enter the number: 5 Sum is : 15</p>	<pre>#using for statement  total=0  for i in range(5):     num = int(input("Enter the number: "))     total= total+num  print("Sum is : ", total)  #End of Program  Output :</pre> <p>Enter the number: 1 Enter the number: 2 Enter the number: 3 Enter the number: 4 Enter the number: 5 Sum is : 15</p>
--	---

**Example 42. Write a program to display first 10 odd numbers.**

<pre>#using while statement  i=1 num=1  while(i&lt;=10):     print(num, end=" ", ")     num= num+2     i=i+1  #End of Program  Output :</pre> <p>1, 3, 5, 7, 9, 11, 13, 15, 17, 19,</p>	<pre>#using for statement  num=1  for i in range(1,11):     print(num, end=" ", ")     num= num+2  #End of Program  Output :</pre> <p>1, 3, 5, 7, 9, 11, 13, 15, 17, 19,</p>
---	--

**Example 43. Write a program to check the entered word is a palindrome.**

Palindrome is a word that reads the same backwards as forwards, e.g. madam, refer, malayalam.

<pre>#using while statement  i=1 num=1  while(i&lt;=10):     print(num, end=", ")     num= num+2     i=i+1  #End of Program  Output :  1, 3, 5, 7, 9, 11, 13, 15, 17, 19,</pre>	<pre>#using for statement  num=1  for i in range(1,11):     print(num, end=", ")     num= num+2  #End of Program  Output :  1, 3, 5, 7, 9, 11, 13, 15, 17, 19,</pre>
---	--

**Example 44. Write a program to print the following format.**

```
*
**
***
****
*****
```

<pre>#using while statement  i=0 while(i&lt;5):     j=0     while(i&gt;=j):         print("*",end=" ")         j=j+1     print()     i=i+1  #End of Program</pre>	<pre>#using for statement  for i in range(5):     for j in range (0,i+1):         print("*",end=" ")     print()  #End of Program</pre>
---	---

### 3.1.13 break STATEMENT

In general, there arises so many situations when a loop is to be terminated on reaching a particular external condition and in such state Python *break* statement is used. Thus, the *'break'* statement is used to come out of the currently running loop upon reaching the desired condition.

Syntax:

```
for i in range(1,10):
    condition
    break;
```

**Example 45.** Program to explain the working of *'break'* statement.

<pre>#using while statement i=1 while(i&lt;=10):     print(i)     if(i==5):         break;     i=i+1 #End of Program Output : 1 2 3 4 5</pre>	<pre>#using for statement for i in range(1,10):     print(i)     if(i==5):         break;  #End of Program Output : 1 2 3 4 5</pre>
---	---

In above example, in case *'break'* statement would have not been written in the program then counting from 1 to 10 would have been the output. However, due to *'break'* statement when the value of *i* becomes 5 the *'break'* statement plays its role and terminates loop in between and prints the counting from 1 to 5 as output and control transfers out of the loop.

**Example 46. Write a program to check a number is prime number.**

<pre> #using while statement num = int(input("Enter a number greater than 1: = ")) i=2; flag=0 while(i&lt;=num):     rem = num%i     if(rem==0 and i==num):         flag=1     else:         if(rem == 0 and i!= num):             break         i=i+1 if(flag==1):     print("Number is PRIME") else:     print("Number is NOT PRIME")  #End of Program  Output : Enter a number greater than 1: = 7 Number is PRIME </pre>	<pre> #using for statement num = int(input("Enter a number greater than 1: = ")) flag=0 for i in range(2,num+1):     rem= num%i     if(rem==0 and i==num):         flag=1     else:         if(rem==0 and i != num):             break if(flag==1):     print("Number is PRIME") else:     print("Number is NOT PRIME") #End of Program  Output : Enter a number greater than 1: = 97 Number is PRIME </pre>
--	--

**Example 47. Write a program for reversing a number.**

<pre> #using while statement  num = int(input("enter a number = "))  rev_num=0  while(num&gt;0):     rem = num%10     rev_num = rev_num *10 + rem     num = int(num/10)  print("Reverse number is :", rev_num)  #End of Program  Output :  enter a number = 123456 Reverse number is : 654321 </pre>	<pre> #using for statement  num = int(input("enter a number = ")) rev_num=0  for i in range (1,num):     rem = num%10     rev_num = rev_num *10 + rem     num = int(num/10)     if(num&lt;=0):         break  #End of Program  Output :  enter a number = 123456 Reverse number is : 654321 </pre>
--	--

**3.1.14 Continue STATEMENT**

In case of the Python continue statement next iteration of the loop takes place while ignoring the statements after the continue statement i.e. continue statement makes the control jumps back at the start of the loop for iterating again according the set condition for the entry into the loop.

Syntax:

Continue

**Example 48. Explaining the working of the continue statement.**

<pre>#using while statement  i=0; sub='Python'  while(i&lt;len(sub)):     if(sub[i]== 'o' or sub[i]=='y'):         i=i+1         continue     print(sub[i])     i=i+1  print("End of Program") #End of Program  Output :  P t h n End of Program</pre>	<pre>#using for statement  for i in 'Python':     if i=='o' or i=='y':         continue     print(i)  print("End of Program")  #End of Program  Output :  P t h n End of Program</pre>
--	--

In above example, it can be seen that when the alphabets 'o' and 'y' occurs in 'Python' the *if* statement becomes *true* and the *continue* statement is executed due to which statement of 'print' is not executed or ignored and control is being transferred back to start of the loop. Thus, alphabet 'o' and 'y' is not printed in output.

**Example 49.** Write a program to enter the marks of five subjects of a student and if the marks in any subject is less than 50 don't print the marks.

<pre>#using while statement  i=1  while(i&lt;=5):     print("Enter the marks of the subject-",i)     marks=int(input())     if(marks&lt;50):         i=i+1         continue      print("Marks = ",marks)     i=i+1  #End of Program  Output :  Enter the marks of the subject- 1 33 Enter the marks of the subject- 2 22 Enter the marks of the subject- 3 11 Enter the marks of the subject- 4 56 Marks = 56 Enter the marks of the subject- 5 78 Marks = 78</pre>	<pre>#using for statement  for i in range (1,6):     print("Enter the marks of the subject-",i)     marks=int(input())     if(marks&lt;50):         continue      print("Marks = ",marks) #End of Program  Output :  Enter the marks of the subject- 1 33 Enter the marks of the subject- 2 22 Enter the marks of the subject- 3 11 Enter the marks of the subject- 4 56 Marks = 56 Enter the marks of the subject- 5 78 Marks = 78</pre>
---	---

### 3.1.15 pass STATEMENT

Python *pass* statement is used when a condition is required to make the code complete but the statements attached with it are not required to be executed.

Syntax:

If(condition):

    pass



**Example 50.** Write a program to if the marks entered is greater than 50 then display 'Great' and if marks are less than 50 then display 'Do Hard work'.

```
marks = int(input("Enter the marks: "))
if(marks>50):
    print("Great")
elif(marks==50):
    pass
    #Yet to decide
else:
    print("Do Hard work")
Output :
Enter the marks: 50
```

In above example, user gave input marks as 50 since *pass* statement is attached with that condition and no code is written in this block so nothing appears in the output.

#### 3.1.15.1 assert STATEMENT

In Python *assert* statement is used to check the whether a condition or a logical expression is *true* or *false*. The *assert* statement is very useful in tracking the errors and terminating the program on occurrence of an error.

Syntax:

```
assert (condition)
```

**Example 51.** Write a program for explaining working of *assert* statement.

```
password = input("Enter the password : ")
assert password == 'PYTHON'
print("Number Entered : ", password)
Output :
Enter the password : Python
Traceback (most recent call last):
  File "C:\Training\CABA-MDTP COURSE MATERIAL\Book Programs.py", line 6, in
<module>
    assert password == 'PYTHON'
AssertionError
```

In above example, in case the user enters any other password than 'PYTHON' the error message occurs and program is not executed further.

## Exercises

### Multiple Choice Questions

**1) What are the features of Python?**

- a. Open Source
- b. Portable
- c. Have extensive library
- d. All of the above

**2) Comments in Python starts with the character:**

- a. %
- b. &
- c. \*
- d. #

**3) Multiline comments in Python starts with:**

- a. {
- b. ]
- c. ""
- d. ^

**4) Which of the following is True in case of compiler?**

- a. Compiler converts bits i.e. 0's and 1's into High level language
- b. Compiler translates source code into machine language.
- c. Compiler translates low level language to high level language
- d. None of the above

**5) Python is an interpreted language because:**

- a. Python programs are first compiled then executed
- b. Python program needs no compilation and executed directly
- c. Python programs need not to be converted into machine language.
- d. None of the above.

**6) Which of the following is not numeric literal?**

- a. "123"
- b. 125.25
- c. 123
- d. None of the above

**7) Which of the following a string literal?**

- a. "abc"
- b. "Python"
- c. 'Python'
- d. None of the above

**8) What will be value of A and B in the given expression:**

$$A = (2+3) * 2 + 6$$

$$B = (6/2) + 3 * 2$$

- a. A = 16, B = 12
- b. A = 40, B = 12
- c. A = 16, B = 9
- d. None of the above

**9) What will be value of A and B in the given expression:**

$$A = 10/2$$

$$B = 10\%2$$

- a.  $A = 0, B = 0$
- b.  $A = 5, B = 5$
- c.  $A = 0, B = 5$
- d.  $A = 5, B = 0$

**10) What will be the output of the following expression?**

$$A = 8 >> 1$$

$$B = 7 << 1$$

- a.  $A = 4, B = 14$
- b.  $A = 3, B = 4$
- c.  $A = 5, B = 7$
- d.  $A = 9, B = 12$

**State whether statement is true or false**

- 1) Python is object-oriented language. (T/F)
- 2) Python is a compiled language. (T/F)
- 3) The arithmetic operator '%' also called as modulus operator returns remainder in integer division. (T/F)
- 4) The logical operator *and* returns *False* when both the expressions are *True*. (T/F)
- 5) The *not* operator is used to reverse the output of an expression. (T/F)
- 6) Looping is defined as block of instructions repeated till the desired condition is achieved. (T/F)
- 7) *while* statement is an entry controlled loop. (T/F)
- 8) Using *break* statement programmer can come out of loop even if the condition is *True*. (T/F)
- 9) In Python *else* statement is optional. (T/F)
- 10) *not* logical operator has the highest precedence. (T/F)

**Fill in the blanks**

- 1) The \_\_\_\_\_ statement is used for decision making.
- 2) \_\_\_\_\_ statement is used to come out of the loop.
- 3) \_\_\_\_\_ statement is used to check the logical expressions.
- 4) \_\_\_\_\_ bitwise operator returns 1 if any of the corresponding bits is 1.
- 5) \_\_\_\_\_ relational operator is used to represent *not equal to*.
- 6) The output of the expression 3 and 4 is \_\_\_\_\_.
- 7) \_\_\_\_\_ allows sections of code to be executed repeatedly under some condition.
- 8) String is a sequence of \_\_\_\_\_.
- 9) The \_\_\_\_\_ statement is an empty statement in Python.
- 10) Operator \_\_\_\_\_ when used with two strings, gives a concatenated string.

**Lab Exercises**

- 1) Write a program to calculate the multiplication and sum of two numbers.
- 2) Write a program to display characters from a string that are present at an even index number.
- 3) Write a program to generate the following output using *for* and *while* statement.

```
1
12
123
1234
12345
```

- 4) Write a program to generate the following output using *for* and *while* statement.

```
1
22
333
4444
55555
```

- 5) Write a program to generate the following output using *for* and *while* statement.

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

- 6) Write a program to display first ten prime numbers using *for* and *while* statement.
- 7) Write a program to find factorial of number using *for* and *while* statement.
- 8) Write a program to count the total number of digits in a number using *for* and *while* statement.
- 9) Write a program to display Fibonacci series up to 10 terms
- 10) Write a program to calculate the cube of all numbers from 1 to a given number

## Chapter 4

### String Handling and Sequence Types

#### 4.1 String Handling

A string is a **series of characters**. In Python, anything inside quotes is a string. And you can use either single or double quotes. It is just like an array in C language and they are stored and accessed using index.

##### 4.1.1. Creating a string

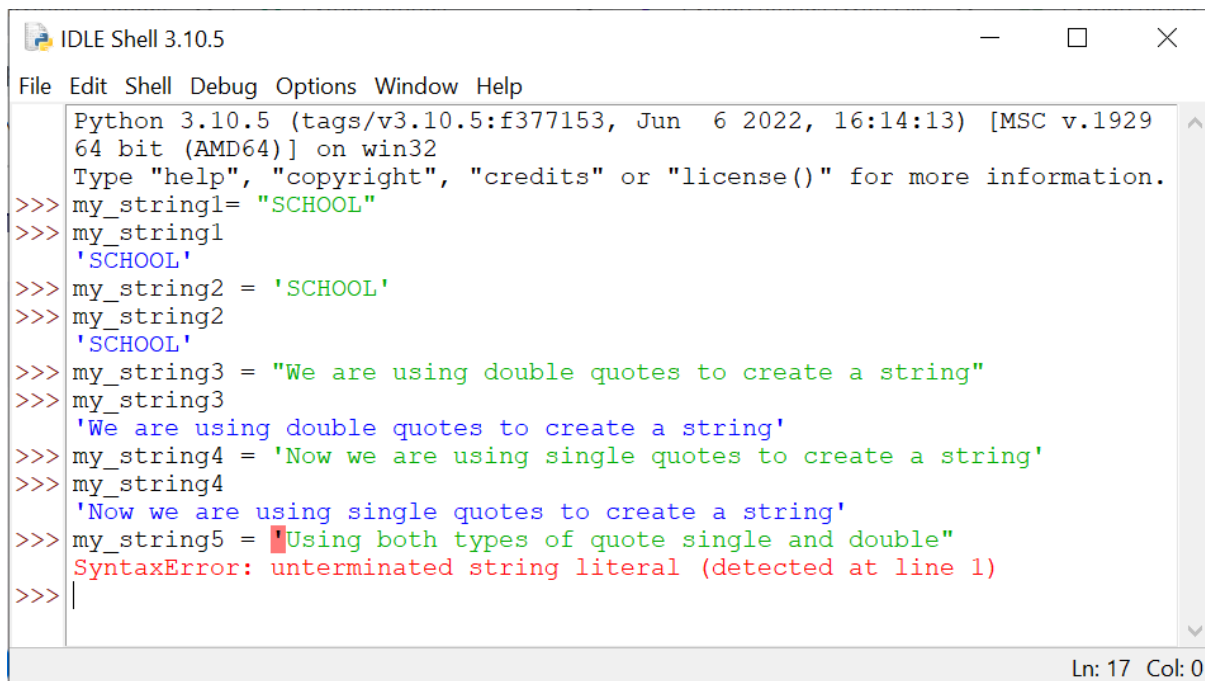
Strings can be created by enclosing characters inside a single quote or double-quotes i.e

```
my_string1 = "SCHOOL"
```

```
my_string2 = 'SCHOOL'
```

In Python by any of the above ways strings can be created.

However, using both single and double quotes simultaneously for creating a string will not work and will generate error.



```

IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> my_string1= "SCHOOL"
>>> my_string1
'SCHOOL'
>>> my_string2 = 'SCHOOL'
>>> my_string2
'SCHOOL'
>>> my_string3 = "We are using double quotes to create a string"
>>> my_string3
'We are using double quotes to create a string'
>>> my_string4 = 'Now we are using single quotes to create a string'
>>> my_string4
'Now we are using single quotes to create a string'
>>> my_string5 = "Using both types of quote single and double"
SyntaxError: unterminated string literal (detected at line 1)
>>> |
  
```

String Figure 1: Creating a String.

##### 4.1.2. String indexing

In programming languages, individual items in an ordered set of data can be accessed directly using a numeric index or key value. This process is referred to as indexing.

In Python, strings are ordered sequences of character data, and thus can be indexed in this way. Individual characters in a string can be accessed by specifying the string name followed by a number in square brackets ([]).

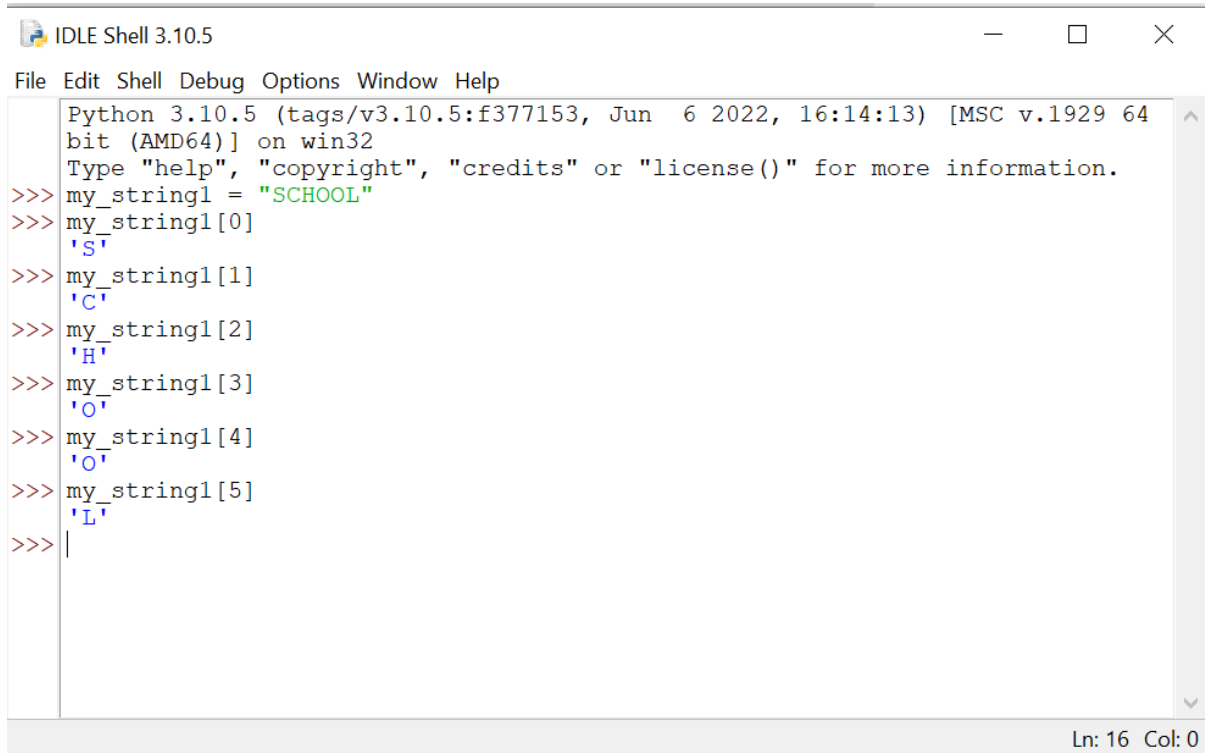
String indexing in Python is zero-based: the first character in the string has index 0, the next has index 1, and so on. The index of the last character will be the length of the string minus one.

For example, a schematic diagram of the indices of the string 'SCHOOL' would look like this:

S	C	H	O	O	L
0	1	2	3	4	5

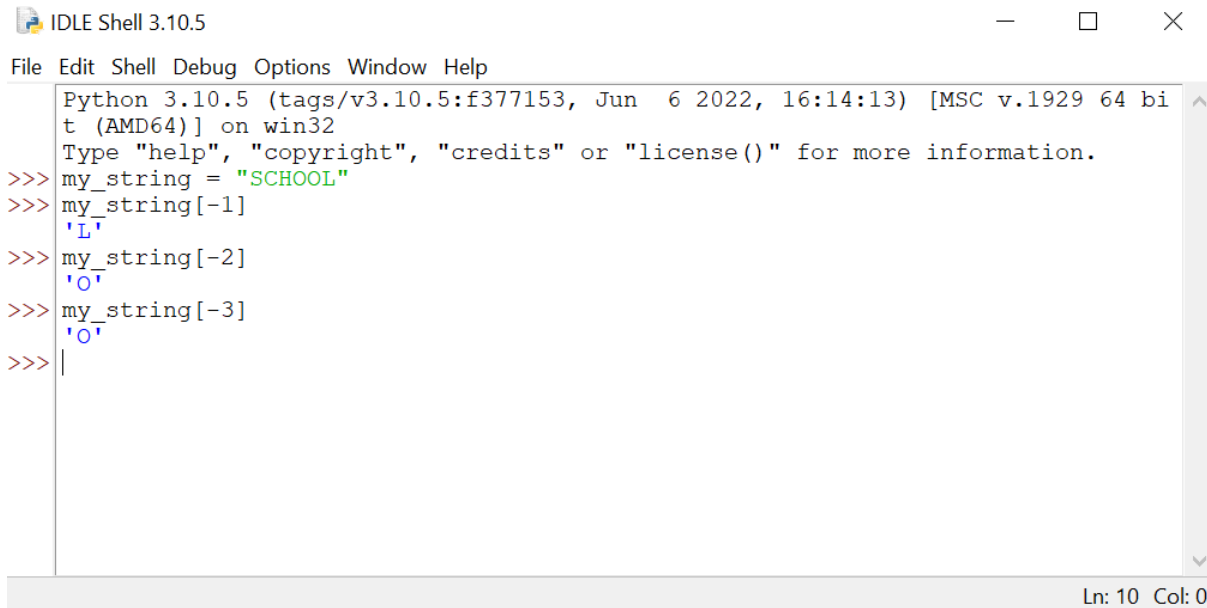
String Example 1: String Indexing

The individual characters can be accessed by index as follows:

A screenshot of the IDLE Shell 3.10.5 window. The window title is 'IDLE Shell 3.10.5'. The menu bar includes 'File', 'Edit', 'Shell', 'Debug', 'Options', 'Window', and 'Help'. The shell displays the following text: 'Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32', 'Type "help", "copyright", "credits" or "license()" for more information.', and a series of commands and outputs: '>>> my\_string1 = "SCHOOL"', '>>> my\_string1[0]' returns ''S'', '>>> my\_string1[1]' returns ''C'', '>>> my\_string1[2]' returns ''H'', '>>> my\_string1[3]' returns ''O'', '>>> my\_string1[4]' returns ''O'', and '>>> my\_string1[5]' returns ''L''. The cursor is on the line '>>> |'. The status bar at the bottom right shows 'Ln: 16 Col: 0'.

String Figure 2: String Indexing

In case a situation arise where we like to access the last element of a string and we are not aware of the length of the string then in such case negative indexing is used as follows:



```

Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bi
t (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> my_string = "SCHOOL"
>>> my_string[-1]
'L'
>>> my_string[-2]
'O'
>>> my_string[-3]
'O'
>>> |

```

Ln: 10 Col: 0

String Figure 3: Negative Indexing

In above example with using negative index we are able to access the last element and second last element easily without requiring any knowledge of length of the string. Negative indexing is as under:

S	C	H	O	O	L
-6	-5	-4	-3	-2	-1

String Example 2: Negative Indexing

### 4.1.3. string slicing

Concept of slicing is about obtaining a sub-string or a part from the given string by slicing it respectively from start to end. The concept is slicing is similar to take a bread slice from a bread packet. In the slicing operation the desired part of the string is obtained using the indexes of the string.

my_string	C	O	M	P	U	T	E	R
Positive Index	0	1	2	3	4	5	6	7
Negative Index	-8	-7	-6	-5	-4	-3	-2	-1

String Example 3: String Slicing Positive and Negative Index

```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> my_string = "COMPUTER"
>>> print("Characters 1 to 4 are: ",my_string[1:4])
Characters 1 to 4 are: OMP
>>> print("Characters 1 to 3 are: ",my_string[1:3])
Characters 1 to 3 are: OM
>>> print("Characters 0 to 7 are: ",my_string[0:7])
Characters 0 to 7 are: COMPUTER
>>> print("Characters -7 to -1 are: ",my_string[-7:-1])
Characters -7 to -1 are: OMPUTE
>>> print("Characters -8 to -1 are: ",my_string[-8:-1])
Characters -8 to -1 are: COMPUTE
>>> print("Characters Start to -1 are: ",my_string[: -1])
Characters Start to -1 are: COMPUTE
>>> print("Characters 0 to end are: ",my_string[0:])
Characters 0 to end are: COMPUTER
>>> print("Characters 0 to end are: ",my_string[:])
Characters 0 to end are: COMPUTER
>>> |

```

Ln: 20 Col: 0

String Figure 4: String Slicing positive and negative

The following points are to be noticed:

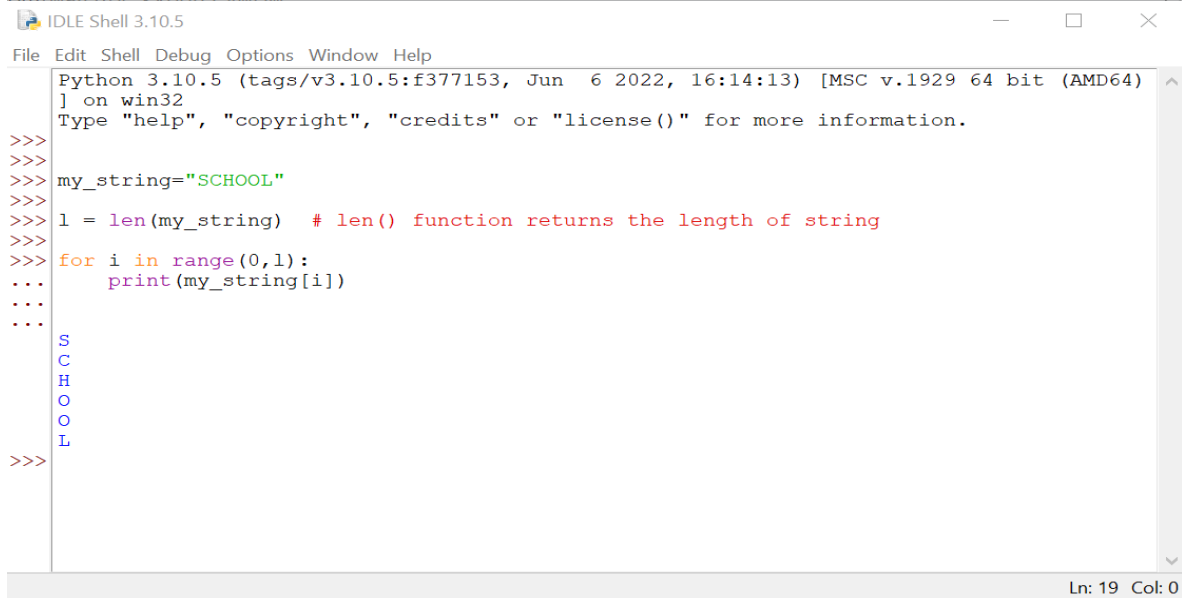
arr[start:stop]	# items start through stop-1
arr[start:]	# items start through the rest of the array
arr[:stop]	# items from the beginning through stop-1
arr[:]	# a copy of the whole array
arr[start:stop:step]	# start through not past stop, by step

Table 3: Slicing points to be noticed.

#### 4.1.4. traversing a string

Traversing means to fetch or access each character of string. So traversing a string can be achieved either one by one as explained in Figure 4.2 or using a loop. Using a loop string can be traversed as below:





```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> my_string="SCHOOL"
>>>
>>> l = len(my_string) # len() function returns the length of string
>>>
>>> for i in range(0,l):
...     print(my_string[i])
...
S
C
H
O
O
L
>>>

```

Ln: 19 Col: 0

String Figure 5: Traversing a string

#### 4.1.5. Concatenation of string

The '+' operator works differently with string and perform concatenation means it joins two or more strings to form a single string. For example:

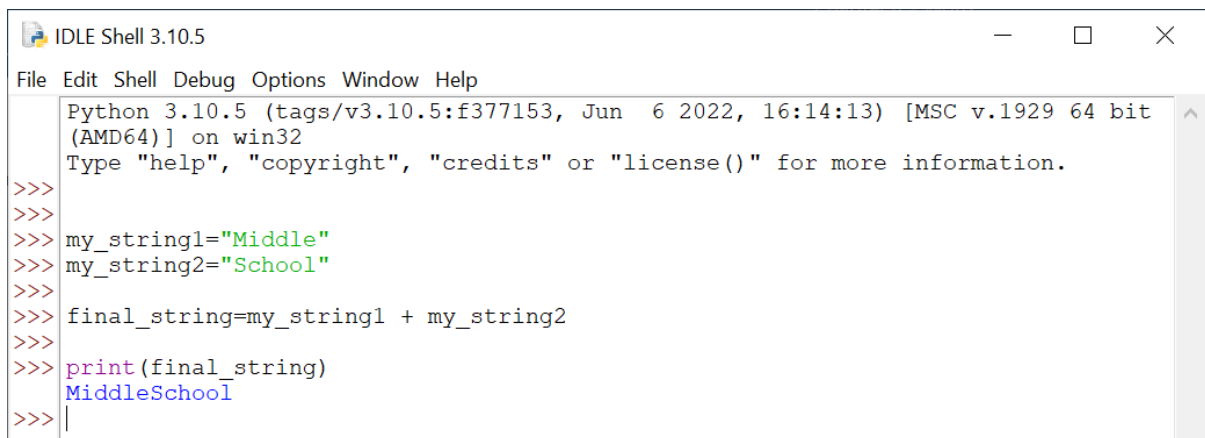
This operation is valid and *final\_string* variable will hold "MiddleSchool". Let us understand with the program as under:

```

my_string1 = "Middle"
my_string2 = "School"
final_string = my_string1 + my_string2

```

String Example 4: Concatenation of String



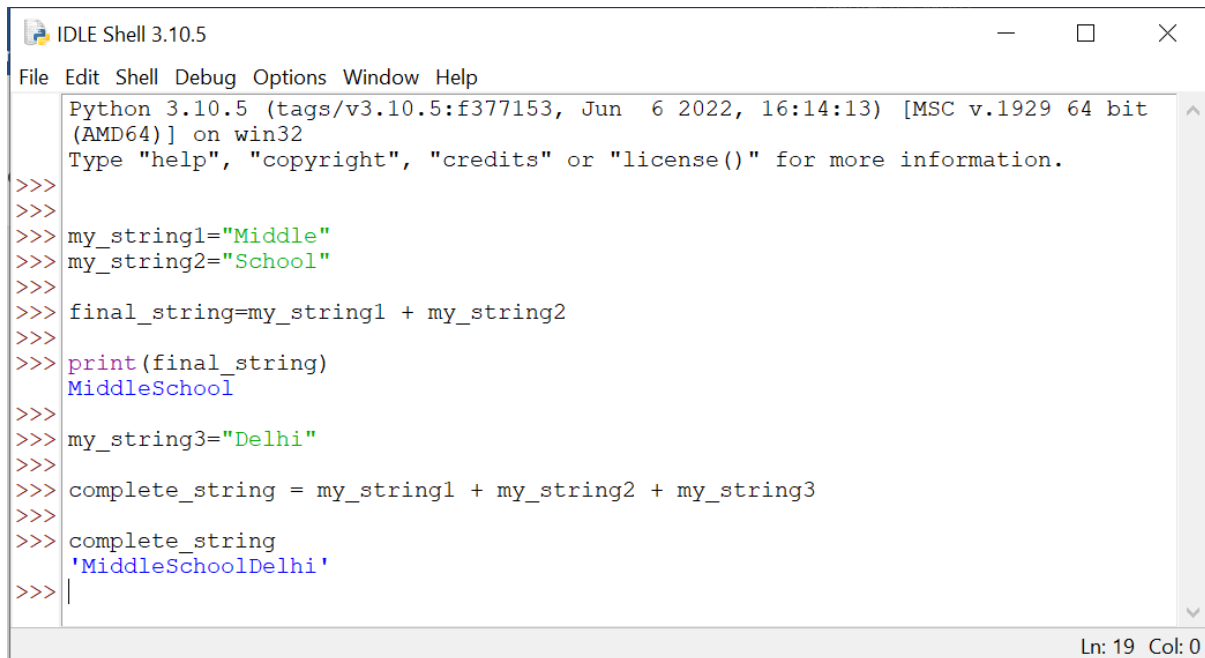
```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>>
>>> my_string1="Middle"
>>> my_string2="School"
>>>
>>> final_string=my_string1 + my_string2
>>>
>>> print(final_string)
MiddleSchool
>>>

```

String Figure 6: Concatenation of String

We can combine more than two strings which are explained with the program as under :



```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> my_string1="Middle"
>>> my_string2="School"
>>>
>>> final_string=my_string1 + my_string2
>>>
>>> print(final_string)
MiddleSchool
>>>
>>> my_string3="Delhi"
>>>
>>> complete_string = my_string1 + my_string2 + my_string3
>>>
>>> complete_string
'MiddleSchoolDelhi'
>>>

```

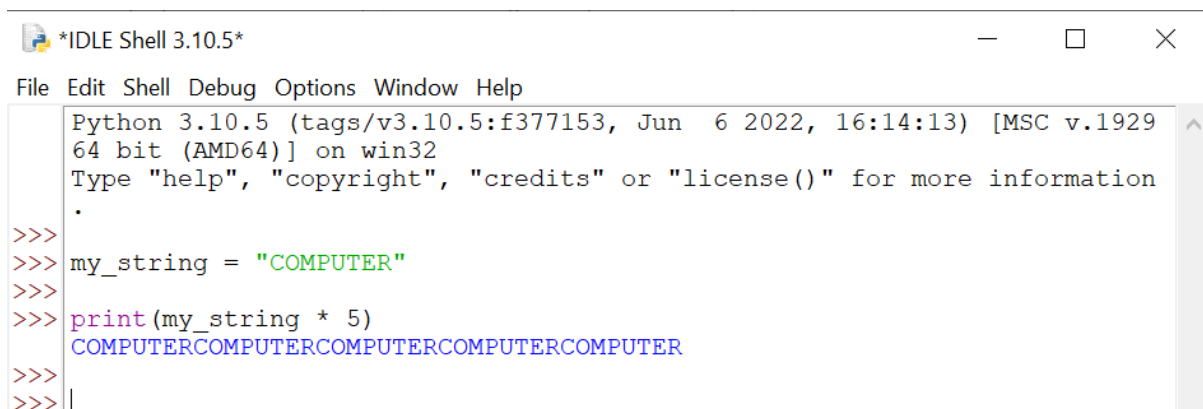
String Figure 7: can combine more than two strings

#### 4.1.6. Other operations on strings

##### 4.1.6.1 Replication (\*) operator

There may be times when you need to use Python to automate tasks, and one way you may do this is through repeating a string several times. You can do so with the ‘\*’ operator. Like the ‘+’ operator, where it is the operator for multiplication. When used with one string and one integer, ‘\*’ is the **string replication operator**, repeating a single string however many times you would like through the integer you provide.

