Python EBook

By @Curious_.programmer

Chapter 1: Introduction

1.1 What Is Python?

Python is a programming language that is commonly used to create websites and applications, automate processes, and do data analysis. Python is a general- purpose programming language, which means it can be used to construct a wide range of applications and isn't specialized for any particular problem. Because of its flexibility and beginner-friendliness, it has become one of the most widely used programming languages today.

Python is a general purpose, dynamic, high-level, and interpreted programming language. It supports Object Oriented programming approach to develop applications. It is simple and easy to learn and provides lots of high-level data structures.

Python is easy to learn yet powerful and versatile scripting language, which makes it attractive for Application Development.
Python's syntax and dynamic typing with its interpreted nature make it an ideal language for scripting and rapid application development.

1.2 Why Python Is Popular?

Python's popularity is due to a variety of factors. Here's a closer look at what makes it so useful to programmers

It features a straightforward syntax that resembles normal English, making it easy to read and comprehend. This speeds up the development of projects as well as the improvement of existing ones.

It's adaptable. Python may be used for a variety of purposes, including web development and machine learning.

It's user-friendly, making it popular among new programmers.

It's free to use and distribute, even for commercial, because it's open source.

The Python module and library archive—bundles of code produced by third-party users to extend Python's capabilities—is large and expanding.

Python has a strong community that adds to the library of modules and libraries and serves as a resource for other programmers. Because of the large support network, finding a solution to a stumbling block is quite simple; someone has almost certainly encountered the same issue before.

1.3 What can be done using Python?

Python is widely used for web and software development, task automation, data analysis, and data visualization. Python has been embraced by many non- programmers, such as accountants and scientists, for a range of common activities, such as arranging money, due to its relative ease of learning.

Data analysis and machine Learning

Python has become a data science standard, allowing data analysts and other professionals to do complicated statistical computations, generate data visualization's, construct machine learning algorithms, manage and analyses data, and perform other data-related activities using the language.

Web development

Python is frequently used to create a website's or application's back end—the bits that the user does not see. Sending data to and from servers, processing data and connecting with databases, URL routing, and maintaining security are all things that Python can help with in web development. For web development, Python has a number of frameworks. Django and Flask are two popular frameworks

Automation or scripting

If you find yourself repeating a job, you may make it more efficient by automating it with Python. Scripting is the process of writing code to create these automated operations. Automation may be used in the coding industry to check for mistakes across many files, convert files, do simple math, and eliminate duplicates in data.

1.4 Why learn Python?

Python provides many useful features to the programmer. These features make it most popular and widely used language. We have listed below few-essential feature of Python.

- Interpreted Language
- Object-Oriented Language
- Open Source Language
- Extensible
- Learn Standard Library
- GUI Programming Support
- Integrated
- Embeddable
- Dynamic Memory Allocation

1.5 Python Installation

1.5.1 Downloading installing Python

1. Go to www.python.org/downloads/
2. Download Python as per your system requirement

Installing python on windows

1. Click on Python Releases for Windows, select the link for the latest Python3 Release – Python 3.x.x

2. Scroll to the bottom and select either Windows x86-64 executable Installer for 64-bit or Windows x86 executable installer for 32-bit

Installing python on Linux

1. Open the Ubuntu Software Center folder
2. Select Developer Tools from the All Software drop-down list box
3. Double-click the Python 3.3.4 entry
4. Click Install
5. Close the Ubuntu Software Center folder

1.6 Python First Program

Unlike the other programming languages, Python provides the facility to execute the code using few lines.

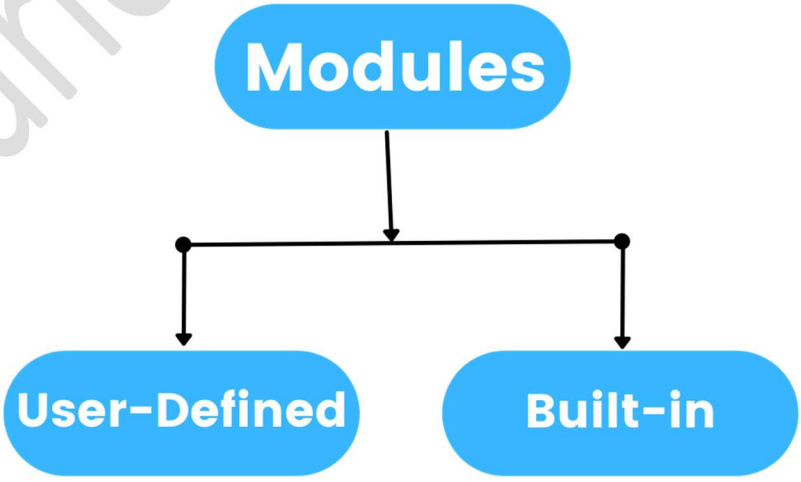
We can do this using one statement in Python.

Chapter 2 :
Modules, Commets,pip

2.1 Modules in Python

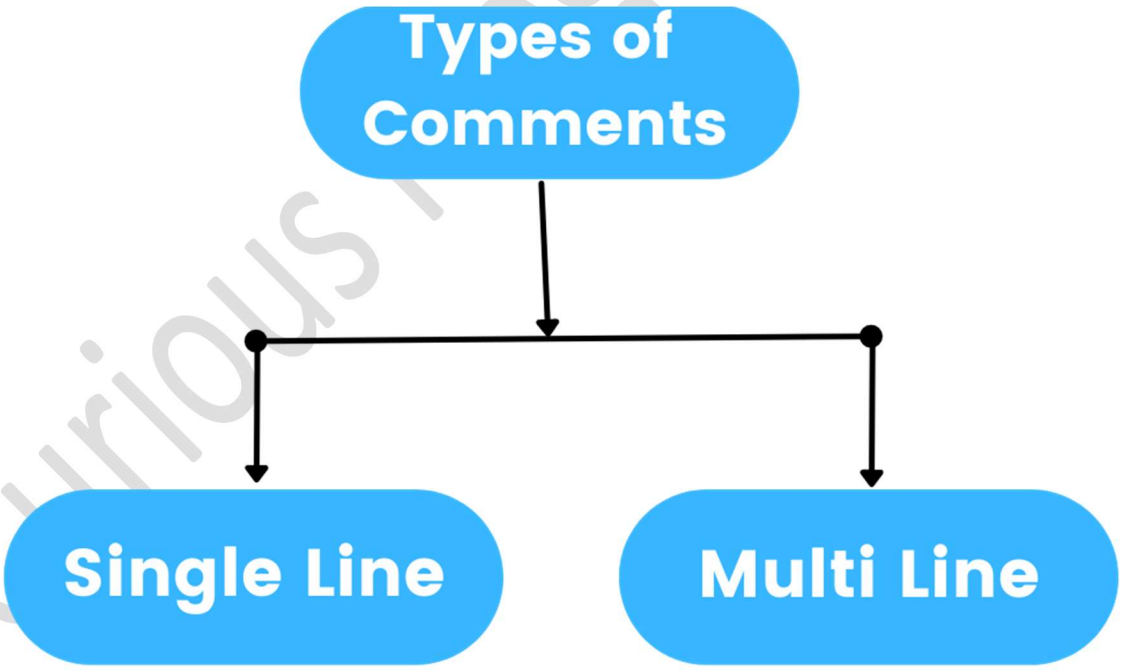
- A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables.

- A module can also include runnable code. Grouping related code into a module makes the code easier to understand and use. It also makes the code logically organized.



2.2 Comments in Python

- Python Comment is an essential tool for the programmers. Comments are generally used to explain the code.
- We can easily understand the code if it has a proper explanation.
- A good programmer must use the comments because in the future anyone wants to modify the code as well as implement the new module; then, it can be done easily.

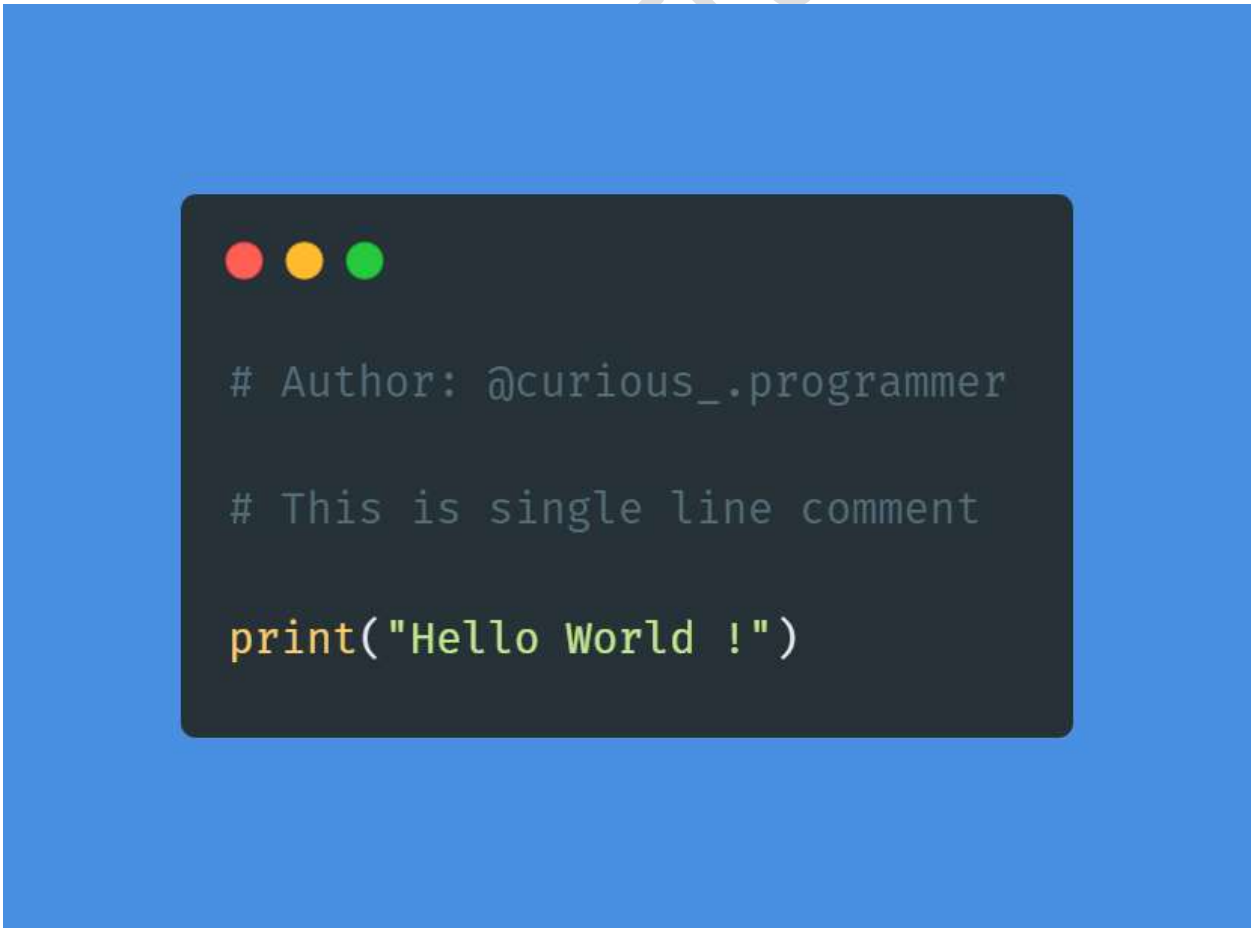


Single line comment

- Python single line comment starts with the hashtag symbol (#) with no white spaces and lasts till the end of the line.

- If the comment exceeds one line then put a hashtag on the next line and continue the comment

- Example:



Multiline Comments

- Python does not provide the option for multiline comments. However, there are different ways through which we can write multiline comments.

- Python ignores the string literals that are not assigned to a variable so we can use these string literals as a comment.

- Example:

2.3 What is pip?

- pip is a package-management system written in Python used to install and manage software packages.

- It connects to an online repository of public packages, called the Python Package Index.

- pip can also be configured to connect to other package repositories, provided that they comply to Python Enhancement Proposal 503.

Chapter 3: Variables, datatypes and Keywords

3.1 Variables in python

- Variable is a name that is used to refer to memory location. Python variable is also known as an identifier and used to hold value.

- In Python, we don't need to specify the type of variable because Python is a infer language and smart enough to get variable type.

- Variable names can be a group of both the letters and digits, but they have to begin with a letter or an underscore.

Example:

3.2 Identifier in python

Variables are the example of identifiers. An Identifier is used to identify the literals used in the program.

Rules:

1. The first character of the variable must be an alphabet or underscore (_).
2. All the characters except the first character may be an alphabet of lower-case (a-z), upper-case (A-Z), underscore, or digit (0-9).
3. Identifier name must not contain any white-space, or special character (!, @, #, %, ^, &, *).

4. Identifier name must not be similar to any keyword defined in the language.
5. Examples of valid identifiers: a123, _n, n_9, etc.
6. Examples of invalid identifiers: 1a, n%4, n 9, etc.

3.3 Data Types in python

- Variables can hold values, and every value has a data-type. Python is a dynamically typed language; hence we do not need to define the type of the variable while declaring it. The interpreter implicitly binds the value with its type.

```
a = 5
```

- The variable a holds integer value five and we did not define its type. Python interpreter will automatically interpret variables a as an integer type.

- Python enables us to check the type of the variable used in the program. Python provides us the type() function, which returns the type of the variable passed.

Python Data Types:

1) **Numbers:** Number stores numeric values. The integer, float, and complex values belong to a Python Numbers data-type.
   a) Python supports three types of numeric data-
      i) **Integer -** Integer value can be any length such as integers 10, 2, 29, -20, -150 etc.
      ii) **Float-** Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc.
      iii) **Complex number-** A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively

2) **Sequence Type:**
      i) **String:** The string can be defined as the sequence of characters represented in the quotation marks. In Python, we can use single, double, or triple quotes to define a string.
      ii) **List:** Python Lists are similar to arrays in C. However, the list can contain data of different types. The items stored in the list are separated with a comma (,) and enclosed within square brackets [].
      iii) **Tuple:** A tuple is similar to the list in many ways. Like lists, tuples also contain the collection of the items of different data types. The items of the tuple are

separated with a comma (,) and enclosed in parentheses ().

3) **Boolean**: Boolean type provides two built-in values, True and False. These values are used to determine the given statement true or false. It denotes by the class bool. True can be represented by any non-zero value or 'T' whereas false can be represented by the 0 or 'F'

4) **Set:** Python Set is the unordered collection of the data type. It is inerrable, mutable (can modify after creation), and has unique elements. In set, the order of the elements is undefined; it may return the changed sequence of the element.

5) **Dictionary:** Dictionary is an unordered set of a key-value pair of items. It is like an associative array or a hash table where each key stores a specific value. Key can hold any primitive data type, whereas value is an arbitrary Python object.

3.4 operator in Python

The operator can be defined as a symbol which is responsible for a particular operation between two operands. Python provides variety of operators, which are described as follows.

- Arithmetic operators
- Comparison operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Arithmetic operators

Arithmetic operators are used with numeric values to perform common mathematical operations

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Python Assignment Operators:

Assignment operators are used to assign values to variables:

Operator	**Example**	**Same As**
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3

>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Python Comparison Operators

Comparison operators are used to compare two values:

Operator	**Name**	**Example**
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python Logical Operators

Logical operators are used to combine conditional statements:

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python Identity Operators

Identity operators are used to compare the objects, not if they are equal, but if they are actually the same object, with the same memory location:

Operator	Description	Example
is	Returns True if both variables are the same object	x is y

is not	Returns True if both variables are not the same object	x is not y

Python Membership Operators

Membership operators are used to test if a sequence is presented in an object:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

Python Bitwise Operators

Operator	Name	Description
&	AND	Sets each bit to 1 if both bits are 1
|	OR	Sets each bit to 1 if one of two bits is 1

^	XOR	Sets each bit to 1 if only one of two bits is 1
~	NOT	Inverts all the bits
<<	Zero fill left shift	Shift left by pushing zeros in from the right and let the leftmost bits fall off
>>	Signed right shift	Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off

3.5 Input In Python

Python input() Function

Python input() function is used to get input from the user. It prompts for the user input and reads a line. After reading data, it converts it into a string and returns that. It throws an error EOFError if EOF is read.

Parameters

prompt: It is a string message which prompts for the user input

Return

It returns user input after converting into a string.
Let's see some examples of input() function to understand it's functionality.

Python input() Function Example

Input-

```
# Author: @curious_.programmer
# Python Input Function Example

val = input("Enter a value : ")
#Showing Result
print("You Entered: ",val)
```

Output:

```
Enter a value : 4
You Entered:  4
>
```

Chapter 4:
List,Tuples,set,Dictionary and Type Conversion

4.1 list In Python

Lists are just like dynamic sized arrays, declared in other languages (vector in C++ and ArrayList in Java).

Lists need not be homogeneous always which makes it a most powerful tool in Python.

A single list may contain Data Types like Integers, Strings, as well as Objects. Lists are mutable, and hence, they can be altered even after their creation.

A list can be defined as a collection of values or items of different types. The items in the list are separated with the comma (,) and enclosed with the square brackets [].

Characteristics of Lists

The list has the following characteristics:

- The lists are ordered.
- The element of the list can access by index.
- The lists are the mutable type.
- The lists are mutable types.
- A list can store the number of various elements.

A list can be define as below

```
# Author: @curious_.programmer
# A list can be Define as Below

L1 = ["Yadnyesh", 102, "INDIA"]
L2 = [1,2,3,4,5,6,7,8]
```

4.2 Tuples in Python

Python Tuple is used to store the sequence of immutable Python objects.

The tuple is similar to lists since the value of the items stored in the list can be changed, whereas the tuple is immutable, and the value of the items stored in the tuple cannot be changed.

Creating a tuple

A tuple can be written as the collection of comma-separated (,) values enclosed with the small () brackets. The parentheses are optional but it is good practice to use. A tuple can be defined as follows.

Example-

```
# Author: @curious_.programmer
# Tuples in Python

T1 = (101, "Yadnyesh", 19)
t2 = ("Apple", "banana", "cat")

T3 = 10, 20, 30, 40, 50
```

4.3 Set in Python

A Python set is the collection of the unordered items. Each element in the set must be unique, immutable, and the sets remove the duplicate elements. Sets are mutable which means we can modify it after its creation.

Unlike other collections in Python, there is no index attached to the elements of the set, i.e., we cannot directly access any element of the set by the index. However, we can print them all together, or we can get the list of elements by looping through the set.

Creating a set

The set can be created by enclosing the comma-separated immutable items with the curly braces {}. Python also provides the set() method, which can be used to create the set by the passed sequence.

Example 1: Using curly braces

```
# Author: @curious_.programmer
# Set In python using Curly Braces

Days = {"Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"}

print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

Example 2: Using set() method

```
# Author: @curious_.programmer
# Set In python using Set Method

Days = set(["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"])
print(Days)
print(type(Days))
print("looping through the set elements ... ")
for i in Days:
    print(i)
```

4.4 Dictionary in Python

Python Dictionary is used to store the data in a key-value pair format. The dictionary is the data type in Python, which can simulate the real-life data arrangement where some specific value exists for some particular key. It is the mutable data-structure. The dictionary is defined into element Keys and values.

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.

Creating the dictionary

The dictionary can be created by using multiple key-value pairs enclosed with the curly brackets {}, and each key is separated from its value by the colon (:).The syntax to define the dictionary is given below.

Syntax:

```
Dict = {"Name": "Tom", "Age": 22}
```

In the above dictionary Dict, The keys Name and Age are the string that is an immutable object.

Let's see an example to create a dictionary and print its content.

Example-

```
# Author: @curious_.programmer
# Dictionary in Python

Employee = {"Name": "John", "Age": 29,
"salary":25000,"Company":"GOOGLE"}
print(type(Employee))
print("printing Employee data .... ")
print(Employee)
```

Accessing the dictionary values

We have discussed how the data can be accessed in the list and tuple by using the indexing.

However, the values can be accessed in the dictionary by using the keys as keys are unique in the dictionary.

The dictionary values can be accessed in the following way.

Code:

```
Employee = {"Name": "YADNYESH", "Age": 29, "salary":25000}
print(type(Employee))
print("printing Employee data .... ")
print("Name : %s" %Employee["Name"])
print("Age : %d" %Employee["Age"])
print("Salary : %d" %Employee["salary"])
print("Company : %s" %Employee["Company"])
```

Output:

```
<class 'dict'>
printing Employee data ....
Name : YADNYESH
Age : 29
Salary : 25000
>
```

Adding dictionary values

The dictionary is a mutable data type, and its values can be updated by using the specific keys.

The value can be updated along with key Dict[key] = value. The update() method is also used to update an existing value.

Note: If the key-value already present in the dictionary, the value gets updated. Otherwise, the new keys added in the dictionary.

Let's see an example to update the dictionary values.

Programme

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Adding elements to dictionary one at a time
Dict[0] = 'Peter'
Dict[2] = 'Joseph'
Dict[3] = 'Ricky'
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Adding set of values
# with a single Key
# The Emp_ages doesn't exist to dictionary
Dict['Emp_ages'] = 20, 33, 24
print("\nDictionary after adding 3 elements: ")
print(Dict)

# Updating existing Key's Value
Dict[3] = 'Java'
print("\nUpdated key value: ")
print(Dict)
```

Output-

```
Empty Dictionary:
{}

Dictionary after adding 3 elements:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky'}

Dictionary after adding 3 elements:
{0: 'Peter', 2: 'Joseph', 3: 'Ricky', 'Emp_ages': (20, 33, 24)}

Updated key value:
{0: 'Peter', 2: 'Joseph', 3: 'Java', 'Emp_ages': (20, 33, 24)}
```

4.5 Python Type Casting

Type Casting is the method to convert the variable data type into a certain data type in order to the operation required to be performed by users. In this article, we will see the various technique for typecasting.

There can be two types of Type Casting in Python –

- Implicit Type Casting
- Explicit Type Casting

Implicit Type Casting

In this, methods, Python converts data type into another data type automatically. In this process, users don't have to involve in this process.

Program

```
# Python program to demonstrate

# implicit type Casting

# Python automatically converts

# a to int

a = 7
```

```
print(type(a))


# Python automatically converts

# b to float

b = 3.0

print(type(b))


# Python automatically converts

# c to float as it is a float addition

c = a + b

print(c)

print(type(c))


# Python automatically converts

# d to float as it is a float multiplication

d = a * b

print(d)

print(type(d))
```

Output:

Explicit Type Casting

In this method, Python need user involvement to convert the variable data type into certain data type in order to the operation required.

Mainly in type casting can be done with these data type function:

Int() : Int() function take float or string as an argument and return int type object.

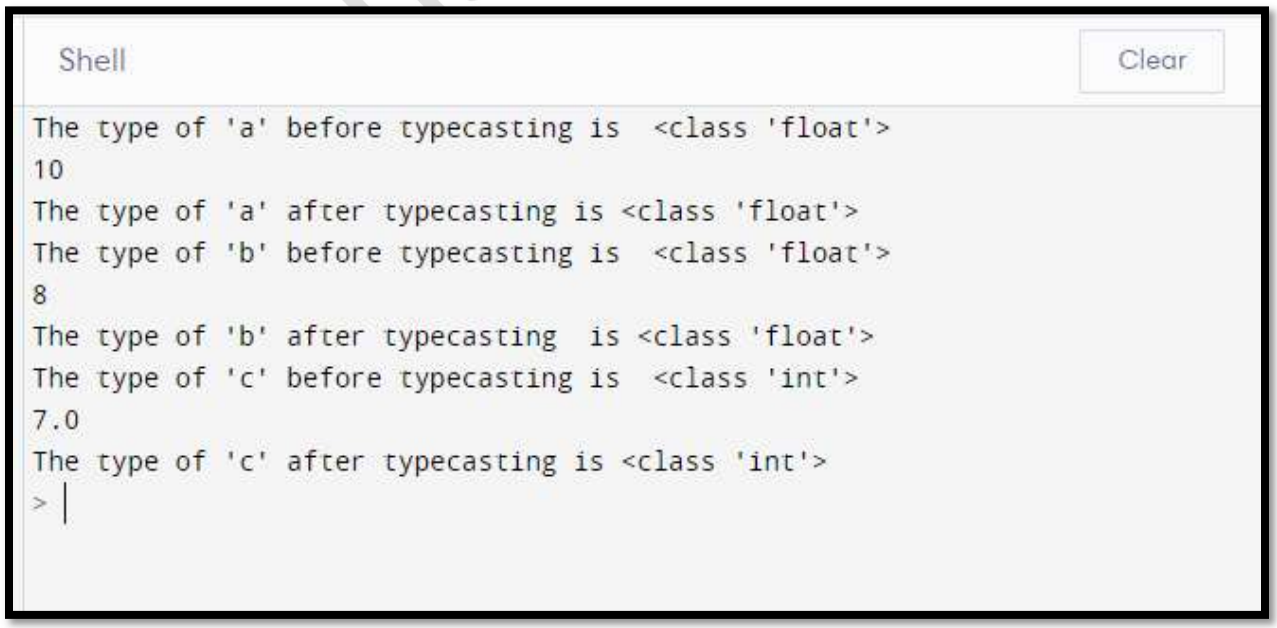
float() : float() function take int or string as an argument and return float type object.

str() : str() function take float or int as an argument and return string type object.

Program

```
#program to demonstrate explicit type conversion
#initializing the value of a
a=10.6
print("The type of 'a' before typecasting is ",type(a))
print(int(a))
print("The type of 'a' after typecasting is",type(a))
#initializing the value of b
b=8.3
print("The type of 'b' before typecasting is ",type(b))
print(int(b))
print("The type of 'b' after typecasting  is",type(b))
#initializing the value of c
c=7
print("The type of 'c' before typecasting is ",type(c))
print(float(c))
print("The type of 'c' after typecasting is",type(c))
```

Output:



Chapter 5: Flow Control

There comes situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. Decision-making statements in programming languages decide the direction of the flow of program execution

Statement	Description
If Statement	The if statement is used to test a specific condition. If the condition is true, a block of code (if-block) will be executed.
If - else Statement	The if-else statement is similar to if statement except the fact that, it also provides the block of the code for the false case of the condition to be checked. If the condition provided in the if statement is false, then the else statement will be executed.
Nested if Statement	Nested if statements enable us to use if ? else statement inside an outer if statement.