Let’s print out “COMPUTER” 5 times without typing out “COMPUTER” 5 times with the ‘\*’ operator:



```

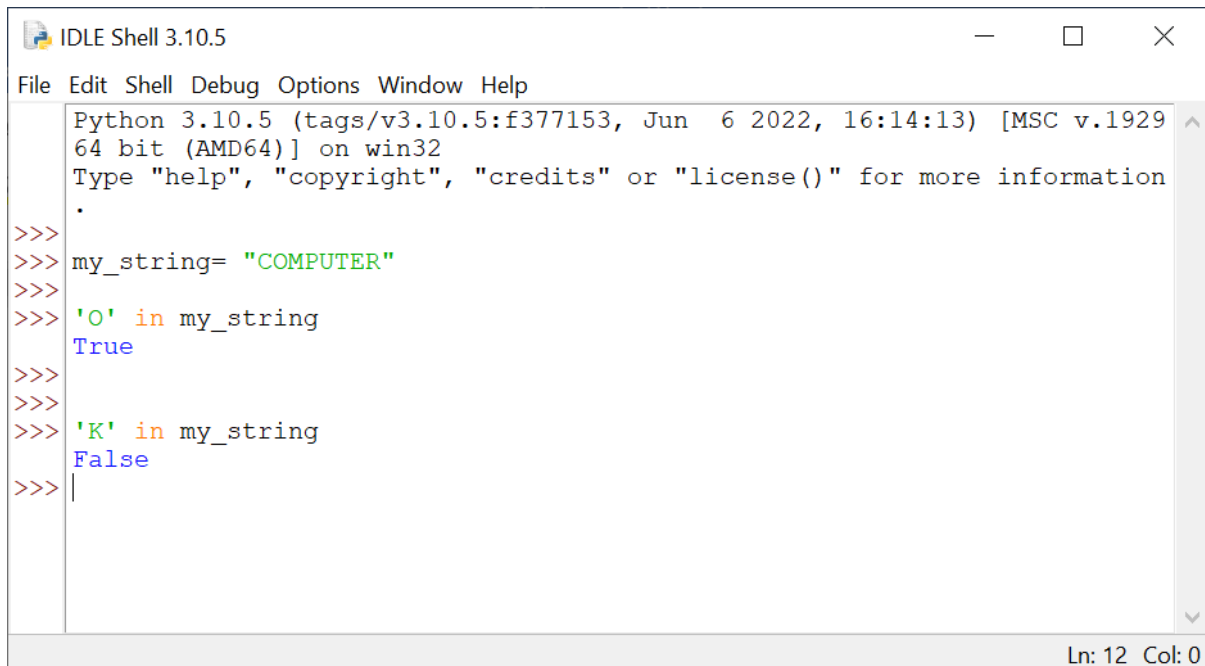
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information
.
>>>
>>> my_string = "COMPUTER"
>>>
>>> print(my_string * 5)
COMPUTERCOMPUTERCOMPUTERCOMPUTERCOMPUTER
>>>
>>>

```

String Figure 8: Replication Operator in strings

### 4.1.6.2 Membership Operator (in)

This operator confirms the presence of a character in a given string and is used as under :



```

Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929
64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information
>>>
>>> my_string= "COMPUTER"
>>>
>>> 'O' in my_string
True
>>>
>>>
>>> 'K' in my_string
False
>>> |
  
```

String Figure 9: Membership Operator (in)

### 4.1.6.3 Comparison Operators

To compare two strings, we mean that we want to identify whether the two strings are equivalent to each other or not, or perhaps which string should be greater or smaller than the other.

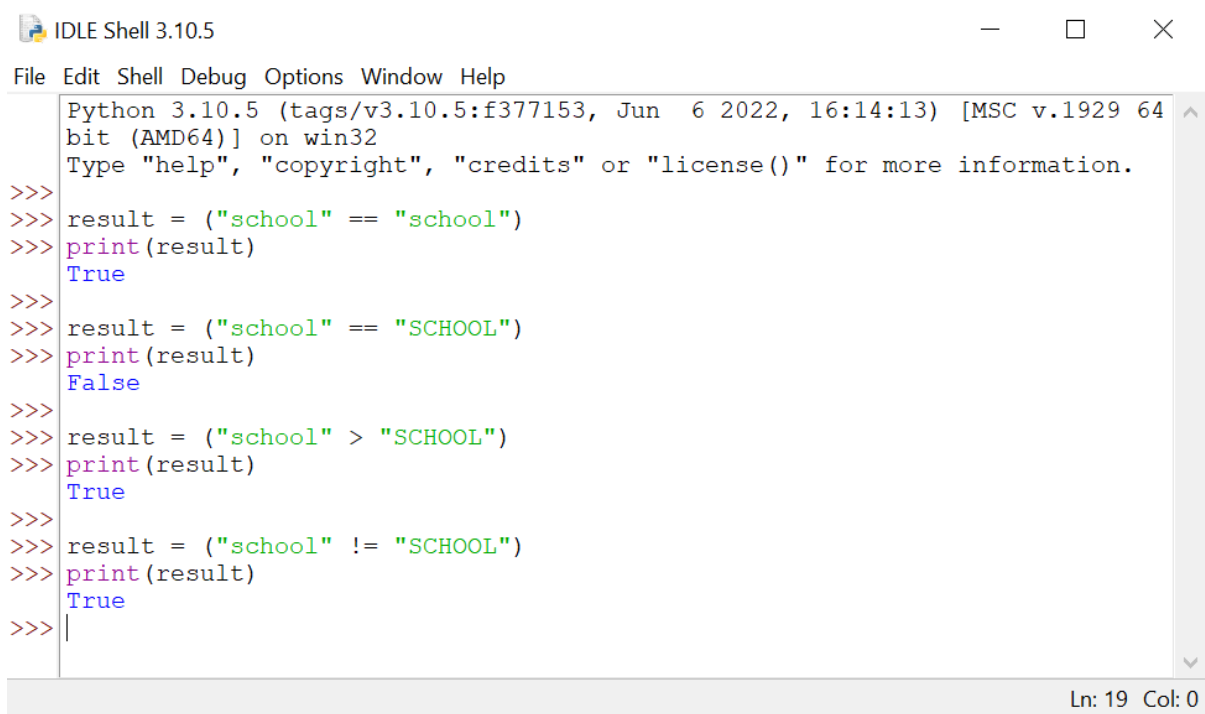
This is done using the following operators:

'=='	<b>This checks whether two strings are equal</b>
'!='	This checks if two strings are not equal
'<'	This checks if the string on its left is smaller than that on its right
'<='	This checks if the string on its left is smaller than or equal to that on its right
'>'	This checks if the string on its left is greater than that on its right
'>='	This checks if the string on its left is greater than or equal to that on its right

Table 4: Comparison Operators

Comparison of strings is performed character by character comparison rules for ASCII and Unicode. ASCII values of numbers 0 to 9 are from 48 to 57, uppercase (A to Z) are from 65 to 90, and lowercase (a to z) are from 97 to 122.

The comparison operators working with strings is explained as under:



```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> result = ("school" == "school")
>>> print(result)
True
>>>
>>> result = ("school" == "SCHOOL")
>>> print(result)
False
>>>
>>> result = ("school" > "SCHOOL")
>>> print(result)
True
>>>
>>> result = ("school" != "SCHOOL")
>>> print(result)
True
>>>

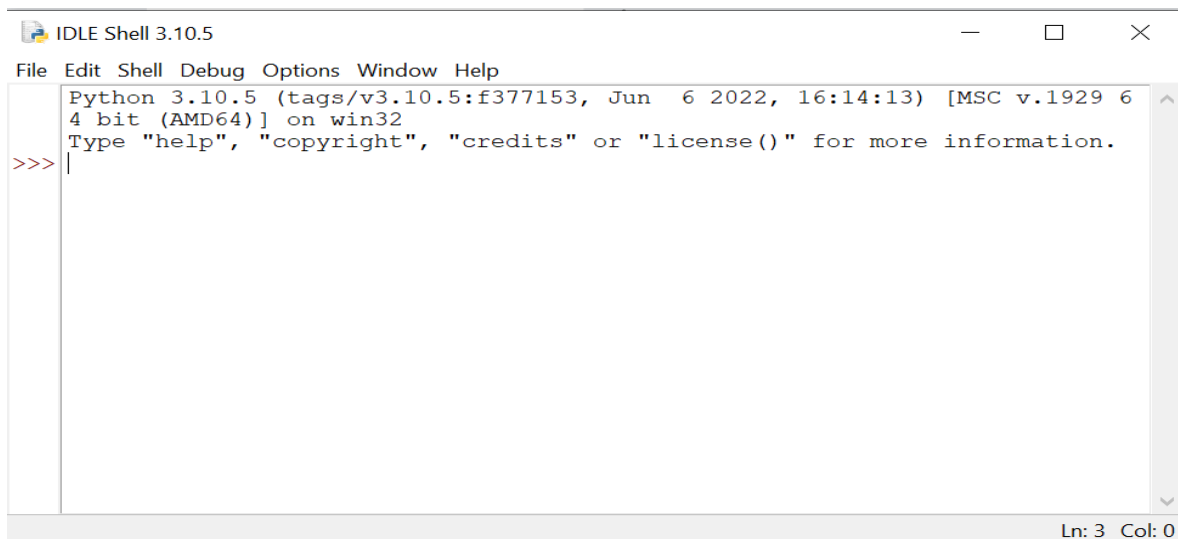
```

Ln: 19 Col: 0

String Figure 10: Comparison operators working with Strings

#### 4.1.7 accepting input from console

Console (also called Shell) is basically a command line interpreter that takes input from the user i.e one command at a time and interprets it. If it is error free then it runs the command and gives required output otherwise shows the error message. Python console is like:



```

Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 6
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>

```

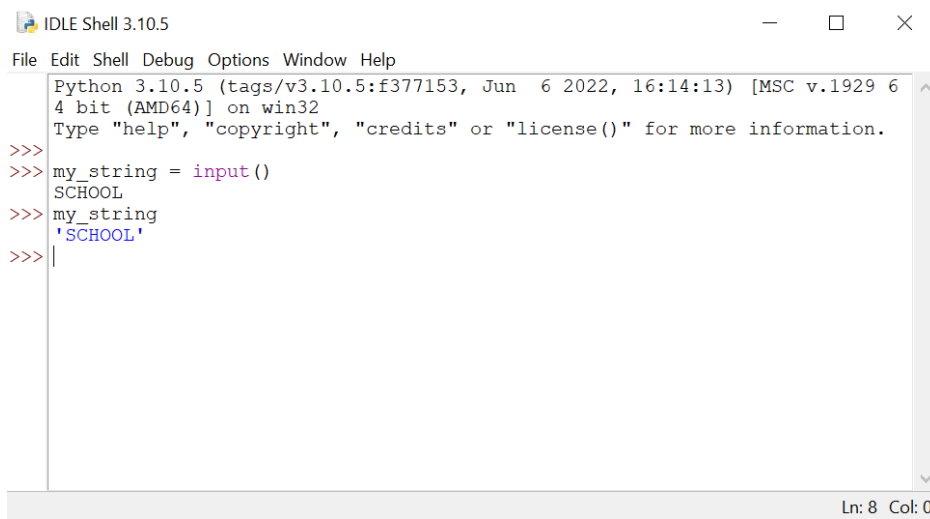
Ln: 3 Col: 0

String Figure 11: Accepting Input from console

Here we write command and to execute the command just press enter key and your command will be interpreted. For coding in Python you must know the basics of the console used in Python. The primary prompt of the python console is the three greater than symbols ">>>"

You are free to write the next command on the shell only when after executing the first command these prompts have appeared. The Python Console accepts command in Python which you write after the prompt.

User enters the values in the Console and that value is then used in the program as it was required. To take input from the user we make use of a built-in function *input()*.



```

IDLE Shell 3.10.5
File Edit Shell Debug Options Window Help
Python 3.10.5 (tags/v3.10.5:f377153, Jun 6 2022, 16:14:13) [MSC v.1929 6
4 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> my_string = input()
>>> SCHOOL
>>> my_string
>>> 'SCHOOL'
>>> |
  
```

String Figure 12: Input from user using build-in function input()

#### 4.1.8 print statements

Python print() function prints the message to the screen or any other standard output device.

Syntax:

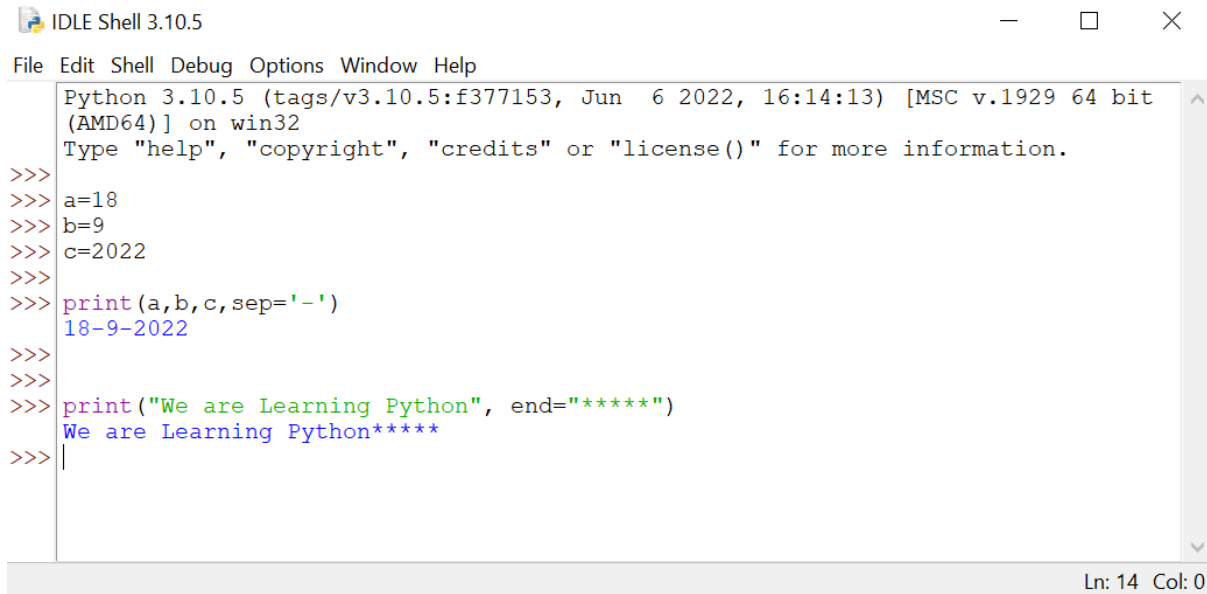
**print(value(s) , sep= ' ', end = '\n')**

Parameters:

<b>value(s)</b>	Any value, and as many as you like. Will be converted to string before printed
<b>sep='separator'</b>	(Optional) Specify how to separate the objects, if there is more than one.Default : ' '
<b>end='end'</b>	(Optional) Specify what to print at the end. Default : '\n'
<b>Return Type</b>	It returns output to the screen

Table 5: Print statement parameters

Though it is not necessary to pass arguments in the print() function, it requires an empty parenthesis at the end that tells python to execute the function rather calling it by name. Now, let's explore the optional arguments that can be used with the print() function.



```

Python 3.10.5 (tags/v3.10.5:f377153, Jun  6 2022, 16:14:13) [MSC v.1929 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
>>> a=18
>>> b=9
>>> c=2022
>>>
>>> print(a,b,c,sep='-')
18-9-2022
>>>
>>>
>>> print("We are Learning Python", end="*****")
We are Learning Python*****
>>> |

```

Ln: 14 Col: 0

String Figure 13: Print() Function

#### 4.1.9 simple programs on strings

##### 4.1.9.1 Write a program to find length of a string.

```

# PROGRAM TO FIND LENGTH OF A STRING

my_string = "We are going to learn Python"

count = 0

for i in my_string:

    count = count+1

print("length of string is : ", count)

Output:

length of string is : 28

```

String Example 5: program to find length of a string

**4.1.9.2 Write a program to confirm presence of a character in string.**

```
# PROGRAM TO CONFIRM PRESENCE OF A CHARACTER IN STRING.
my_string = input("Enter the string: ")
my_char = input("Enter the character to confirm: ")
flag = my_char in my_string
print(flag)
Output:
Enter the string: COMPUTER
Enter the character to confirm: E
True
```

String Example 6: confirm presence of a character in string

**4.2 Sequence Data Types****4.2.1. list**

In simple language, a list is a collection of things, enclosed in [ ] and separated by commas. Lists are used to store multiple items in a single variable.

Creation of list

Lists are created using square brackets:

```
# PROGRAM TO CREATE A LIST
My_list = ["apple", "banana", "cherry"]
print(My_list)
Output:
['apple', 'banana', 'cherry']
```

Sequence Example 1: Creation of list

**4.2.2. tuple**

A tuple in Python is similar to a list. The difference between the two is that we cannot change the elements of a tuple once it is assigned whereas we can change the elements of a list. Tuples are also used to store multiple items in a single variable.

Creating a Tuple

A tuple is created by placing all the items (elements) inside parentheses (), separated by commas. The parentheses are optional, however, it is a good practice to use them.

A tuple can have any number of items and they may be of different types (integer, float, list, etc.).

Tuple is created using parenthesis () as explained below:

<pre> # PROGRAM TO CREATE DIFFERENT TYPES OF   TUPLES # Empty tuple my_tuple = () print(my_tuple) # Tuple having integers my_tuple = (10, 100, 1000) print(my_tuple) # tuple with mixed datatypes my_tuple = (1, "SCHOOL", 3.4) print(my_tuple) # nested tuple my_tuple = ("COMPUTER", [20, 100, 6], (1, 10, 3)) print(my_tuple) </pre>	<pre> Output: () (10, 100, 1000) (1, 'SCHOOL', 3.4) ('COMPUTER', [20, 100, 6], (1, 10, 3)) </pre>
---	---

Sequence Example 2: Create Different types of tuple

### 4.2.3. Dictionary

Python dictionary is an unordered collection of items. Each item of a dictionary has a key/value pair.

Creating a dictionary

Creating a dictionary is as simple as placing items inside curly braces {} separated by commas.

An item has a key and a corresponding value that is expressed as a pair (key: value). While the values can be of any data type and can repeat, keys must be of immutable type (with immutable elements) and must be unique.



<pre># PROGRAM TO CREATE DIFFERENT TYPES OF   DICTIONARIES  # empty dictionary my_dict = {} print(my_dict) # dictionary with integer keys my_dict = {1: 'male', 2: 'female'} print(my_dict) # dictionary with mixed keys my_dict = {'name': 'Amit', 1: [100, 200, 3]} print(my_dict) # using dict() my_dict = dict({1:'glass', 2:'bat'}) print(my_dict) # from sequence having each item as a pair my_dict = dict([(1,'a'), (2,'b')]) print(my_dict)</pre>	<p>Output:</p> <pre>{}</pre> <pre>{1: 'male', 2: 'female'}</pre> <pre>{'name': 'Amit', 1: [100, 200, 3]}</pre> <pre>{1: 'glass', 2: 'bat'}</pre> <pre>{1: 'a', 2: 'b'}</pre>
--	--

Sequence Example 3: Different types of Dictionaries

#### 4.2.4. Indexing and accessing elements of lists, tuples and dictionaries

##### Accessing elements of Lists

We can use the index operator `[]` to access an item in a list. In Python, indices start at 0. So, a list having 5 elements will have an index from 0 to 4.

Trying to access indexes other than these will raise an `IndexError`. The index must be an integer. We can't use float or other types, this will result in `TypeError`.

Nested lists are accessed using nested indexing.

<pre> # PROGRAM TO ACCESS ELEMENTS OF   LIST my_list = ['a', 'b', 'c', 'd', 'e'] # first item print(my_list[0]) # third item print(my_list[2]) # fifth item print(my_list[4]) # Nested List n_list = ["Python", [2, 10, 11, 25]] # Nested indexing print(n_list[0][1]) print(n_list[1][3]) # Using Negative Indexing print(my_list[-1]) # Last Element print(my_list[-5]) # 5th Element # Error! Only integer can be used for indexing print(my_list[4.0]) </pre>	<pre> Output: a c e y 25 e a Traceback (most recent call last):   File "C:\Training\CABA-MDTP COURSE MATERIAL\Book Programs.py", line 24, in &lt;module&gt;     print(my_list[4.0]) TypeError: list indices must be integers or slices, not float </pre>
---	--

Sequence Example 4: Program to access elements of list

### Accessing elements of Tuples

We can use the index operator `[]` to access an item in a tuple, where the index starts from 0. So, a tuple having 6 elements will have indices from 0 to 5. Trying to access an index outside of the tuple index range (6, 7, ... in this example) will raise an `IndexError`.

The index must be an integer, so we cannot use float or other types. This will result in `TypeError`. Likewise, nested tuples are accessed using nested indexing, as shown in the example below.

<pre> # PROGRAM TO ACCESS ELEMENTS OF TUPLE  # Different types of tuples # Empty tuple my_tuple = () print(my_tuple)  # Tuple having integers my_tuple = (100, 200, 300) print(my_tuple)  # tuple with mixed datatypes my_tuple = (1, "PYTHON", 5.4) print(my_tuple)  # nested tuple my_tuple = ("COMPUTER", [80, 40, 60], (10, 20, 30)) print(my_tuple)  my_tuple = ('A', 'B', 'C', 'D', 'E', 'F') #Using Negative Indexing print(my_tuple[-1]) print(my_tuple[-6]) </pre>	<pre> Output: () (100, 200, 300) (1, 'PYTHON', 5.4) ('COMPUTER', [80, 40, 60], (10, 20, 30)) F A </pre>
---	---

Sequence Example 5: Access Elements of Tuple

### Accessing elements of Dictionaries

While indexing is used with other data types to access values, a dictionary uses keys. Keys can be used either inside square brackets [] or with the get() method.

If we use the square brackets [], KeyError is raised in case a key is not found in the dictionary. On the other hand, the get() method returns None if the key is not found.

<pre> my_list = [1, 2, 7, 4, 5] print(my_list[2:]) </pre>	<pre> Output: [7, 4, 5] </pre>
---	--------------------------------

<pre># PROGRAM TO ACCESS ELEMENTS OF   DICTIONARIES # Different types of dictionary # get vs [] for retrieving elements my_dict = {'name': 'Tushar', 'age': 36} print(my_dict['name']) print(my_dict.get('age')) # Trying to access keys which doesn't exist throws error # Output None print(my_dict.get('school')) # KeyError print(my_dict['school'])</pre>	<pre>Output: Tushar 36 None Traceback (most recent call last):   File "C:\Training\CABA-MDTP COURSE MATERIAL\Book Programs.py", line 15, in &lt;module&gt;     print(my_dict['school']) KeyError: 'school'</pre>
--	--

Sequence Example 6: Accessing elements of Dictionaries

#### 4.2.5. slicing in list, tuple

Concept of slicing is same as of string slicing and is implemented as also in the similar manner. Slicing in List

The format for list slicing is [start:stop:step].

- start is the index of the list where slicing starts.
- stop is the index of the list where slicing ends.
- step allows you to select nth item within the range start to stop.

<pre>my_list = [1, 2, 5, 4, 5] print(my_list[:])</pre>	<pre>Output: [1, 2, 5, 4, 5]</pre>
--	------------------------------------

Sequence Example 7: Get all the items in a list.

Sequence Example 8: To get all the items after a specific position

<pre>my_list = [1, 3, 7, 4, 5] print(my_list[:2])</pre>	<pre>Output: [1, 3]</pre>
---	---------------------------

Sequence Example 9: get all the items before a specific position

```
my_list = [1, 2, 7, 4, 5]
print(my_list[2:4])
```

Output:  
[7, 4]

Sequence Example 10: get all the items from one position to another position

```
my_list = [0, 2, 7, 4, 5]
print(my_list[::2])
```

Output:  
[0, 7, 5]

Sequence Example 11: Get the Items at Specified Intervals

### Slicing in Tuple

```
# Accessing tuple elements using slicing
my_tuple = ('c','o','m','p','u','t','e','r','s')
('p','r','o','g','r','a','m','i','z')
# elements 2nd to 4th
print(my_tuple[1:4])

# elements beginning to 2nd
print(my_tuple[:-7])

# elements 8th to end
print(my_tuple[7:])
# elements beginning to end
print(my_tuple[:])
```

Output:  
( 'o', 'm', 'p' )  
( 'c', 'o' )  
( 'r', 's' )  
( 'c', 'o', 'm', 'p', 'u', 't', 'e', 'r', 's' )

Sequence Example 12: Slicing in Tuple with code.

### 4.2.6. Concatenation on list, tuple and dictionary

Concatenation of List.

Two separate list can be combined or joined using '+' operation.

```
list_1 = [5, 'c']
list_2 = [3, 9, 8]
list_joined = list_1 + list_2
print(list_joined)
```

Output:  
[5, 'c', 3, 9, 8]

Sequence Example 13: Concatenation of list

Two list can be combined using set () and list() functions so that the final list contains only the unique value.

<pre>list_1 = [5, 'P'] list_2 = [5,8,0,3] list_joined = list(set(list_1 + list_2)) print(list_joined)</pre>	<pre>Output: [0, 3, 5, 8, 'P']</pre>
---	--------------------------------------

Sequence Example 14: Concatenation of two lists using set()

In above example, the set () selects the unique values and list () converts the set into list. We can concatenate a list to another list or simply merge two lists using extend() function.

<pre>list_1 = [5, 'a'] list_2 = [7, 5, 5] list_2.extend(list_1) print(list_2)</pre>	<pre>Output: [7, 5, 5, 5, 'a']</pre>
---	--------------------------------------

Sequence Example 15: Concatenation of two lists using extend()

Concatenation of Tuple.

Concatenation of two separate tuples can be done using '+' operation. Concatenation of two tuples.

<pre>my_tuple_1 = (22, 14, 0, 67, 98, 11) my_tuple_2 = (11, 96, 0, 1, 23, 44) print("The first tuple is: ") print(my_tuple_1) print("The second tuple is: ") print(my_tuple_2) my_result = my_tuple_1 + my_tuple_2 print("The tuple after concatenation is: ") print(my_result) (22, 14, 0, 67, 98, 11, 11, 96, 0, 1, 23, 44)</pre>	<pre>Output: The first tuple is: (22, 14, 0, 67, 98, 11) The second tuple is: (11, 96, 0, 1, 23, 44) The tuple after concatenation is:</pre>
---	--

Sequence Example 16: Concatenation of two tuples.

Concatenation of Dictionary.

Concatenation of two separate dictionaries can be done using '|' operation. Concatenation of dictionaries using '|' operator:

<pre>dict_1 = {5: 't', 7: 'y'} dict_2 = {5: 'r', 6: 'd'} print(dict_1   dict_2)</pre>	<pre>Output: {5: 'r', 7: 'y', 6: 'd'}</pre>
---	---

Sequence Example 17: Concatenation of dictionaries using '|' operator

#### 4.2.7. Concept of mutability

##### Mutable Definition

Mutable is when something is changeable or has the ability to change. In Python, 'mutable' is the ability of objects to change their values. These are often the objects that store a collection of data.

##### Immutable Definition

Immutable is the when no change is possible over time. In Python, if the value of an object cannot be changed over time, then it is known as immutable. Once created, the value of these objects is permanent.

##### List of Mutable and Immutable objects

Objects of built-in type that are mutable are:

- Lists
- Sets
- Dictionaries
- User-Defined Classes (It purely depends upon the user to define the characteristics)

Objects of built-in type that are immutable are:

- Numbers (Integer, Rational, Float, Decimal, Complex & Booleans)
- Strings
- Tuples
- Frozen Sets
- User-Defined Classes (It purely depends upon the user to define the characteristics)

##### Objects in Python

In Python, everything is treated as an object. Every object has these three attributes:

- Identity – This refers to the address that the object refers to in the computer's memory.
- Type – This refers to the kind of object that is created. For example- integer, list, string etc.
- Value – This refers to the value stored by the object. For example – List=[1,2,3] would hold the numbers 1,2 and 3

Identity and Type cannot be changed once it's created, values can be changed for Mutable objects.

Explanation of mutable objects using List.

Lists are mutable, meaning their elements can be changed unlike We can use the assignment operator = to change an item or a range of items.

<pre>odd = [1, 3, 5, 11] # change the 1st item odd[0] = 1 print(odd) # change 2nd to 4th items odd[1:4] = [13, 17, 19] print(odd)</pre>	<pre>Output: [1, 3, 5, 11] [1, 13, 17, 19]</pre>
---	--

Example 11: mutable objects using List

Explanation of mutable objects using Dictionary.

Dictionaries are mutable. We can add new items or change the value of existing items using an assignment operator.

If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

<pre># Changing and adding Dictionary Elements my_dict = {'name': 'Tushar', 'age': 29} # update value my_dict['age'] = 30 print(my_dict) # add item my_dict['address'] = 'Sarojini Nagar' print(my_dict)</pre>	<pre>Output: {'name': 'Tushar', 'age': 30} {'name': 'Tushar', 'age': 30, 'address': 'Sarojini Nagar'}</pre>
--	---

Example 12: mutable objects using Dictionary

Explanation of immutable objects using String and Tuple.

Strings and Tuple both are immutable object means once they are created their elements values can't be changed.

<pre># Changing tuple values my_tuple = (5, 7, 10, 9) my_tuple[2]=100 print(my_tuple)</pre>	<pre>Output: Traceback (most recent call last):   File "&lt;string&gt;", line 3, in &lt;module&gt; TypeError: 'tuple' object does not support item assignment</pre>
---	---

Example 13: Explanation of immutable objects using Tuple.

<pre># Changing a string value my_string = "PYTHON" my_string[1]='K' print(my_string )</pre>	<pre>Output: Traceback (most recent call last):   File "&lt;string&gt;", line 3, in &lt;module&gt; TypeError: 'str' object does not support item assignment</pre>
--	---

Example 14: Explanation of immutable objects using String



However, we can also assign a tuple or string to different values (reassignment).

Other operations on list, tuple and dictionary

Adding item to a list using the append() method

<pre>list_1 = [5, 'a'] list_1.append(10) print(list_1)</pre>	<p>Output:</p> <pre>[5, 'a', 10]</pre>
--	--

Example 15: Adding item to a list using the append()

The \* operator repeats a list for the given number of times.

<pre>list_1 = [5, 'a'] print(list_1*3)</pre>	<p>Output:</p> <pre>[5, 'a', 5, 'a', 5, 'a']</pre>
--	--

Example 16: repeat a list for the given number of times

Inserting one item at a desired location by using the method insert()

<pre>list1 = [1, 9, 10, 11] list1.insert(3,23) #inserting 23 at 3rd location print(list1)</pre>	<p>Output:</p> <pre>[1, 9, 10, 23, 11]</pre>
---	--

Example 17: Inserting one item at a desired location

Delete operation on List

<pre># Deleting list items my_list = ['c', 'o', 'm', 'p', 'u', 't', 'e', 'r'] # delete one item del my_list[2] print(my_list) # delete multiple items del my_list[1:5] print(my_list) # delete the entire list del my_list # Error: List not defined print(my_list)</pre>	<p>Output:</p> <pre>['c', 'o', 'p', 'u', 't', 'e', 'r'] ['c', 'e', 'r']</pre> <p>Traceback (most recent call last):</p> <pre>File "&lt;string&gt;", line 18, in &lt;module&gt; NameError: name 'my_list' is not defined</pre>
---	---

Example 18: Delete operation on List

Usage of pop() , remove() and clear() method on List.

<pre>my_list = ['c', 'o', 'm', 'p', 'u', 't', 'e', 'r'] my_list.remove('p') # Delete a particular item print(my_list) print(my_list.pop(1)) # Delete item at index [1] print(my_list) print(my_list.pop()) # Delete last item of the list print(my_list) my_list.clear() # Delete complete list print(my_list)</pre>	<pre>Output: ['c', 'o', 'm', 'u', 't', 'e', 'r'] o ['c', 'm', 'u', 't', 'e', 'r'] r ['c', 'm', 'u', 't', 'e'] []</pre>
--	--

Example 19: Usage of pop() , remove() and clear() method on List

Changing and Adding Dictionary elements

If the key is already present, then the existing value gets updated. In case the key is not present, a new (key: value) pair is added to the dictionary.

<pre># Changing and adding Dictionary Elements my_dict = {'name': 'Kapil', 'age': 39} # update value my_dict['age'] = 27 print(my_dict) # add item my_dict['address'] = 'Delhi' print(my_dict)</pre>	<pre>Output: {'name': 'Kapil', 'age': 27} {'name': 'Kapil', 'age': 27, 'address': 'Delhi'}</pre>
--	--

Example 20: Changing and Adding Dictionary elements

Removing elements from Dictionary

<pre># Removing elements from a dictionary # create a dictionary cubes = {1: 1, 2: 6, 3: 27, 4: 64, 5: 125} # remove a particular item, returns its value print(cubes.pop(4)) print(cubes)</pre>	<pre>Output: 64 {1: 1, 2: 6, 3: 27, 5: 125} (5, 125) {1: 1, 2: 6, 3: 27} {} Traceback (most recent call last):</pre>
--	--

<pre># remove an arbitrary item, return (key,value) print(cubes.popitem()) print(cubes)  # remove all items cubes.clear()  # Output: {} print(cubes) # delete the dictionary itself del cubes # Throws Error print(cubes)</pre>	<pre>File "&lt;string&gt;", line 22, in &lt;module&gt; NameError: name 'cubes' is not defined</pre>
---	---

Example 21: Removing elements from Dictionary

Finding Minimum and Maximum in List and Tuple.

<pre># Maximum and Minimum in a List even = [2,4,6,8,10] print(max(even)) print(min(even)) print(" End of Operation on List ") # Maximum and Minimum in a Tuple odd = (1,3,5,7,9) print(max(odd)) print(min(odd)) print("End of Operation on Tuple ")</pre>	<pre>Output: 10 2 End of Operation on List 9 1 End of Operation on Tuple</pre>
---	--

Example 22: Finding Minimum and Maximum in List and Tuple

Finding Mean in List and Tuple.

<pre>import statistics # Mean of a List even = [2,4,6,8,10] print(statistics.mean(even)) print(" End of Operation on List ")</pre>	<pre>Output: 10 2 End of Operation on List 9</pre>
--	--

<pre># Mean of a Tuple odd = (1,3,5,7,9) print(statistics.mean(odd)) print(" End of Operation on Tuple ")</pre>	<pre>1 End of Operation on Tuple</pre>
---	--

Example 23: Finding Mean in List and Tuple

Linear search on list of numbers.

<pre># Linear Search on List num= int(input("Enter the number to search: ")) even = [2,4,6,8,10] flag=0 for i in even:     if(i==num):         flag=1         break if(flag==1):     print("Entered Number is Present") else:     print("Entered Number is not Present")</pre>	<pre>Output: Enter the number to search: 10 Entered Number is Present</pre>
--	---

Example 24: Linear search on list of numbers

Counting the frequency of elements in a list using a dictionary.

<pre>my_list =[2, 2, 2, 1, 1, 3, 2, 3, 3, 1, 7, 7, 7, 2, 2, 2, 2] print(my_list) print() freq = {}# Creating an empty dictionary for item in my_list:     if (item in freq):         freq[item] += 1     else:         freq[item] = 1 for key, value in freq.items():     print ("% d : % d"%(key, value))</pre>	<pre>Output: [2, 2, 2, 1, 1, 3, 2, 3, 3, 1, 7, 7, 7, 2, 2, 2, 2]  2 : 8 1 : 3 3 : 3 7 : 3</pre>
--	---

Example 25: Counting the frequency of elements in a list using a dictionary

## Exercises

### Multiple Choice Questions

1)A string is series of \_\_\_\_\_

- a. characters
- b. integers
- c. double
- d. float

2)String index starts from :

- a. 1
- b. 0
- c. -1
- d. None of the above

3)Slicing in string is used to extract :

- a. part of the string
- b. complete string
- c. used to empty string
- d. None of the above

4)In negative indexing the last character of a string can be accessed using index value

- a. Length of string
- b. 0
- c. -1
- d. None of the above

5)Which of the following are sequence data type:

- a. List
- b. Tuple
- c. Dictionary
- d. All of the above

6)Which is the Replication operator for string

- a. '+'
- b. '-'
- c. '='
- d. '\*'

**7)Membership operator 'in' confirms :**

- a. Duplicate elements
- b. Presence of an element
- c. Absence of an element
- d. None of the above

**8)Elements in a List are enclosed in which type of bracket:**

- a. [ ]
- b. ()
- c. {}
- d. Any of the above

**9)pop() function is used to :**

- a. add an item
- b. delete an item
- c. merge items
- d. None of the above

**10)Concatenation operator is :**

- a. '+'
- b. '-'
- c. '&'
- d. '%'

**State whether statement is true or false**

- 1)Integer values can't be stored as strings. (T/F)
- 2)Indexing of string is done manually by the programmer. (T/F)
- 3)We can use positive and negative indexing to access string elements. (T/F)
- 4)Comparison operators cannot be used for comparing strings. (T/F)
- 5)Lists are mutable. (T/F)
- 6)Tuple are mutable. (T/F)

- 7) Dictionaries are mutable. (T/F)
- 8) insert () function is used to add an element at desired location in a List. (T/F)
- 9) In dictionary elements are stored as key:value pair. (T/F)
- 10) Membership operator is not available in dictionaries. (T/F)

### Fill in the blanks

- 1. A string can be traversed using an \_\_\_\_\_.
- 2. A string can be accessed using positive and \_\_\_\_\_ index.
- 3. A \_\_\_\_\_ function is used to add element in a List at last location.
- 4. Concatenation of two Dictionaries can be done using \_\_\_\_\_ operator.
- 5. \_\_\_\_\_ function is used to delete complete elements in a list.
- 6. Elements in a dictionary are stored as key and \_\_\_\_\_ pair.
- 7. The method of extracting part of tuple is called \_\_\_\_\_.
- 8. To display the message on monitor \_\_\_\_\_ function is used.
- 9. \_\_\_\_\_ function is used to take input from console.
- 10. To check two strings are equal \_\_\_\_\_ operator is used.

### LAB exercises

- 1. Write a program to display index value against each character in string.
- 2. Write a program to maximize frequency in a string..
- 3. Write a program to split / join string
- 4. Write a program to find length of list.
- 5. Write a program for reversing a list.
- 6. Write a program for finding largest and smallest number in a list.
- 7. Write a program to print all odd numbers in tuple.
- 8. Write a program to join two tuples if their first element is same.
- 9. Write a program to explain min(), max() and mean() functions using List.
- 10. Write a program to explain mutability using List, Tuple and Dictionary.

## Chapter 5

### Functions

#### 5.1 Top-down Approach of Problem Solving

Top **down** analysis is a problem solving mechanism whereby a given problem is successively broken down into smaller and smaller sub-problems or operations until a set of easily solvable (by computer) sub-problems is arrived at.

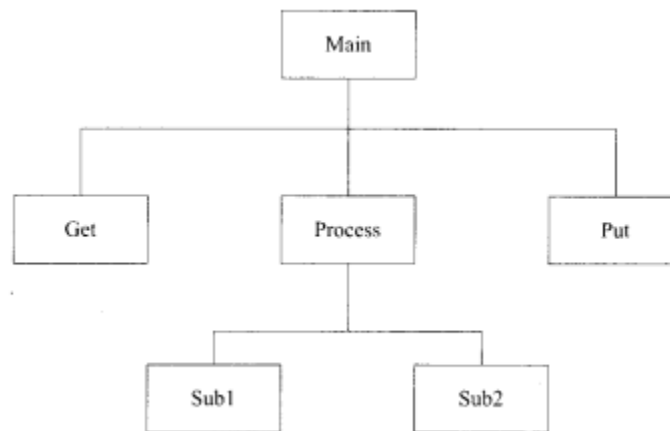
Using the top-down approach. It is possible to achieve a very detailed breakdown, however, it should be remembered that our aim is to identify easily solvable sub-problems.

➤ The top-down approach is used in the system analysis and design process.

The top-down approach, starting at the general levels to gain an understanding of the system and gradually moving down to levels of greater detail is done in the analysis stage. In the process of moving from top to bottom, each component is exploded into more and more details. Thus, the problem at hand is analysed or broken down into major components, each of which is again broken down if necessary.

➤ The top-down process involves working from the most general down to the most specific.

The design of modules is reflected in hierarchy charts such as the one shown in Figure below:



The purpose of procedure Main is to coordinate the three branch operations e.g. Get, Process, and Put routines. These three routines communicate only through Main. Similarly, Sub1 and Sub2 can communicate only through the Process routine.

#### Advantages of Top-down Approach

The advantages of the top-down approach are as follows:

This approach allows a programmer to remain “on top of” a problem and view the developing solution in context. The solution always proceeds from the highest level downwards.

By dividing the problem into a number of sub-problems, it is easier to share problem development. For example, one person may solve one part of the problem and the other person may solve another part of the problem.



Since debugging time grows quickly when the program is longer, it will be to our advantage to debug a long program divided into a number of smaller segments or parts rather than one big chunk. The top-down development process specifies a solution in terms of a group of smaller, individual subtasks. These subtasks thus become the ideal units of the program for testing and debugging.

## 5.2 Modular Programming and Functions

### 5.2.1. Modular Programming

Modular programming is defined as a software design technique that focuses on separating the program functionality into independent, interchangeable methods/modules. Each of them contains everything needed to execute only one aspect of functionality.

Talking of modularity in terms of files and repositories, modularity can be on different levels -

- Libraries in projects
- Function in the files

Files in the libraries or repositories

Modularity is all about making blocks, and each block is made with the help of other blocks. Every block in itself is solid and testable and can be stacked together to create an entire application. Therefore, thinking about the concept of modularity is also like building the whole architecture of the application.

Examples of modular programming languages - All the object-oriented programming languages like C++, Java, etc., are modular programming languages.

### 5.2.2. Module

A module is defined as a part of a software program that contains one or more routines. When we merge one or more modules, it makes up a program. Whenever a product is built on an enterprise level, it is a built-in module, and each module performs different operations and business. Modules are implemented in the program through interfaces. The introduction of modularity allowed programmers to reuse prewritten code with new applications. Modules are created and merged with compilers, in which each module performs a business or routine operation within the program.

For example – SAP (System, Applications, and Products) comprises large modules like finance, payroll, supply chain, etc. In terms of softwares example of a module is Microsoft Word which uses Microsoft paint to help users create drawings and paintings.

### 5.2.3. Advantages of Modular Design

- Rather than focusing on the entire problem at hand, a module typically focuses on one relatively small portion of the problem.
- Since module is small, it is simpler to understand it as a unit of code. It is therefore easier to test and debug, especially if its purpose is clearly defined and documented.
- Program maintenance becomes much easier because the modules that are likely to be affected are quickly identified.
- In a very large project, several programmers may be working on a single problem. Using a modular approach, each programmer can be given a specific set of modules to work on. This enables the whole project to be completed faster.

- More experienced programmers can be given a more complex module to write, and the junior programmers can work on simpler modules. Modules can be tested independently, thereby shortening the time taken to get the whole project working.
- If a programmer leaves a project, it is easier for someone else to take over a set of self-contained modules.
- A large project becomes easier to monitor as well as to control.

### 5.3 Function and function parameters

A function is an isolated block of code that performs a specific task.

Functions are useful in programming because they eliminate needless and excessive copying and pasting of code in a program. If a certain action is required often and in different places, that is a good indicator that you can write a function for it. Functions are meant to be reusable.

Functions also help organize your code. If you need to make a change, you'll only need to update that certain function. This saves you from having to search for different pieces of the same code that have been scattered in different locations in your program by copying and pasting.

This complies with the DRY (Don't Repeat Yourself) principle in software development. The code inside a function runs only when the function is called. Functions can accept arguments and defaults and may or not return values back to the caller once the code has run.

#### 5.3.1. HOW to Define a Function in Python

The general syntax for creating a function in Python looks something like this:

```
def
function_name(parameters):
    function body
```

Syntax 1: Define a Function

Let's break down what's happening here:

- `def` is a keyword that tells Python a new function is being defined.
- Next comes a valid function name of your choosing. Valid names start with a letter or underscore but can include numbers. Words are lowercase and separated by underscores. It's important to know that function names can't be a Python reserved keyword.
- Then we have a set of opening and closing parentheses, `()`. Inside them, there can be zero, one, or more optional comma separated parameters with their optional default values. These are passed to the function.
- Next is a colon, `:`, which ends the function's definition line.
- Then there's a new line followed by a level of indentation (you can do this with 4 spaces using your keyboard or with 1 Tab instead). Indentation is important since it lets Python know what code will belong in the function.
- Then we have the function's body. Here goes the code to be executed – the contents with the actions to be taken when the function is called.
- Finally, there's an optional return statement in the function's body, passing back a value to the caller when the function is exited.

Keep in mind that if you forget the parentheses `()` or the colon `:` when trying to define a new function, Python will let you know with a Syntax Error.

### 5.3.2. How to Define and Call a Basic Function in Python

Below is an example of a basic function that has no return statement and doesn't take in any parameters.

#### 5.3.3. It just prints hello world whenever it is called.

```
def hello_world_func():  
    print("hello world")
```

Code 1: defining a function

Once you've defined a function, the code will not run on its own. To execute the code inside the function, you have to make a function invocation or else a function call.

You can then call the function as many times as you want. To call a function you need to do this:

```
Function_name(arguments)
```

Code 2: Calling a function

Here's a breakdown of the code:

- Type the function name.
- The function name has to be followed by parentheses. If there are any required arguments, they have to be passed in the parentheses. If the function doesn't take in any arguments, you still need the parentheses.

To call the function from the example above, which doesn't take in any arguments, do the following.

### 5.3.4. How to Define and Call Functions with Parameters

So far you've seen simple functions that don't really do much besides printing something to the console. What if you want to pass in some extra data to the function? We've used terms here like *parameter* and *arguments*. What are their definitions exactly?

```
hello_world_func()  
#Output  
#hello world
```

```
def hello_to_you(name):  
    print("Hello " + name)
```

Code 3: Defining Functions with parameters

Parameters are a named placeholder that pass information into functions. They act as variables that are defined locally in the function's definition line.

In the example above, there is one parameter, name. We can pass more than one parameters in the function as shown below :

```
def hello_to_you(name):
    print(f"Hello {name}")
hello_to_you("Timmy")
#Output
# Hello Timmy
```

Code 4: Calling function with parameter

The function can be called many times, passing in different values each time.

```
def hello_to_you(name):
    print(f"Hello {name}")
    hello_to_you("Timmy")
hello_to_you("Kristy")
hello_to_you("Jackie")
hello_to_you("Liv")

#Output:
#"Hello Timmv"
```

Code 5: Function can be called many times

## 5.4 Local Variables

Python local variable plays an important role in the entire python programming language as it is used for any scope definition and manipulation. A local variable in Python is always declared within a specific scope like it is mostly present within any function's body where other members can access it. Therefore, it is very difficult and rare that local variables will be present outside the scope or outside the function. If a variable is present outside the scope, it is considered a global variable, and all the members become unreachable to a local variable.

### 5.4.1. Syntax of Local Variable in Python

The syntax flow for the local variable declaration in function for Python includes the following representation:

```
Function_declaration():
Variable= "var_assign"
Logic statement ()
Function_declaration() //calling of the function
```

Syntax 2: Declaration of the Local Variable

```
def dinner_prep():
    dine = 'Pizza_with_extra_topping'
    print('Please have a pizza with extra_topping', dine)
dinner_prep()
```

Output

Please have a pizza with extra\_topping Pizza\_with\_extra\_topping

Example 26: Working of Local Variable in Python

A function is declared, and then the variable is taken, which creates the memory, and on top of it, a variable is assigned, which makes it a local variable after which the function is called and then the following logic statement is called to perform a lot of manipulation and work.

### 5.4.2 How Local Variable Works in python?

This program demonstrates the local variable when defined within the function where the variable is declared within function and then a statement followed by the function calling as shown in the output below.

A local variable in Python plays a significant role in the sense it helps in making the function and the code snippet access to other member variables with manipulation simple and easy. In addition, local variables help in making the entire workflow with the global variable compatible and less complex. Also, the nested functions or statements play a very nice blend with local variables.

## 5.5 The Return Statement

A **return statement** is used to end the execution of the function call and “returns” the result (value of the expression following the return keyword) to the caller. The statements after the return statements are not executed. If the return statement is without any expression, then the special value None is returned. A **return statement** is overall used to invoke a function so that the passed statements can be executed.

**Note:** Return statement can not be used outside the function.

```
defun():
    statements
    return [expression]
```

Syntax 3: The return statement

## 5.6 Default argument values

Function arguments can have default values in Python. We can provide a default value to an argument by using the assignment operator (=). Here is an example.

```
# Python program to
# demonstrate return statement
def add(a, b):
    return a + b

print(add(5,7))
Output
12
```

Code 6: Python Program to Demonstrate Return statement

<pre>def greet(name, msg="Greeting of Day!"):     """     This function greets to     the person with the     provided message.      If the message is not provided,     it defaults to "Greeting of Day!"     """     print("Hello", name + ', ' + msg)  greet("Rajesh") greet("Kapil", "How do you do?")</pre>	<p>Output:</p> <p>Hello Rajesh, Greeting of Day!</p> <p>Hello Kapil, How do you do?</p>
--	---

Example 27: Example to explain default arguments value.

In this function, the parameter name does not have a default value and is required (mandatory) during a call.

On the other hand, the parameter msg has a default value of " Greeting of Day!". So, it is optional during a call. If a value is provided, it will overwrite the default value.

Any number of arguments in a function can have a default value. But once we have a default argument, all the arguments to its right must also have default values.

## 5.7 keyword arguments

When we call a function with some values, these values get assigned to the arguments according to their position.

For example, in the above function greet(), when we called it as greet("Kapil", "How do you do?"), the value "Kapil" gets assigned to the argument name and similarly "How do you do?" to msg.

Python allows functions to be called using keyword arguments. When we call functions in this way, the order (position) of the arguments can be changed. Following calls to the above function are all valid and produce the same result.

<pre>def greet(name, msg="Greeting of Day!"):     """     This function greets to     the person with the     provided message.      If the message is not provided,     it defaults to "Greeting of Day!"     """     print("Hello", name + ', ' + msg)  greet("Rajesh") greet("Kapil", "How do you do?")</pre>	<p>Output:</p> <p>Hello Rajesh, Greeting of Day!</p> <p>Hello Kapil, How do you do?</p>
--	---

Example 28: Example to explain default arguments value.

## 5.8 VArArgs parameters.

Python has \*args which allow us to pass the variable number of non keyword arguments to function.

In the function, we should use an asterisk \* before the parameter name to pass variable length arguments. The arguments are passed as a tuple and these passed arguments make tuple inside the function with same name as the parameter excluding asterisk \*.

<pre>def adder(*num):     sum = 0      for n in num:         sum = sum + n     print("Sum:",sum) adder(3,5) adder(4,5,6,7) adder(1,2,3,5,6)</pre>	<pre>Output: Sum: 8 Sum: 22 Sum: 17</pre>
---	---

Example 29:Example to explain VArArgs parameters.

## 5.9 Library function:

A library is a collection of modules or functions in a python that allows doing specific tasks to fulfill user's needs.

### 5.9.1. input()

The input() function reads a line from the input (usually from the user), converts the line into a string by removing the trailing newline, and returns it.

If EOF is read, it raises an EOFError exception.

<pre>inputString = input() # get input from user  print('The inputted string is:', inputString)</pre>	<pre>Output: Yes we are learning The inputted string is: Yes we are learning</pre>
---	--

Example 30:Example to explain input()

<pre>inputString = input('Enter a string:') print('The inputted string is:', inputString)</pre>	<pre>Output: Enter a string:Yes we are learning The inputted string is: Yes we are learning</pre>
---	---

Example 31:input() with a message .

### 5.9.2. eval()

The eval() method parses the expression passed to this method and runs python expression (code) within the program.

<pre>number = 10 # eval performs the multiplication passed as argument square_number = eval('number * number') print(square_number)</pre>	<pre>Output: 100</pre>
---	------------------------

Example 32: Explanation eval() function

<pre># Perimeter of Square def calculatePerimeter(l):     return 4*l  # Area of Square def calculateArea(l):     return l*l  expression = input("Type a function: ")  for l in range(1, 5):     if (expression == 'calculatePerimeter(l)'):         print("If length is ", l, ", Perimeter = ",               eval(expression))     elif (expression == 'calculateArea(l)'):         print("If length is ", l, ", Area = ", eval(expression))     else:         print('Wrong Function')         break</pre>	<pre>Output: Type a function: calculateArea(l) If length is 1 , Area = 1 If length is 2 , Area = 4 If length is 3 , Area = 9 If length is 4 , Area = 16</pre>
---	---

Example 33: Explanation eval() function using code

### 5.9.3. print() function

The print() function prints the given object to the standard output device (screen) or to the text stream file.

<pre>message = 'We are Learning Python' # print the string message print(message)</pre>	<pre>Output: We are Learning Python</pre>
---	---

Example 34: Explanation of print() function.



### 5.9.4. print() Parameters

- objects - object to the printed. \* indicates that there may be more than one object
- sep - objects are separated by sep. Default value: ' '
- end - `end` is printed at last
- file - must be an object with write(string) method. If omitted, sys.stdout will be used which prints objects on the screen.
- flush - If True, the stream is forcibly flushed. Default value: False

<pre>print("Computer") a = 10 # Two objects are passed print("a =", a) b = a # Three objects are passed print('a =', a, '= b')</pre>	<p>Output:</p> <pre>Computer a = 10 a = 10 = b</pre>
--	--

Example 35: explanation of print()

<pre>a = 10 print("a =", a, sep='*****', end='\n\n\n') print("a =", a, sep='+++++', end='\n\n\n')</pre>	<p>Output:</p> <pre>a =*****10  a =+++++10</pre>
---	--

Example 36: print() with separator and end parameters

### 5.9.5 String Functions:

Count function()

The count() method returns the number of occurrences of a substring in the given string.

<pre>message = 'Computer programming' # number of occurrence of 'p' print('Number of occurrence of p:', message.count('p'))</pre>	<p>OUTPUT:</p> <pre>Number of occurrence of p: 2</pre>
---	--

Example 37: Explanation of count() function.

## Syntax of String count

The syntax of count() method is:

```
string.count(substring, start=..., end=...)
```

Syntax 4: Syntax of string count

count() Parameters : count() method only requires a single parameter for execution. However, it also has two optional parameters:

- substring - string whose count is to be found.
- start (Optional) - starting index within the string where search starts.
- end (Optional) - ending index within the string where search ends.

Note: Index in Python starts from 0, not 1. count() method returns the number of occurrences of the substring in the given string.

### find() function

The find() method returns the index of first occurrence of the substring (if found). If not found, it returns -1.

<pre>message = 'Computer Programming is good'  # check the index of 'good'  print(message.find('good'))</pre>	<pre>Output:  24</pre>
---	------------------------

Example 38: Explanation of find() function.

### find() Syntax:

```
str.find(sub[, start[, end]] )
```

Syntax 5: The syntax of the find() method

### find() Parameters

The find() method takes maximum of three parameters:

- sub - It is the substring to be searched in the str string.
- start and end (optional) - The range str[start:end] within which substring is searched.
- find() Return Value

### The find() method returns an integer value:

- If the substring exists inside the string, it returns the index of the first occurrence of the substring.
- If a substring doesn't exist inside the string, it returns -1.

### 5.9.6 rfind() function

The `rfind()` method returns the highest index of the substring (if found). If not found, it returns -1.

The syntax of `rfind()` is:

```
str.rfind(sub[, start[, end]])
```

Syntax 6: The Syntax of `rfind()` method.

#### **rfind() Parameters**

`rfind()` method takes a maximum of three parameters:

- `sub` - It's the substring to be searched in the `str` string.
- `start` and `end` (optional) - substring is searched within `str[start:end]`

#### **Return Value from rfind()**

`rfind()` method returns an integer value.

- If substring exists inside the string, it returns the highest index where substring is found.
- If substring doesn't exist inside the string, it returns -1.

<pre>quote = 'This is what, This is what, This is what' result = quote.rfind('This is what') print("Substring 'This is what':", result) result = quote.rfind('Big') print("Substring 'Big ':", result)</pre>	<p>Output:</p> <p>Substring 'This is what': 28</p> <p>Substring 'Big ': -1</p>
--	--

Example 39: Explanation of `rfind()` function.

Various string functions `capitalize()`, `title()`, `lower()`, `upper()` and `swapcase()`:-

- The `capitalize()` method converts the first character of a string to an uppercase letter and all other alphabets to lowercase.
- The `islower()` method returns `True` if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns `False`.
- The `upper()` method converts all lowercase characters in a string into uppercase characters and returns it.
- The `title()` method returns a string with first letter of each word capitalized; a title cased string.

- The `swapcase()` method returns the string by converting all the characters to their opposite letter case (uppercase to lowercase and vice versa).

<pre>my_string = "computer programming is GOOD" print(my_string.capitalize()) # Computer programming is good print(my_string.title()) # Computer Programming Is Good print(my_string.lower()) #computer programming is good print(my_string.upper()) # COMPUTER PROGRAMMING IS GOOD print(my_string.swapcase()) # COMPUTER PROGRAMMING IS good</pre>	<p>Output:</p> <pre>Computer programming is good Computer Programming Is Good computer programming is good COMPUTER PROGRAMMING IS GOOD COMPUTER PROGRAMMING IS good</pre>
--	--

Example 40: Various string functions `capitalize()`, `title()`, `lower()`, `upper()` and `swapcase()`

#### Various string functions `islower()`, `isupper()` and `istitle()`:-

- The `islower()` method returns True if all alphabets in a string are lowercase alphabets. If the string contains at least one uppercase alphabet, it returns False.
- The `upper()` method converts all lowercase characters in a string into uppercase characters and returns it.
- The `istitle()` returns True if the string is a titlecased string. If not, it returns False.

- `my_string1 = "computer programming"`
- `my_string2 = "COMPUTER"`
- `my_string3 = "Computer"`
- `print("---Check String1---")`
- `print(my_string1.islower())`
- `print(my_string1.isupper())`
- `print(my_string1.istitle())`
- `print("---Check String2---")`
- `print(my_string2.islower())`
- `print(my_string2.isupper())`
- `print(my_string2.istitle())`

- `print("---Check String3---")`
- `print(my_string3.islower())`
- `print(my_string3.isupper())`
- `print(my_string3.istitle())`
- Output:
- Computer programming is good
- ---Check String1---
- True
- False
- False
- ---Check String2---
- False
- True
- False
- ---Check String3---
- False
- False
- True

Example 41: Various string functions `islower()`, `isupper()` and `istitle()`.**Replace() and strip() function:-**

The `replace()` method replaces each matching occurrence of the old character/text in the string with the new character/text.

<pre>text = 'Cricket Football' # replace c with b replaced_text = text.replace('b', 'c') print(replaced_text)</pre>	<p>Output:</p> <p>Cricket Footcall</p>
---	--

Example 42: Explanation for `replace()` function usage.

The `strip()` method returns a copy of the string by removing both the leading and the trailing characters (based on the string argument passed).

<pre>message = '    This is Python    ' # remove leading and trailing whitespaces print('Message:', message.strip())</pre>	<p>Output:</p> <p>Message: This is Python</p>
--	---

Example 43: Explanation for `strip()` function usage.

**numeric Functions:**

The `min()` function returns the smallest item in an iterable. It can also be used to find the smallest item between two or more parameters.

The `max()` function returns the largest item in an iterable. It can also be used to find the largest item between two or more parameters.

<pre>number = [10, 20, 8, 5, 100, 600] largest_number = max(number); print("The largest number is:", largest_number) smallest_number = min(number) print("The smallest number is:", smallest_number)</pre>	<p>Output:</p> <p>The largest number is: 600</p> <p>The smallest number is: 5</p>
--	---

Example 44: Explanation of `min()`, `max()`.

The `pow()` method computes the power of a number by raising the first argument to the second argument

<pre>number = [10, 20, 8, 5, 100, 600] for i in number:     print(pow(i,2))</pre>	<p>Output:</p> <p>100</p> <p>400</p> <p>64</p> <p>25</p> <p>10000</p> <p>360000</p>
---	---

Example 45: Explanation of `pow()`.**5.9.6 Date and time functions**

Python has a module named **`datetime`** to work with dates and times. Let's create a few simple programs related to date and time before we dig deeper.

<pre>import datetime datetime_object = datetime.datetime.now() print(datetime_object)</pre>	<p>Output:</p> <p>2022-09-25 12:03:39.574930</p>
---	--

Example 46: Code to get Current Date and Time

<pre>import datetime date_object = datetime.date.today() print(date_object)</pre>	<p>Output:</p> <p>2022-09-25</p>
---	----------------------------------

Example 47: Code to get Current Date

<pre>from datetime import date # date object of today's date today = date.today() print("Current year:", today.year) print("Current month:", today.month) print("Current day:", today.day)</pre>	<b>Output:</b> Current year: 2022 Current month: 9 Current day: 25
--	---

Example 48: Code to print today's date, month and year.

### 5.9.7 recursion

Recursion is the process of defining something in terms of itself.

In Python, we know that a function can call other functions. It is even possible for the function to call itself. These types of constructs are termed as recursive functions.

The following image shows the working of a recursive function called recurse.

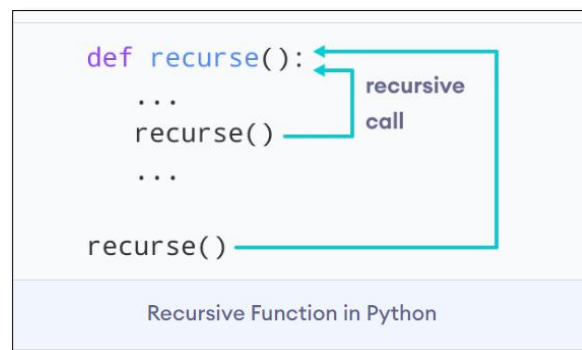


Figure 3: Recursive Function

Following is an example of a recursive function to find the factorial of an integer.

Factorial of a number is the product of all the integers from 1 to that number. For example, the factorial of 5 (denoted as 5!) is  $1*2*3*4*5 = 120$ .

<pre>def factorial(x):     """This is a recursive function     to find the factorial of an integer"""     if x == 1:         return 1     else:         return (x * factorial(x-1)) num = 5 print("The factorial of", num, "is", factorial(num))</pre>	<b>Output:</b> 120
--	-----------------------

Example 49: Code to implement recursive functions.

When we call this function with a positive integer, it will recursively call itself by decreasing the number.

Each function multiplies the number with the factorial of the number below it until it is equal to one. This recursive call can be explained in the following steps.

```
factorial(5)      # 1st call with 5
5 * factorial(4)  # 2nd call with 4
5 * 4 * factorial(3) # 3rd call with 3
5 * 4 * 3 * factorial(2) # 4th call with 2
5 * 4 * 3 * 2 * factorial(1) # 5th call with 1
```

Our recursion ends when the number reduces to 1. This is called the base condition. Every recursive function must have a base condition that stops the recursion or else the function calls itself infinitely.

Advantages of Recursion:-

- Recursive functions make the code look clean and elegant.
- A complex task can be broken down into simpler sub-problems using recursion.
- Sequence generation is easier with recursion than using some nested iteration.

Disadvantages of Recursion:-

- Sometimes the logic behind recursion is hard to follow through.
- Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
- Recursive functions are hard to debug.

### 5.9.8 Packages and modules

We don't usually store all of our files on our computer in the same location. We use a well-organized hierarchy of directories for easier access.

Similar files are kept in the same directory, for example, we may keep all the songs in the "music" directory. Analogous to this, Python has packages for directories and for files.

As our application program grows larger in size with a lot of modules, we place similar modules in one package and different modules in different packages. This makes a project (program) easy to manage and conceptually clear.

Similarly, as a directory can contain subdirectories and files, a Python package can have sub-packages and modules.

A directory must contain a file named `__init__.py` in order for Python to consider it as a package. This file can be left empty but we generally place the initialization code for that package in this file.

Here is an example. Suppose we are developing a game. One possible organization of packages and modules could be as shown in the figure below.



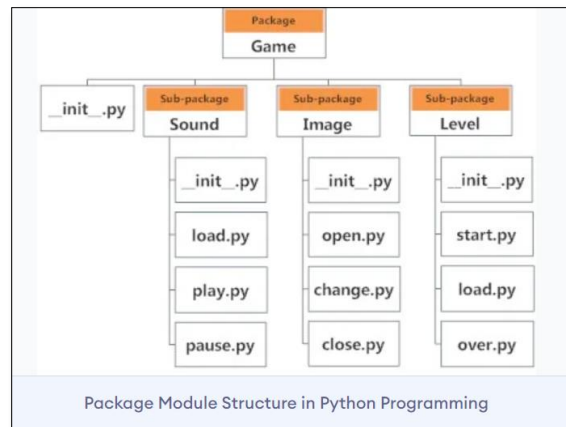


Figure 4: package Module Structure in Python Programming

### Scope of Objects and Names in Python :

Scope refers to the coding region from which a particular Python object is accessible. Hence one cannot access any particular object from anywhere from the code, the accessing has to be allowed by the scope of the object.