Indentation in Python

For the ease of programming and to achieve simplicity, python doesn't allow the use of parentheses for the block level code. In Python, indentation is used to declare a block.

If two statements are at the same indentation level, then they are the part of the same block.

Generally, four spaces are given to indent the statements which are a typical amount of indentation in python Indentation is the most used part of the python language since it declares the block of code.

All the statements of one block are intended at the same level indentation. We will see how the actual indentation takes place in decision making and other stuff in python.

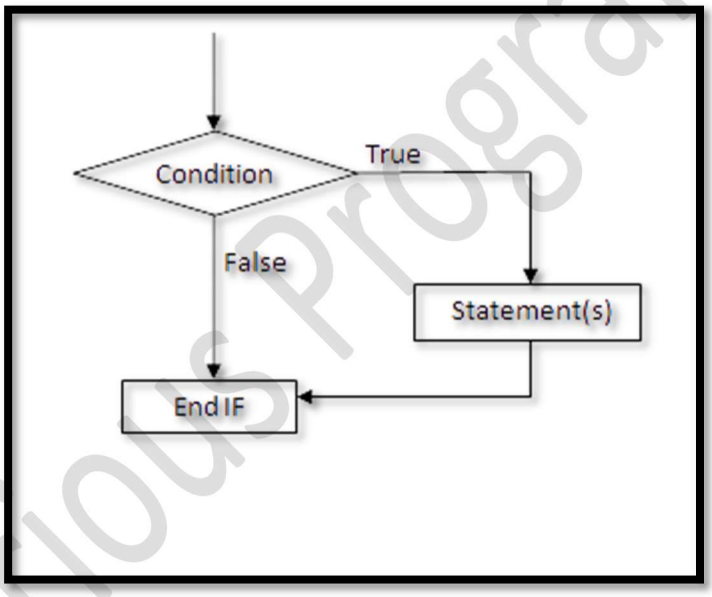
5.1 If Statement

The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.

Syntax:

```
# Author @Curious_.programmer
# Syntx of if
if expression:
  satatements
```

Flowchart:

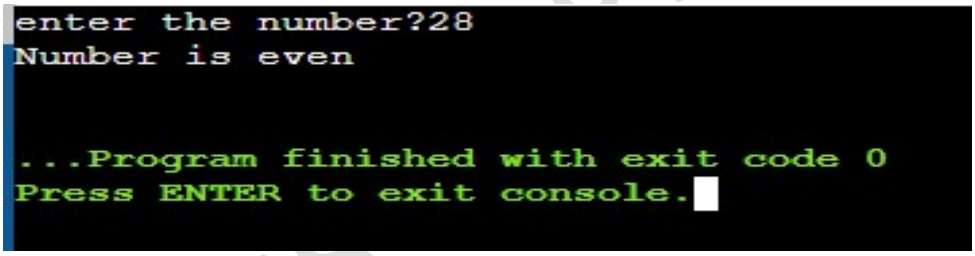


Example:

```
# Author @Curious_.programmer
# Example of if

num = int(input("Enter a Number : "))
if num%2 ==0:
  print("Number is Even
```

Output:



5.2 If-else Statement

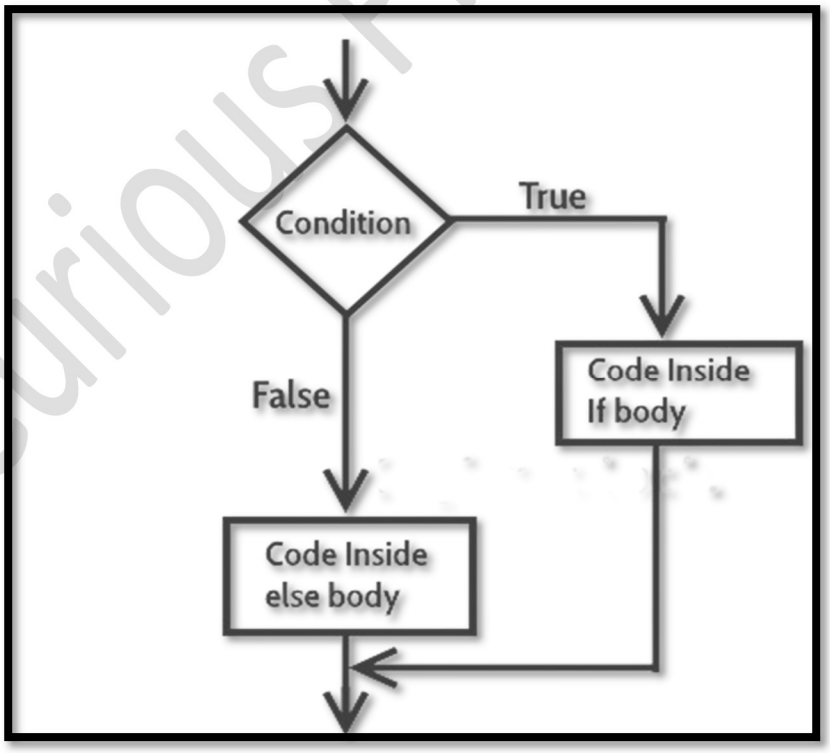
The if-else statement provides an else block combined with the if statement which is executed in the false case of the condition.

If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.

Syntax:

```
# Author @Curious_.programmer
# Syntx of if-else
if expression:
  # Block of code
else:
  # Another Block of Statement (else-Block)
```

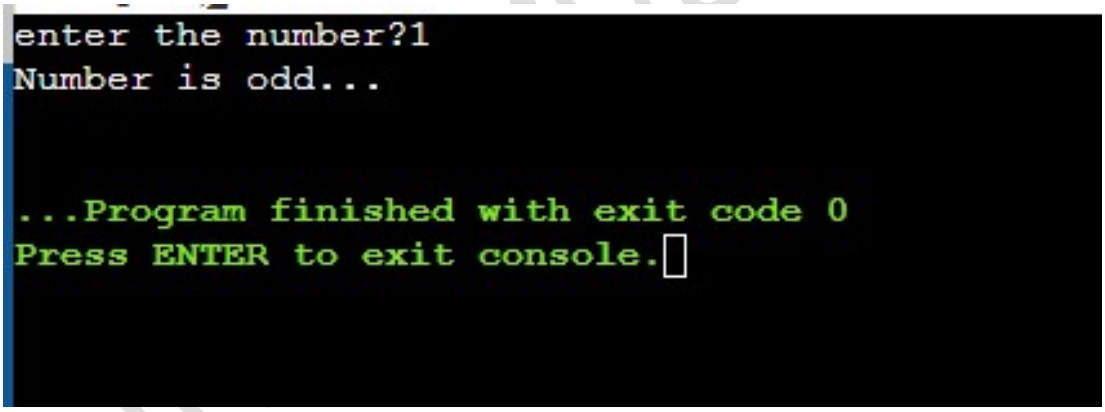
Flowchart:



Program

```python
# Author @Curious_Programmer
num = int(input("enter the number?"))
if num%2 == 0:
    print("Number is even ... ")
else:
    print("Number is odd ... ")
```

Output:



5.3 elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them.

We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

The elif statement works like an if-else-if ladder statement in C. It must be succeeded by an if statement.

Syntax

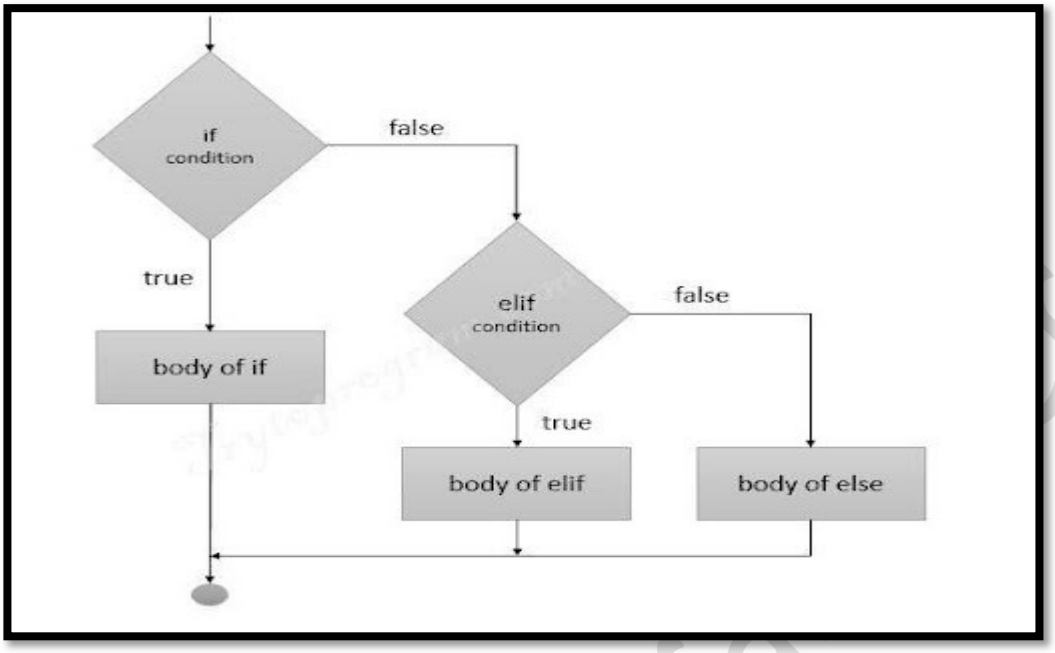
```
# Author @Curious_Programmer
if expression 1:
    # block of statements

elif expression 2:
    # block of statements

elif expression 3:
    # block of statements

else:
    # block of statements
```

Flowchart:



Program

```
# Author @Curious_Programmer
number = int(input("Enter the number?"))
if number==10:
    print("number is equals to 10")
elif number==50:
    print("number is equal to 50");
elif number==100:
    print("number is equal to 100");
else:
    print("number is not equal to 10, 50 or 100");
```

Output

```
Enter the number?10
number is equals to 10


...Program finished with exit code 0
Press ENTER to exit console.
```

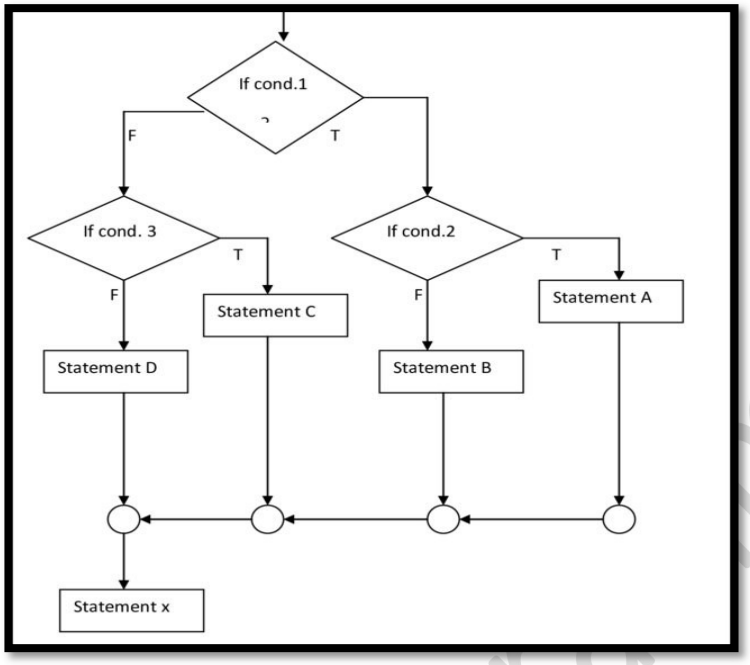
5.4 Nested If

A nested if is an if statement that is the target of another if statement. Nested if statements mean an if statement inside another if statement. Yes, Python allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1):
   # Executes when condition1 is true
   if (condition2):
      # Executes when condition2 is true
   # if Block is end here
# if Block is end here
```

Flowchart:



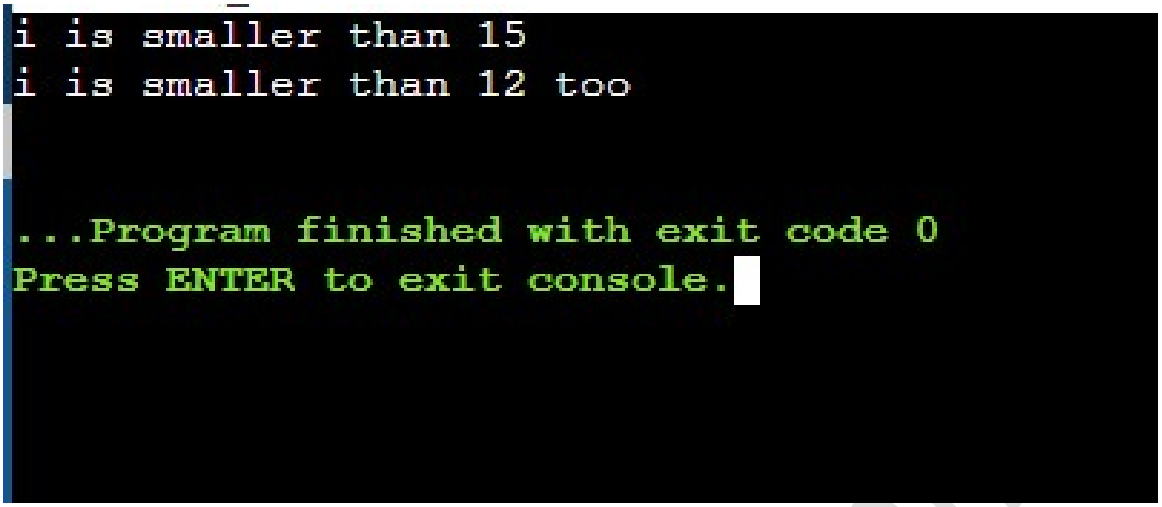
Program