Let's take an example to have a detailed understanding of the same:

<pre> # Python program showing # a scope of object def some_func():     print("Inside some_func")     def some_inner_func():         var = 15         print("Inside inner function, value of var:",var)     some_inner_func()     print("Try printing var from outer function: ",var) some_func() </pre>	<p>Output:</p> <pre> Inside some_func Inside inner function, value of var: 15 Traceback (most recent call last):   File "&lt;string&gt;", line 12, in &lt;module&gt;   File "&lt;string&gt;", line 11, in some_func NameError: name 'var' is not defined </pre>
--	---

Example 50: Example to explain scope of object.

### What is namespace:

A namespace is a system that has a unique name for each and every object in Python. An object might be a variable or a method. Python itself maintains a namespace in the form of a Python dictionary. Let's go through an example, a directory-file system structure in computers. Needless to say, that one can have multiple directories having a file with the same name inside every directory. But one can get directed to the file, one wishes, just by specifying the absolute path to the file. Real-time example, the role of a namespace is like a surname. One might not find a single "Alice" in the class there might be multiple "Alice" but when you particularly ask for "Alice Lee" or "Alice Clark" (with a surname), there will be only one (time being don't think of both first name and surname are same for multiple students)

On similar lines, the Python interpreter understands what exact method or variable one is trying to point to in the code, depending upon the namespace. So, the division of the word itself gives a little more information. Its Name (which means name, a unique identifier) + Space (which talks something related to scope). Here, a name might be of any Python method or variable and space depends upon the location from where is trying to access a variable or a method.

### Types of namespaces :

When Python interpreter runs solely without any user-defined modules, methods, classes, etc. Some functions like `print()`, `id()` are always present, these are built-in namespaces. When a user creates a module, a global namespace gets created, later the creation of local functions creates the local namespace. The built-in namespace encompasses the global namespace and the global namespace encompasses the local namespace.

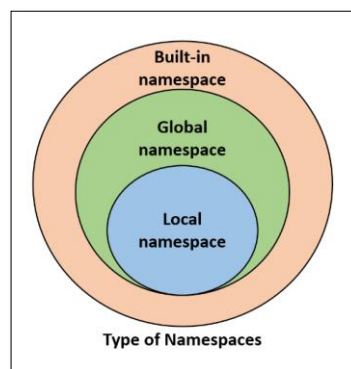


Figure 5: Type of Namespaces

<pre># Python program showing # a scope of object # Python program showing # a scope of name def some_func():     var=10 # Outer function variable     print("Inside some_func")     def some_inner_func():         var = 15         print("Inside inner function, value of var:",var)     some_inner_func()     print("var from outer function: ",var) some_func()</pre>	<p>Output:</p> <pre>Inside some_func Inside inner function, value of var: 15 var from outer function: 10</pre>
---	--

Example 51: Code to explain namespace concept.

## LEGB Rule module basics

Python namespaces can be divided into four types.

- **Local Namespace:** A function, for-loop, try-except block are some examples of a local namespace. The local namespace is deleted when the function or the code block finishes its execution.
- **Enclosed Namespace:** When a function is defined inside a function, it creates an enclosed namespace. Its lifecycle is the same as the local namespace.
- **Global Namespace:** It belongs to the python script or the current module. The global namespace for a module is created when the module definition is read. Generally, module namespaces also last until the interpreter quits.
- **Built-in Namespace:** The built-in namespace is created when the Python interpreter starts up and it's never deleted.

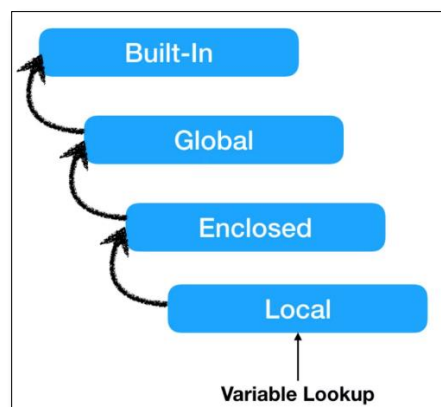


Figure 6: Four types of Python namespaces

## Importing Module

We can import the definitions inside a module to another module or the interactive interpreter in Python. We use the *import* keyword to do this. To import our previously defined module *example*, we type the following in the Python prompt.

<pre># to import standard module math import math print("The value of pi is", math.pi)</pre>	<p>Output:</p> <p>The value of pi is 3.141592653589793</p>
--	--

Example 52: Code to explain importing module.

<pre>import math as m print("The value of pi is", m.pi)</pre>	<p>Output:</p> <p>The value of pi is 3.141592653589793</p>
---	--

Example 53: Code to explain import with renaming.

## Reloading a Module

The Python interpreter imports a module only once during a session. This makes things more efficient. Here is an example to show how this works.

Suppose we have the following code in a module named `my_module`.

```
# This module shows the effect of  
# multiple imports and reload  
print("This code got executed")
```

Now we see the effect of multiple imports.

```
>>> import my_module  
This code got executed  
>>> import my_module  
>>> import my_module
```

We can see that our code got executed only once. This goes to say that our module was imported only once. Now if our module changed during the course of the program, we would have to reload it. One way to do this is to restart the interpreter. But this does not help much.

Python provides a more efficient way of doing this. We can use the `reload()` function inside the `imp` module to reload a module. We can do it in the following ways:

```
>>> import imp  
>>> import my_module  
This code got executed  
>>> import my_module  
>>> imp.reload(my_module)  
This code got executed  
<module 'my_module' from './my_module.py'>
```

**Exercise****Multiple choice questions**

- 11) In Top-down approach a problem \_\_\_\_\_
- a. Combined to form bigger modules
  - b. Divided into smaller modules
  - c. No change done in problem
  - d. None of the above
- 12) The keyword that tell Python that a new function is defined :
- a. def
  - b. next
  - c. import
  - d. None of the above
- 13) To pass some extra data to a function \_\_\_\_\_ is used:
- a. import
  - b. export
  - c. parameter
  - d. module
- 14) A variable with a specific scope is called \_\_\_\_\_.
- a. Local variable
  - b. Global variable
  - c. Argument
  - d. None of the above
- 15) Function used to take input from the console is \_\_\_\_\_.
- a. print()
  - b. isupper()
  - c. istitle()
  - d. None of the above
- 16) \_\_\_\_\_ function used to convert method converts the first character of a string to an uppercase letter and all other alphabets to lowercase.
- a. capitalize()
  - b. upper()
  - c. lower()
  - d. None of the above
- 17) Using today() function we will be able to know
- a. Current day
  - b. Current month
  - c. Current year
  - d. All of the above

**18) Which of the function helps in finding power of a number:**

- a. pow()
- b. powmin()
- c. maxpow()
- d. None of the above

**19) With the help of which keyword we are able to include modules in our program:**

- a. Bypass
- b. break
- c. import
- d. None of the above

**With the help of reloading module we need not to do \_\_\_\_\_**

- e. Shutdown interpreter
- f. Shutdown compiler
- g. Restart interpreter
- h. Restart compiler

**State whether statement is true or false**

- 1) Function helps in achieving Top-down approach. (T/F)
- 2) More than one parameter cannot be accepted by a function. (T/F)
- 3) Using return statement we are able to return value to a caller function. (T/F)
- 4) Local variable can be used in a specific block. (T/F)
- 5) VarArgs parameters are used to pass the variable number of non-keyword arguments to function. (T/F)
- 6) input() function used to display message on screen. (T/F)
- 7) print() function is used to take input from the console. (T/F)
- 8) Recursive function calls to itself until particular condition is met. (T/F)
- 9) Reloading a module is an efficient way than restarting an interpreter. (T/F)
- 10) import doesn't allow modules include into a program. (T/F)

**Fill in the blanks**

- 1) The islower() method returns \_\_\_\_\_ if all alphabets in a string are lowercase alphabets.
- 2) The rfind() method returns the \_\_\_\_\_ index of the substring (if found).
- 3) A \_\_\_\_\_ statement is used to end the execution of the function call and “returns” the result to the caller.
- 4) The \_\_\_\_\_ method returns the number of occurrences of a substring in the given string.
- 5) The \_\_\_\_\_ method returns a string with first letter of each word capitalized; a title cased string.

- 6) The \_\_\_\_\_ method returns the string by converting all the characters to their opposite letter case (uppercase to lowercase and vice versa).
- 7) A \_\_\_\_\_ is a system that has a unique name for each and every object in Python.
- 8) Recursive functions are \_\_\_\_\_ to debug.
- 9) \_\_\_\_\_ keyword used to include packages or modules to a program.
- 10) The \_\_\_\_\_ method replaces each matching occurrence of the old character/text in the string with the new character/text.

### **LAB exercise**

- 1) Write a program to find sum of first 10 numbers using function.
- 2) Write a program to find first 10 prime numbers using function.
- 3) Write a program to explain the concept of local variable.
- 4) Write a program to explain return statement.
- 5) Write a program explain input() and print() function.
- 6) Write a program to explain the working of eval() function.
- 7) Write a program to explain the working of min() and max() functions.
- 8) Write a program to explain count(), find(), replace() functions.
- 9) Write a program to explain the upper(), lower(), title(), capitalize() functions.
- 10) Write a program to explain importing of module /package

## Chapter 6

### File Processing

#### 6.1 Concept of Files

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short.

#### 6.2 File opening in various modes and closing of a file

The file can be opened in the following modes:

Mode	Description
<b>r</b>	Opens a file for reading only. (It's a default mode.)
<b>w</b>	Opens a file for writing. (If a file doesn't exist already, then it creates a new file. Otherwise, it's truncate a file.)
<b>x</b>	Opens a file for exclusive creation. (Operation fails if a file does not exist in the location.)
<b>a</b>	Opens a file for appending at the end of the file without truncating it. (Creates a new file if it does not exist in the location.)
<b>t</b>	Opens a file in text mode. (It's a default mode.)
<b>b</b>	Opens a file in binary mode.
<b>+</b>	Opens a file for updating (reading and writing.)

Table 6: Various modes for file handling

Syntax:

```
file_object = open("filename", "mode")
```

Closing a file:

In Python, it is not system critical to close all your files after using them, because the file will auto close after Python code finishes execution. You can close a file by using the close() method.

Syntax:

```
file_object.close()
```

#### 6.3 Reading from a file

For reading text data, different text-encoding schemes are used, such as ASCII (American Standard Code for Information Interchange), UTF-8 (Unicode Transformation Format), UTF-16.

Syntax:

```
file = open("file.txt", "r")
print (file.read())
```



## 6.4 Writing onto a file, File functions - open()

Similarly, for writing data to files, we have to use `open()` with 'wt' mode, clearing and overwriting the previous content. Also, we have to use the `write()` function to write into a file.

Syntax:

```
file = open("file.txt", "wt")
file.write('hi there, this is a first line of file.\n')
file.write('and another line\n')
```

Output:

```
hi there, this is a first line of file.
and another line.
```

The `open()` function returns a file object which can be used to read, write and modify the file. If the file is not found, it raises the `FileNotFoundError` exception.

Example 1: How to open a file in Python?

```
# opens test.text file of the current directory
f = open("test.txt")

# specifying the full path
f = open("C:/Python33/README.txt")
```

Since the mode is omitted, the file is opened in 'r' mode; opens for reading.

Example 2: Providing mode to open()

```
# opens the file in reading mode
f = open("path_to_file", mode='r')

# opens the file in writing mode
f = open("path_to_file", mode = 'w')

# opens for writing to the end
f = open("path_to_file", mode = 'a')
```

Python's default encoding is ASCII. You can easily change it by passing the encoding parameter.

## 6.5 close()

Python file method `close()` closes the opened file. A closed file cannot be read or written any more. You should always close your files, in some cases, due to buffering, changes made to a file may not show until you close the file. Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the `close()` method to close a file.

Syntax:

```
f.close()
```

## 6.6 read()

The `read()` method in Python is a **pre-defined** function which returns the read data in the form of a **string**.

**Syntax:**

```
f.read(n)
```

Where **f** is the object created while opening a specific file, and **'n'** is the number of bytes to be read from the file. In the case where **n** is not specified, the `read()` function reads the whole file.

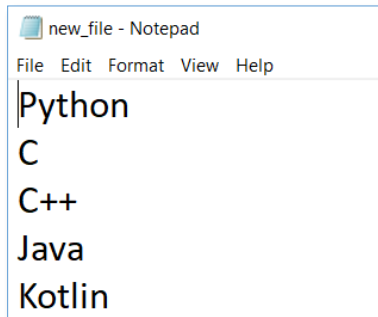


Figure 7: new\_file.txt

Consider the contents to be read belong to the above-shown file, named **new\_file.txt**. Hence using `read()` we can read the information present inside **new\_file**. Let us see how we can do that,

```
file = open("new_file.txt", "r")
print(file.read())
```

**Output:**

```
Python
C
C++
Java
Kotlin
```

Again for reading a specific number of bytes, we can use `read()` in the following way,

```
file = open("new_file.txt", "r")
print(file.read(6))
```

**Output:**

```
Python
```

## 6.7 readline()

`readline()` is yet another pre-defined method in Python, which returns a read line in the form of a **string**. Below is the syntax for `readline()` function,

```
f.readline( n )
```

Similarly, here **f** is the object created while opening the file and '**n**' is the number of bytes which the function would read almost. Noteworthy, if **n** exceeds the length of a line, the function doesn't consider the next line.

```
file = open("new_file.txt", "r")
print(file.readline())
```

**Output:**

```
Python\n
```

## 6.8 readlines()

`readlines()` reads all the **lines** present inside a specified file and returns a list containing the string forms of the read lines.

```
file.readlines()
```

Using the `readlines()` method,

```
file = open("new_file.txt", "r")
print(file.readlines())
```

**Output:**

```
['Python\n', 'C\n', 'C++\n', 'Java\n', 'Kotlin']
```

## 6.9 write()

The `write()` method writes a specified text to the file. Where the specified text will be inserted depends on the file mode and stream position.

"a": The text will be inserted at the current file stream position, default at the end of the file.

"w": The file will be emptied before the text will be inserted at the current file stream position, default 0.

**Syntax:**

```
f.write(" ")
```

**Example**

```
f = open("new_file.txt", "a")
f.write("\nSee you soon!")
f.close()

#open and read the file after the appending:
f = open("new_file.txt", "r")
print(f.read())
```

Code 7: write to a file

**Output:**

```
Python
Java
C
C++
Kotlin
See you again!
```

Output 1: write a file

**6.10 writelines()**

The writelines() method writes the items of a list to the file. Where the texts will be inserted depends on the file mode and stream position.

"a": The texts will be inserted at the current file stream position, default at the end of the file.

"w": The file will be emptied before the texts will be inserted at the current file stream position, default 0.

Syntax : **f.writelines(list)**

**Example**

```
f = open("new_file.txt", "a")
f.writelines(["\nSee you soon!", "\nOver and out."])
f.close()

#open and read the file after the appending:
f = open("new_file.txt", "r")
print(f.read())
```

Code 8: writelines

**Output:**

```
Python
    Java
    C
    C++
    Kotlin
    See you again!
Over and out.
```

Output 2: Writelines

## 6.11 tell()

The tell() function in Python, used to find the current position of the file handler or file object. Most of the time, the tell() function becomes useful to check whether the position of the file handler is at the beginning of the file or not.

**Syntax:** `file.tell()`

### Example

```
file = open("myfile.txt", "w")
text = "What's Up!"
file.write(text)
print("The current position of file handler is:", file.tell())
file.close()
```

Code 9: tell () Function

### Output:

The current position of file handler is: 10

Output 3: tell() Function Output

Here are the list of 10 characters of the text written to the file:

1. W
2. h
3. a
4. t
5. '
6. s
7. (a space)
8. U
9. p
10. !

So the current position of file handler is 10.

## 6.12 seek()

If we want to move the file pointer to another position, we can use the seek() method.

**Syntax:**

This method takes two arguments:

- **file. Seek(offset, from where)**, where offset represents how many bytes to move
- **from where**, represents the position from where the bytes are moving.

**Example 1**

```
>>> # test.txt contents:
>>> # ABCDE
>>> f = open(r'C:\test.txt')
>>> f.seek(3)
>>> f.read() # starts reading from the 3rd character
'DE'
```

**Example 2**

```
>>> f = open(r'C:\test.txt')
>>> f.seek(2) # move two characters ahead
>>> f.seek(2, 1) #move two characters ahead from the current pos
>>> f.read()
'E'
```

**Example 3**

```
>>> f = open(r'C:\test.txt')
>>> f.seek(-3, 2) # move to the 3rd character from the end of the file
>>> f.read()
```

## 6.13 Command Line arguments

The arguments that are given after the name of the program in the command line shell of the operating system are known as **Command Line Arguments**. Python provides various ways of dealing with these types of arguments. The three most common are:

**a) Using sys.argv:**

The sys module provides functions and variables used to manipulate different parts of the Python runtime environment. This module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. One such variable is sys.argv which is a simple list structure. Its main purpose is:

- It is a list of command line arguments.
- len(sys.argv) provides the number of command line arguments.
- sys.argv[0] is the name of the current Python script.

**Example:** Let's suppose there is a Python script for adding two numbers and the numbers are passed as command-line arguments.

```
import sys
# total arguments
n = len(sys.argv)
print("Total arguments passed:", n)
# Arguments passed
print("\nName of Python script:", sys.argv[0])
print("\nArguments passed:", end = " ")
for i in range(1, n):
    print(sys.argv[i], end = " ")
# Addition of numbers
Sum = 0
# Using argparse module
for i in range(1, n):
    Sum += int(sys.argv[i])
print("\n\nResult:", Sum)
```

Code 10: Using sys.argv

**Output:**

```
Name of Python script: gfg.py
Arguments passed: 2 3 5 6
Result: 16
```

Output 4: Python Output for sys.argv

**b) Using getopt module**

Python **getopt module** is similar to the function of C. Unlike sys module getopt module extends the separation of the input string by parameter validation. It allows both short, and long options including a value assignment. However, this module requires the use of the sys module to process input data properly. To use getopt module, it is required to remove the first element from the list of command-line arguments.

**Syntax:** getopt.getopt(args, options, [long\_options])

**Parameters:**

**args:** List of arguments to be passed.

**options:** String of option letters that the script want to recognize. Options that require an argument should be followed by a colon (:).

**long\_options:** List of string with the name of long options. Options that require arguments should be followed by an equal sign (=).

**Return Type:** Returns value consisting of two elements: the first is a list of (option, value) pairs. The second is the list of program arguments left after the option list was stripped.

Example:

```
# Python program to demonstrate
# command line argument
import getopt, sys
# Remove 1st argument from the
# list of command line arguments
argumentList = sys.argv[1:]
# Options
options = "hmo:"
# Long options
long_options = ["Help", "My_file", "Output="]
try:
    # Parsing argument
    arguments, values = getopt.getopt(argumentList, options, long_options)
    # checking each argument
    for currentArgument, currentValue in arguments:
        if currentArgument in ("-h", "--Help"):
            print ("Displaying Help")
        elif currentArgument in ("-m", "--My_file"):
            print ("Displaying file_name:", sys.argv[0])
        elif currentArgument in ("-o", "--Output"):
            print (("Enabling special output mode (% s)") % (currentValue))
except getopt.error as err:
    # output error, and return with an error code
    print (str(err))
```

**Output**

Code 11: Code for getopt module

```
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py -h
Displaying Help
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py --Help -m
Displaying Help
Displaying file_name: gfg.py
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py --My_file -o Blue
Displaying file_name: gfg.py
Enabling special output mode (Blue)
geeks@GFGLTTCE0026:~/Desktop$
```

Output 5: Output for getopt module



**c) Using argparse module:**

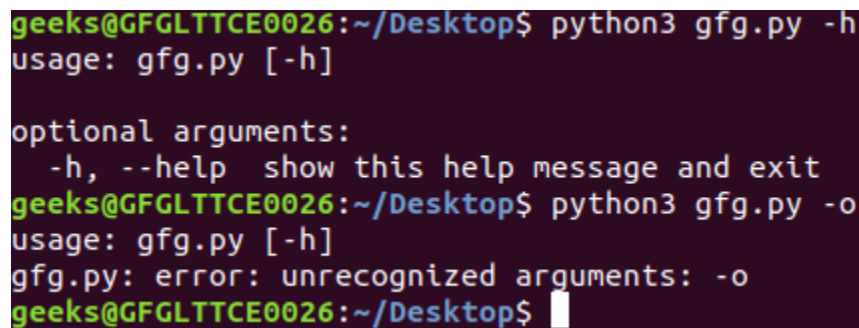
Using argparse module is a better option than the above two options as it provides a lot of options such as positional arguments, default value for arguments, help message, specifying data type of argument etc.

**Note:** As a default optional argument, it includes -h, along with its long version --help.

**Example 1:** Basic use of argparse module.

```
# Python program to demonstrate
# command line arguments
import argparse
# Initialize parser
parser = argparse.ArgumentParser()
parser.parse_args()
```

Code 12: Using argparse module.

**Output:**

```
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py -h
usage: gfg.py [-h]

optional arguments:
  -h, --help  show this help message and exit
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py -o
usage: gfg.py [-h]
gfg.py: error: unrecognized arguments: -o
geeks@GFGLTTCE0026:~/Desktop$
```

Output 6: argparse module

**Example 2:** Adding description to the help message.

```
# Python program to demonstrate
# command line arguments
import argparse
msg = "Adding description"
# Initialize parser
parser = argparse.ArgumentParser(description = msg)
parser.parse_args()
```

Code 13: Using argparse adding description to the help message.

Output:

```
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py -h
usage: gfg.py [-h]

Adding description

optional arguments:
  -h, --help  show this help message and exit
geeks@GFGLTTCE0026:~/Desktop$
```

Output 7: Adding description to the help message using argparse

Example 3: Defining optional value

```
# Python program to demonstrate
# command line arguments
import argparse
# Initialize parser
parser = argparse.ArgumentParser()
# Adding optional argument
parser.add_argument("-o", "--Output", help = "Show Output")
# Read arguments from command line
args = parser.parse_args()
if args.Output:
    print("Displaying Output as: % s" % args.Output)
```

Output:

Code 14: Defining optional value

```
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py -o Green
Displaying Output as: Green
geeks@GFGLTTCE0026:~/Desktop$ python3 gfg.py -o
usage: gfg.py [-h] [-o OUTPUT]
gfg.py: error: argument -o/--Output: expected one argument
geeks@GFGLTTCE0026:~/Desktop$
```

Output 8: Defining optional value

---

**Exercise****Multiple choice Question**

1. **To open a file in python language ..... function is used .**
  - a) Begin()
  - b) Create()
  - c) Open()
  - d) File()
2. **To read all contents from file object FILE at once we may use**
  - a) FILE.read(\*)
  - b) FILE.readlines()
  - c) FILE.read()
  - d) FILE.readline()
3. **To read 20 characters from file object FILE at once we may use**
  - a) FILE.read(20)
  - b) FILE.readlines(20)
  - c) FILE.read(char=20)
  - d) FILE.readline(char=20)
4. **readlines() will return .....**
  - a) list of characters
  - b) list of strings
  - c) list of lines
  - d) list of tuples
5. **read() will return .....**
  - a) List
  - b) String
  - c) tuple
  - d) dictionary
6. **In file handling , what does this term means "r,a"**
  - a) append append,
  - b) read read
  - c) append
  - d) error
7. **Which of the following file mode will refer to the BINARY mode?**
  - a) Binary
  - b) B
  - c) Bin
  - d) W
8. **Which of the following file mode will refer to the writing and reading both BINARY file?**
  - a) wb
  - b) wb+
  - c) w+
  - d) w

**9. Which of the following file mode is not a valid file mode?**

- a) Rw
- b) Ab
- c) w+
- d) r+
- e) ra

**10. Which of the following function is used to write LIST OF STRINGS in a file?**

- a) write()
- b) writeline()
- c) writelines()
- d) write(all)

**11. Which of the following function is used to write GROUP OF CHARACTERS in a file?**

- a) write()
- b) writeChars()
- c) writelines()
- d) write(all)

**12. If we do not specify file mode while opening a file, the file will open in .....mode  
read**

- a) Write
- b) append
- c) will give an error

**13. If we want to add more contents in an existing file, file must be opened in.....mode  
binary**

- a) append
- b) Write
- c) it is not possible

**14. What is to be added as mode to open file in text mode?**

- a) Text
- b) T
- c) t+
- d) nothing to specify , by default it will open in text mode

**15. To open a file Myfile.txt ,which is stored at d:\Myfolder, for WRITING , we can use**

- a) F=open("d:\Myfolder\Myfile.txt","w")
- b) F=open(file="d:\Myfolder\Myfile.txt","w")
- c) F=open("d:\\Myfolder\\Myfile.txt","w")
- d) F=open("d:\\Myfolder\\Myfile.txt","w")
- e) F=open(r"d:\Myfolder\Myfile.txt","w")

---

**SUBJECTIVE TYPE QUESTIONS**

- 1.What do you mean by file? What do you mean by file handling?
- 2.Does python create itself if the file doesn't exist in the memory? Illustrate your answer with an example.
- 3.List out the basic file modes available in python.
- 4.Write a python program to create and read the city.txt file in one go and print the contents on the output screen.
- 5.Explain open() function with its syntax in detail
- 6.Write one basic difference between text file and binary file.
- 7.What is the difference between write and append mode.
- 8.What is the method used to close the file in python?
- 9.What is the difference between read() and read(n) function?
- 10.Write a program to read first 5 character from a file("data.txt")
- 11.Write a program to read first line from the file("data.txt")
- 12.Name 2 functions which are used to write data into files
- 13.Write a program to read entire content from the file named("test.txt")
- 14.Which method is used to close a file in python?
15. Another name of file object is?

## Chapter 7

### Machine Learning and AI

Artificial Intelligence (AI) is when a computer algorithm does intelligent work. On the other hand, Machine Learning is a part of AI that learns from the data that also involves the information gathered from previous experiences and allows the computer program to change its behavior accordingly. **Artificial Intelligence is the superset of Machine Learning** i.e. all Machine Learning is Artificial Intelligence but not all AI is Machine Learning.

Artificial Intelligence	Machine Learning
AI manages more comprehensive issues of automating a system. This computerization should be possible by utilizing any field such as image processing, cognitive science, neural systems, machine learning, etc.	Machine Learning (ML) manages to influence users' machines to gain from the external environment. This external environment can be sensors, electronic segments, external storage gadgets, and numerous other devices.
AI manages the making of machines, frameworks, and different gadgets savvy by enabling them to think and do errands as all people generally do.	What ML does, depends on the user input or a query requested by the client, the framework checks whether it is available in the knowledge base or not. If it is available, it will restore the outcome to the user related to that query, however, if it isn't stored initially, the machine will take in the user input and will enhance its knowledge base, to give a better value to the end-user

Table 7: Artificial Intelligence vs Machine Learning

Future Scope –

- Artificial Intelligence and Machine Learning are likely to replace the current model of technology that we see these days, for example, traditional programming packages like ERP and CRM are certainly losing their charm.
- Firms like Facebook, and Google are investing a hefty amount in AI to get the desired outcome at a relatively lower computational time.
- Artificial Intelligence is something that is going to redefine the world of software and IT in the near future.

#### 7.1. Types of Machine Learning Algorithms (supervised, unsupervised)

Based on the methods and way of learning, machine learning is divided into mainly four types, which are:

1. Supervised Machine Learning
2. Unsupervised Machine Learning

3. Semi-Supervised Machine Learning
4. Reinforcement Learning

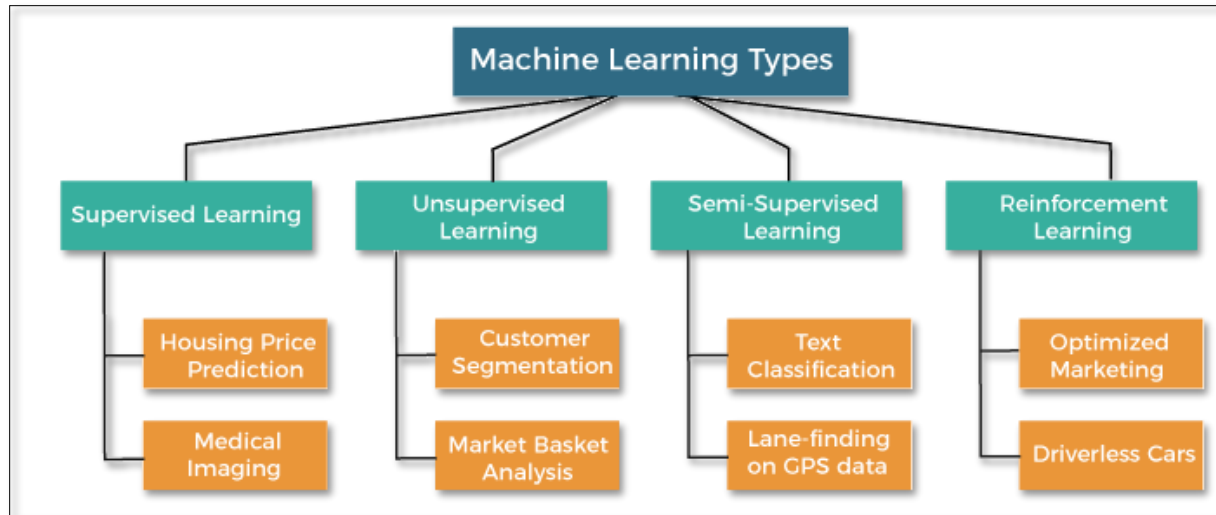


Figure 8: Types of Machine Learning Algorithms

### 7.1.1 Supervised Machine Learning

As its name suggests, is based on supervision. It means in the supervised learning technique, we train the machines using the "labelled" dataset, and based on the training, the machine predicts the output. Here, the labelled data specifies that some of the inputs are already mapped to the output. More preciously, we can say; first, we train the machine with the input and corresponding output, and then we ask the machine to predict the output using the test dataset. Let's use an illustration to clarify supervised learning. Assume we have a dataset of photos of dogs and cats as our input. Therefore, we will first train the machine to comprehend the photos, teaching it things like the size and shape of a dog's tail, the shape of a cat's eyes, their colour, and their height (dogs are taller than cats, for example). After training, we input a cat image and ask the computer to recognise the object and forecast the outcome. Now that the machine is educated, it will examine every characteristic of the thing, including height, form, colour, eyes, ears, tail, and so on, and determine that it is a cat. As a result, it will be classified as a cat. This is how it works.

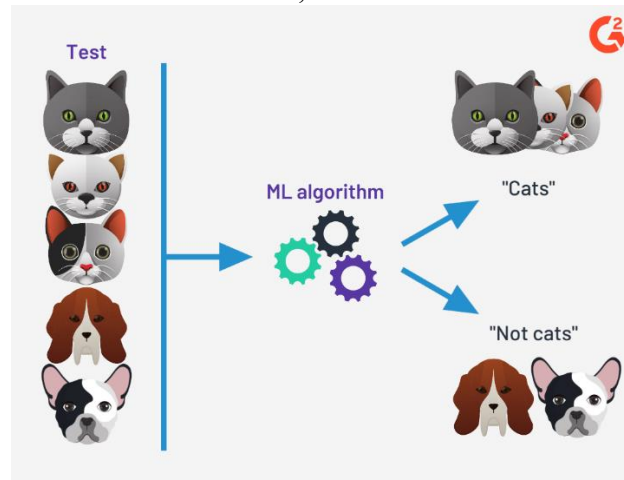


Figure 9: Classification of dog vs cat (Supervised Machine Learning)

The main goal of the supervised learning technique is to map the input variable( $x$ ) with the output variable( $y$ ). Some real-world applications of supervised learning are Risk Assessment, Fraud Detection, Spam filtering, etc.

### Categories of Supervised Machine Learning

Supervised machine learning can be classified into two types of problems, which are given below:

- Classification
- Regression

### 7.1.2. Unsupervised Machine Learning

Unsupervised learning is distinct from the supervised learning method because, as its name implies, supervision is not required. In unsupervised machine learning, this means that the system is trained on an unlabeled dataset and makes output predictions without any human supervision.

In unsupervised learning, the models are trained on data that has neither been classified nor labelled, and they are then allowed to behave autonomously on that data.

The main objective of the unsupervised learning method is to categorise or group the unsorted dataset according to similarities, differences, and patterns. The machines are to find the hidden patterns in the input dataset. To better comprehend it, let's use an example. Suppose we feed the machine learning model photographs of a basket of fruit. The model has no prior knowledge of the photos, and its job is to identify patterns and groups of items.

As a result, when the machine is tested with the test dataset, it will now learn its patterns and distinctions, such as colour differences and form differences, and anticipate the output.