```
# python program to illustrate nested Ifstatement
i = 10
if (i == 10):

    # First if statement
    if (i < 15):
        print("i is smaller than 15")

    # Nested - if statement
    # Will only be executed if statement above
    # it is true
    if (i < 12):
        print("i is smaller than 12 too")
    else:
        print("i is greater than 15")
```

Output:



Chapter 6: Loops

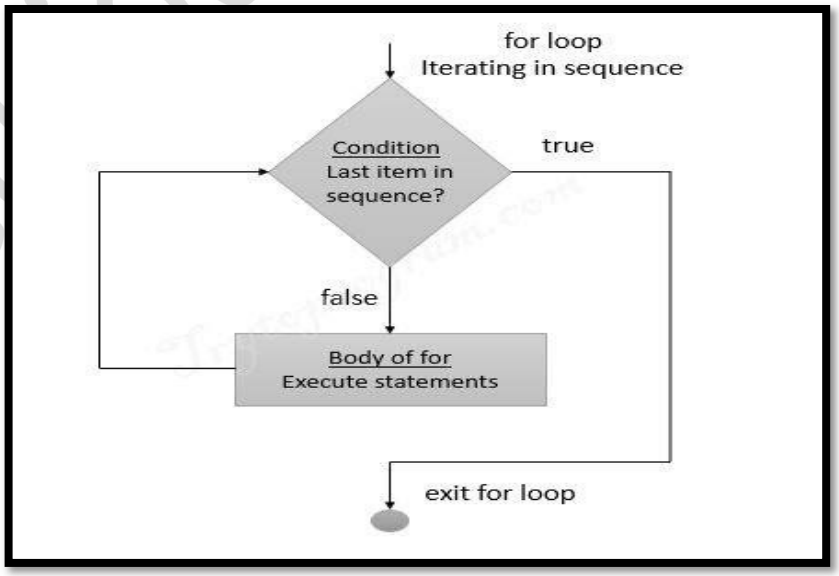
The for loop in Python is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like list, tuple, or dictionary.

The syntax of for loop in python is given below.

```
# Author @Curious_Programmer

for iterating_var in sequance:
  statement(S)
```

Flowchart:



For loop Using Sequence

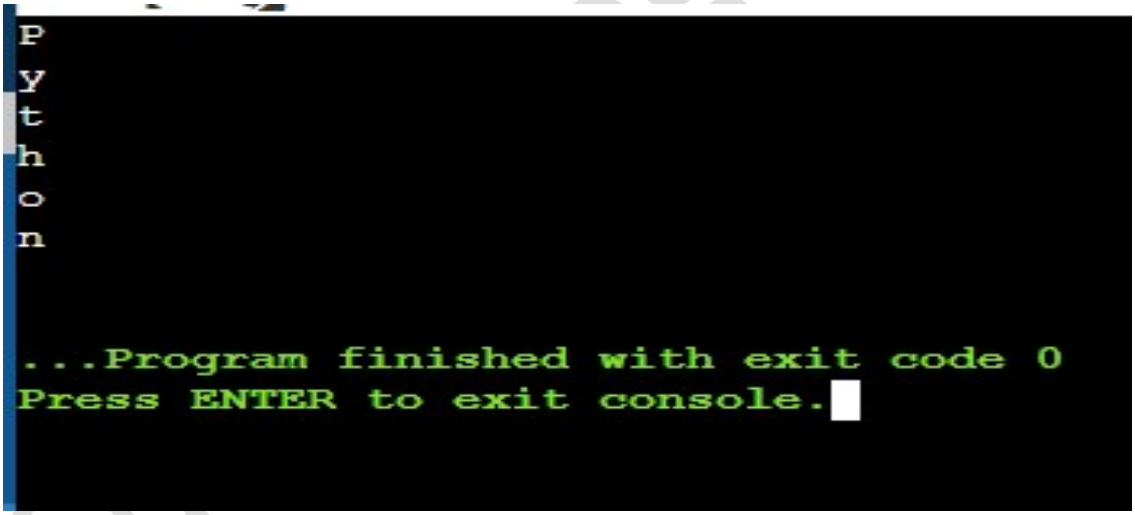
Example-1: Iterating string using for loop

Program

```python
# Author @Curious_Programmer

str = "python"
for i in str:
  print(i)
```

Output



Example- 2: Program to print the table of the given number.

Program:

```
# Author @Curious_Programmer

list = [1,2,3,4,5,6,7,8,9,10]
n = 5
for i in list:
  c = n*i
  print(c)
```

Output

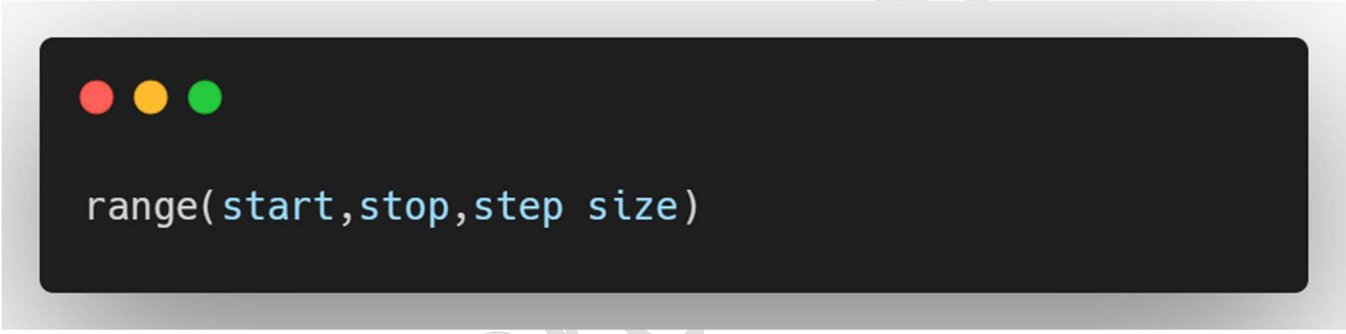
```
5
10
15
20
25
30
35
40
45
50


...Program finished with exit code 0
Press ENTER to exit console.
```

For loop Using range() function

The **range()** function is used to generate the sequence of the numbers. If we pass the range(10), it will generate the numbers from 0 to 9. The syntax of the range() function is given below.

Syntax:



- The start represents the beginning of the iteration.
- The stop represents that the loop will iterate till stop-1. The range(1,5) will generate numbers 1 to 4 iterations. It is optional.
- The step size is used to skip the specific numbers from the iteration. It is optional to use. By default, the step size is 1. It is optional.

Example

```
for i in range(10):
    print(i,end = ' ')
```

Output

```
0 1 2 3 4 5 6 7 8 9

...Program finished with exit code 0
Press ENTER to exit console.
```

Nested for loop

Python allows us to nest any number of for loops inside a **for** loop. The inner loop is executed n number of times for every iteration of the outer loop. The syntax is given below.

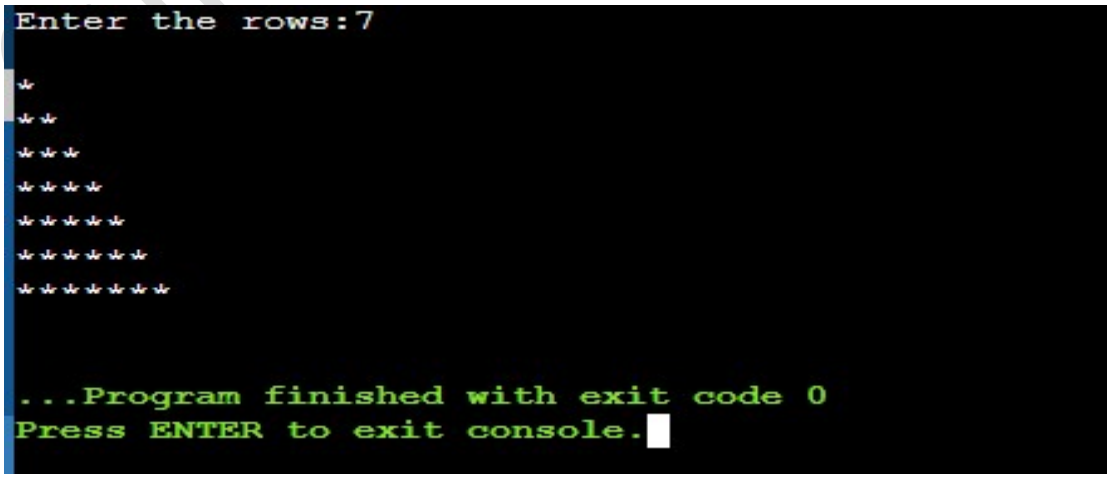
Syntax:

```
for iterating_var1 in sequence:  #outer loop
    for iterating_var2 in sequence:  #inner loop
        #block of statements
#Other statements
```

Example

```
# User input for number of rows
rows = int(input("Enter the rows:"))
# Outer loop will print number of rows
for i in range(0,rows+1):
# Inner loop will print number of Astrisk
    for j in range(i):
        print("*",end = '')
    print()
```

output:



Chapter 7: Strings

In Python, Strings are arrays of bytes representing Unicode characters. However, Python does not have a character data type, a single character is simply a string with a length of 1. Square brackets can be used to access elements of the string.

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's.

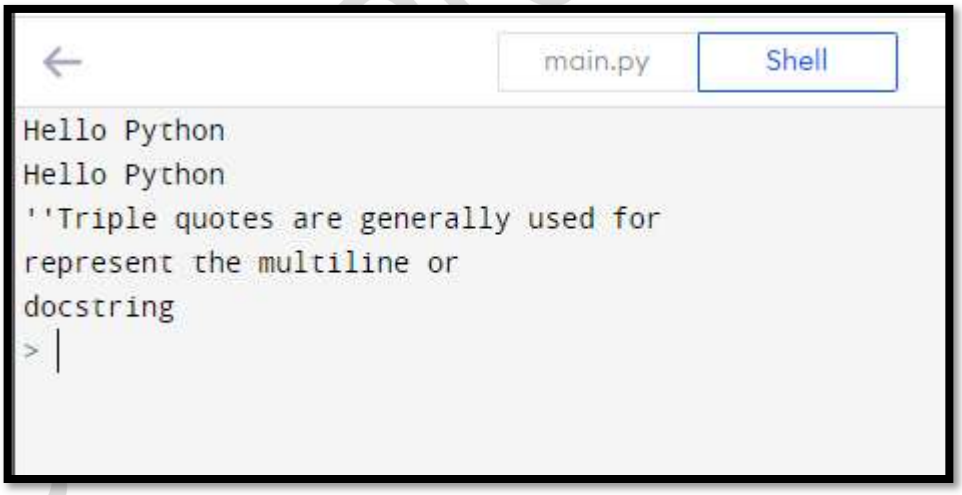
Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters.

7.1 Creating String in Python

```
#Using single quotes
str1 = 'Hello Python'
print(str1)
#Using double quotes
str2 = "Hello Python"
print(str2)

#Using triple quotes
str3 = '''Triple quotes are generally used for
        represent the multiline or
docstring'''
print(str3)
```

Output:



7.2 Python String Formatting

Escape Sequence

Let's suppose we need to write the text as - They said, "Hello what's going on?"- the given statement can be written in single quotes or double quotes but it will raise the **SyntaxError** as it contains both single and double-quotes.

Example:

```
1. str = "They said, "Hello what's going on?""
2. print(str)
```

Output:

```
SyntaxError: invalid syntax
```

We can use the triple quotes to accomplish this problem but Python provides the escape sequence.

The backslash(/) symbol denotes the escape sequence. The backslash can be followed by a special character and it interpreted differently. The single quotes inside the string must be escaped. We can apply the same as in the double quotes.

Example

```
# using triple quotes
print('''''They said, "What's there?"''')

# escaping single quotes
print('They said, "What\'s going on?"')

# escaping double quotes
print("They said, \"What's going on?\"")
```

Output:

```
''They said, "What's there?"
They said, "What's going on?"
They said, "What's going on?"


...Program finished with exit code 0
Press ENTER to exit console.
```

The list of an escape sequence is given below:

Sr.	Escape Sequence	Description	Example

1.	\newline	It ignores the new line.	`print("Python1 \`
			`Python2 \`
			`Python3")`
			Output:
			`Python1 Python2 Python3`

2.	\\	Backslash	`print("\\")`
			Output:
			`\`

3.	\'	Single Quotes	`print('\'')`
			Output:
			`'`

4.	\\''	Double Quotes	`print("\"")`
			Output:
			`"`

5.	\a	ASCII Bell	`print("\a")`

6.	\b	ASCII Backspace(BS)	`print("Hello \b World")`
			Output:
			`Hello World`

7.	\f	ASCII Formfeed	`print("Hello \f World!")` `Hello  World!`

8.	\n	ASCII Linefeed	`print("Hello \n World!")`
			Output:
			`Hello`
			` World!`

9.	\r	ASCII Carriege Return(CR)	`print("Hello \r World!")` **Output:**

			`World!`

10.	\t	ASCII Horizontal Tab	`print("Hello \t World!")`
			Output:
			`Hello  World!`

11.	\v	ASCII Vertical Tab	`print("Hello \v World!")`
			Output:
			`Hello`
			`World!`

12.	\ooo	Character with octal value	`print("\110\145\154\154\157")`
			Output:
			`Hello`

13	\xHH	Character with hex value.	`print("\x48\x65\x6c\x6c\x6f")`
			Output:
			`Hello`

7.3 The format() method

The **format()** method is the most flexible and useful method in formatting strings. The curly braces {} are used as the placeholder in the string and replaced by the **format()** method argument. Let's have a look at the given an example.

```
# Using Curly braces
print("{} and {} both are the best
friend".format("Devansh","Abhishek"))

#Positional Argument
print("{1} and {0} best players
".format("Virat","Rohit"))

#Keyword Argument
print("{a},{b},{c}".format(a = "James", b =
"Peter", c = "Ricky"))
```

Output:

```
Devansh and Abhishek both are the best friend
Rohit and Virat best players
James,Peter,Ricky


...Program finished with exit code 0
Press ENTER to exit console.
```

7.4 Python String functions

Method	Description
capitalize()	It capitalizes the first character of the String. This function is deprecated in python3

casefold()	It returns a version of s suitable for case-less comparisons.
center(width ,fillchar)	It returns a space padded string with the original string centred with equal number of left and right spaces.
count(string,begin,end)	It counts the number of occurrences of a substring in a String between begin and end index.
decode(encoding = 'UTF8', errors = 'strict')	Decodes the string using codec registered for encoding.
encode()	Encode S using the codec registered for encoding. Default encoding is 'utf-8'.
endswith(suffix ,begin=0,end=len(string))	It returns a Boolean value if the string terminates with given suffix between begin and end.
expandtabs(tabsize = 8)	It defines tabs in string to multiple spaces. The default space value is 8.
find(substring ,beginIndex, endIndex)	It returns the index value of the string where substring is found between begin index and end index.

format(value)	It returns a formatted version of S, using the passed value.
index(subsring, beginIndex, endIndex)	It throws an exception if string is not found. It works same as find() method.
isalnum()	It returns true if the characters in the string are alphanumeric i.e., alphabets or numbers and there is at least 1 character. Otherwise, it returns false.
isalpha()	It returns true if all the characters are alphabets and there is at least one character, otherwise False.
isdecimal()	It returns true if all the characters of the string are decimals.
isdigit()	It returns true if all the characters are digits and there is at least one character, otherwise False.
isidentifier()	It returns true if the string is the valid identifier.

Chapter 8: Functions

Functions are the most important aspect of an application. A function can be defined as the organized block of reusable code, which can be called whenever required.

Python allows us to divide a large program into the basic building blocks known as a function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the Python program.

The Function helps to programmer to break the program into the smaller part. It organizes the code very effectively and avoids the repetition of the code. As the program grows, function makes the program more organized.

Python provide us various inbuilt functions like range() or print(). Although, the user can create its functions, which can be called user-defined functions.
There are mainly two types of functions.

1. **User-define functions** - The user-defined functions are those define by the user to perform the specific task.
2. **Built-in functions** - The built-in functions are those functions that are pre-defined in Python.

Advantage of Functions in Python

- There are the following advantages of Python functions.
- Using functions, we can avoid rewriting the same logic/code again and again in a program.
- We can call Python functions multiple times in a program and anywhere in a program.
- We can track a large Python program easily when it is divided into multiple functions.
- Reusability is the main achievement of Python functions.
- However, Function calling is always overhead in a Python program.

8.1 Creating a Function

```
# Author @Curious_Programmer

def my_function(parametres):
  function_block
return expression
```

- Let's understand the syntax of functions definition.
- The def keyword, along with the function name is used to define the function.
- The identifier rule must follow the function name.
- A function accepts the parameter (argument), and they can be optional.
- The function block is started with the colon (:), and block statements must be at the same indentation.
- The return statement is used to return the value. A function can have only one return

8.2 Function Calling

In Python, after the function is created, we can call it from another function. A function must be defined before the function call; otherwise, the Python interpreter gives an error. To call the function, use the function name followed by the parentheses.

Example:

```
#function definition
def hello_world():
    print("hello world")
# function calling
hello_world()
```

8.3 The return statement

The return statement is used at the end of the function and returns the result of the function. It terminates the function execution and transfers the result where the function is called. The return statement cannot be used outside of the function

Syntax:

return [expression_list]

Example:

```
# Author @Curious_Programmer

def hello_world():
  print("hello World")
# function calling
hello_world()
```

8.4 Arguments in function

The arguments are types of information which can be passed into the function. The arguments are specified in the parentheses. We can pass any number of arguments, but they must be separate them with a comma.

Consider the following example, which contains a function that accepts a string as the argument.

Example

```
1. #defining the function
2. def func (name):
3.     print("Hi ",name)
4. #calling the function
5. func("yash")
```

types of argument

1. Required arguments
2. Keyword arguments
3. Default arguments
4. Variable-length arguments

8.5 Scope of variables

The scopes of the variables depend upon the location where the variable is being declared. The variable declared in one part of the program may not be accessible to the other parts.

In python, the variables are defined with the two types of scopes.

1. Global variables
2. Local variables

The variable defined outside any function is known to have a global scope, whereas the variable defined inside a function is known to have a local scope.

Example 1 Local Variable

```
def print_message():
    message = "hello !! I am going to print a message." # the
variable message is local to the function itself
    print(message)
print_message()
print(message) # this will cause an error since a local
variable cannot be accessible here.
```

Output:

```
hello !! I am going to print a message.
Traceback (most recent call last):
  File "main.py", line 5, in <module>
    print(message) # this will cause an error since a local variable cannot be accessible here.
NameError: name 'message' is not defined


...Program finished with exit code 0
Press ENTER to exit console.
```

Example 2 Global Variable

```python
def calculate(*args):
    sum=0
    for arg in args:
        sum = sum +arg
    print("The sum is",sum)
sum=0
calculate(10,20,30) #60 will be printed as the sum
print("Value of sum outside the function:",sum) # 0 will be
printed  Output:
```

Output:

```
The sum is 60
Value of sum outside the function: 0


...Program finished with exit code 0
Press ENTER to exit console.
```

THE ULTIMATE PYTHON HANDBOOK

CodeWithHarry

PREFACE

Welcome to the “Ultimate Python Programming Handbook," your comprehensive guide to mastering Python programming. This handbook is designed for beginners and anyone looking to strengthen their foundational knowledge of Python, a versatile and user-friendly programming language.

PURPOSE AND AUDIENCE

This handbook aims to make programming accessible and enjoyable for everyone. Whether you're a student new to coding, a professional seeking to enhance your skills, or an enthusiast exploring Python, this handbook will definitely be helpful. Python's simplicity and readability make it an ideal starting point for anyone interested in programming.

STRUCTURE AND CONTENT

The handbook is divided into clear, concise chapters, each focused on a specific aspect of Python:

- Fundamental Concepts: Start with the basics, such as installing Python and writing your first program.
- Practical Examples: Illustrative examples and sample code demonstrate the application of concepts.
- Hands-On Exercises: End-of-chapter exercises reinforce learning and build confidence.
- Additional Resources: References to official Python documentation for deeper exploration.

WHY PYTHON?

Python is known for its simplicity and readability, making it perfect for beginners. It is a high-level, interpreted language with a broad range of libraries and frameworks, supporting applications in web development, data analysis, AI, and more. Python's versatility and ease of use make it a valuable tool for both novice and experienced programmers.

ACKNOWLEDGEMENTS

I extend my gratitude to the educators, programmers, and contributors who have shared their knowledge and insights, shaping the content of this handbook. Special thanks to all the students watching my content on YouTube and Python community for maintaining a supportive and inspiring environment for learners worldwide.

CONCLUSION

Learning programming can be both exciting and challenging. The “Ultimate Python Programming Handbook" aims to make your journey smooth and rewarding. Watch my video along with following this handbook for optimal learning. Let this guide be your stepping stone to success in the world of programming.

CONTENTS

PYTHON PROGRAMMING HANDBOOK

WHAT IS PROGRAMMING?

Just like we use Hindi or English to communicate with each other, we use a programming language like Python to communicate with the computer.

Programming is a way to instruct the computer to perform various tasks.

WHAT IS PYTHON?

Python is a simple and easy to understand language which feels like reading simple English. This Pseudo code nature is easy to learn and understandable by beginners.

FEATURES OF PYTHON

- Easy to understand = Less development time
- Free and open source
- High level language
- Portable: Works on Linux / Windows / Mac.
- Fun to work with!

INSTALLATION

Python can be easily installed from python.org. When you click on the download button, python can be installed right after you complete the setup by executing the file for your platform.

CHAPTER 1 – MODULES, COMMENTS & PIP

Let's write our very first python program. Create a file called hello.py and paste the below code in it.

```
print("hello world") # print is a function (more later)
```

Execute this file (.py file) by typing python hello.py and you will see Hello World printed on the screen.

MODULES

A module is a file containing code written by somebody else (usually) which can be imported and used in our programs.

PIP

Pip is the package manager for python. You can use pip to install a module on your system.

```
pip install flask # Installs Flask Module
```

TYPES OF MODULES

There are two types of modules in Python.

1. Built in Modules (Preinstalled in Python)
2. External Modules (Need to install using pip)

Some examples of built in modules are os, random etc.

Some examples of external modules are tensorflow, flask etc.

USING PYTHON AS A CALCULATOR

We can use python as a calculator by typing "python" + ↵ on the terminal.

This opens **REPL** or Read Evaluate Print Loop.

COMMENTS

Comments are used to write something which the programmer does not want to execute. This can be used to mark author name, date etc.

TYPES OF COMMENTS

There are two types of comments in python.

1. Single Line Comments: To write a single line comment just add a '#' at the start of the line.

```
# This is a Single-Line Comment
```

2. Multiline Comments: To write multi-line comments you can use '#' at each line or you can use the multiline string (""" """)

```
"""This is an amazing
    example of a Multiline
    comment!"""
```

CHAPTER 1 – PRACTICE SET

1. Write a program to print Twinkle twinkle little star poem in python.
2. Use REPL and print the table of 5 using it.
3. Install an external module and use it to perform an operation of your interest.
4. Write a python program to print the contents of a directory using the os module. Search online for the function which does that.
5. Label the program written in problem 4 with comments.

CHAPTER 2 – VARIABLES AND DATATYPE

A variable is the name given to a memory location in a program. For example.

```
a= 30      # variables = container to store a value.
b= "harry" # keywords = reserved words in python
c= 71.22   # identifiers = class/function/variable name
```

DATA TYPES

Primarily these are the following data types in Python:

1. Integers
2. Floating point numbers
3. Strings
4. Booleans
5. None

Python is a fantastic language that automatically identifies the type of data for us.

```
a= 71          # identifies a as class <int>
b=88.44        # identifies b as class <float>
name= "harry" # identifies name as class <str>
```

RULES FOR CHOOSING AN IDENTIFIER

- A variable name can contain alphabets, digits, and underscores.
- A variable name can only start with an alphabet and underscores.
- A variable name can't start with a digit.
- No while space is allowed to be used inside a variable name.

Examples of a few variable names are: harry, one8, seven, _seven etc.

OPERATORS IN PYTHON

Following are some common operators in python:

1. Arithmetic operators: +, -, *, / etc.
2. Assignment operators: =, +=, -= etc.
3. Comparison operators: ==, >, >=, <, != etc.
4. Logical operators: and, or, not.

TYPE() FUNCTION AND TYPECASTING.

type() function is used to find the data type of a given variable in python.

```
a = 31
type(a) # class <int>

b = "31"
type (b) # class <str>
```

A number can be converted into a string and vice versa (if possible)

There are many functions to convert one data type into another.

```
str(31)    =>"31"   # integer to string conversion
int("32")  => 32    # string to integer conversion
float(32)  => 32.0  # integer to float conversion
```

... and so, on

Here "31" is a string literal and 31 a numeric literal.

INPUT () FUNCTION

This function allows the user to take input from the keyboard as a string.