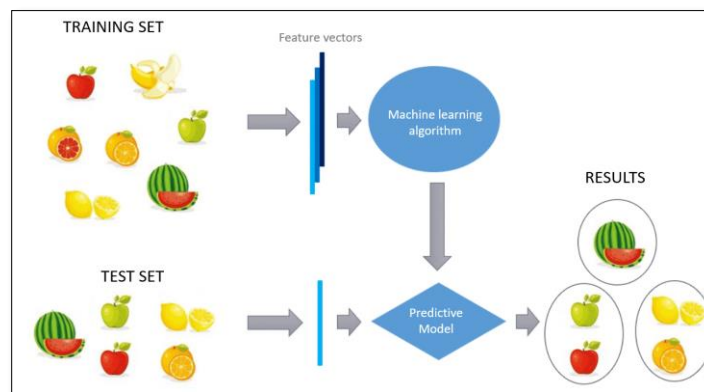


Figure 10: Unsupervised Learning clustering of different fruits

### 7.1.3 Categories of Unsupervised Machine Learning

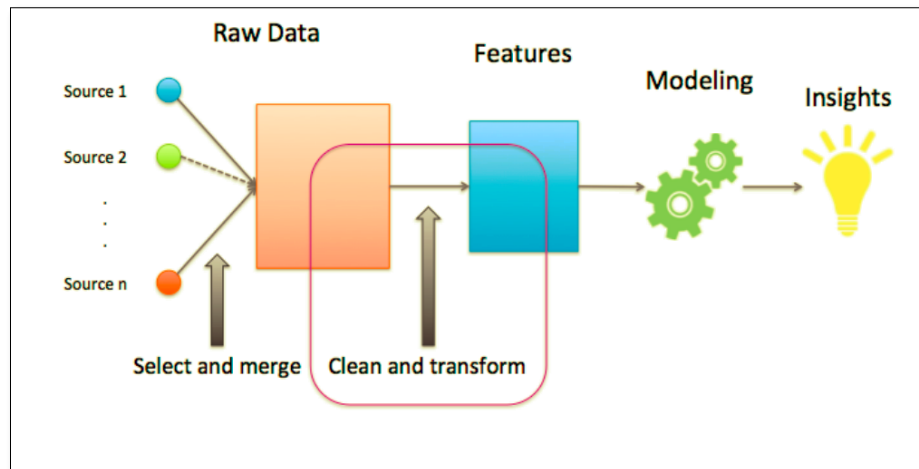
Unsupervised Learning can be further classified into two types, which are given below:

- Clustering
- Association



## 7.2. Feature engineering

The act of choosing, modifying, and transforming raw data into features that can be used in supervised learning is referred to as feature engineering. It could be necessary to develop and train better features in order to make machine learning effective on new tasks. A "feature," as you may know, is any quantifiable input that may be used in a predictive model; examples include the colour of an object's surface or the sound of a person's voice. Simply put, feature engineering is the technique of employing statistical or machine learning techniques to transform unprocessed observations into desired features. A machine learning technique called feature engineering uses data to generate new variables that aren't present in the training set. It can generate fresh features for supervised and unsupervised learning.



learning.

Figure 11: Feature Engineering diagram

## 7.3. Preparing Data

Data preparation may be one of the most difficult steps in any machine learning project. The reason is that each dataset is different and highly specific to the project. Nevertheless, there are enough commonalities across predictive modelling projects that we can define a loose sequence of steps and subtasks that you are likely to perform.

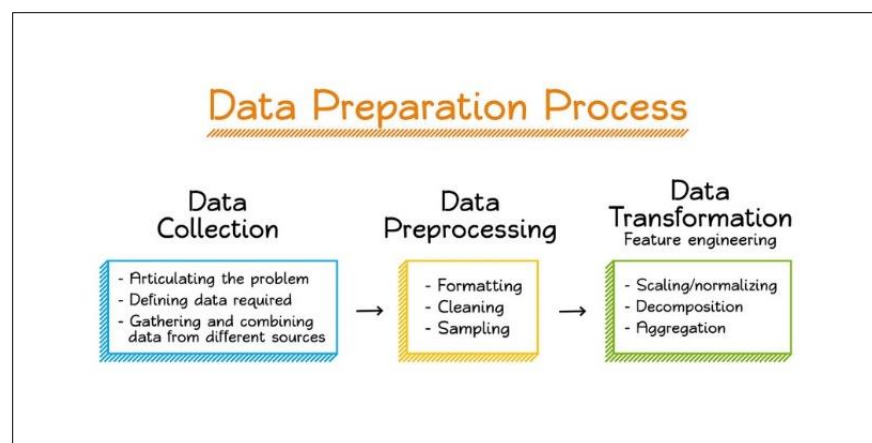


Figure 12: Data Preparation Process

### 7.3.1 Training Data, Test data

#### 7.3.1.1 What is Training Data?

Algorithms are used in machine learning to extract knowledge from datasets. They identify patterns, gain insight, make judgments, and assess those judgments.

The datasets are divided into two groups for machine learning.

The first subset, referred to as the training data, is a part of our actual data that is utilized to train a machine learning model. It trains our model in this way. The testing data refers to the other subset. Normally, training data is larger than test dataset. This is because we want to provide the model with as much information as we can in order for it to discover and learn useful patterns. When our datasets' data are supplied to a machine learning algorithm, the programme recognises patterns in the data and draws conclusions.

#### 7.3.1.2 What is Testing Data?

Once your machine learning model is built (with your training data), you need unseen data to test your model. This data is called testing data, and you can use it to evaluate the performance and progress of your algorithms' training and adjust or optimize it for improved results.

Testing data has two main criteria. It should:

- Represent the actual dataset
- Be large enough to generate meaningful predictions

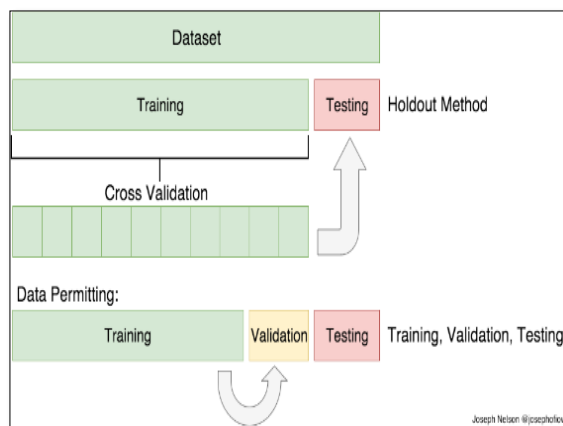


Figure 13: understanding training, test, and validation data

#### 7.3.1.2 Data Validation

Validation data provides an initial check that the model can return useful predictions in a real-world setting, which training data cannot do. The ML algorithm can assess training data and validation data at the same time.

Validation data is an entirely separate segment of data, though a data scientist might carve out part of the training dataset for validation — as long as the datasets are kept separate throughout the entirety of training and testing.

For example, let's say an ML algorithm is supposed to analyze a picture of a vertebrate and provide its scientific classification. The training dataset would include lots of pictures of mammals, but not all pictures of all mammals, let alone all pictures of all vertebrates. So, when the validation data provides a picture of a squirrel, an animal the model hasn't seen before, the data scientist can assess how well the algorithm performs in that task. This is a check against an entirely different dataset than the one it was trained on.

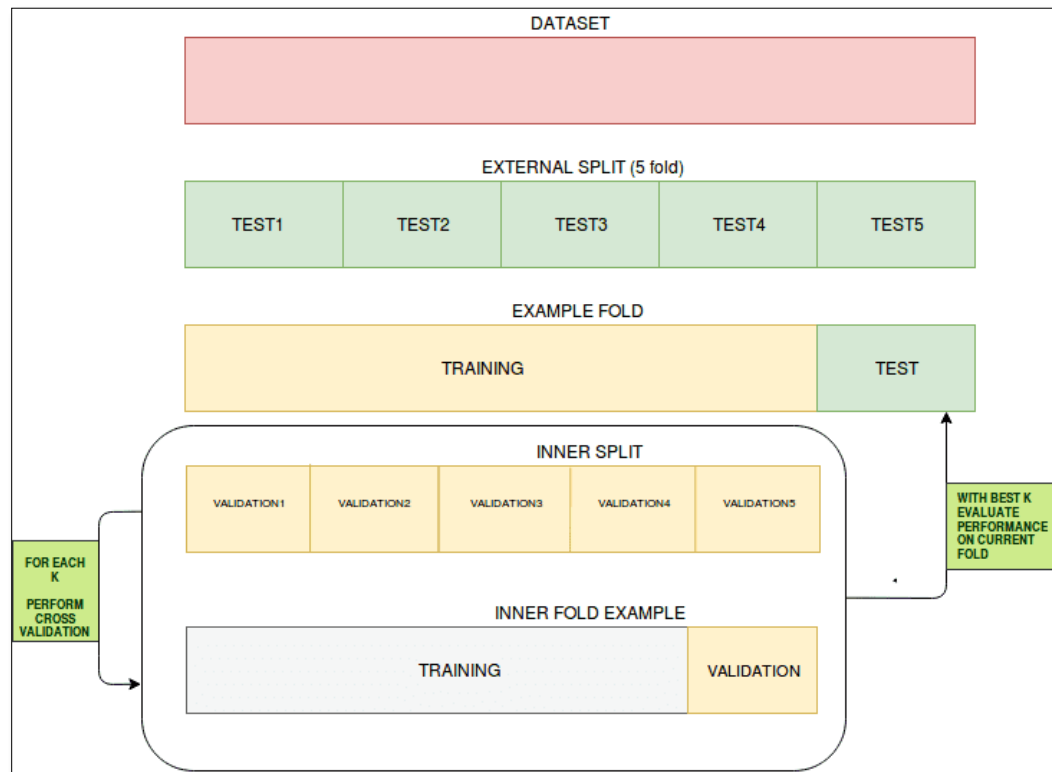


Figure 14: Understanding Training, Test and Validation data

## 7.4. Introduction to different Machine Learning Algorithms

List of commonly used Machine Learning (ML) Algorithms:

1. **Linear regression:** Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables such as sales, salary, age, product price, etc.
2. **Logistic regression :** Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.

3. Decision tree : Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
4. SVM algorithm : Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
5. Naive Bayes algorithm : Naïve Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems. It is mainly used in text classification that includes a high-dimensional training dataset.
6. KNN algorithm : K-nearest neighbours (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry.
7. K-means : K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called flat clustering algorithm. The number of clusters identified from data by algorithm is represented by 'K' in K-means. In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

## 7.5. Training the Machine learning model and predicting the results

Problem Statement:

Create the model that can classify the different species of the Iris flower.

Problem solving:

1. create the dataset.
2. Build the model
3. Train the model
4. Make predictions.

Iris Flower:

Iris is the family in the flower which contains the several species such as the *iris.setosa*, *iris.versicolor*, *iris.virginica* etc.



Figure 15: Iris Flower species

## 1) Create the datasets:

In order to classify the different species of the Iris, We should prepare the datasets with features and labels. But sklearn comes with the inbuilt datasets for the iris classification problem.

Let us first understand the datasets

The data set consists of:

- 150 samples
- 3 labels: species of Iris (Iris setosa, Iris virginica and Iris versicolor)
- 4 features: Sepal length, Sepal width, Petal length, Petal Width in cm

Scikit learn only works if data is stored as numeric data, irrespective of it being a regression or a classification problem. It also requires the arrays to be stored at numpy arrays for optimization. Since, this dataset is loaded from scikit learn, everything is appropriately formatted.

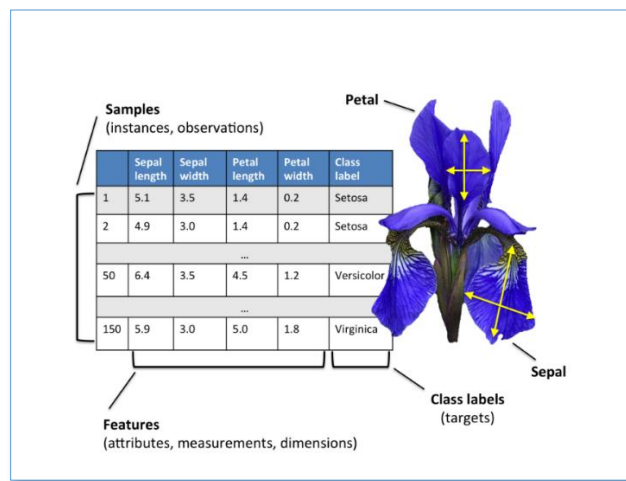


Figure 16: how the data looks like

So now let us write the python code to load the Iris dataset.

```
from sklearn import datasets
```

```
iris=datasets.load_iris()
```

Assign the data and target to separate variables.

```
x=iris.data
```

```
y=iris.target
```

x contains the features and y contains the labels

Splitting the dataset:

Since our process involve training and testing, We should split our dataset. It can be executed by the following code

```
from sklearn.model_selection import train_test_split
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.5)
```

x\_train contains the training features

x\_test contains the testing features

y\_train contains the training label

y\_test contains the testing labels

## 2) Build the mode

We can use any classification algorithm to solve the problem. we have solved the previous problem with decision tree algorithm, I will go with that.

```
from sklearn import tree
classifier=tree.DecisionTreeClassifier()
```

The above code will create the empty model. In order to provide the operations to the model we should train them.

**Note:** We can also use KNeighborsClassifier (efficiency is higher)

```
from sklearn import neighbors
classifier=neighbors.KNeighborsClassifier()
```

At this point, We have just made the model. But it cannot able to predict whether the given flower belongs to which species of Iris. If our model has to predict the flower, We have to train the model with the Features and the Labels.

## 3) Train the Model.

We can train the model with **fit** function.

```
classifier.fit(x_train,y_train)
```

Now the model is ready to make predictions

## 4) Make predictions:

Predictions can be done with **predict** function

```
predictions=classifier.predict(x_test)
```

these predictions can be matched with the expected output to measure the accuracy value.

```
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,predictions))
```

Full code:

```
from sklearn import datasets
iris=datasets.load_iris()
x=iris.data
y=iris.target
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.5)
from sklearn import tree
classifier=tree.DecisionTreeClassifier()
classifier.fit(x_train,y_train)
predictions=classifier.predict(x_test)
from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,predictions))
```

Code 15: Code for Iris flower classification using Decision tree classifier

Output:

```
D:\iris_classification>python classifier.py
0.96
```

Output 9: Result for iris flower classification

So the accuracy is 96%

## 7.6. Applications of Machine Learning. Introduction to Artificial Intelligence

### 7.6.1 Applications of Machine Learning.

Machine learning is a buzzword for today's technology, and it is growing very rapidly day-by-day. We are using machine learning in our daily life even without knowing it such as Google Maps, Google assistant, Alexa, etc. Below are some most trending real-world applications of Machine Learning:

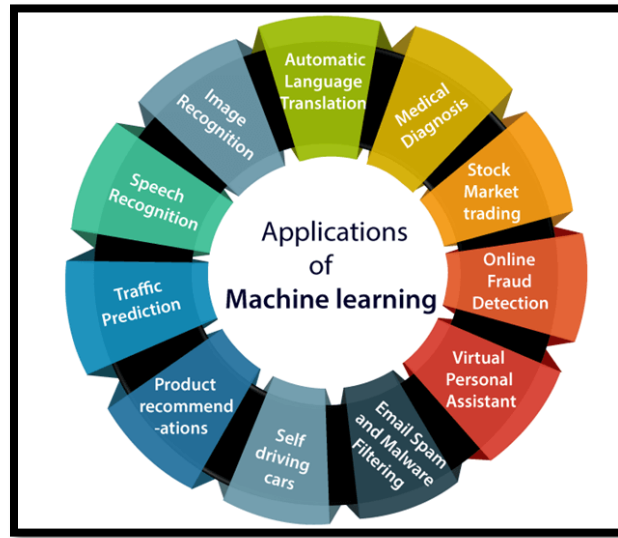


Figure 17: Applications of Machine Learning.

#### 1. Image Recognition:

Image recognition is one of the most common applications of machine learning. It is used to identify objects, persons, places, digital images, etc. The popular use case of image recognition and face detection is, Automatic friend tagging suggestion: Facebook

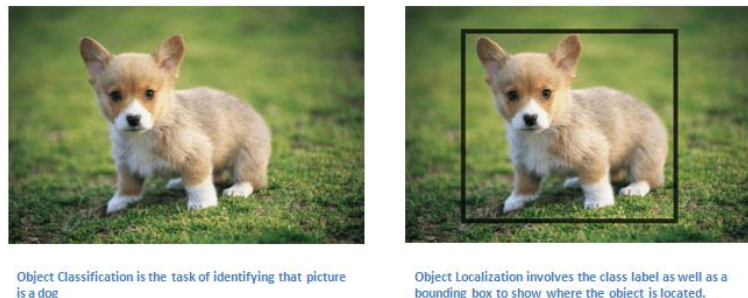


Figure 18: Image Recognition



## 2. Speech Recognition:

While using Google, we get an option of "Search by voice," it comes under speech recognition, and it's a popular application of machine learning.

Speech recognition is a process of converting voice instructions into text, and it is also known as "Speech to text", or "Computer speech recognition." At present, machine-learning algorithms are widely used by various applications of speech recognition. Google assistant, Siri, Cortana, and Alexa are using speech recognition technology to follow the voice instructions.

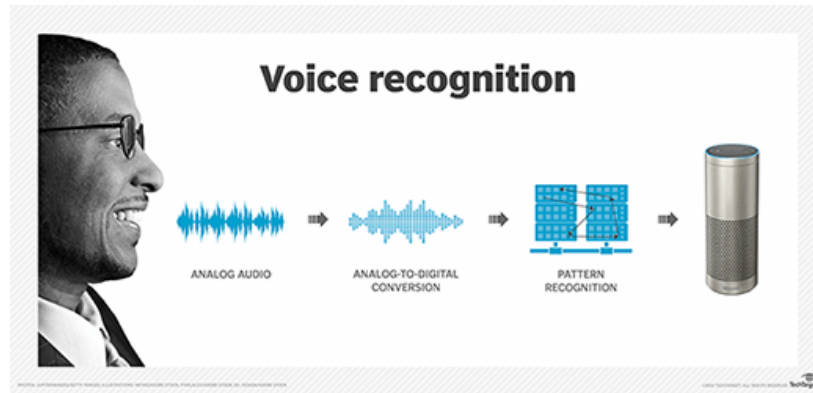


Figure 19: Speech Recognition

## 3. Traffic prediction:

If we want to visit a new place, we take help of Google Maps, which shows us the correct path with the shortest route and predicts the traffic conditions.

It predicts the traffic conditions such as whether traffic is cleared, slow moving, or heavily congested with the help of two ways:

- Real Time location of the vehicle form Google Map app and sensors
- Average time has taken on past days at the same time.

Everyone who is using Google Map is helping this app to make it better. It takes information from the user and sends back to its database to improve the performance.

## 4. Product recommendations:

Machine learning is widely used by various e-commerce and entertainment companies such as Amazon, Netflix, etc., for product recommendation to the user. Whenever we search for some product on Amazon, then we started getting an advertisement for the same product while internet surfing on the same browser and this is because of machine learning. Google understands the user interest using various machine-learning algorithms and suggests the product as per customer interest. As similar, when we use Netflix, we find some recommendations for entertainment series, movies, etc., and this is also done with the help of machine learning.

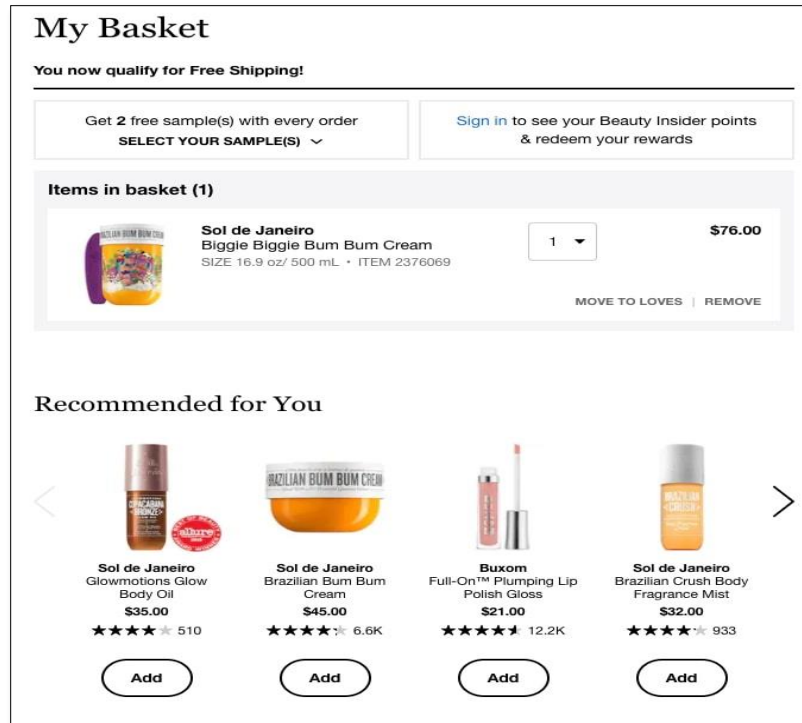


Figure 20: Product recommendations

## 5. Self-driving cars:

One of the most exciting applications of machine learning is self-driving cars. Machine learning plays a significant role in self-driving cars. Tesla, the most popular car manufacturing company is working on self-driving car. It is using unsupervised learning method to train the car models to detect people and objects while driving.



Figure 21: Self-driving cars

## 6. Email Spam and Malware Filtering:

Whenever we receive a new email, it is filtered automatically as important, normal, and spam. We always receive an important mail in our inbox with the important symbol and spam emails in our spam box, and the technology behind this is Machine learning. Below are some spam filters used by Gmail:

- Content Filter
- Header filter
- General blacklists filter
- Rules-based filters
- Permission filters

Some machine learning algorithms such as Multi-Layer Perceptron, Decision tree, and Naïve Bayes classifier are used for email spam filtering and malware detection.

### **7. Virtual Personal Assistant:**

We have various virtual personal assistants such as Google assistant, Alexa, Cortana, Siri. As the name suggests, they help us in finding the information using our voice instruction. These assistants can help us in various ways just by our voice instructions such as Play music, call someone, open an email, Scheduling an appointment, etc.

These virtual assistants use machine learning algorithms as an important part. These assistant record our voice instructions, send it over the server on a cloud, and decode it using ML algorithms and act accordingly.

### **8. Online Fraud Detection:**

Machine learning is making our online transaction safe and secure by detecting fraud transaction. Whenever we perform some online transaction, there may be various ways that a fraudulent transaction can take place such as fake accounts, fake ids, and steal money in the middle of a transaction. So to detect this, Feed Forward Neural network helps us by checking whether it is a genuine transaction or a fraud transaction.

For each genuine transaction, the output is converted into some hash values, and these values become the input for the next round. For each genuine transaction, there is a specific pattern which gets change for the fraud transaction hence, it detects it and makes our online transactions more secure.

### **9. Stock Market trading:**

Machine learning is widely used in stock market trading. In the stock market, there is always a risk of up and downs in shares, so for this machine learning's long short term memory neural network is used for the prediction of stock market trends.

### **10. Medical Diagnosis:**

In medical science, machine learning is used for diseases diagnoses. With this, medical technology is growing very fast and able to build 3D models that can predict the exact position of lesions in the brain.

It helps in finding brain tumors and other brain-related diseases easily.

## 11. Automatic Language Translation:

Nowadays, if we visit a new place and we are not aware of the language then it is not a problem at all, as for this also machine learning helps us by converting the text into our known languages. Google's GNMT (Google Neural Machine Translation) provide this feature, which is a Neural Machine Learning that translates the text into our familiar language, and it called as automatic translation.

The technology behind the automatic translation is a sequence to sequence learning algorithm, which is used with image recognition and translates the text from one language to another language.

### 7.6.2 Introduction to Artificial Intelligence:

In today's world, technology is growing very fast, and we are getting in touch with different new technologies day by day.

Here, one of the booming technologies of computer science is Artificial Intelligence which is ready to create a new revolution in the world by making intelligent machines. The Artificial Intelligence is now all around us. It is currently working with a variety of subfields, ranging from general to specific, such as self-driving cars, playing chess, proving theorems, playing music, Painting, etc. AI is one of the fascinating and universal fields of Computer science which has a great scope in future. AI holds a tendency to cause a machine to work as a human.

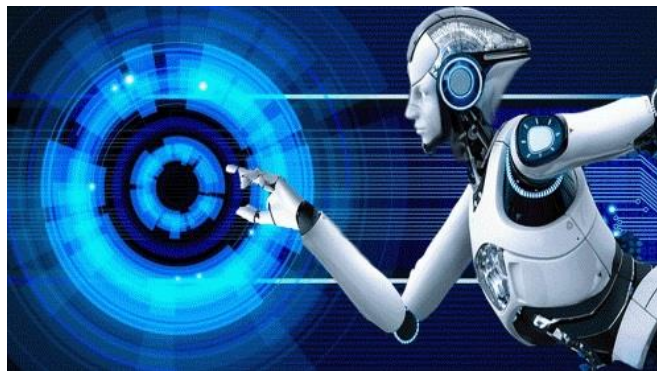


Figure 22: Introduction to Artificial Intelligence

Artificial Intelligence is composed of two words **Artificial** and **Intelligence**, where Artificial defines "*man-made*," and intelligence defines "*thinking power*", hence AI means "*a man-made thinking power*."

So, we can define AI as:

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like humans, and able to make decisions."

Artificial Intelligence exists when a machine can have human based skills such as learning, reasoning, and solving problems

With Artificial Intelligence you do not need to pre-program a machine to do some work, despite that you can create a machine with programmed algorithms which can work with own intelligence, and that is the awesomeness of AI.

It is believed that AI is not a new technology, and some people says that as per Greek myth, there were Mechanical men in early days which can work and behave like humans.

## **Why Artificial Intelligence?**

Before Learning about Artificial Intelligence, we should know that what is the importance of AI and why should we learn it. Following are some main reasons to learn about AI:

- With the help of AI, you can create such software or devices which can solve real-world problems very easily and with accuracy such as health issues, marketing, traffic issues, etc.
- With the help of AI, you can create your personal virtual Assistant, such as Cortana, Google Assistant, Siri, etc.
- With the help of AI, you can build such Robots which can work in an environment where survival of humans can be at risk.
- AI opens a path for other new technologies, new devices, and new Opportunities.

Artificial Intelligence is not just a part of computer science even it's so vast and requires lots of other factors which can contribute to it. To create the AI first, we should know that how intelligence is composed, so the Intelligence is an intangible part of our brain which is a combination of Reasoning, learning, problem-solving perception, language understanding, etc.

To achieve the above factors for a machine or software Artificial Intelligence requires the following discipline:

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science
- Neurons Study
- Statistics

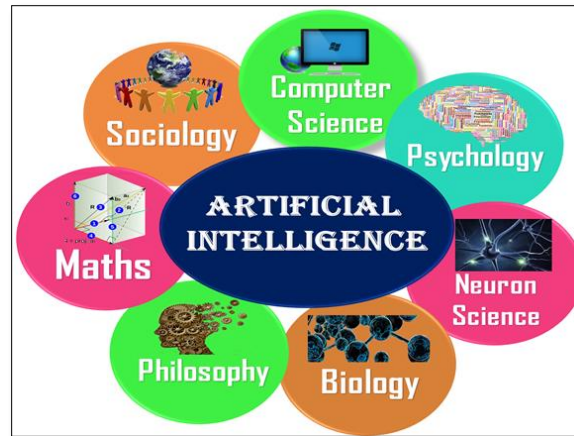


Figure 23: Factors needed to learn Artificial Intelligence

## 7.7. Common Applications of AI:

The function and popularity of Artificial Intelligence are soaring by the day. Artificial intelligence is the ability of a system or a program to think and learn from the experience. AI applications have significantly evolved over the past few years and has found its applications in almost every business sector. Top artificial intelligence applications in the real world are:

### 7.7.1. AI Application in E-Commerce:

- **Personalized Shopping**  
Artificial Intelligence technology is used to create recommendation engines through which you can engage better with your customers. These recommendations are made in accordance with their browsing history, preference, and interests. It helps in improving your relationship with your customers and their loyalty towards your brand.
- **AI-powered Assistants**  
Virtual shopping assistants and chatbots help improve the user experience while shopping online. Natural Language Processing is used to make the conversation sound as human and personal as possible. Moreover, these assistants can have real-time engagement with your customers. Did you know that on amazon.com, soon, customer service could be handled by chatbots?

### 7.7.2. Applications of Artificial Intelligence in Education:

Although the education sector is the one most influenced by humans, Artificial Intelligence has slowly begun to seep its roots in the education sector as well. Even in the education sector, this slow transition of Artificial Intelligence has helped increase productivity among faculties and helped them concentrate more on students than office or administration work.

Some of these applications in this sector include:

- **Administrative Tasks Automated to Aid Educators**  
Artificial Intelligence can help educators with non-educational tasks like task-related duties like facilitating and automating personalized messages to students, back-office tasks like grading paperwork, arranging and facilitating parent and guardian interactions, routine issue feedback facilitating, managing enrolment, courses, and HR-related topics.
- **Creating Smart Content**  
Digitization of content like video lectures, conferences, and text book guides can be made using Artificial Intelligence. We can apply different interfaces like animations and learning content through customization for students from different grades.  
Artificial Intelligence helps create a rich learning experience by generating and providing audio and video summaries and integral lesson plans.
- **Voice Assistants**  
Without even the direct involvement of the lecturer or the teacher, a student can access extra learning material or assistance through Voice Assistants. Through this, printing costs of temporary handbooks and also provide answers to very common questions easily.
- **Personalized Learning**  
Using AI technology, hyper-personalization techniques can be used to monitor students' data thoroughly, and habits, lesson plans, reminders, study guides, flash notes, frequency or revision, etc., can be easily generated.

### 7.7.3. Applications of Artificial Intelligence in Lifestyle:

Artificial Intelligence has a lot of influence on our lifestyle. Let us discuss a few of them.

- **Autonomous Vehicles**  
Automobile manufacturing companies like Toyota, Audi, Volvo, and Tesla use machine learning to train computers to think and evolve like humans when it comes to driving in any environment and object detection to avoid accidents.
- **Spam Filters**  
The email that we use in our day-to-day lives has AI that filters out spam emails sending them to spam or trash folders, letting us see the filtered content only. The popular email provider, Gmail, has managed to reach a filtration capacity of approximately 99.9%.
- **Facial Recognition**  
Our favorite devices like our phones, laptops, and PCs use facial recognition techniques by using face filters to detect and identify in order to provide secure access. Apart from personal usage, facial recognition is a widely used Artificial Intelligence application even in high security-related areas in several industries.
- **Recommendation System**  
Various platforms that we use in our daily lives like e-commerce, entertainment websites, social media, video sharing platforms, like YouTube, etc., all use the recommendation system to get user data and provide customized recommendations to users to increase engagement. This is a very widely used Artificial Intelligence application in almost all industries.



#### **7.7.4. Applications of Artificial intelligence in Navigation:**

Based on research from MIT, GPS technology can provide users with accurate, timely, and detailed information to improve safety. The technology uses a combination of Convolutional Neural Network and Graph Neural Network, which makes lives easier for users by automatically detecting the number of lanes and road types behind obstructions on the roads. AI is heavily used by Uber and many logistics companies to improve operational efficiency, analyze road traffic, and optimize routes.

#### **7.7.5. Applications of Artificial Intelligence in Robotics:**

Robotics is another field where artificial intelligence applications are commonly used. Robots powered by AI use real-time updates to sense obstacles in its path and pre-plan its journey instantly.

It can be used for -

- Carrying goods in hospitals, factories, and warehouses
- Cleaning offices and large equipment
- Inventory management

#### **7.7.6 Applications of Artificial Intelligence in Human Resource**

Did you know that companies use intelligent software to ease the hiring process?

Artificial Intelligence helps with blind hiring. Using machine learning software, you can examine applications based on specific parameters. AI drive systems can scan job candidates' profiles, and resumes to provide recruiters an understanding of the talent pool they must choose from.

#### **7.7.7. Applications of Artificial Intelligence in Healthcare**

Artificial Intelligence finds diverse applications in the healthcare sector. AI applications are used in healthcare to build sophisticated machines that can detect diseases and identify cancer cells. Artificial Intelligence can help analyze chronic conditions with lab and other medical data to ensure early diagnosis. AI uses the combination of historical data and medical intelligence for the discovery of new drugs.

#### **7.7.8. Applications of Artificial Intelligence in Agriculture**

Artificial Intelligence is used to identify defects and nutrient deficiencies in the soil. This is done using computer vision, robotics, and AI can analyze where weeds are growing. AI bots can help to harvest crops at a higher volume and faster pace than human laborers.



### 7.7.9. Applications of Artificial Intelligence in Gaming

The gaming industry is another area where AI technologies have gained popularity. AI can be utilised to develop intelligent, human-like NPCs that communicate with players. In order to improve game design and testing, it can also be used to forecast human behaviour. The 2014 Alien Isolation video games employ AI to follow the player around at all times. Two artificial intelligence systems are used in the game: the "Director AI," who frequently knows where you are, and the "Alien AI," which is controlled by sensors and behaviours and persistently pursues the player.