```
A = input ("enter name")   # if a is "harry", the user entered harry
```

It is important to note that the output of input is always a string (even is a number is entered).

CHAPTER 2 – PRACTICE SET

1. Write a python program to add two numbers.
2. Write a python program to find remainder when a number is divided by z.
3. Check the type of variable assigned using input () function.
4. Use comparison operator to find out whether ‘a’ given variable a is greater than ‘b’ or not. Take a = 34 and b = 80
5. Write a python program to find an average of two numbers entered by the user.
6. Write a python program to calculate the square of a number entered by the user.

CHAPTER 3 – STRINGS

String is a data type in python.

String is a sequence of characters enclosed in quotes.

We can primarily write a string in these three ways.

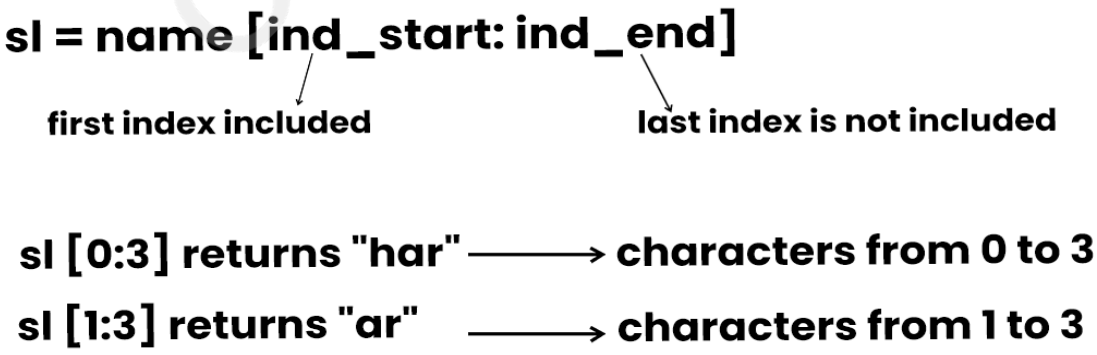
```
a ='harry'        # Single quoted string
b = "harry"       # Double quoted string
c = '''harry'''   # Triple quoted string
```

STRING SLICING

A string in python can be sliced for getting a part of the strings.

Consider the following string:

The index in a sting starts from 0 to (length -1) in Python. In order to slice a string, we use the following syntax:



Negative Indices: Negative indices can also be used as shown in the figure above. -1 corresponds to the (length - 1) index, -2 to (length - 2).

SLICING WITH SKIP VALUE

We can provide a skip value as a part of our slice like this:

```
word = "amazing"

word[1: 6: 2] # "mzn"
```

Other advanced slicing techniques:

```
Word = "amazing"
Word = [:7]  # word [0:7] – 'amazing'
Word = [0:]  # word [0:7] – 'amazing'
```

STRING FUNCTIONS

Some of the commonly used functions to perform operations on or manipulate strings are as follows. Let us assume there is a string ‘str’ as follows:

```
str = 'harry'
```

Now when operated on this string ‘str’, these functions do the following:

1. len () function – This function returns the length of the strings.

```
str = "harry"
print(len(str))  # Output: 5
```

2. String.endswith("rry") – This function_ tells whether the variable string ends with the string "rry" or not. If string is "harry", it returns true for "rry" since Harry ends with rry.

```
str = "harry"
print(str.endswith("rry"))  # Output: True
```

3. string.count("c") – counts the total number of occurrences of any character.

```
str = "harry"
count = str.count("r")
print(count)  # Output: 2
```

4. the first character of a given string.

```
str = "harry"
capitalized_string = str.capitalize()
print(capitalized_string)  # Output: "Harry"
```

5. string.find(word) – This function friends a word and returns the index of first occurrence of that word in the string.

```
str = "harry"
```

```
index = str.find("rr")
print(index)  # Output: 2
```

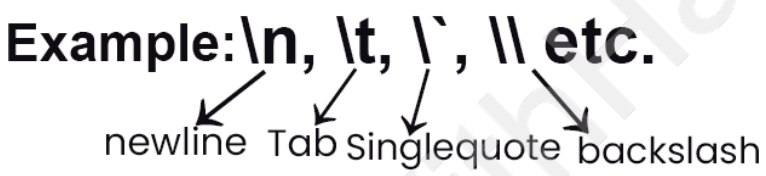
6. string.replace (old word, new word) – This function replace the old word with new word in the entire string.

```
str = "harry"
replaced_string = str.replace("r", "l")
print(replaced_string)  # Output: "hally"
```

ESCAPE SEQUENCE CHARACTERS

Sequence of characters after backslash "\" → Escape Sequence characters

Escape Sequence characters comprise of more than one character but represent one character when used within the strings.



CHAPTER 3 – PRACTICE SET

1. Write a python program to display a user entered name followed by Good Afternoon using input () function.
2. Write a program to fill in a letter template given below with name and date.

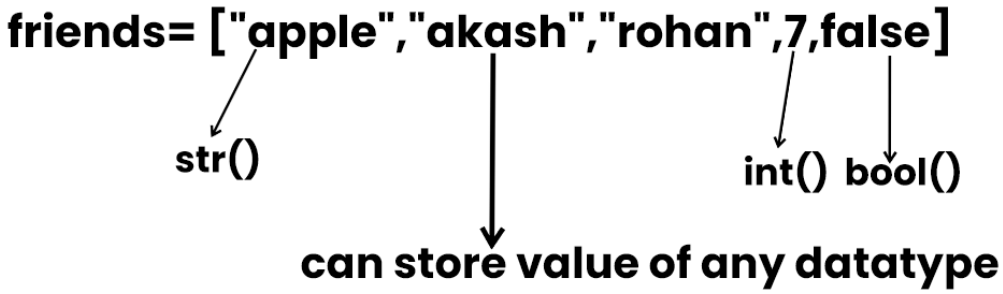
```
letter = '''
       Dear <|Name|>,
       You are selected!
       <|Date|>
        '''
```

3. Write a program to detect double space in a string.
4. Replace the double space from problem 3 with single spaces.
5. Write a program to format the following letter using escape sequence characters.

```
letter = "Dear Harry, this python course is nice. Thanks!"
```

CHAPTER 4 – LISTS AND TUPLES

Python lists are containers to store a set of values of any data type.



LIST INDEXING

A list can be indexed just like a string.

```
l1 = [7,9,"harry"]

l1[0] # 7
l1[1] # 9
l1[70]  # error
l1[0:2] # [7,9]  #list slicing
```

LIST METHODS.

Consider the following list:

```
l1 = [1,8,7,2,21,15]
```

- l1.sort(): updates the list to [1,2,7,8,15,21]
- l1.reverse(): updates the list to [15,21,2,7,8,1]
- l1.append(8): adds 8 at the end of the list
- l1.insert(3,8): This will add 8 at 3 index
- l1.pop(2): Will delete element at index 2 and return its value.
- l1.remove(21): Will remove 21 from the list.

TUPLES IN PYTHON

A tuple is an immutable data type in python.

```
a = () # empty tuple
a = (1,) # tuple with only one element needs a comma
a = (1,7,2) # tuple with more than one element
```

TUPLE METHODS

Consider the following tuple.

```
a = (1, 7, 2)
```

- a.count (1): a count (1) will return number of times 1 occurs in a.
- a.index (1) will return the index of first occurrence of 1 in a.

CHAPTER 4 - PRACTICE SET

1. Write a program to store seven fruits in a list entered by the user.
2. Write a program to accept marks of 6 students and display them in a sorted manner.
3. Check that a tuple type cannot be changed in python.
4. Write a program to sum a list with 4 numbers.
5. Write a program to count the number of zeros in the following tuple:

```
a = (7, 0, 8, 0, 0, 9)
```

CHAPTER 5 – DICTIONARY & SETS

Dictionary is a collection of keys-value pairs.

Syntax:

```
a = {
    "key": "value",
    "harry": "code",
    "marks": "100",
    "list": [1, 2, 9]
}

print(a["key"])  # Output: "value"
print(a["list"])  # Output: [1, 2, 9]
```

PROPERTIES OF PYTHON DICTIONARIES

1. It is unordered.
2. It is mutable.
3. It is indexed.
4. Cannot contain duplicate keys.

DICTIONARY METHODS

Consider the following dictionary.

```
a={"name":"harry"
   "from":"india"
   "marks":[92,98,96]}
```

- a.items(): Returns a list of (key,value)tuples.
- a.keys(): Returns a list containing dictionary's keys.
- a.update({"friends":}): Updates the dictionary with supplied key-value pairs.
- a.get("name"): Returns the value of the specified keys (and value is returned eg."harry" is returned here).

More methods are available on [docs.python.org]

SETS IN PYTHON.

Set is a collection of non-repetitive elements.

```
s = set()           # no repetition allowed!
s.add(1)
s.add(2)           # or set ={1,2}
```

If you are a programming beginner without much knowledge of mathematical operations on sets, you can simply look at sets in python as data types containing unique values.

PROPERTIES OF SETS

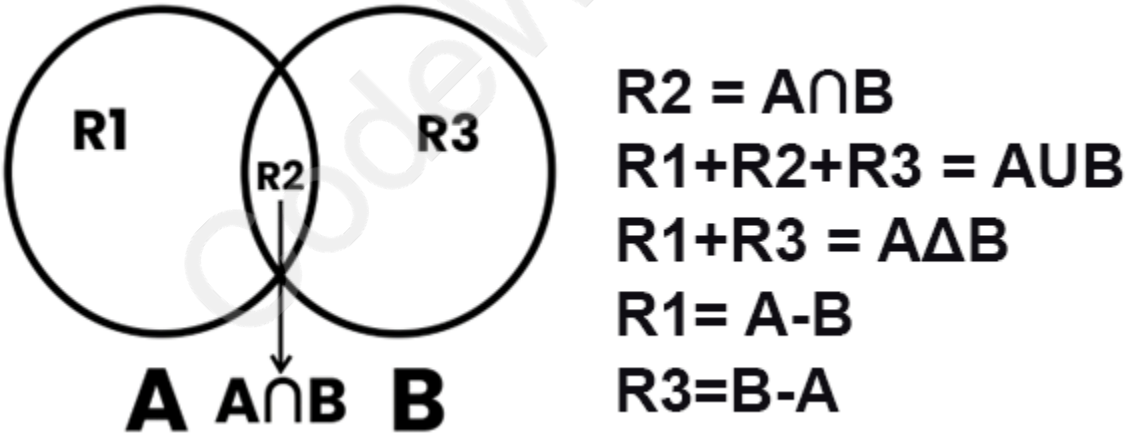
1. Sets are unordered => Element's order doesn't matter
2. Sets are unindexed => Cannot access elements by index
3. There is no way to change items in sets.
4. Sets cannot contain duplicate values.

OPERATIONS ON SETS

Consider the following set:

```
s = {1,8,2,3}
```

- len(s): Returns 4, the length of the set
- s.remove(8): Updates the set s and removes 8 from s.
- s.pop(): Removes an arbitrary element from the set and return the element removed.
- s.clear():empties the set s.
- s.union({8,11}): Returns a new set with all items from both sets. {1,8,2,3,11}.
- s.intersection({8,11}): Return a set which contains only item in both sets {8}.



CHAPTER 5 – PRACTICE SET

1. Write a program to create a dictionary of Hindi words with values as their English translation. Provide user with an option to look it up!
2. Write a program to input eight numbers from the user and display all the unique numbers (once).
3. Can we have a set with 18 (int) and `'18'` (str) as a value in it?
4. What will be the length of following set s:

```
s = set()
s.add(20)
s.add(20.0)
s.add('20') # length of s after these operations?
```

5. s = {}
   What is the type of `'s'`?
6. Create an empty dictionary. Allow 4 friends to enter their favorite language as value and use key as their names. Assume that the names are unique.
7. If the names of 2 friends are same; what will happen to the program in problem 6?
8. If languages of two friends are same; what will happen to the program in problem 6?
9. Can you change the values inside a list which is contained in set S?

```
s = {8, 7, 12, "Harry", [1,2]}
```

CHAPTER 6 – CONDITIONAL EXPRESSION

Sometimes we want to play PUBG on our phone if the day is Sunday.

Sometimes we order Ice Cream online if the day is sunny.

Sometimes we go hiking if our parents allow.

All these are decisions which depend on a condition being met.

In python programming too, we must be able to execute instructions on a condition(s) being met.

This is what conditionals are for!

IF ELSE AND ELIF IN PYTHON

If else and elif statements are a multiway decision taken by our program due to certain conditions in our code.

Syntax:

```
if (condition1): # if condition1 is True
    print ("yes")

elif(condition2): # if condition2 is True
    print("no")

else:             # otherwise
    print("maybe")
```

CODE EXAMPLE.

```
a=22
if(a>9):
    print("greater")
else:
    print("lesser")
```

Quick Quiz: Write a program to print yes when the age entered by the user is greater than or equal to 18.

RELATIONAL OPERATORS

Relational Operators are used to evaluate conditions inside the if statements. Some examples of relational operators are:

==: equals.

> =: greater than/ equal to.

< =: lesser than/ equal to.

LOGICAL OPERATORS

In python logical operators operate on conditional statements. For Example:

- and – true if both operands are true else false.
- or – true if at least one operand is true or else false.
- not – inverts true to false & false to true.

ELIF CLAUSE

elif in python means [else if]. An if statements can be chained together with a lot of these elif statements followed by an else statement.

```
if (condition1):
    #code
elif (condition2):   # this ladder will stop once a condition in an if or
elif is met.
    #code
elif(condition3):
    #code
else:
    #code
```


IMPORTANT NOTES:

1. There can be any number of elif statements.
2. Last else is executed only if all the conditions inside elifs fail.

CHAPTER 6 – PRACTICE SET

1. Write a program to find the greatest of four numbers entered by the user.
2. Write a program to find out whether a student has passed or failed if it requires a total of 40% and at least 33% in each subject to pass. Assume 3 subjects and take marks as an input from the user.
3. A spam comment is defined as a text containing following keywords:
   "Make a lot of money", "buy now", "subscribe this", "click this". Write a program to detect these spams.
4. Write a program to find whether a given username contains less than 10 characters or not.
5. Write a program which finds out whether a given name is present in a list or not.
6. Write a program to calculate the grade of a student from his marks from the following scheme:
   90 – 100 => Ex
   80 – 90 => A
   70 – 80 => B
   60 – 70 =>C
   50 – 60 => D
   <50 => F
7. Write a program to find out whether a given post is talking about "Harry" or not.

CHAPTER 7 – LOOPS IN PYTHON

Sometimes we want to repeat a set of statements in our program. For instance: Print 1 to 1000.

Loops make it easy for a programmer to tell the computer which set of instructions to repeat and how!

TYPES OF LOOPS IN PYTHON

Primarily there are two types of loops in python.

- while loops
- for loops

We will look into these one by one.

WHILE LOOP

Syntax:

```
while (condition): # The block keeps executing until the condition is true
    #Body of the loop
```

In while loops, the condition is checked first. If it evaluates to true, the body of the loop is executed otherwise not!

If the loop is entered, the process of [condition check & execution] is continued until the condition becomes False.

Quick Quiz: Write a program to print 1 to 50 using a while loop.

Example:

```
i = 0
while i < 5: # print "Harry" – 5 times!
    print("Harry")
    i = i + 1
```

Note: If the condition never become false, the loop keeps getting executed.

Quick Quiz: Write a program to print the content of a list using while loops.

FOR LOOP

A for loop is used to iterate through a sequence like list, tuple, or string [iterables]

Syntax:

```
l = [1, 7, 8]
for item in l:
    print(item) # prints 1, 7 and 8
```

RANGE FUNCTION IN PYTHON

The range() function in python is used to generate a sequence of number.

We can also specify the start, stop and step-size as follows:

```
range(start, stop, step_size)
# step_size is usually not used with range()
```

AN EXAMPLE DEMONSTRATING RANGE () FUNCTION.

```
for i in range(0,7): # range(7) can also be used.

    print(i) # prints 0 to 6
```

FOR LOOP WITH ELSE

An optional else can be used with a for loop if the code is to be executed when the loops exhausts.

Example:

```
l= [1,7,8]
for item in l:
    print(item)
else:
    print("done") # this is printed when the loop exhausts!
```

Output:

```
1
7
8
done
```

THE BREAK STATEMENT

'break' is used to come out of the loop when encountered. It instructs the program to – exit the loop now.

Example:

```
for i in range (0,80):
    print(i)     # this will print 0,1,2 and 3
    if i==3
        break
```

THE CONTINUE STATEMENT

'continue' is used to stop the current iteration of the loop and continue with the next one. It instructs the Program to "skip this iteration".

Example:

```
for i in range(4):
    print("printing")
    if i == 2:   # if i is 2, the iteration is skipped
        continue
    print(i)
```

PASS STATEMENT

pass is a null statement in python.

It instructs to "do nothing".

Example:

```
l = [1,7,8]
for item in l:
    pass          # without pass, the program will throw an error
```

CHAPTER 7 – PRACTICE SET

1. Write a program to print multiplication table of a given number using for loop.
2. Write a program to greet all the person names stored in a list ‘l’ and which starts with S.
   l = ["Harry", "Soham", "Sachin", "Rahul"]
3. Attempt problem 1 using while loop.
4. Write a program to find whether a given number is prime or not.
5. Write a program to find the sum of first n natural numbers using while loop.
6. Write a program to calculate the factorial of a given number using for loop.
7. Write a program to print the following star pattern.
   ```
    *
   ***
   ***** for n = 3
   ```
8. Write a program to print the following star pattern:
   ```
   *
   **
   ***   for n = 3
   ```
9. Write a program to print the following star pattern.
   ```
   * * *
   *   *   for n = 3
   * * *
   ```
10. Write a program to print multiplication table of n using for loops in reversed order.

CHAPTER 8 – FUNCTIONS & RECURSIONS

A function is a group of statements performing a specific task.

When a program gets bigger in size and its complexity grows, it gets difficult for a program to keep track on which piece of code is doing what!

A function can be reused by the programmer in a given program any number of

EXAMPLE AND SYNTAX OF A FUNCTION

The syntax of a function looks as follows:

```
def func1():
    print('hello')
```

This function can be called any number of times, anywhere in the program.

FUNCTION CALL

Whenever we want to call a function, we put the name of the function followed by parentheses as follows:

```
func1() # This is called function call.
```

FUNCTION DEFINITION

The part containing the exact set of instructions which are executed during the function call.

Quick Quiz: Write a program to greet a user with “Good day” using functions.

TYPES OF FUNCTIONS IN PYTHON

There are two types of functions in python:

- Built in functions (Already present in python)
- User defined functions (Defined by the user)

Examples of built in functions includes len(), print(), range() etc.

The func1() function we defined is an example of user defined function.

FUNCTIONS WITH ARGUMENTS

A function can accept some value it can work with. We can put these values in the parentheses.

A function can also return value as shown below:

```
def greet(name):
    gr = "hello"+ name
    return gr

a = greet ("harry")
# a will now contain "hello harry"
```

DEFAULT PARAMETER VALUE

We can have a value as default as default argument in a function.

If we specify name = “stranger” in the line containing def, this value is used when no argument is passed.

Example:

```
def greet(name = "stranger"):
    # function body
greet() # name will be "stranger" in function body (default)
greet("harry") # name will be "harry" in function body (passed)
```

RECURSION

Recursion is a function which calls itself.

It is used to directly use a mathematical formula as function.

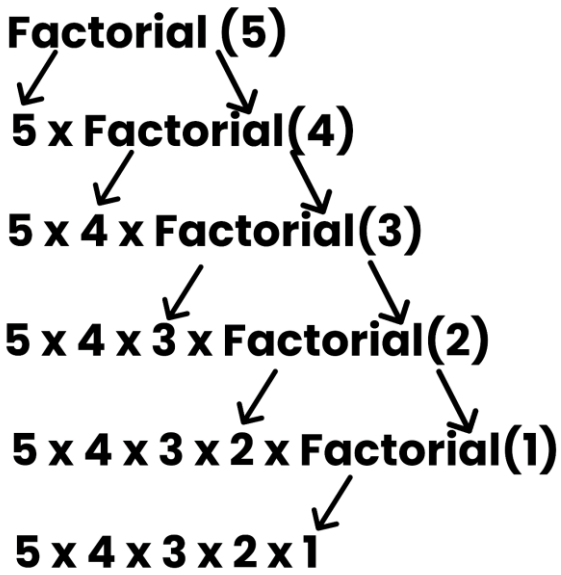
Example:

```
factorial(n) = n x factorial (n-1)
```

This function can be defined as follows:

```
def factorial(n)
    if i == 0 or i==1: # base condition which doesn’t call the function
any further
    return 1
else:
    return n*factorial(n-1) # function calling itself
```

This works as follows:



The programmer needs to be extremely careful while working with recursion to ensure that the function doesn't infinitely keep calling itself. Recursion is sometimes the most direct way to code an algorithm.

CHAPTER 8 – PRACTICE SET

1. Write a program using functions to find greatest of three numbers.
2. Write a python program using function to convert Celsius to Fahrenheit.
3. How do you prevent a python print() function to print a new line at the end.
4. Write a recursive function to calculate the sum of first n natural numbers.
5. Write a python function to print first n lines of the following pattern:

   ***
   ** - for n = 3
   *

6. Write a python function which converts inches to cms.
7. Write a python function to remove a given word from a list ad strip it at the same time.
8. Write a python function to print multiplication table of a given number.

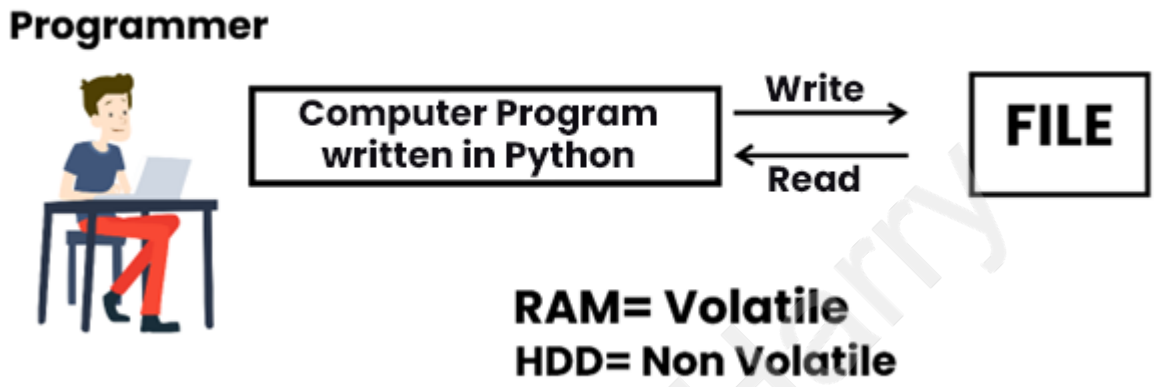
PROJECT 1: SNAKE, WATER, GUN GAME

We all have played snake, water gun game in our childhood. If you haven't, google the rules of this game and write a python program capable of playing this game with the user.

CHAPTER 9 – FILE I/O

The random-access memory is volatile, and all its contents are lost once a program terminates. In order to persist the data forever, we use files.

A file is data stored in a storage device. A python program can talk to the file by reading content from it and writing content to it.



TYPE OF FILES.

There are 2 types of files:

1. Text files (.txt, .c, etc)
2. Binary files (.jpg, .dat, etc)

Python has a lot of functions for reading, updating, and deleting files.

OPENING A FILE

Python has an open() function for opening files. It takes 2 parameters: filename and mode.

```
# open("filename", "mode of opening(read mode by default)")
open("this.txt", "r")
```

READING A FILE IN PYTHON

```
# Open the file in read mode
f = open("this.txt", "r")
# Read its contents
text = f.read()
# Print its contents
print(text)
```

```
# Close the file
f.close()
```

OTHER METHODS TO READ THE FILE.

We can also use f.readline() function to read one full line at a time.

```
f.readline() # Read one line from the file.
```

MODES OF OPENING A FILE

r – open for reading

w - open for writing

a - open for appending

+ - open for updating.

'rb' will open for read in binary mode.

'rt' will open for read in text mode.

WRITE FILES IN PYTHON

In order to write to a file, we first open it in write or append mode after which, we use the python's f.write() method to write to the file!

```
# Open the file in write mode
f = open("this.txt", "w")
# Write a string to the file
f.write("this is nice")
# Close the file
f.close()
```


WITH STATEMENT

The best way to open and close the file automatically is the with statement.

```
# Open the file in read mode using 'with', which automatically closes the
file
with open("this.txt", "r") as f:
    # Read the contents of the file
    text = f.read()

# Print the contents
print(text)
```

CHAPTER 9 – PRACTICE SET

1. Write a program to read the text from a given file ‘poems.txt’ and find out whether it contains the word ‘twinkle’.
2. The game() function in a program lets a user play a game and returns the score as an integer. You need to read a file ‘Hi-score.txt’ which is either blank or contains the previous Hi-score. You need to write a program to update the Hi-score whenever the game() function breaks the Hi-score.
3. Write a program to generate multiplication tables from 2 to 20 and write it to the different files. Place these files in a folder for a 13 – year old.
4. A file contains a word “Donkey” multiple times. You need to write a program which replace this word with ***#####*** by updating the same file.
5. Repeat program 4 for a list of such words to be censored.
6. Write a program to mine a log file and find out whether it contains ‘python’.
7. Write a program to find out the line number where python is present from ques 6.
8. Write a program to make a copy of a text file “this. txt”
9. Write a program to find out whether a file is identical & matches the content of another file.
10. Write a program to wipe out the content of a file using python.
11. Write a python program to rename a file to “renamed_by_ python.txt.

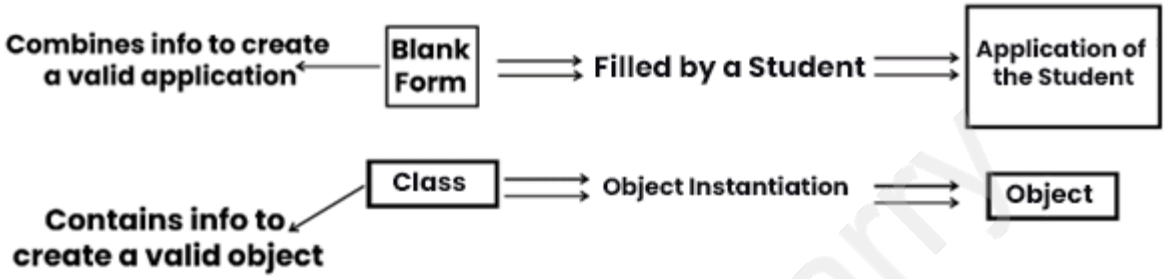
CHAPTER 10 - OBJECT ORIENTED PROGRAMMING

Solving a problem by creating object is one of the most popular approaches in programming. This is called object-oriented programming.

This concept focuses on using reusable code (DRY Principle).

CLASS

A class is a blueprint for creating object.



Syntax:

```
class Employee: # Class name is written in pascal case
    # Methods & Variables
```

OBJECT

An object is an instantiation of a class. When class is defined, a template (info) is defined. Memory is allocated only after object instantiation.

Objects of a given class can invoke the methods available to it without revealing the implementation details to the user. – Abstractions & Encapsulation!

MODELLING A PROBLEM IN OOPS

We identify the following in our problem.

- Noun → Class → Employee
- Adjective → Attributes → name, age, salary
- Verbs → Methods → getSalary(), increment()

CLASS ATTRIBUTES

An attribute that belongs to the class rather than a particular object.

Example:

```
class Employee:
    company = "Google" # Specific to Each Class
harry = Employee() # Object Instatiation
harry.company
Employee.company = "YouTube" # Changing Class Attribute
```

INSTANCE ATTRIBUTES

An attribute that belongs to the Instance (object). Assuming the class from the previous example:

```
harry.name = "harry"
harry.salary = "30k"  # Adding instance attribute
```

Note: Instance attributes, take preference over class attributes during assignment & retrieval.

When looking up for harry.attribute it checks for the following:

1) Is attribute present in object?

2) Is attribute present in class?

SELF PARAMETER

self refers to the instance of the class. It is automatically passed with a function call from an object.