## 7.8. Advantages and Disadvantages of AI

### Advantages of Artificial Intelligence

Following are some main advantages of Artificial Intelligence:

- High Accuracy with less errors: AI machines or systems are prone to less errors and high accuracy as it takes decisions as per pre-experience or information.
- High-Speed: AI systems can be of very high-speed and fast-decision making, because of that AI systems can beat a chess champion in the Chess game.
- High reliability: AI machines are highly reliable and can perform the same action multiple times with high accuracy.
- Useful for risky areas: AI machines can be helpful in situations such as defusing a bomb, exploring the ocean floor, where to employ a human can be risky.
- Digital Assistant: AI can be very useful to provide digital assistant to the users such as AI technology is currently used by various E-commerce websites to show the products as per customer requirement.
- Useful as a public utility: AI can be very useful for public utilities such as a self-driving car which can make our journey safer and hassle-free, facial recognition for security purpose, Natural language processing to communicate with the human in human-language, etc.

### Disadvantages of Artificial Intelligence

Every technology has some disadvantages, and the same goes for Artificial intelligence. Being so advantageous technology still, it has some disadvantages which we need to keep in our mind while creating an AI system. Following are the disadvantages of AI:

- High Cost: The hardware and software requirement of AI is very costly as it requires lots of maintenance to meet current world requirements.
- Can't think out of the box: Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- No feelings and emotions: AI machines can be an outstanding performer, but still it does not have the feeling so it cannot make any kind of emotional attachment with human, and may sometime be harmful for users if the proper care is not taken.

- Increase dependency on machines: With the increment of technology, people are getting more dependent on devices and hence they are losing their mental capabilities.
- No Original Creativity: As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

## 7.9. Common examples of AI using python

### a) Chatbot

In the past few years, chatbots in Python have become wildly popular in the tech and business sectors. These intelligent bots are so adept at imitating natural human languages and conversing with humans, that companies across various industrial sectors are adopting them. From e-commerce firms to healthcare institutions, everyone seems to be leveraging this nifty tool to drive business benefits.

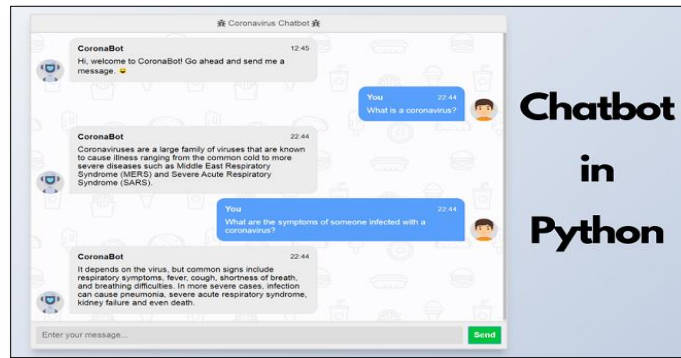


Figure 24: example of Chatbot

### b) Search and Recommendation Algorithms

When you want to watch a movie or shop online, have you noticed that the items suggested to you are often aligned with your interests or recent searches? These smart recommendation systems have learned your behavior and interests over time by following your online activity. The data is collected at the front end (from the user) and stored and analyzed through machine learning and deep learning. It is then able to predict your preferences, usually, and offer recommendations for things you might want to buy or listen to next.

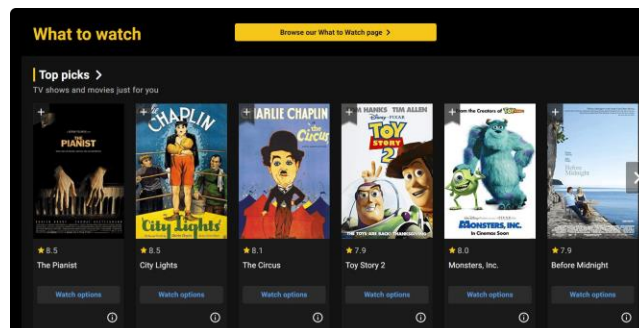


Figure 25: Example of Movies recommendation System

### c) Face Recognition Python Project

Face Recognition is a technology in computer vision. In Face recognition / detection we locate and visualize the human faces in any digital image.

It is a subdomain of Object Detection, where we try to observe the instance of semantic objects. These objects are of particular class such as animals, cars, humans, etc. Face Detection technology has importance in many fields like marketing and security.



Figure 26: Example of Face Recognition

### d) Digital voice assistants

A digital assistant, also referred to as a predictive chatbot, is a sophisticated computer programme that mimics conversations with its users, usually online.

Digital assistants learn as they go and offer a tailored, conversational experience by combining machine learning, sophisticated artificial intelligence (AI), natural language processing, and NLP. Algorithms can develop data models that discover patterns of behaviour and then enhance those patterns as data is supplied by combining previous information such as buying preferences, home ownership, geography, family size, and so forth. Digital assistants can deliver complicated answers, suggestions, forecasts, and even start conversations by learning about a user's past, preferences, and other information.

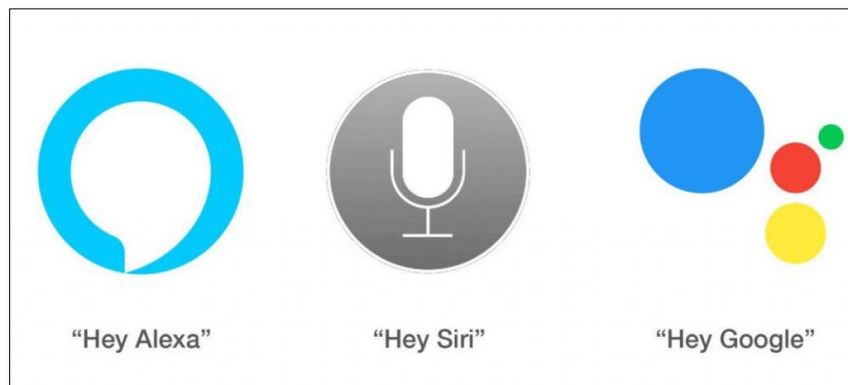


Figure 27: Example of Digital Voice Assistant

### e) Artificial Intelligence in Medical Diagnosis

Medical imaging and diagnosis powered by AI should witness more than 40% growth to surpass USD 2.5 billion by 2024.” – Global Market Insights. With the help of Neural Networks and Deep learning models, Artificial Intelligence is revolutionizing the image diagnosis field in medicine. It has taken over the complex analysis of MRI scans and made it a simpler process.



Figure 28: Artificial Intelligence in Medical Diagnosis

### f) Artificial Intelligence in Decision Making

The decision-making process has benefited greatly from artificial intelligence. AI has helped businesses by researching client demands and assessing any hazards, not just in the healthcare sector.

Using surgical robots to reduce errors and variances and ultimately aid in improving the effectiveness of surgeons is a potent application of artificial intelligence in decision-making. The Da Vinci is one such surgical robot that gives experienced surgeons more flexibility and control than they would have with traditional techniques to carry out difficult operations.

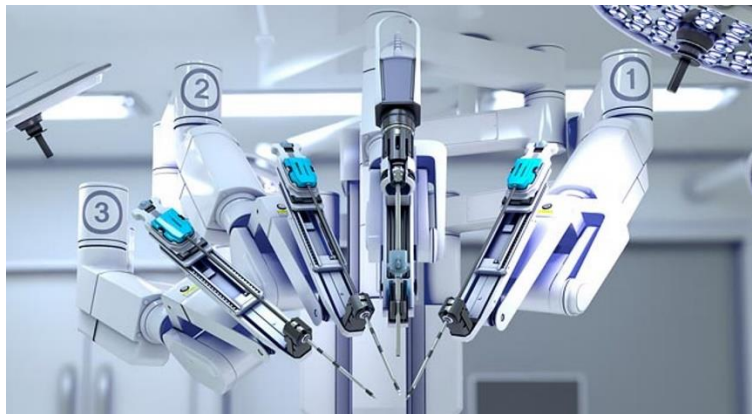


Figure 29: Artificial Intelligence in Decision Making

## **Exercise**

### **Objective Type Questions**

#### **1. What is Artificial Intelligence?**

- a) Artificial Intelligence is a field that aims to make humans more intelligent
- b) Artificial Intelligence is a field that aims to improve the security
- c) Artificial Intelligence is a field that aims to develop intelligent machines
- d) Artificial Intelligence is a field that aims to mine the data

#### **2. Which of the following is the branch of Artificial Intelligence?**

- a) Machine Learning
- b) Cyber forensics
- c) Full-Stack Developer
- d) Network Design

#### **3. In how many categories process of Artificial Intelligence is categorized?**

- a) categorized into 5 categories
- b) processes are categorized based on the input provided
- c) categorized into 3 categories
- d) process is not categorized

#### **4. \_\_\_\_\_ number of informed search method are there in Artificial Intelligence.**

- a) 4
- b) 3
- c) 2
- d) 1

#### **5. Select the most appropriate situation for that a blind search can be used.**

- a) Real-life situation
- b) Small Search Space
- c) Complex game
- d) All of the above

#### **6. A robot is able to change its own trajectory as per the external conditions, then the robot is considered as the\_\_**

- a) Mobile
- b) Non-Servo
- c) Open Loop
- d) Intelligent

**7. Which language is not commonly used for AI?**

- a) LIST
- b) PROLOG
- c) Python
- d) Perl

**8. Artificial Intelligence is about \_\_\_\_.**

- a) Playing a game on Computer
- b) Making a machine Intelligent
- c) Programming on Machine with your Own Intelligence
- d) Putting your intelligence in Machine

**9. Which of the following is an application of Artificial Intelligence?**

- a) It helps to exploit vulnerabilities to secure the firm
- b) Language understanding and problem-solving (Text analytics and NLP)
- c) Easy to create a website
- d) It helps to deploy applications on the cloud

**10. Which of the following is not an application of artificial intelligence?**

- a) Face recognition system
- b) Chatbots
- c) LIDAR
- d) DBMS

**11. Which of the following is an advantage of artificial intelligence?**

- a) Reduces the time taken to solve the problem
- b) Helps in providing security
- c) Have the ability to think hence makes the work easier
- d) All of the above

**12. What is Weak AI?**

- a) the study of mental faculties using mental models implemented on a computer
- b) the embodiment of human intellectual capabilities within a computer
- c) a set of computer programs that produce output that would be considered to reflect intelligence if it were generated by humans
- d) all of the mentioned

**13. Which of the following environment is strategic?**

- a) Rational
- b) Deterministic
- c) Partial
- d) Stochastic

**14. What is the function of the system Student?**

- a) program that can read algebra word problems only
- b) system which can solve algebra word problems but not read
- c) system which can read and solve algebra word problems
- d) None of the mentioned

**15. What is the goal of Artificial Intelligence?**

- a) To solve artificial problems
- b) To extract scientific causes
- c) To explain various sorts of intelligence
- d) To solve real-world problems

**Subjective Type Questions**

1. Which programming language is used for AI?
2. What is Deep Learning, and how is it used in real-world?
3. What are the types of Machine Learning?
4. What are the types of AI?
5. How is machine learning related to AI?
6. What are parametric and non-parametric model?
7. What is Strong AI, and how is it different from the Weak AI?
8. What is overfitting? How can it be overcome in Machine Learning?
9. What is NLP? What are the various components of NLP?
10. Give a brief introduction to the Turing test in AI?



## Chapter 8

### Data Science and Analytics Concepts

#### 8.1. What is Data Science and Analytics? The Data Science Process

Data Science is a field that gives insights from structured and unstructured data, using different scientific methods and algorithms, and consequently helps in generating insights, making predictions and devising data driver solutions. It uses a large amount of data to get meaningful insights using statistics and computation for decision making.

The data used in Data Science is usually collected from different sources, such as e-commerce sites, surveys, social media, and internet searches. All this access to data has become possible due to the advanced technologies for data collection. This data helps in making predictions and providing profits to the businesses accordingly. Data Science is the most discussed topic in today's time and is a hot career option due to the great opportunities it has to offer.



Figure 30: Data Science and Analytics



## Life Cycle of Data Science.

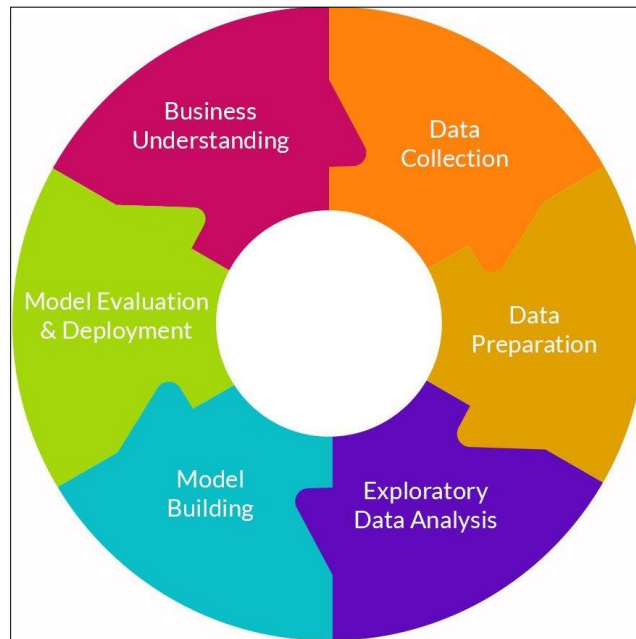



Figure 31: Life Cycle of Data Science

- Phase 1: Business Understanding
- Phase 2: Data Collection
- Phase 3: Data Preparation
- Phase 4: Exploratory Data Analysis
- Phase 5: Model Building
- Phase 6: Model Deployment and Maintenance

## Data Analyst:



### Data Analyst

**Description:** Partake in exploratory data analysis, data visualization and modeling.

**Required Background:** Candidates from various quantitative background like mathematics, stats, commerce and management.

**How To Get Started:** Analytics Edge course provides the practical knowledge of statistical tools such as R among others including Excel, SQL, etc. with visualizations tools such as Tableau. Theoretical knowledge of statistics and model development is also deeply explained.

Figure 32: Who is Data Analyst?

The role of a Data Analyst is quite similar to a Data Scientist in terms of responsibilities, and skills required. The skills shared between these two roles include SQL and data query knowledge, data preparation and cleaning, applying statistical and mathematical methods to find the insights, data visualizations, and data reporting.

The main difference between the two roles is that Data Analysts do not need to be skilled in programming languages and do not need to perform data modeling or have the knowledge of machine learning.

The tools used by both Data Scientists and Data Analysts are also different. The tools used by Data Analysts Are Tableau, Microsoft Excel, SAP, SAS, and Qlik. Data Analysts also perform the task of data mining and data modeling, but they use SAS, Rapid Miner, KNIME, and IBM SPSS Moderator. They are provided with the problem statement and the goal. They just have to perform the data analysis and deliver data reporting to the managers.

## 8.2. Framing the problem

When it comes to the problem framing process, there are four key steps to follow once the problem statement is introduced. These can help you better understand and visualize the problem as it relates to larger business needs. Using a visual aid to look at a problem can give your team a bigger picture view of the problem you're trying to solve. By contextualizing, prioritizing, and understanding the details on a deeper level, your team can develop a different point of view when reviewing the problem with stakeholders.

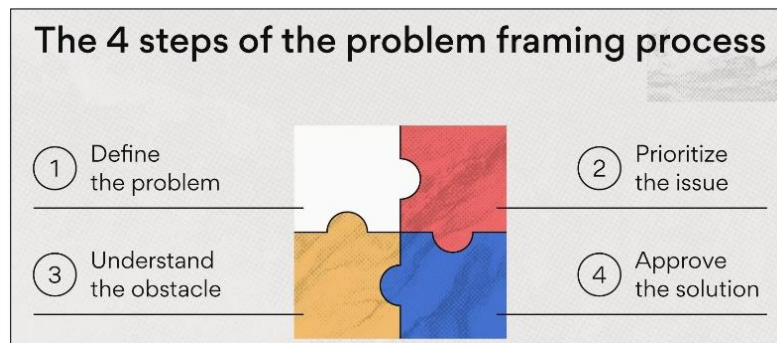


Figure 33: Problem Framing Process

### 1. Define the problem

Analyze your problem in context with the system or process it presents itself in. Ask questions such as, "Where does this problem live within the system?" and, "What is the root cause of the problem?"

## 2. Prioritize the problem

Next, prioritize the pain points based on other issues and Questions such as, “Does this problem prevent objectives from being met?” and, “Will this problem deplete necessary resources?” are good ones to get you started.

## 3. Understand the problem

To understand the problem, collect information from diverse stakeholders and department leaders. This will ensure you have a wide range of data.

## 4. Approve the solution

Finally, it's time to get your solution approved. Quality assure your solution by testing in one or more internal scenarios. This way you can be sure it works before introducing it to external customers.

### 8.3. Collecting

Before an analyst begins collecting data, they must answer three questions first:

- What’s the goal or purpose of this research?
- What kinds of data are they planning on gathering?
- What methods and procedures will be used to collect, store, and process the information?

Additionally, we can break up data into qualitative and quantitative types. Qualitative data covers descriptions such as color, size, quality, and appearance. Quantitative data, unsurprisingly, deals with numbers, such as statistics, poll numbers, percentages, etc. Data collection could mean a telephone survey, a mail-in comment card, or even some guy with a clipboard asking passersby some questions. Data collection breaks down into two methods.

The two methods are:

- **Primary.:** As the name implies, this is original, first-hand data collected by the data researchers. This process is the initial information gathering step, performed before anyone carries out any further or related research. Primary data results are highly accurate provided the researcher collects the information. However, there’s a downside, as first-hand research is potentially time-consuming and expensive.
- **Secondary.:** Secondary data is second-hand data collected by other parties and already having undergone statistical analysis. This data is either information that the researcher has tasked other people to collect or information the researcher has looked up. Simply put, it’s second-hand information. Although it’s easier and cheaper to obtain than primary information, secondary information raises concerns regarding accuracy and authenticity. Quantitative data makes up a majority of secondary data.

## 8.4. Processing

Data processing occurs when data is collected and translated into usable information. Usually performed by a data scientist or team of data scientists, it is important for data processing to be done correctly as not to negatively affect the end product, or data output. Data processing starts with data in its raw form and converts it into a more readable format (graphs, documents, etc.), giving it the form and context necessary to be interpreted by computers and utilized by employees throughout an organization.

### 8.4.1 Six stages of data processing

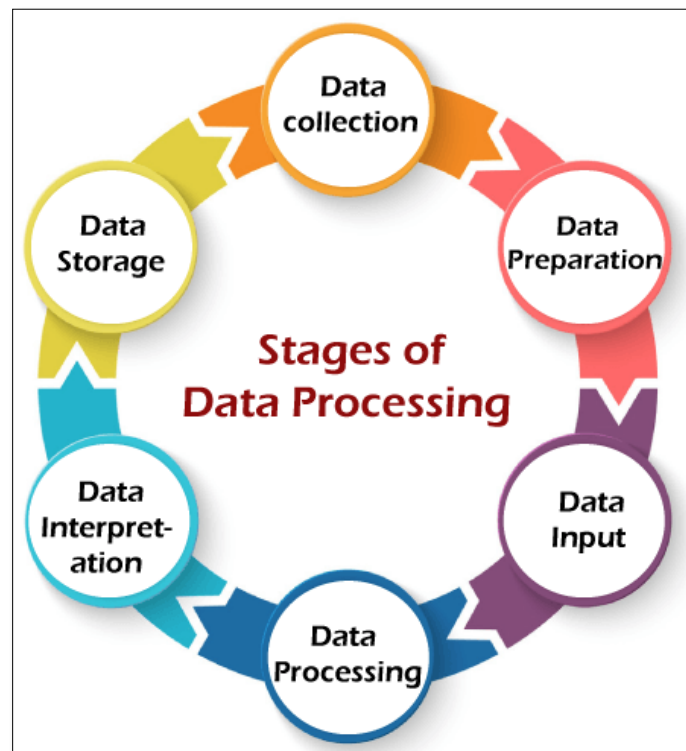


Figure 34: Stages of data processing

#### 1. Data collection

Collecting data is the first step in data processing. Data is pulled from available sources, including . It is important that the data sources available are trustworthy and well-built so the data collected (and later used as information) is of the highest possible quality.

#### 2. Data preparation

Once the data is collected, it then enters the stage. Data preparation, often referred to as “pre-processing” is the stage at which raw data is cleaned up and organized for the following stage of data processing. During preparation, raw data is diligently checked for any errors. The purpose of this step is to eliminate bad data (incomplete, or incorrect data) and begin to create high-quality data for the best .

### 3. Data input

The clean data is then entered into its destination (perhaps a CRM like Salesforce or a data warehouse like and translated into a language that it can understand. Data input is the first stage in which raw data begins to take the form of usable information.

### 4. Processing

During this stage, the data inputted to the computer in the previous stage is actually processed for interpretation. Processing is done using algorithms, though the process itself may vary slightly depending on the source of data being processed (data lakes, social networks, connected devices etc.) and its intended use (examining advertising patterns, medical diagnosis from connected devices, determining customer needs, etc.).

### 5. Data output/interpretation

The output/interpretation stage is the stage at which data is finally usable to non-data scientists. It is translated, readable, and often in the form of graphs, videos, images, plain text, etc.). Members of the company or institution can now begin for their own data analytics projects.

### 6. Data storage

The final stage of data processing is After all of the data is processed, it is then stored for future use. While some information may be put to use immediately, much of it will serve a purpose later on. Plus, properly stored data is a necessity for compliance with data protection legislation like When data is properly stored, it can be quickly and easily accessed by members of the organization when needed.

## 8.5. Cleaning and Munging Data



Figure 35: Cleaning and Munging Data

When working with data, your analysis and insights are only as good as the data you use. If you're performing data analysis with dirty data, your organization can't make efficient and effective decisions with that data. Data cleaning is a critical part of data management that allows you to validate that you have a high quality of data.

Data cleaning includes more than just fixing spelling or syntax errors. It's a fundamental aspect of data science analytics and an important machine learning technique. Today, we'll learn more about data cleaning, its benefits, issues that can arise with your data

Data cleaning, or data cleansing, is the important **process of correcting or removing incorrect, incomplete, or duplicate data within a dataset**. Data cleaning **should be the first step in your workflow**. When working with large datasets and combining various data sources, there's a strong possibility you may duplicate or mislabel data. If you have inaccurate or incorrect data, it will lose its quality, and your algorithms and outcomes become unreliable.

Data cleaning **differs from data transformation because you're actually removing data that doesn't belong in your dataset**. With data transformation, you're changing your data to a different format or structure. Data transformation processes are sometimes referred to as *data wrangling* or *data munging*. The data cleaning process is what we'll focus on today.

To determine data quality, you can study its features and weigh them according to what's important to your organization and your project.

There are five main features to look for when evaluating your data:

- **Consistency:** Is your data consistent across your datasets?
- **Accuracy:** Is your data close to the true values?
- **Completeness:** Does your data include all required information?
- **Validity:** Does your data correspond with business rules and/or restrictions?
- **Uniformity:** Is your data specified using consistent units of measurement?

Now that we know how to recognize high-quality data, let's dive deeper into the process of data science cleaning, why it's important, and how to do it effectively.

## 8.6. Exploratory Data Analysis

Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns, to spot anomalies, to test hypothesis and to check assumptions with the help of summary statistics and graphical representations. It is a good practice to understand the data first and try to gather as many insights from it. EDA is all about making sense of data in hand, before getting them dirty with it.

To understanding the concept and techniques we'll take an example of white variant of which is available on UCI Machine Learning Repository and try to catch hold of as many insights from the data set using EDA.

To starts with, import necessary libraries (for this example pandas, numpy, matplotlib and seaborn) and loaded the data set.

Note : Whatever inferences are extracted, is mentioned with bullet points.

```
In [2]: df = pd.read_csv('winequality-white.csv', sep=';')
df.head()
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

Figure 36: Winequality-white.csv dataset

- Original data is separated by delimiter “;” in given data set.
- To take a closer look at the data took help of “. head ()” function of pandas library which returns first five observations of the data set. Similarly, “. tail ()” returns last five observations of the data set.

find out the total number of rows and columns in the data set using “.shape”.

```
In [3]: df.shape
Out[3]: (4898, 12)
```

Figure 37: Total number of rows and columns in the data

- Dataset comprises of 4898 observations and 12 characteristics.
- Out of which one is dependent variable and rest 11 are independent variables — physico-chemical characteristics.

It is also a good practice to know the columns and their corresponding data types, along with finding whether they contain null values or not.

```
In [5]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4898 entries, 0 to 4897
Data columns (total 12 columns):
fixed acidity      4898 non-null float64
volatile acidity   4898 non-null float64
citric acid        4898 non-null float64
residual sugar     4898 non-null float64
chlorides          4898 non-null float64
free sulfur dioxide 4898 non-null float64
total sulfur dioxide 4898 non-null float64
density            4898 non-null float64
pH                4898 non-null float64
sulphates          4898 non-null float64
alcohol            4898 non-null float64
quality            4898 non-null int64
dtypes: float64(11), int64(1)
memory usage: 459.3 KB
```

Figure 38: finding whether contains null values or not.

- Data has only float and integer values.
- No variable column has null/missing values.



The describe() function in pandas is very handy in getting various summary statistics. This function returns the count, mean, standard deviation, minimum and maximum values and the quantiles of the data.

```
In [6]: df.describe()
```

Out[6]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
count	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000	4898.000000
mean	6.854788	0.278241	0.334192	6.391415	0.045772	35.308085	138.360657	0.994027	3.188267	0.489847	10.514267
std	0.843868	0.100795	0.121020	5.072058	0.021848	17.007137	42.498065	0.002991	0.151001	0.114126	1.230621
min	3.800000	0.080000	0.000000	0.600000	0.009000	2.000000	9.000000	0.987110	2.720000	0.220000	8.000000
25%	6.300000	0.210000	0.270000	1.700000	0.036000	23.000000	108.000000	0.991723	3.090000	0.410000	9.500000
50%	6.800000	0.260000	0.320000	5.200000	0.043000	34.000000	134.000000	0.993740	3.180000	0.470000	10.400000
75%	7.300000	0.320000	0.390000	9.900000	0.050000	46.000000	167.000000	0.996100	3.280000	0.550000	11.400000
max	14.200000	1.100000	1.660000	65.800000	0.345000	289.000000	440.000000	1.038960	3.820000	1.080000	14.200000

Figure 39: Describing the dataset

## 8.7. Visualizing results

Data Visualization techniques involve the generation of graphical or pictorial representation of DATA, from which leads you to understand the insight of a given data set. This visualisation technique aims to identify the Patterns, Trends, Correlations, and Outliers of data sets.

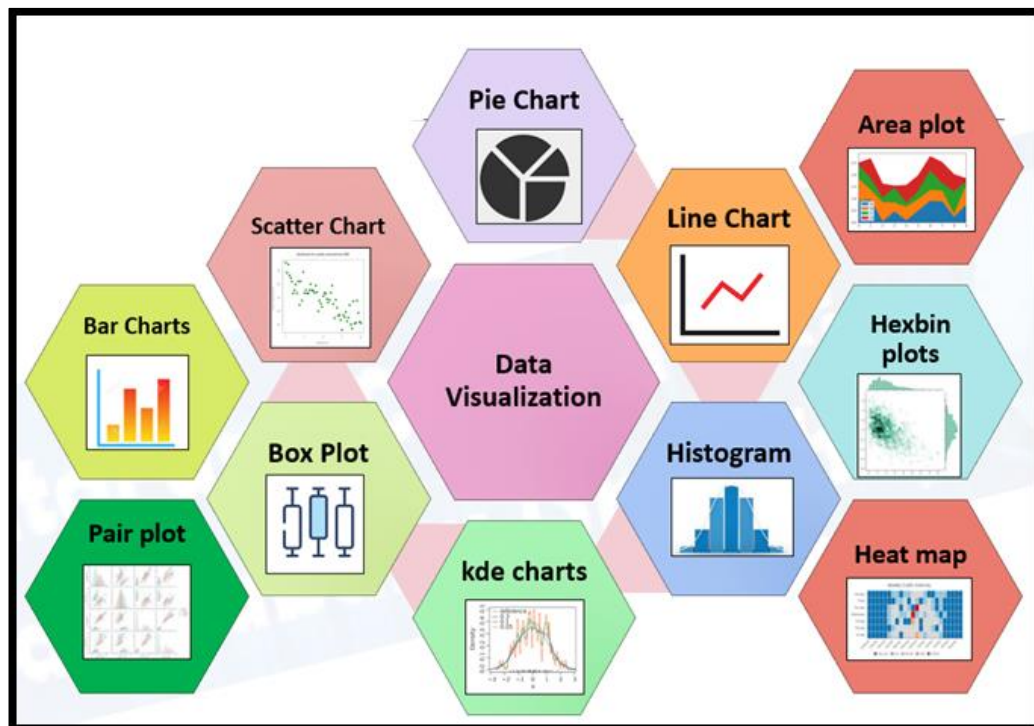


Figure 40: Data Visualization

Data visualization techniques most important part of Data Science, there won't be any doubt about it. And even in the Data Analytics space as well the Data visualization doing a major role. We will discuss this in detail with help of Python packages and how it helps during the Data Science process flow. This is a very interesting topic for every Data Scientist and Data Analyst.



## Exercise

### OBJECTIVE TYPE QUESTIONS

**1. Which of the following input can be accepted by DataFrame?**

- a) Structured ndarray
- b) Series
- c) DataFrame
- d) All of the mentioned

**2. Which of the following input can be accepted by DataFrame?**

- a) Structured ndarray
- b) Series
- c) DataFrame
- d) All of the mentioned

**3. If data is an ndarray, index must be the same length as data.**

- a) True
- b) False

**4. Point out the wrong statement.**

- a) A DataFrame is like a fixed-size dict in that you can get and set values by index label
- b) Series can be passed into most NumPy methods expecting an ndarray
- c) A key difference between Series and ndarray is that operations between Series automatically align the data based on label
- d) None of the mentioned

**5. The result of an operation between unaligned Series will have the \_\_\_\_\_ of the indexes involved.**

- a) intersection
- b) union
- c) total
- d) all of the mentioned

**6. Pandas is an open-source \_\_\_\_\_ Library?**

- a). Ruby
- b). Javascript
- c). Java
- d). Python

**7. Pandas key data structure is called?**

- a). Keyframe
- b). DataFrame
- c). Statistics
- d). Econometrics

**8. In pandas, Index values must be?**

- a) unique
- b). hashable
- c). Both A and B
- d) None of the above

**9. What will be syntax for pandas dataframe?**

- a). pandas.DataFrame( data, index, dtype, copy)
- b). pandas.DataFrame( data, index, rows, dtype, copy)
- c). pandas\_DataFrame( data, index, columns, dtype, copy)
- d) pandas.DataFrame( data, index, columns, dtype, copy)

**10. Which of the following thing can be data in Pandas?**

- a). a python dict
- b). an ndarray
- c). a scalar value
- d). All of the above

**11. Which of the following makes use of pandas and returns data in a series or dataframe?**

- a)PandaSDMX
- b). freedapi
- c). OutPy
- d). Inpy

**12. What will be output for the following code?**

```
import pandas as pd
import numpy as np
s = pd.Series(np.random.randn(2))
print s.size
```

- a).0
- b). 1
- c). 2
- d). 3

**13. What will be output for the following code?**

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print s['a']
```

- a). 1
- b). 2
- c). 3
- d). 4

**14. Which of the following takes a dict of dicts or a dict of array-like sequences and returns a DataFrame?**

- a) DataFrame.from\_items
- b) DataFrame.from\_records
- c) DataFrame.from\_dict
- d) All of the mentioned

**15. Which of the following operation works with the same syntax as the analogous dict operations?**

- a) Getting columns
- b) Setting columns
- c) Deleting columns
- d) All of the mentioned

#### ANSWERS

<u>1. d</u>	<u>6. d</u>	<u>11. b</u>
<u>2. d</u>	<u>7. b</u>	<u>12. c</u>
<u>3. a</u>	<u>8. c</u>	<u>13. a</u>
<u>4. a</u>	<u>9. d</u>	<u>14. a</u>
<u>5. b</u>	<u>10. d</u>	<u>15. d</u>

#### SUBJECTIVE TYPE QUESTIONS

1. Mention the different types of Data Structures in Pandas?
2. Define the different ways a DataFrame can be created in pandas?
3. Explain Categorical data in Pandas?
4. How can we create a copy of the series in Pandas?
5. How will you add a column to a pandas DataFrame?
6. How to Delete Indices, Rows or Columns From a Pandas Data Frame?
7. How to get the items of series A not present in series B?
8. How to get frequency counts of unique items of a series?
9. Mention The Different Types Of Data Structures In pandas?
10. What Are The Key Features Of pandas Library?
11. Explain Categorical Data In pandas
12. Define Python pandas
13. What Is A pandas DataFrame? How Will You Create An Empty DataFrame In pandas?
14. What Are The Key Features Of pandas Library?
15. What is Python pandas used for?

## Chapter 9

### Introduction to NumPy (7 Hrs.)

#### 9.1. Array Processing Package

NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is the fundamental package for scientific computing with Python. It is open-source software. It contains various features including these important ones:

1. A powerful N-dimensional array object
2. Sophisticated (broadcasting) functions
3. Tools for integrating C/C++ and Fortran code
4. Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

#### Installation:

pip install Numpy

#### Example:

```
# Python program to demonstrate
# basic array characteristics
import numpy as np

# Creating array object
arr = np.array( [[ 1, 2, 3],
                 [ 4, 2, 5]] )

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)

# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)
```

Code 16: Numpy array example

Output:

```
Array is of type: <class 'numpy.ndarray'>  
No. of dimensions: 2  
Shape of array: (2, 3)  
Size of array: 6  
Array stores elements of type: int64
```

Output 10: Numpy Array

## 9.2. Array types

### 9.3. Array slicing

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: `[start:end]`.
- We can also define the step, like this: `[start:end:step]`.
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension.
- If we don't pass step its considered 1

Example 1

Slice elements from index 1 to index 5 from the following array:

```
import numpy as np  
  
arr = np.array([1, 2, 3, 4, 5, 6, 7])  
  
print(arr[1:5])
```

Code 17 : Numpy array Slicing

Output:

```
[ 2  3  4  5]
```

Output 11: Slicing output

**Example 2**

Slice elements from index 4 to the end of the array:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[4:])
```

Code 18 : Numpy array Slicing

**Output:**

```
[5 6 7]
```

Output 12: Slicing output

**Example 3**

Slice elements from the beginning to index 4 (not included):

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[:4])
```

Code 19: Numpy array Slicing

**Output:**

```
[1 2 3 4]
```

Output 13: Slicing output

**9.3.1 Negative Slicing**

Use the minus operator to refer to an index from the end:

**Example 1**

Slice from the index 3 from the end to index 1 from the end:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print(arr[-3:-1])
```

Code 20: Negative Slicing

**Output:**

```
[5 6]
```

Output 14: Negative Slicing

### 9.3.3 Slicing 2-D Array

**Example 1**

From the second element, slice elements from index 1 to index 4 (not included):

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[1, 1:4])
```

Code 21: Slicing 2D-Array

**Output:**

```
[7 8 9]
```

Output 15: 2D-array Slicing Output

**Example 2**

From both elements, return index 2:

```
import numpy as np

arr = np.array([[1, 2, 3, 4, 5], [6, 7, 8, 9, 10]])

print(arr[0:2, 2])
```

Code 22: Slicing 2D-Array

**Output:**

```
[3 8]
```

Output 16: 2D-array Slicing Output

**9.4. Computation on NumPy Arrays – Universal functions**

Up until now, we have been discussing some of the basic nuts and bolts of NumPy; in the next few sections, we will dive into the reasons that NumPy is so important in the Python data science world. Namely, it provides an easy and flexible interface to optimized computation with arrays of data.

Computation on NumPy arrays can be very fast, or it can be very slow. The key to making it fast is to use *vectorized* operations, generally implemented through NumPy's *universal functions* (ufuncs).

Universal functions in Numpy are simple mathematical functions. It is just a term that we gave to mathematical functions in the Numpy library. Numpy provides various universal functions that cover a wide variety of operations. These functions include standard trigonometric functions, functions for arithmetic operations, handling complex numbers, statistical functions, etc. Universal functions have various characteristics which are as follows-

- These functions operate on ndarray (N-dimensional array) i.e. Numpy's array class.
- It performs fast element-wise array operations.
- It supports various features like array broadcasting, type casting etc.
- Numpy, universal functions are objects those belongs to **numpy.ufunc** class.
- Python functions can also be created as a universal function using **frompyfunc** library function.
- Some ufuncs are called automatically when the corresponding arithmetic operator is used on arrays. For example when addition of two array is performed element-wise using '+' operator then **np.add()** is called internally.



Some of the basic universal functions in NumPy are-

### 9.4.1 Array arithmetic

NumPy's ufuncs feel very natural to use because they make use of Python's native arithmetic operators. The standard addition, subtraction, multiplication, and division can all be used:

```
import numpy as np
x = np.arange(4)
print("x      =", x)
print("x + 5 =", x + 5)
print("x - 5 =", x - 5)
print("x * 2 =", x * 2)
print("x / 2 =", x / 2)
print("x // 2 =", x // 2) # floor division
```

Code 23: Array Arithmetic

**Output:**

```
x      = [0 1 2 3]
x + 5 = [5 6 7 8]
x - 5 = [-5 -4 -3 -2]
x * 2 = [0 2 4 6]
x / 2 = [0.  0.5 1.  1.5]
x // 2 = [0 0 1 1]
```

Output 17: Array Arithmetic

The following table lists the arithmetic operators implemented in NumPy:

Operator	Equivalent ufunc	Description
+	np.add	Addition (e.g., $1 + 1 = 2$ )
-	np.subtract	Subtraction (e.g., $3 - 2 = 1$ )
-	np.negative	Unary negation (e.g., $-2$ )
*	np.multiply	Multiplication (e.g., $2 * 3 = 6$ )
/	np.divide	Division (e.g., $3 / 2 = 1.5$ )
//	np.floor_divide	Floor division (e.g., $3 // 2 = 1$ )
**	np.power	Exponentiation (e.g., $2 ** 3 = 8$ )
%	np.mod	Modulus/remainder (e.g., $9 \% 4 = 1$ )

Table 8: List of Arithmetic operators implemented

## 9.5. Aggregations: Min, Max, etc.

In the Python numpy module, we have many aggregate functions or statistical functions to work with a single-dimensional or multi-dimensional array. The Python numpy aggregate functions are sum, min, max, mean, average, product, median, standard deviation, variance, argmin, argmax and percentile.

To demonstrate these Python numpy aggregate functions, we use the below-shown arrays.

```
import numpy as np
arr1 = np.array([10, 20, 30, 40, 50])
arr1

arr2 = np.array([[0, 10, 20], [30, 40, 50], [60, 70, 80]])
arr2

arr3 = np.array([[14, 6, 9, -12, 19, 72], [-9, 8, 22, 0, 99, -11]])
arr3
```

Python Numpy Aggregate Functions Examples:

### 9.5.1. Python numpy sum:

```
import numpy as np
arr1 = np.array([10, 20, 30, 40, 50])
arr1

arr2 = np.array([[0, 10, 20], [30, 40, 50], [60, 70, 80]])
arr2

arr3 = np.array([[14, 6, 9, -12, 19, 72], [-9, 8, 22, 0, 99, -11]])
arr3

print("Sum of arr1:", arr1.sum())
print("Sum of arr2:", arr2.sum())
print("Sum of arr3:", arr3.sum())
```

Code 24: Numpy sum()

**Output:**

```
Sum of arr1: 150
Sum of arr2: 360
Sum of arr3: 217
```

Output 18: Numpy sum()

### 9.5.2. Python numpy average:

Python numpy average function returns the average of a given array.

```
print(np.average(arr1))
print(np.average(arr2))
print(np.average(arr3))
```

Code 25: Numpy average()

Output:

```
30.0
40.0
18.083333333333332
```

Output 19: Numpy average()

Average of x and Y axis:

```
print("Average of columns in arr2 :",np.average(arr2, axis = 0))
print("Average of rows in arr2 :",np.average(arr2, axis = 1))
```

Code 26: Numpy average along with axis (with Axis name)

Output:

```
Average of columns in arr2 : [30. 40. 50.]
Average of rows in arr2 : [10. 40. 70.]
```

Output 20: Numpy average along with axis (with Axis name)

### 9.5.3. Python numpy min :

The Python numpy min function returns the minimum value in an array or a given axis.

```
import numpy as np
arr1 = np.array([10, 20, 30, 40, 50])

arr2 = np.array([[0, 10, 20], [30, 40, 50], [60, 70, 80]])

arr3 = np.array([[14, 6, 9, -12, 19, 72],[-9, 8, 22, 0, 99, -11]])

print("Minimum value in arr 1:",arr1.min())
print("Minimum value in arr 2:",arr2.min())
print("Minimum value in arr 3:",arr3.min())
```

Code 27: Numpy minimum function min()

**Output:**

```

➤ Minimum value in arr 1: 10
   Minimum value in arr 2: 0
   Minimum value in arr 3: -12

```

*Output 21: Numpy minimum function min()*

We are finding the numpy array minimum value in the X and Y-axis.

```

▶ print("Minimum value in arr 2:",arr2.min(axis = 0))
  print("Minimum value in arr 2:",arr2.min(axis = 1))
  print("Minimum value in arr 3:",arr3.min(axis = 0))
  print("Minimum value in arr 3:",arr3.min(axis = 1))

```

Code 28: Numpy minimum function with and without axis name

**Output:**

```

➤ Minimum value in arr 2: [ 0 10 20]
   Minimum value in arr 2: [ 0 30 60]
   Minimum value in arr 3: [-9  6  9 -12 19 -11]
   Minimum value in arr 3: [-12 -11]

```

Output 22: Numpy minimum function with and without axis name

#### 9.5.4. Python numpy max

The Python numpy max function returns the maximum number from a given array or in a given axis.

```

▶ import numpy as np
  arr1 = np.array([10, 20, 30, 40, 50])

  arr2 = np.array([[0, 10, 20], [30, 40, 50], [60, 70, 80]])

  arr3 = np.array([[14, 6, 9, -12, 19, 72],[-9, 8, 22, 0, 99, -11]])

  print("Maximum value in arr 1:",arr1.max())
  print("Maximum value in arr 2:",arr2.max())
  print("Mamimum value in arr 3:",arr3.max())

```

Code 29: Numpy maximum function max()

Output:

```
Maximum value in arr 1: 50
Maximum value in arr 2: 80
Maximum value in arr 3: 99
```

Output 23: Numpy maximum function max()

Find the maximum value in the X and Y-axis using numpy max function.

```
print("Maximum value in arr 2:",arr2.max(axis = 0))
print("Maximum value in arr 2:",arr2.max(axis = 1))
print("Maximum value in arr 3:",arr3.max(axis = 0))
print("Maximum value in arr 3:",arr3.max(axis = 1))
```

Code 30: Numpy maximum function max() with axis name

Output:

```
Maximum value in arr 2: [60 70 80]
Maximum value in arr 2: [20 50 80]
Maximum value in arr 3: [14  8 22  0 99 72]
Maximum value in arr 3: [72 99]
```

Output 24: Numpy maximum function max() with axis name

## 9.6. N-Dimensional arrays

An ndarray is a (usually fixed-size) multidimensional container of items of the same type and size. The number of dimensions and items in an array is defined by its shape, which is a tuple of N positive integers that specify the sizes of each dimension.

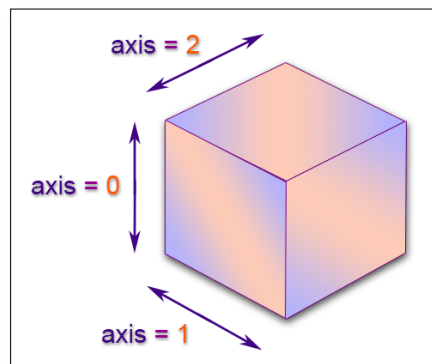


Figure 41: N-Dimensional arrays

The type of items in the array is specified by a separate data-type object (dtype), one of which is associated with each ndarray.

Like other container objects in Python, the contents of an ndarray can be accessed and modified by indexing or slicing the array (using, for example, N integers), and via the methods and attributes of the ndarray.

The basic ndarray is created using an array function in NumPy as follows –

- `numpy.array`

It creates an ndarray from any object exposing array interface, or from any method that returns an array.

- `numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)`

The above constructor takes the following parameters –

Sr.No.	Parameter & Description
1	<b>object</b> Any object exposing the array interface method returns an array, or any (nested) sequence.
2	<b>dtype</b> Desired data type of array, optional
3	<b>copy</b> Optional. By default (true), the object is copied
4	<b>order</b> C (row major) or F (column major) or A (any) (default)
5	<b>subok</b> By default, returned array forced to be a base class array. If true, sub-classes passed through
6	<b>ndmin</b> Specifies minimum dimensions of resultant array

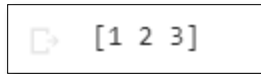
Table 9: Numpy array Parameters

**Example:**

```
import numpy as np
a = np.array([1,2,3])
print(a)
```

Code 31: 1D array

**Output:**



[1 2 3]

Output 25: 1D array

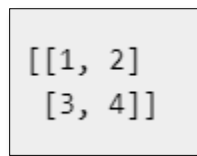
**Example:**

**More than one dimensions:**

```
# more than one dimensions
import numpy as np
a = np.array([[1, 2], [3, 4]])
print a
```

Code 32: 2D array

**Output:**



[[1, 2]  
 [3, 4]]

Output 26: 2D array

## 9.7. Broadcasting

The term broadcasting describes how numpy treats arrays with different shapes during arithmetic operations. Subject to certain constraints, the smaller array is “broadcast” across the larger array so that they have compatible shapes. Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

For example, suppose that we want to add a constant vector to each row of a matrix. We could do like this:

```

import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = np.empty_like(x) # 4,3 Create an empty matrix with the same shape as x

for i in range(4):

    y[i, :] = x[i, :] + v

print(y)

```

Code 33: Adding a constant vector to each row of a matrix

Now y is the following:

```

[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]

```

*Output 27: Adding a constant vector to each row of a matrix*

- This works; however, when the matrix x is very large, computing an explicit loop in Python could be slow.
- Note that adding the vector v to each row of the matrix x is equivalent to forming a matrix vv by stacking multiple copies of v vertically.
- Then performing elementwise summation of x and vv. We could implement this approach like this:

**Using Tile:**

```

import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
vv = np.tile(v, (4, 1)) # Stack 4 copies of v on top of each other
print(vv)
print("_____\n")
y = x + vv # Add x and vv elementwise
print(y)

```

Code 34: Adding a constant vector to each row of a matrix using tile() function



**Output:**

```

↳ [[1 0 1]
    [1 0 1]
    [1 0 1]
    [1 0 1]]

    [[ 2  2  4]
     [ 5  5  7]
     [ 8  8 10]
     [11 11 13]]

```

Output 28: Adding a constant vector to each row of a matrix using tile() function

### Broadcasting:

Numpy broadcasting allows us to perform this computation without actually creating multiple copies of *v*. Consider this version, using broadcasting:

```

▶ import numpy as np

# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])
y = x + v # Add v to each row of x using broadcasting
print(y)

```

Code 35: Adding to each row of array using broadcasting

**Output:**

```

↳ [[ 2  2  4]
    [ 5  5  7]
    [ 8  8 10]
    [11 11 13]]

```

Output 29: Adding to each row of array using broadcasting

- The line `y = x + v` works even though *x* has shape (4, 3) and *v* has shape (3,) due to broadcasting; this line works as if *v* actually had shape (4, 3), where each row was a copy of *v*, and the sum was performed elementwise.

**Broadcasting Rules:**

The trailing axes of both arrays must either be 1 or have the same size for broadcasting to occur. Otherwise, a **“ValueError: frames are not aligned”** exception is thrown.

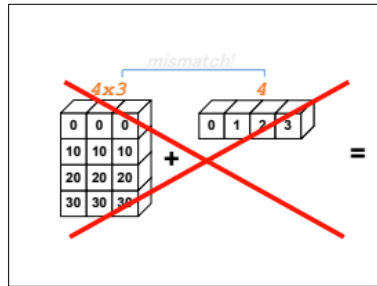


Figure 42: Broadcasting rules

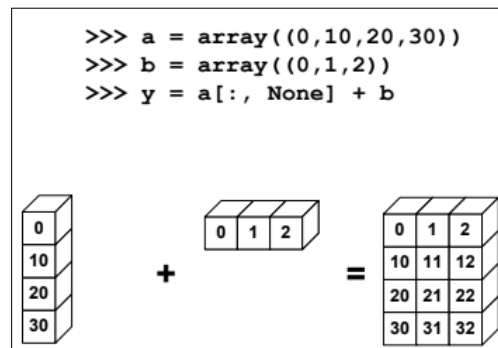
**Broadcasting in Action:**

Figure 43: Broadcasting in Action

## 9.8. Fancy indexing

In Fancy Indexing, we pass array of indices instead of single scalar(numbers) to fetch elements at different index points. Remember that the shape of the output depends on the shape of the index arrays rather than the shape of the array being indexed.

Let's go through some examples to understand this concept:

```
import numpy as np

rand = np.random.RandomState(42)

# creating 1d array for demonstration
arr1 = rand.randint(100, size=10)
print(f"1D array:\n{arr1}")

#creating 2d array for demonstration
arr2 = rand.randint(100, size=(3,5))
print(f"\n2D array:\n{arr2}")
```

Code 36: Fancy indexing

Output:

```
1D array:
[51 92 14 71 60 20 82 86 74 74]

2D array:
[[87 99 23  2 21]
 [52  1 87 29 37]
 [ 1 63 59 20 32]]
```

Output 30: Fancy Indexing

**Case A****a. 1D Array:**

For 1D array, let's suppose we want to access elements at index position of 0, 4 and -1.

```
# method 1

arr1[0],arr1[4],arr1[-1]

(51, 60, 74)
```

Figure 44: Fancy indexing 1D array method 1

```
# method 2

i = np.array([0,4,-1])
arr1[i]

array([51, 60, 74])
```

Figure 45: Fancy indexing 1D array Method 2

**9.9. Sorting Arrays**

Sorting means putting elements in an ordered sequence. Ordered sequence is any sequence that has an order corresponding to elements, like numeric or alphabetical, ascending or descending. The NumPy ndarray object has a function called `sort()`, that will sort a specified array.

**Example:**

```
import numpy as np

arr = np.array([3, 2, 0, 1])

print(np.sort(arr))
```

Code 37: Example 1. Sorting arrays 1D

Output:

```
[0 1 2 3]
```

Output 31: Example 1. Sorting arrays 1D

**Example:**

```
import numpy as np

arr = np.array(['banana', 'cherry', 'apple'])
print("Sorted Array")
print(np.sort(arr))
```

Code 38: Example 2. Sorting arrays 1D

**Output:**

```
Sorted Array
['apple' 'banana' 'cherry']
```

Output 32: Example 2. Sorting arrays 1D

**Sorting a 2-D Array:**

```
import numpy as np

arr = np.array([[3, 2, 4], [5, 0, 1]])
print("Sorted Array")
print(np.sort(arr))
```

Code 39: Sorting 2D arrays

**Output:**

```
Sorted Array
[[2 3 4]
 [0 1 5]]
```

Output 33: Sorting 2D arrays

## Exercise

### Objective Type Question

**1. NumPy stands for?**

- a) Numerical Python
- b) Number In Python
- c) Numbering Python
- d) None Of the above

**2. Numpy developed by?**

- a) Jim Hugunin
- b) Wes McKinney
- c) Travis Oliphant
- d) Guido van Rossum

**3. Which of the following Numpy operation are correct?**

- a) Operations related to linear algebra.
- b) Mathematical and logical operations on arrays.
- c) Fourier transforms and routines for shape manipulation.
- d) All of the above

**4. NumPy is often used along with packages like?**

- a) Node.js
- b) SciPy
- c) Matplotlib
- d) Both B and C

**5. Which of the following is contained in NumPy library?**

- a) n-dimensional array object
- b) tools for integrating C/C++ and Fortran code
- c) All of the mentioned
- d) Fourier transform

**6. which of the following function is used to combine different vectors so as to obtain result for**

**each n-uplet?**

- a) iid\_
- b) ix\_
- c) ixd\_
- d) None of the above

**7. Which of the following sets the size of the buffer used in ufuncs?**

- a) setsize(size)
- b) bufsize(size)
- c) setbufsize(size)
- d) All of the mentioned

**8. Which of the following attribute should be used while checking for type combination input and output?**

- a) .types
- b) .class
- c) .type
- d) None of the above

**9. Which of the following function stacks 1D arrays as columns into a 2D array?**

- a) column\_stack
- b) com\_stack
- c) row\_stack
- d) All of the above

**10. The \_\_\_\_\_ function returns its argument with a modified shape, whereas the \_\_\_\_\_ method modifies the array itself.**

- a) resize, reshape
- b) reshape, resize
- c) reshape2, resize
- d) None of the above

**11. Point out the correct statement in NumPy.**

- a) Numpy array class is called ndarray
- b) In Numpy, dimensions are called axes
- c) NumPy main object is the homogeneous multidimensional array
- d) All of the mentioned

**12. Which of the following method creates a new array object that looks at the same data?**

- a) copy
- b) paste
- c) view
- d) All of the above

**13. Which of the following function take only single value as input?**

- a) fmin
- b) minimum
- c) iscomplex
- d) None of the above

**14. Which of the following set the floating-point error callback function or log object?**

- a) seterrcall
- b) setter
- c) setterstack
- d) All of the above

**15. What will be output for the following code?**

```
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print a
```

- a) [ 1.+0.j]
- b) [[ 1.+0.j, 3.+0.j]]
- c) [[ 1.+0.j, 2.+0.j, 3.+0.j]]
- d) [ 1.+0.j, 2.+0.j, 3.+0.j]

### **Subjective Type Questions**

1. Why to use NumPy?
2. Explain what is ndarray in NumPy
3. What is the difference between ndarray and array in NumPy?
4. How to convert a numeric array to a categorical (text) array?
5. How would you reverse a NumPy array?
6. What are the differences between NumPy arrays and matrices?
7. What are the advantages of NumPy over regular Python lists?
8. What are the differences between np.mean() vs np.average() in Python NumPy?
9. What are the differences between NumPy arrays and matrices?
10. Explain what is Vectorization in NumPy
11. Why is NumPy Array good compared to Python Lists?
12. How can you reshape NumPy array?
13. How many dimensions can a NumPy array have?
14. How are NumPy Arrays better than Lists in Python?
15. Explain the data types supported by NumPy.



## Chapter 10.

### Data Analysis Tool: Pandas

#### 10.1. Introduction to the Data Analysis Library Pandas

Pandas is an open-source library designed primarily for working quickly and logically with relational or labelled data. It offers a range of data structures and procedures for working with time series and numerical data. The NumPy library serves as the foundation for this library. Pandas is quick and offers its users exceptional performance & productivity.

Processing steps including merging, cleansing, and restructuring are all necessary for data analysis. For quick data processing, a variety of tools are available, including Numpy, Scipy, Cython, and Panda. However, we favour Pandas since they are quicker, easier, and more expressive to utilise than other tools.

Given that Pandas is built on top of the Numpy package, Numpy is necessary in order to use Pandas. Before Pandas, Python was capable for data preparation, but it only provided limited support for data analysis. So, Pandas came into the picture and enhanced the capabilities of data analysis. It can perform five significant steps required for processing and analysis of data irrespective of the origin of the data, i.e., load, manipulate, prepare, model, and analyze.



Figure 46: Pandas

#### History:



Pandas were initially developed by Wes McKinney in 2008 while he was working at AQR Capital Management. He convinced the AQR to allow him to open source the Pandas. Another AQR employee, Chang She, joined as the second major contributor to the library in 2012. Over time many versions of pandas have been released. The latest version of the pandas is 1.4.4.

Figure 47: Developer of Pandas Wes McKinney (2008)

#### 10.2. Pandas objects – Series and Data frames

Integer, text, boolean, float, and other datatypes, including Python objects, can all be stored in a Pandas Series, a one-dimensional indexed data structure. One data type can only be stored in a Pandas Series at a time. The index of the series is the name given to the data's axis label. The labels must be a hashable type even though they don't have to be unique. Integer, text, or even time-series data can be used as the series' index. In general, a Pandas Series is just an Excel sheet's column, with the row index serving as the series' index.

### 10.2.1 Pandas Series

We can create a Pandas Series by using the following `pandas.Series()` constructor:-

```
pandas.Series([data, index, dtype, name, copy, ...])
```

The parameters for the constructor of a Python Pandas Series are detailed as under:-

Parameters	Remarks
data : array-like, Iterable, dict, or scalar value	Contains data stored in Series. Changed in version 0.23.0: If data is a dict, argument order is maintained for Python 3.6 and later.
index : array-like or Index (1d)	Values must be hashable and have the same length as data. Non-unique index values are allowed. Will default to RangeIndex (0, 1, 2, ..., n) if not provided. If both a dict and index sequence are used, the index will override the keys found in the dict.
dtype : str, numpy.dtype, or ExtensionDtype, optional	Data type for the output Series. If not specified, this will be inferred from data.
copy : bool, default False	Copy input data.

Table 10: Pandas Series Parameters

#### 10.2.1.1 How to create an empty Pandas Series?

```
import pandas as pd
empty_series = pd.Series()
print(empty_series)
```

Code 40: Empty Pandas Series

**Output:**

```
Series([], dtype: float64)
```

Output 34: Empty Pandas Series

#### 10.2.1.2 How to create a Pandas Series from a list?

```
import pandas as pd
data = pd.Series(['a', 'b', 'c', 'd'])
print (data)
```

Code 41: Pandas series from a list

**Output:**

```

0    a
1    b
2    c
3    d
dtype: object

```

Output 35: Pandas series from a list

**10.2.2 Pandas Dataframe**

One of pandas' main data structures is the pandas dataframe. A two-dimensional size changeable array with adjustable row indices and column names is known as a "Pandas dataframe." Generally speaking, it resembles an excel sheet or SQL table. It can also be thought of as a container for series objects similar to Python's dict. Various techniques for constructing a Pandas Dataframe

The following pandas can be used to create or construct a Pandas Dataframe.

The function `Object() { [native code] }` for `DataFrame()`:

```
pd.DataFrame([data, index, columns, dtype, name, copy, ...])
```

A Pandas Dataframe can be created from:-

- Dict of 1D ndarrays, lists, dicts, or Series
- 2-D numpy.ndarray
- Structured or record ndarray
- A Series
- Another DataFrame

The parameters for the constructor of a Pandas Dataframe are detailed as under:-

Parameters	Remarks
data : ndarray (structured or homogeneous), Iterable, dict, or DataFrame	Dict can contain Series, arrays, constants, or list-like objects Changed in version 0.23.0: If data is a dict, column order follows insertion-order for Python 3.6 and later. Changed in version 0.25.0: If data is a list of dicts, column order follows insertion-order for Python 3.6 and later.
index : Index or array-like	Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided
columns : Index or array-like	Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, ..., n) if no column labels are provided
dtype, default None	Data type to force. Only a single dtype is allowed. If None, infer
copy : bool, default False	Copy data from inputs. Only affects DataFrame / 2d ndarray input

Table 11: Parameters for Pandas Dataframe

You can create an empty Pandas Dataframe using `pandas.DataFrame()` and later on you can add

```
import pandas as pd
df = pd.DataFrame()
df
```

Code 42: Empty Pandas Dataframe

the columns using `df.columns = [list of column names]` and to it

**Output:**



Output 36: Empty Pandas Dataframe

A pandas dataframe can be created from a 2 dimensional by using the following code:-

```
import pandas as pd
import numpy as np

df = pd.DataFrame(np.random.rand(3, 2))
print(df)
```

Code 43: Pandas Dataframe from 2D Numpy array

**Output:**

```
      0      1
0  0.059926  0.119440
1  0.548637  0.232405
2  0.343573  0.809589
```

Output 37: Pandas Dataframe from 2D Numpy array

## 10.3 Data indexing and selection

### Indexing in Pandas:

In Pandas, picking specific rows and columns of data from a DataFrame constitutes indexing. Selecting all the rows and part of the columns, some of the rows and all the columns, or a portion of each row and each column is what is referred to as indexing. Another name for indexing is subset selection.

There are a lot of ways to pull the elements, rows, and columns from a DataFrame. There are some indexing method in Pandas which help in getting an element from a DataFrame. These indexing methods appear very similar but behave very differently. Pandas support four types of Multi-axes indexing they are:

- **Dataframe.[ ]** ; This function also known as indexing operator

Collectively, they are called the **indexers**. These are by far the most common ways to index data. These are four function which help in getting the elements, rows, and columns from a DataFrame.

Indexing a Dataframe using indexing operator [ ] :Indexing operator is used to refer to the square brackets following an object. The indexers also use the indexing operator to make selections.

In this indexing operator to refer to df[ ].

Selecting a single column

In order to select a single column, we simply put the name of the column in-between the brackets

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving columns by indexing operator
first = data["Age"]

print(first)
```

Code 44: Indexing: selecting a single column

**Output:**

Name	
Avery Bradley	25.0
Jae Crowder	25.0
John Holland	27.0
R.J. Hunter	22.0
Jonas Jerebko	29.0
Amir Johnson	29.0
Jordan Mickey	21.0
Kelly Olynyk	25.0
Terry Rozier	22.0
Marcus Smart	22.0
Jared Sullinger	24.0
Isaiah Thomas	27.0
▪	▪
▪	▪
▪	▪
Joe Ingles	28.0
Chris Johnson	26.0
Trey Lyles	20.0
Shelvin Mack	26.0
Raul Neto	24.0
Tibor Pleiss	26.0
Jeff Withey	26.0
NaN	NaN

Name: Age, Length: 458, dtype: float64

Output 38: Indexing: selecting a single column

### Selecting multiple columns

In order to select multiple columns, we have to pass a list of columns in an indexing operator.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving multiple columns by indexing operator
first = data[["Age", "College", "Salary"]]

first
```

Code 45: selecting multiple columns

### Output:

	Age	College	Salary
Name			
Avery Bradley	25.0	Texas	7730337.0
Jae Crowder	25.0	Marquette	6796117.0
John Holland	27.0	Boston University	NaN
R.J. Hunter	22.0	Georgia State	1148640.0
Jonas Jerebko	29.0	NaN	5000000.0
Amir Johnson	29.0	NaN	12000000.0
Jordan Mickey	21.0	LSU	1170960.0
Kelly Olynyk	25.0	Gonzaga	2165160.0
Terry Rozier	22.0	Louisville	1824360.0
...	...	...	...
Joe Ingles	28.0	NaN	2050000.0
Chris Johnson	26.0	Dayton	981348.0
Trey Lyles	20.0	Kentucky	2239800.0
Shelvin Mack	26.0	Butler	2433333.0
Raul Neto	24.0	NaN	900000.0
Tibor Pleiss	26.0	NaN	2900000.0
Jeff Withey	26.0	Kansas	947276.0
NaN	NaN	NaN	NaN

458 rows x 3 columns

Output 39: selecting multiple columns

### Indexing a DataFrame using

This function selects data by the **label** of the rows and columns. The `df.loc` indexer selects data in a different way than just the indexing operator. It can select subsets of rows or columns. It can also simultaneously select subsets of rows and columns.

### Selecting a single row

In order to select a single row using `.loc[]`, we put a single row label in a `.loc` function.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving row by loc method
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]

print(first, "\n\n", second)
```

Code 46: Selecting a single row using `.loc[]`

### Output:

As shown in the output image, two series were returned since there was only one parameter both of the times

```
Team      Boston Celtics
Number      0
Position    PG
Age         25
Height      6-2
Weight      180
College     Texas
Salary      7.73034e+06
Name: Avery Bradley, dtype: object

Team      Boston Celtics
Number      28
Position    SG
Age         22
Height      6-5
Weight      185
College     Georgia State
Salary      1.14864e+06
Name: R.J. Hunter, dtype: object
```

Output 40: Selecting a single row using `.loc[]`

### Selecting multiple rows

In order to select multiple rows, we put all the row labels in a list and pass that to function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving multiple rows by loc method
first = data.loc[["Avery Bradley", "R.J. Hunter"]]

print(first)
```

Code 47: Selecting multiple row using `.loc[]`

## Output

	Team	Number	Position	Age	Height	Weight	College	Salary
Name								
Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0

Output 41: Selecting multiple row using .loc[]

Selecting two rows and three columns

In order to select two rows and three columns, we select a two rows which we want to select and three columns and put it in a separate list like this:

**Syntax:**

`Dataframe.loc[["row1", "row2"], ["column1", "column2", "column3"]]`

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving two rows and three columns by loc method
first = data.loc[["Avery Bradley", "R.J. Hunter"],
                 ["Team", "Number", "Position"]]

print(first)
```

Code 48: Selecting two rows and three columns using .loc[]

**Output:**

	Team	Number	Position
Name			
Avery Bradley	Boston Celtics	0.0	PG
R.J. Hunter	Boston Celtics	28.0	SG

Output 42: Selecting two rows and three columns using .loc[]

Selecting all of the rows and some columns

In order to select all of the rows and some columns, we use single colon [:] to select all of rows and list of some columns which we want to select like this:

**Syntax:**

`Dataframe.loc[:, ["column1", "column2", "column3"]]`

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving all rows and some columns by loc method
first = data.loc[:, ["Team", "Number", "Position"]]

print(first)
```

Code 49: Selecting all rows and some columns using .loc[]



**Output:**

Name	Team	Number	Position
Avery Bradley	Boston Celtics	0.0	PG
Jae Crowder	Boston Celtics	99.0	SF
John Holland	Boston Celtics	30.0	SG
R.J. Hunter	Boston Celtics	28.0	SG
Jonas Jerebko	Boston Celtics	8.0	PF
Amir Johnson	Boston Celtics	90.0	PF
Jordan Mickey	Boston Celtics	55.0	PF
Kelly Olynyk	Boston Celtics	41.0	C
Terry Rozier	Boston Celtics	12.0	PG
Marcus Smart	Boston Celtics	36.0	PG
Jared Sullinger	Boston Celtics	7.0	C
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
.	.	.	.
Rudy Gobert	Utah Jazz	27.0	C
Gordon Hayward	Utah Jazz	20.0	SF
Rodney Hood	Utah Jazz	5.0	SG
Joe Ingles	Utah Jazz	2.0	SF
Chris Johnson	Utah Jazz	23.0	SF
Trey Lyles	Utah Jazz	41.0	PF
Shelvin Mack	Utah Jazz	8.0	PG
Raul Neto	Utah Jazz	25.0	PG
Tibor Pleiss	Utah Jazz	21.0	C
Jeff Withey	Utah Jazz	24.0	C
NaN	NaN	NaN	NaN

[458 rows x 3 columns]

Output 43: Selecting all rows and some columns using .loc[]

**Indexing a DataFrame:**

This function allows us to retrieve rows and columns by position. In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well. The `df.iloc` indexer is very similar to `df.loc` but only uses integer locations to make its selections.

**Selecting a single row**

In order to select a single row using `.iloc[]`, we can pass a single integer to `.iloc[]` function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("nba.csv", index_col = "Name")

# retrieving rows by iloc method
row2 = data.iloc[3]

print(row2)
```

Code 50: Selecting a single row using .iloc[]

**Output:**

Team	Boston Celtics
Number	28
Position	SG
Age	22
Height	6-5
Weight	185
College	Georgia State
Salary	1.14864e+06
Name: R.J. Hunter, dtype: object	

Output 44: Selecting a single row using .iloc[]

## Methods for indexing in DataFrame

Function	Description
	Return top n rows of a data frame.
	Return bottom n rows of a data frame.
	Access a single value for a row/column label pair.
	Access a single value for a row/column pair by integer position.
	Purely integer-location based indexing for selection by position.
DataFrame.lookup()	Label-based “fancy indexing” function for DataFrame.
	Return item and drop from frame.
DataFrame.xs()	Returns a cross-section (row(s) or column(s)) from the DataFrame.
	Get item from object for given key (DataFrame column, Panel slice, etc.).
	Return boolean DataFrame showing whether each element in the DataFrame is contained in values.
	Return an object of same shape as self and whose corresponding entries are from self where cond is True and otherwise are from other.
	Return an object of same shape as self and whose corresponding entries are from self where cond is False and otherwise are from other.
	Query the columns of a frame with a boolean expression.
	Insert column into DataFrame at specified location.

Table 12: Methods for indexing in DataFrame

## 10.4 Nan objects

Missing Data can occur when no information is provided for one or more items or for a whole unit. Missing Data is a very big problem in a real-life scenarios. Missing Data can also refer to as NA(Not Available) values in pandas. In DataFrame sometimes many datasets simply arrive with missing data, either because it exists and was not collected or it never existed. For Example, Suppose different users being surveyed may choose not to share their income, some users may choose not to share the address in this way many datasets went missing.

In Pandas missing data is represented by two value:

- **None:** None is a Python singleton object that is often used for missing data in Python code.
- **NaN :** NaN (an acronym for Not a Number), is a special floating-point value recognized by all systems that use the standard IEEE floating-point representation

Pandas treat None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful functions for detecting, removing, and replacing null values in Pandas DataFrame:

### Checking for missing values using isnull() and notnull()

In order to check missing values in Pandas DataFrame, we use a function isnull() and notnull(). Both function help in checking whether a value is NaN or not. These function can also be used in Pandas Series in order to find null values in a series.

### Checking missing values using isnull()

In order to check null values in Pandas DataFrame, we use isnull() function this function return dataframe of Boolean values which are True for NaN values.

**Code #1:**

```
# importing pandas as pd
import pandas as pd

# importing numpy as np
import numpy as np

# dictionary of lists
dict = {'First Score':[100, 90, np.nan, 95],
        'Second Score': [30, 45, 56, np.nan],
        'Third Score':[np.nan, 40, 80, 98]}

# creating a dataframe from list
df = pd.DataFrame(dict)

# using isnull() function
df.isnull()
```

Code 51: Checking missing values using isnull()

**Output:**

	First Score	Second Score	Third Score
0	False	False	True
1	False	False	False
2	True	False	False
3	False	True	False

Output 45: Checking missing values using isnull()

## 10.5 Manipulating Data Frames

Before manipulating the dataframe with pandas we have to understand what is data manipulation. The data in the real world is very unpleasant & unordered so by performing certain operations we can make data understandable based on one's requirements, this process of converting unordered data into meaningful information can be done by data manipulation.

Pandas is an open-source library that is used from data manipulation to data analysis & is very powerful, flexible & easy to use tool which can be imported using `import pandas as pd`. Pandas deal essentially with data in 1-D and 2-D arrays; Although, pandas handles these two differently. In pandas, 1-D arrays are stated as a series & a dataframe is simply a 2-D array.

**Below are various operations used to manipulate the dataframe:**

- First, import the library which is used in data manipulation i.e. pandas then assign and read the dataframe:

```
# import module
import pandas as pd

# assign dataset
df = pd.read_csv("country_code.csv")

# display
print("Type-", type(df))
df
```

Code 52: Importing the library for data manipulation

**Output:**

```
Type- <class 'pandas.core.frame.DataFrame'>
Out[84]:
```

	Name	Code
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
...	...	...
244	Wallis and Futuna	WF
245	Western Sahara	EH
246	Yemen	YE
247	Zambia	ZM
248	Zimbabwe	ZW

249 rows × 2 columns

Output 46: Importing the library for data manipulation

- We can read the dataframe by using **head()** function also which is having an argument (n) i.e. number of rows to be displayed.

```
df.head(10)
```

Code 53: read the dataframe by using head() function

**Output:**

```
Out[85]:
```

	Name	Code
0	Afghanistan	AF
1	Åland Islands	AX
2	Albania	AL
3	Algeria	DZ
4	American Samoa	AS
5	Andorra	AD
6	Angola	AO
7	Anguilla	AI
8	Antarctica	AQ
9	Antigua and Barbuda	AG

Output 47: read the dataframe by using head() function

- Counting the rows and columns in DataFrame using **shape()**. It returns the no. of rows and columns enclosed in a tuple.

```
df.shape
```

Code 54: Counting the rows and columns in DataFrame using shape().

**Output:**

```
Out[35]: (249, 2)
```

Output 48: Counting the rows and columns in DataFrame using shape().

- Summary of Statistics of DataFrame using **describe()** method.

```
df.describe()
```

Code 55: Summary of Statistics of DataFrame using describe() method

**Output:**

```
Out[36]:
```

	Name	Code
count	249	248
unique	249	248
top	Dominican Republic	LA
freq	1	1

Output 49: Summary of Statistics of DataFrame using describe() method

- Merging DataFrames using **merge()**, arguments passed are the dataframes to be merged along with the column name.

```
df1 = pd.read_csv("country_code.csv")
merged_col = pd.merge(df, df1, on='Name')
merged_col
```

Code 56: Merging DataFrames using merge()

**Output:**

```
Out[91]:
```

	Name	Code_x	Code_y
0	Afghanistan	AF	AF
1	Åland Islands	AX	AX
2	Albania	AL	AL
3	Algeria	DZ	DZ
4	American Samoa	AS	AS
...	...	...	...
244	Wallis and Futuna	WF	WF
245	Western Sahara	EH	EH
246	Yemen	YE	YE
247	Zambia	ZM	ZM
248	Zimbabwe	ZW	ZW

249 rows x 3 columns

Output 50: Merging DataFrames using merge()

**Creating a dataframe manually:**

```
student = pd.DataFrame({'Name': ['Rohan', 'Rahul', 'Gaurav',
                                'Ananya', 'Vinay', 'Rohan',
                                'Vivek', 'Vinay'],
                        'Score': [76, 69, 70, 88, 79, 64, 62, 57]})

# Reading Dataframe
student
```

Code 57: Creating a dataframe manually

**Output:**

Out[228]:

	Name	Score
0	Rohan	76
1	Rahul	69
2	Gaurav	70
3	Ananya	88
4	Vinay	79
5	Rohan	64
6	Vivek	62
7	Vinay	57

Output 51: Creating a dataframe manually

- Sorting the DataFrame using **sort\_values()** method.

```
student.sort_values(by=['Score'], ascending=True)
```

Code 58: Sorting the DataFrame using sort\_values()

**Output:**

Out[229]:

	Name	Score
7	Vinay	57
6	Vivek	62
5	Rohan	64
1	Rahul	69
2	Gaurav	70
0	Rohan	76
4	Vinay	79
3	Ananya	88

Output 52: Sorting the DataFrame using sort\_values()

- Creating another column in DataFrame, Here we will create column name percentage which will calculate the percentage of student score by using aggregate function sum().

```
student['Percentage'] = (student['Score'] / student['Score'].sum()) * 100
student
```

Code 59: Creating another column in DataFrame

**Output:**

Out[233]:

	Name	Score	Percentage
0	Rohan	76	13.451327
1	Rahul	69	12.212389
2	Gaurav	70	12.389381
3	Ananya	88	15.575221
4	Vinay	79	13.982301
5	Rohan	64	11.327434
6	Vivek	62	10.973451
7	Vinay	57	10.088496

Output 53: Creating another column in DataFrame

- Selecting DataFrame rows using logical operators:

```
# Selecting rows where score is
# greater than 70
print(student[student.Score>70])

# Selecting rows where score is greater than 60
# OR less than 70
print(student[(student.Score>60) | (student.Score<70)])
```

Code 60: Selecting DataFrame rows using logical operators

**Output:**

	Name	Score	Percentage
0	Rohan	76	13.451327
3	Ananya	88	15.575221
4	Vinay	79	13.982301
0	Rohan	76	13.451327
1	Rahul	69	12.212389
2	Gaurav	70	12.389381
3	Ananya	88	15.575221
4	Vinay	79	13.982301
5	Rohan	64	11.327434
6	Vivek	62	10.973451
7	Vinay	57	10.088496

Output 54: Selecting DataFrame rows using logical operators

## 10.6 Grouping

Pandas `groupby` is used for grouping the data according to the categories and apply a function to the categories. It also helps to aggregate data efficiently.

Pandas **`dataframe.groupby()`** function is used to split the data into groups based on some criteria. pandas objects can be split on any of their axes. The abstract definition of grouping is to provide a mapping of labels to group names.

### Syntax:

`DataFrame.groupby(by=None, axis=0, level=None, as_index=True, sort=True, group_keys=True, squeeze=False, **kwargs)`

Parameters :
<ul style="list-style-type: none"> <li>• <b>by:</b> mapping, function, str, or iterable</li> <li>• <b>axis:</b> int, default 0</li> <li>• <b>level:</b> If the axis is a MultiIndex (hierarchical), group by a particular level or levels</li> <li>• <b>as_index:</b> For aggregated output, return object with group labels as the index. Only relevant for DataFrame input. as_index=False is effectively “SQL-style” grouped output</li> <li>• <b>sort:</b> Sort group keys. Get better performance by turning this off. Note this does not influence the order of observations within each group. groupby preserves the order of rows within each group.</li> <li>• <b>group_keys:</b> When calling apply, add group keys to index to identify pieces</li> <li>• <b>squeeze:</b> Reduce the dimensionality of the return type if possible, otherwise return a consistent type</li> <li>• <b>Returns:</b> GroupBy object</li> </ul>

Table 13: Grouping parameters

**Example #1:** Use `groupby()` function to group the data based on the “Team”.

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# Print the dataframe
df
```

Code 61: Reading CSV file



**Output:**

Team	Name	Number	Position	Age	Height	Weight	College	Salary
Atlanta Hawks	Kent Bazemore	24.0	SF	26.0	6-5	201.0	Old Dominion	2000000.0
Boston Celtics	Avery Bradley	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
Brooklyn Nets	Bojan Bogdanovic	44.0	SG	27.0	6-8	216.0	Oklahoma State	3425510.0
Charlotte Hornets	Nicolas Batum	5.0	SG	27.0	6-8	200.0	Virginia Commonwealth	13125306.0
Chicago Bulls	Cameron Bairstow	41.0	PF	25.0	6-9	250.0	New Mexico	845059.0
Cleveland Cavaliers	Matthew Dellavedova	8.0	PG	25.0	6-4	198.0	Saint Mary's	1147276.0
Dallas Mavericks	Justin Anderson	1.0	SG	22.0	6-6	228.0	Virginia	1449000.0
Denver Nuggets	Darrell Arthur	0.0	PF	28.0	6-9	235.0	Kansas	2814000.0
Detroit Pistons	Joel Anthony	50.0	C	33.0	6-9	245.0	UNLV	2500000.0
Golden State Warriors	Leandro Barbosa	19.0	SG	33.0	6-3	194.0	North Carolina	2500000.0
Houston Rockets	Trevor Ariza	1.0	SF	30.0	6-8	215.0	UCLA	8193030.0
Indiana Pacers	Lavoy Allen	5.0	PF	27.0	6-9	255.0	Temple	4050000.0
Los Angeles Clippers	Cole Aldrich	45.0	C	27.0	6-11	250.0	Kansas	1100602.0
Los Angeles Lakers	Brandon Bass	2.0	PF	31.0	6-8	250.0	LSU	3000000.0
Memphis Grizzlies	Jordan Adams	3.0	SG	21.0	6-5	209.0	UCLA	1404600.0
Miami Heat	Chris Bosh	1.0	PF	32.0	6-11	235.0	Georgia Tech	22192730.0
Milwaukee Bucks	Giannis Antetokounmpo	34.0	SF	21.0	6-11	222.0	Arizona	1953960.0
Minnesota Timberwolves	Nemanja Bjelica	88.0	PF	28.0	6-10	240.0	Louisville	3950001.0

Output 55: CSV file

**Example #2:** Use `groupby()` function to form groups based on more than one category (i.e. Use more than one column to perform the splitting).

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# First grouping based on "Team"
# Within each team we are grouping based on "Position"
gkk = df.groupby(['Team', 'Position'])

# Print the first value in each group
gkk.first()
```

Code 62: `groupby()` function to form groups based on more than one category

**Output:**

		Name	Number	Age	Height	Weight	College	Salary
Team	Position							
Atlanta Hawks	C	Al Horford	15.0	30.0	6-10	245.0	Florida	12000000.0
	PF	Kris Humphries	43.0	31.0	6-9	235.0	Minnesota	1000000.0
	PG	Dennis Schroder	17.0	22.0	6-1	172.0	Wake Forest	1763400.0
	SF	Kent Bazemore	24.0	26.0	6-5	201.0	Old Dominion	2000000.0
	SG	Tim Hardaway Jr.	10.0	24.0	6-6	205.0	Michigan	1304520.0
Boston Celtics	C	Kelly Olynyk	41.0	25.0	7-0	238.0	Gonzaga	2165160.0
	PF	Jonas Jerebko	8.0	29.0	6-10	231.0	LSU	5000000.0
	PG	Avery Bradley	0.0	25.0	6-2	180.0	Texas	7730337.0
	SF	Jae Crowder	99.0	25.0	6-6	235.0	Marquette	6796117.0
	SG	John Holland	30.0	27.0	6-5	205.0	Boston University	1148640.0
Brooklyn Nets	C	Brook Lopez	11.0	28.0	7-0	275.0	Stanford	19689000.0
	PF	Chris McCullough	1.0	21.0	6-11	200.0	Syracuse	1140240.0
	PG	Jarrett Jack	2.0	32.0	6-3	200.0	Georgia Tech	6300000.0
	SG	Bojan Bogdanovic	44.0	27.0	6-8	216.0	Oklahoma State	3425510.0
Charlotte Hornets	C	Al Jefferson	25.0	31.0	6-10	289.0	Wisconsin	13500000.0
	PF	Tyler Hansbrough	50.0	30.0	6-9	250.0	North Carolina	947276.0
	PG	Jorge Gutierrez	12.0	27.0	6-3	189.0	California	189455.0
	SF	Michael Kidd-Gilchrist	14.0	22.0	6-7	232.0	Kentucky	6331404.0

Output 56: groupby() function to form groups based on more than one category

groupby() is a very powerful function with a lot of variations. It makes the task of splitting the dataframe over some criteria really easy and efficient.

## 10.7 Filtering

Python is a great language for doing data analysis, primarily because of the fantastic ecosystem of data-centric python packages. **Pandas** is one of those packages and makes importing and analyzing data much easier.

Pandas **dataframe.filter()** function is used to Subset rows or columns of dataframe according to labels in the specified index. Note that this routine does not filter a dataframe on its contents. The filter is applied to the labels of the index.

**Syntax:**

**DataFrame.filter(items=None, like=None, regex=None, axis=None)**

Parameters:

- items: List of info axis to restrict to (must not all be present)
- like: Keep info axis where “arg in col == True”
- regex: Keep info axis with re.search(regex, col) == True
- axis: The axis to filter on. By default, this is the info axis, ‘index’ for Series, ‘columns’ for DataFrame

Table 14: Filtering parameters

The items, like, and regex parameters are enforced to be mutually exclusive. axis defaults to the info axis that is used when indexing with [].

**Example #1:** Use filter() function to filter out any three columns of the dataframe.

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# Print the dataframe
df
```

Code 63: Reading CSV file

**Output:**

	Name	Team	Number	Position	Age	Height	Weight	College	Salary
0	Avery Bradley	Boston Celtics	0.0	PG	25.0	6-2	180.0	Texas	7730337.0
1	Jae Crowder	Boston Celtics	99.0	SF	25.0	6-6	235.0	Marquette	6796117.0
2	John Holland	Boston Celtics	30.0	SG	27.0	6-5	205.0	Boston University	NaN
3	R.J. Hunter	Boston Celtics	28.0	SG	22.0	6-5	185.0	Georgia State	1148640.0
4	Jonas Jerebko	Boston Celtics	8.0	PF	29.0	6-10	231.0	NaN	5000000.0
5	Amir Johnson	Boston Celtics	90.0	PF	29.0	6-9	240.0	NaN	12000000.0
6	Jordan Mickey	Boston Celtics	55.0	PF	21.0	6-8	235.0	LSU	1170960.0
7	Kelly Olynyk	Boston Celtics	41.0	C	25.0	7-0	238.0	Gonzaga	2165160.0
8	Terry Rozier	Boston Celtics	12.0	PG	22.0	6-2	190.0	Louisville	1824360.0
9	Marcus Smart	Boston Celtics	36.0	PG	22.0	6-4	220.0	Oklahoma State	3431040.0
10	Jared Sullinger	Boston Celtics	7.0	C	24.0	6-9	260.0	Ohio State	2569260.0
11	Isaiah Thomas	Boston Celtics	4.0	PG	27.0	5-9	185.0	Washington	6912869.0
12	Evan Turner	Boston Celtics	11.0	SG	27.0	6-7	220.0	Ohio State	3425510.0
13	James Young	Boston Celtics	13.0	SG	20.0	6-6	215.0	Kentucky	1749840.0
14	Tyler Zeller	Boston Celtics	44.0	C	26.0	7-0	253.0	North Carolina	2616975.0
15	Bojan Bogdanovic	Brooklyn Nets	44.0	SG	27.0	6-8	216.0	NaN	3425510.0

Output 57: CSV file

Now filter the “Name”, “College” and “Salary” columns.

```
# applying filter function
df.filter(["Name", "College", "Salary"])
```

Code 64: Use filter() function to filter out any three columns of the dataframe.

**Output:**

	Name	College	Salary
0	Avery Bradley	Texas	7730337.0
1	Jae Crowder	Marquette	6796117.0
2	John Holland	Boston University	NaN
3	R.J. Hunter	Georgia State	1148640.0
4	Jonas Jerebko	NaN	5000000.0
5	Amir Johnson	NaN	12000000.0
6	Jordan Mickey	LSU	1170960.0
7	Kelly Olynyk	Gonzaga	2165160.0
8	Terry Rozier	Louisville	1824360.0
9	Marcus Smart	Oklahoma State	3431040.0
10	Jared Sullinger	Ohio State	2569280.0
11	Isaiah Thomas	Washington	8912869.0
12	Evan Turner	Ohio State	3425510.0
13	James Young	Kentucky	1749840.0
14	Tyler Zeller	North Carolina	2516975.0
15	Bojan Bogdanovic	NaN	3425510.0
16	Markel Brown	Oklahoma State	845059.0

Output 58: Use filter() function to filter out any three columns of the dataframe.

**Example #2:** Use filter() function to subset all columns in a dataframe which has the letter 'a' or 'A' in its name

**Note :** filter() function also takes a regular expression as one of its parameter.

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# Using regular expression to extract all
# columns which has letter 'a' or 'A' in its name.
df.filter(regex = '[aA]')
```

Code 65: Use filter() function to subset all columns in a dataframe which has the letter 'a' or 'A' in its name.

**Output:**

	Name	Team	Age	Salary
0	Avery Bradley	Boston Celtics	25.0	7730337.0
1	Jae Crowder	Boston Celtics	25.0	6796117.0
2	John Holland	Boston Celtics	27.0	NaN
3	R.J. Hunter	Boston Celtics	22.0	1148640.0
4	Jonas Jerebko	Boston Celtics	29.0	5000000.0
5	Amir Johnson	Boston Celtics	29.0	12000000.0
6	Jordan Mickey	Boston Celtics	21.0	1170960.0
7	Kelly Olynyk	Boston Celtics	25.0	2165160.0
8	Terry Rozier	Boston Celtics	22.0	1824360.0
9	Marcus Smart	Boston Celtics	22.0	3431040.0
10	Jared Sullivan	Boston Celtics	24.0	2569260.0
11	Isaiah Thomas	Boston Celtics	27.0	6912669.0
12	Evan Turner	Boston Celtics	27.0	3425510.0
13	James Young	Boston Celtics	20.0	1749840.0
14	Tyler Zeller	Boston Celtics	26.0	2616975.0
15	Bojan Bogdanovic	Brooklyn Nets	27.0	3425510.0
16	Markal Brown	Brooklyn Nets	24.0	845059.0
17	Wayne Ellington	Brooklyn Nets	28.0	1500000.0

Output 59: Use filter() function to subset all columns in a dataframe which has the letter 'a' or 'A' in its name.

The regular expression '[aA]' looks for all column names which has an 'a' or an 'A' in its name.

## 10.8 Slicing

With the help of Pandas, we can perform many functions on data set like Slicing, Indexing, Manipulating, and Cleaning Data frame.

**Case 1:** Slicing Pandas Data frame

**Example 1:** Slicing Rows

```
# importing pandas library
import pandas as pd

# Initializing the nested list with Data set
player_list = [['M.S.Dhoni', 36, 75, 5428000],
               ['A.B.D Villers', 38, 74, 3428000],
               ['V.Kholi', 31, 70, 8428000],
               ['S.Smith', 34, 80, 4428000],
               ['C.Gayle', 40, 100, 4528000],
               ['J.Root', 33, 72, 7028000],
               ['K.Peterson', 42, 85, 2528000]]

# creating a pandas dataframe
df = pd.DataFrame(player_list, columns=['Name', 'Age', 'Weight', 'Salary'])

# data frame before slicing
df
```

Code 66: Creating Dataframe for slicing

**Output:**

	Name	Age	Weight	Salary
0	M.S.Dhoni	36	75	5428000
1	A.B.D Villers	38	74	3428000
2	V.Kholi	31	70	8428000
3	S.Smith	34	80	4428000
4	C.Gayle	40	100	4528000
5	J.Root	33	72	7028000
6	K.Peterson	42	85	2528000

Output 60: Dataframe for slicing

**Slicing rows in data frame**

```
# Slicing rows in data frame
df1 = df.iloc[0:4]

# data frame after slicing
df1
```

Code 67: Slicing rows in data frame

**Output:**

	Name	Age	Weight	Salary
0	M.S.Dhoni	36	75	5428000
1	A.B.D Villers	38	74	3428000
2	V.Kholi	31	70	8428000
3	S.Smith	34	80	4428000

Output 61: Slicing rows in data frame

**Example 2: Slicing Columns**

```
# importing pandas library
import pandas as pd

# Initializing the nested list with Data set
player_list = [['M.S.Dhoni', 36, 75, 5428000],
               ['A.B.D Villers', 38, 74, 3428000],
               ['V.Kholi', 31, 70, 8428000],
               ['S.Smith', 34, 80, 4428000],
               ['C.Gayle', 40, 100, 4528000],
               ['J.Root', 33, 72, 7028000],
               ['K.Peterson', 42, 85, 2528000]]

# creating a pandas dataframe
df = pd.DataFrame(player_list, columns=['Name', 'Age', 'Weight', 'Salary'])

# data frame before slicing
df
```

Code 68: Slicing Columns

**Output:**

	Name	Age	Weight	Salary
0	M.S.Dhoni	36	75	5428000
1	A.B.D Villers	38	74	3428000
2	V.Kholi	31	70	8428000
3	S.Smith	34	80	4428000
4	C.Gayle	40	100	4528000
5	J.Root	33	72	7028000
6	K.Peterson	42	85	2528000

Output 62: Slicing Columns

**Slicing columns in data frame:**

```
# Slicing columnss in data frame
df1 = df.iloc[:,0:2]

# data frame after slicing
df1
```

Code 69: Slicing Columns

**Output:**

	Name	Age
0	M.S.Dhoni	36
1	A.B.D Villers	38
2	V.Kholi	31
3	S.Smith	34
4	C.Gayle	40
5	J.Root	33
6	K.Peterson	42

Output 63: Slicing Columns

In the above example, we sliced the columns from the data frame.

## 10.9 Sorting

**DataFrame.sort\_values(*by*, *axis*=0, *ascending*=True, *inplace*=False, *kind*='quicksort', *na\_position*='last', *ignore\_index*=False, *key*=None)**

Parameters:
By: <i>str</i> or list of <i>str</i>
Name or list of names to sort by.
<ul style="list-style-type: none"> <li>if <i>axis</i> is 0 or '<i>index</i>' then <i>by</i> may contain index levels and/or column labels.</li> <li>if <i>axis</i> is 1 or '<i>columns</i>' then <i>by</i> may contain column levels and/or index labels.</li> </ul>
Axis: {0 or ' <i>index</i> ', 1 or ' <i>columns</i> '}, default 0
Axis to be sorted.
Ascending: bool or list of bool, default True
Sort ascending vs. descending. Specify list for multiple sort orders. If this is a list of bools, must match the length of the by.
Inplace: bool, default False
If True, perform operation in-place.
Kind: {'quicksort', 'mergesort', 'heapsort', 'stable'}, default 'quicksort'
Choice of sorting algorithm. See also numpy.sort() for more information. <i>mergesort</i> and <i>stable</i> are the only stable algorithms. For Data Frames, this option is only applied when sorting on a single column or label.
na_position: {'first', 'last'}, default 'last'
Puts NaNs at the beginning if <i>first</i> ; <i>last</i> puts NaNs at the end.
ignore_index: bool, default False
If True, the resulting axis will be labelled 0, 1, ..., n - 1.
Key: callable, optional
Apply the key function to the values before sorting. This is similar to the <i>key</i> argument in the built-in sorted() function, with the notable difference that this <i>key</i> function should be <i>vectorised</i> . It should expect a Series and return a Series with the same shape as the input. It will be applied to each column in <i>by</i> independently.

Table 15: Sorting parameters



## Creating a dataframe for demonstration

```
# importing pandas library
import pandas as pd

# creating and initializing a nested list
age_list = [['Afghanistan', 1952, 8425333, 'Asia'],
            ['Australia', 1957, 9712569, 'Oceania'],
            ['Brazil', 1962, 76039390, 'Americas'],
            ['China', 1957, 637408000, 'Asia'],
            ['France', 1957, 44310863, 'Europe'],
            ['India', 1952, 3.72e+08, 'Asia'],
            ['United States', 1957, 171984000, 'Americas']]

# creating a pandas dataframe
df = pd.DataFrame(age_list, columns=['Country', 'Year',
                                   'Population', 'Continent'])

df
```

Code 70: Creating a dataframe for demonstration

## Output:

	Country	Year	Population	Continent
0	Afghanistan	1952	8425333.0	Asia
1	Australia	1957	9712569.0	Oceania
2	Brazil	1962	76039390.0	Americas
3	China	1957	637408000.0	Asia
4	France	1957	44310863.0	Europe
5	India	1952	372000000.0	Asia
6	United States	1957	171984000.0	Americas

Output 64: Dataframe for demonstration

## Sorting Pandas Data Frame

In order to sort the data frame in pandas, function `sort_values()` is used. Pandas `sort_values()` can sort the data frame in Ascending or Descending order.

```
# Sorting by column 'Country'
df.sort_values(by=['Country'])
```

Code 71: Pandas `sort_values()` function

**Output:**

	Country	Year	Population	Continent
0	Afghanistan	1952	8425333.0	Asia
1	Australia	1957	9712569.0	Oceania
2	Brazil	1962	76039390.0	Americas
3	China	1957	637408000.0	Asia
4	France	1957	44310863.0	Europe
5	India	1952	372000000.0	Asia
6	United States	1957	171984000.0	Americas

Output 65: Pandas sort\_values() function

## 10.10 Ufunc

**Universal functions** in Numpy are simple mathematical functions. It is just a term that we gave to mathematical functions in the Numpy library. Numpy provides various universal functions that cover a wide variety of operations. These functions include standard trigonometric functions, functions for arithmetic operations, handling complex numbers, statistical functions, etc. Universal functions have various characteristics which are as follows:

- These functions operate on **ndarray** (N-dimensional array) i.e Numpy's array class.
- It performs fast element-wise array operations.
- It supports various features like array broadcasting, type casting etc.
- Numpy, universal functions are objects that belong to numpy.ufunc class.
- Python functions can also be created as a universal function using **frompyfunc** library function.
- Some **ufuncs** are called automatically when the corresponding arithmetic operator is used on arrays. For example when addition of two arrays is performed element-wise using '+' operator then np.add() is called internally.

```
# First, define two series whose index are not identical
A = pd.Series([1,2,3], index=[0,1,2]) #index[0,1,2]
B = pd.Series([10,20,30], index=[1,2,3]) #index[1,2,3]
# Second, perform addition of these two series
print(A); print(B)
print(A.add(B))
```

## **Exercise**

### **OBJECTIVE TYPE QUESTIONS**

**1. Which is performed by a Data Scientist?**

- a). Defining the question
- b). Creating reproducible code
- c). Challenging the results
- d). All of these
- e). None of these

**2. Which is one of the significant data science skills?**

- a). Statistics
- b). Machine Learning
- c). Data Visualization
- d). All of these
- e). None of these

**3. Which is performed by a Data Scientist?**

- a). Defining the question
- b). Creating reproducible code
- c). Challenging the results
- d). All of these
- e). None of these

**4. Which of the following is the correct statement.**

- a). Pre-processed data is original source of data
- b). Raw data is original source of data
- c). Raw data is the data obtained after processing steps
- d). None of these

**5. Data Analysis is a process of?**

- a). inspecting data
- b). cleaning data
- c). transforming data
- d). All of the above

**6. Which of the following is not a major data analysis approaches?**

- a). Data Mining
- b). Predictive Intelligence
- c). Business Intelligence
- d). Text Analytics

**7. Which of the following is true about regression analysis?**

- a). answering yes/no questions about the data
- b). estimating numerical characteristics of the data
- c). modeling relationships within the data
- d). describing associations within the data

**8. What is true about Data Visualization?**

- a). Data Visualization is used to communicate information clearly and efficiently to users by the usage of information graphics such as tables and charts.
- b). Data Visualization helps users in analyzing a large amount of data in a simpler way.
- c). Data Visualization makes complex data more accessible, understandable, and usable.
- d). All of the above

**9. Data can be visualized using?**

- a). graphs
- b). charts
- c). maps
- d). All of the above

**10. Which of the following is false?**

- a). data visualization include the ability to absorb information quickly
- b). Data visualization is another form of visual art
- c). Data visualization decrease the insights and take solwer decisions
- d). None Of the above

**11. Which of the following function provides unsupervised prediction?**

- a). cl\_forecast
- b). cl\_nowcast
- c). cl\_precast
- d). None of the Mentioned

**12. A model of language consists of the categories which does not include\_\_\_\_\_.**

- a). System Unit
- b). structural units.
- c). data units
- d). empirical units

**13. Which of the following are ML methods?**

- a). based on human supervision
- b). supervised Learning
- c). semi-reinforcement Learning
- d). All of the above

**14. Data Analytics uses \_\_\_\_ to get insights from data.**

- a).Statistical figures
- b).Numerical aspects
- c).Statistical methods
- d).None of the mentioned above

**15. Amongst which of the following is / are the types of Linear Regression,**

- a).Simple Linear Regression
- b).Multiple Linear Regression
- c).Both A and B
- d).None of the mentioned above

1.d	6.b	11.d
2.d	7.c	12.b
3.d	8.d	13.a
4.b	9.d	14.c
5.b	10.c	15.c

## **SUBJECTIVE TYPE QUESTIONS**

1. Differentiate Between Data Analytics and Data Science
2. How can you avoid overfitting your model?
3. How can you select k for k-means?
4. What is the significance of p-value?
5. How can you calculate accuracy using a confusion matrix?
7. Explain the steps in making a decision tree.
8. What are the feature selection methods used to select the right variables?
9. For the given points, how will you calculate the Euclidean distance in Python?
10. Which of the following machine learning algorithms can be used for inputting missing values of both categorical and continuous variables?
11. What is Data Science? List the differences between supervised and unsupervised learning.
12. What is Selection Bias?
13. What is a confusion matrix?
14. What do you understand by true positive rate and false-positive rate?
15. What are dimensionality reduction and its benefits?