```
harry.getSalary() # here self is harry
# equivalent to Employee.getSalary(harry)
```

The function getSalary() is defined as:

```
class Employee:
    company = "Google"
    def getSalary(self):
        print("Salary is not there")
```

STATIC METHOD

Sometimes we need a function that does not use the self-parameter. We can define a static method like this:

```
@staticmethod   # decorator to mark greet as a static method
def greet():
    print("Hello user")
```

__INIT__() CONSTRUCTOR

__init__() is a special method which is first run as soon as the object is created.

__init__() method is also known as constructor.

It takes 'self' argument and can also take further arguments.

For Example:

```
class Employee:
    def __init__(self, name):
        self.name=name
    def getSalary(self):
        ...

harry = Employee("Harry")
```

CHAPTER 10 – PRACTICE SET

1. Create a class *"Programmer"* for storing information of few programmers working at Microsoft.
2. Write a class *"Calculator"* capable of finding square, cube and square root of a number.
3. Create a class with a class attribute a; create an object from it and set 'a' directly using 'object.a = 0'. Does this change the class attribute?
4. Add a static method in problem 2, to greet the user with hello.
5. Write a Class 'Train' which has methods to book a ticket, get status (no of seats) and get fare information of train running under Indian Railways.
6. Can you change the self-parameter inside a class to something else (say "harry"). Try changing self to "slf" or "harry" and see the effects.

CHAPTER 11 - INHERITANCE & MORE ON OOPS

Inheritance is a way of creating a new class from an existing class.

Syntax:

```
class Employee:  # Base class
    # Code

class Programmer(Employee): # Derived or child class
    # Code
```

We can use the method and attributes of 'Employee' in 'Programmer' object.

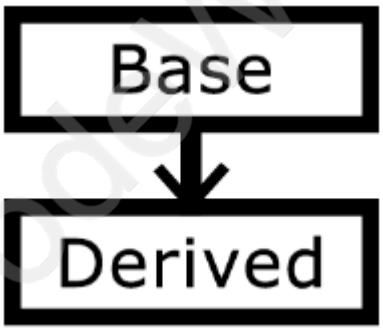
Also, we can overwrite or add new attributes and methods in 'Programmer' class.

TYPES OF INHERITANCE

- Single inheritance
- Multiple inheritance
- Multilevel inheritance

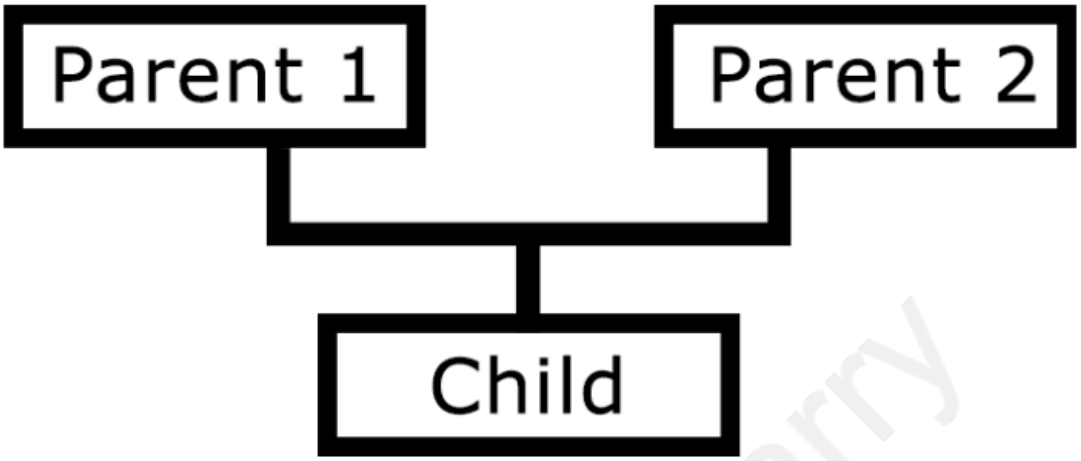
SINGLE INHERITANCE

Single inheritance occurs when child class inherits only a single parent class.



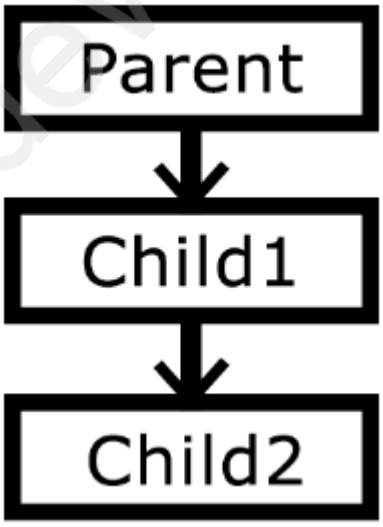
MULTIPLE INHERITANCE

Multiple Inheritance occurs when the child class inherits from more than one parent classes.



MULTILEVEL INHERITANCE

When a child class becomes a parent for another child class.



SUPER() METHOD

super() method is used to access the methods of a super class in the derived class.

```
super().__init__()
# __init__() Calls constructor of the base class
```

CLASS METHOD

A class method is a method which is bound to the class and not the object of the class.

@classmethod decorator is used to create a class method.

Syntax:

```
@classmethod
    def(cls,p1,p2):
```

@PROPERTY DECORATORS

Consider the following class:

```
class Employee:
    @property
    def name(self):
        return self.ename
```

If e = Employee() is an object of class employee, we can print (e.name) to print the ename by internally calling name() function.

@.GETTERS AND @.SETTERS

The method name with '@property' decorator is called getter method.

We can define a function + @ name.setter decorator like below:

```
@name.setter
def name (self,value):
    self.ename = value
```

OPERATOR OVERLOADING IN PYTHON

Operators in Python can be overloaded using dunder methods.

These methods are called when a given operator is used on the objects.

Operators in Python can be overloaded using the following methods:

```
p1+p2 # p1.__add__(p2)
p1-p2 # p1.__sub__(p2)
p1*p2 # p1.__mul__(p2)
p1/p2 # p1.__truediv__(p2)
p1//p2 # p1.__floordiv__(p2)
```

Other dunder/magic methods in Python:

```
str__() # used to set what gets displayed upon calling str(obj)
```

```
__len__() # used to set what gets displayed upon calling.__len__() or
len(obj)
```

CHAPTER 11- PRACTICE SET

1. Create a class (2-D vector) and use it to create another class representing a 3-D vector.

2. Create a class ‘Pets’ from a class ‘Animals’ and further create a class ‘Dog’ from ‘Pets’. Add a method ‘bark’ to class ‘Dog’.

3. Create a class ‘Employee’ and add salary and increment properties to it.

Write a method ‘salaryAfterIncrement’ method with a @property decorator with a setter which changes the value of increment based on the salary.

4. Write a class ‘Complex’ to represent complex numbers, along with overloaded operators ‘+’ and ‘*’ which adds and multiplies them.

5. Write a class vector representing a vector of n dimensions. Overload the + and * operator which calculates the sum and the dot(.) product of them.

6. Write __str__() method to print the vector as follows:

```
7i + 8j +10k
```

Assume vector of dimension 3 for this problem.

7. Override the __len__() method on vector of problem 5 to display the dimension of the vector.

PROJECT 2 – THE PERFECT GUESS

We are going to write a program that generates a random number and asks the user to guess it.

If the player's guess is higher than the actual number, the program displays "Lower number please". Similarly, if the user's guess is too low, the program prints "higher number please" When the user guesses the correct number, the program displays the number of guesses the player used to arrive at the number.

Hint: *Use the random module*.

CHAPTER 12 – ADVANCED PYTHON 1

NEWLY ADDED FEATURES IN PYTHON

Following are some of the newly added features in Python programming language

WALRUS OPERATOR

The walrus operator (`:=`), introduced in Python 3.8, allows you to assign values to variables as part of an expression. This operator, named for its resemblance to the eyes and tusks of a walrus, is officially called the "assignment expression."

```
# Using walrus operator
if (n := len([1, 2, 3, 4, 5])) > 3:
    print(f"List is too long ({n} elements, expected <= 3)")

# Output: List is too long (5 elements, expected <= 3)
```

In this example, `n` is assigned the value of `len([1, 2, 3, 4, 5])` and then used in the comparison within the `if` statement.

TYPES DEFINITIONS IN PYTHON

Type hints are added using the colon (`:`) syntax for variables and the `->` syntax for function return types.

```
# Variable type hint
age: int = 25

# Function type hints
def greeting(name: str) -> str:
    return f"Hello, {name}!"

# Usage
print(greeting("Alice"))  # Output: Hello, Alice!
```

ADVANCED TYPE HINTS

Python's typing module provides more advanced type hints, such as List, Tuple, Dict, and Union.

You can import List, Tuple and Dict types from the typing module like this:

```
from typing import List, Tuple, Dict, Union
```

The syntax of types looks something like this:

```
from typing import List, Tuple, Dict, Union

# List of integers
numbers: List[int] = [1, 2, 3, 4, 5]

# Tuple of a string and an integer
person: Tuple[str, int] = ("Alice", 30)

# Dictionary with string keys and integer values
scores: Dict[str, int] = {"Alice": 90, "Bob": 85}

# Union type for variables that can hold multiple types
identifier: Union[int, str] = "ID123"
identifier = 12345  # Also valid
```

These annotations help in making the code self-documenting and allow developers to understand the data structures used at a glance.

MATCH CASE

Python 3.10 introduced the match statement, which is similar to the switch statement found in other programming languages.

The basic syntax of the match statement involves matching a variable against several cases using the case keyword.

```
def http_status(status):
    match status:
        case 200:
            return "OK"
        case 404:
            return "Not Found"
        case 500:
            return "Internal Server Error"
        case _:
            return "Unknown status"

# Usage
print(http_status(200))  # Output: OK
print(http_status(404))  # Output: Not Found
print(http_status(500))  # Output: Internal Server Error
print(http_status(403))  # Output: Unknown status
```

DICTIONARY MERGE & UPDATE OPERATORS

New operators `|` and `|=` allow for merging and updating dictionaries.

```
dict1 = {'a': 1, 'b': 2}
dict2 = {'b': 3, 'c': 4}
merged = dict1 | dict2
print(merged)  # Output: {'a': 1, 'b': 3, 'c': 4}
```

You can now use multiple context managers in a single `with` statement more cleanly using the parenthesised context manager

```
with (
    open('file1.txt') as f1,
    open('file2.txt') as f2
):
    # Process files
```

EXCEPTION HANDLING IN PYTHON

There are many built-in exceptions which are raised in python when something goes wrong.

Exception in python can be handled using a try statement. The code that handles the exception is written in the except clause.

```
try:
   # Code which might throw exception
except Exception as e:
    print(e)
```

When the exception is handled, the code flow continues without program interruption.

We can also specify the exception to catch like below:

```
try:
    # Code
except ZeroDivisionError:
    # Code
except TypeError:
    # Code
except:
    # Code       # All other exceptions are handled here.
```

RAISING EXCEPTIONS

We can raise custom exceptions using the ‘raise’ keyword in python.

TRY WITH ELSE CLAUSE

Sometimes we want to run a piece of code when try was successful.

```
try:
    # Somecode
except:
    # Somecode
else:
    # Code          # This is executed only if the try was successful
```

TRY WITH FINALLY

Python offers a 'finally' clause which ensures execution of a piece of code inspective of the exception.

```
try:
    # Some Code
except:
    # Some Code
finally:
    # Some Code           # Executed regardless of error!
```

IF __NAME__== '__MAIN__' IN PYTHON

'__name__' evaluates to the name of the module in python from where the program is ran.

If the module is being run directly from the command line, the ' __name__' is set to string "__main__". Thus, this behaviour is used to check whether the module is run directly or imported to another file.

THE GLOBAL KEYWORD

'global' keyword is used to modify the variable outside of the current scope.

ENUMERATE FUNCTION IN PYTHON

The 'enumerate' function adds counter to an iterable and returns it

```
for i,item in list1:
    print(i,item)  # Prints the items of list 1 with index
```

LIST COMPREHENSIONS

List Comprehension is an elegant way to create lists based on existing lists.

```
list1 = [1,7,12,11,22,]
list2 = [i for item in list 1 if item > 8]
```

CHAPTER 12 – PRACTICE SET

1. Write a program to open three files 1.txt, 2.txt and 3.txt if any these files are not present, a message without exiting the program must be printed prompting the same.

2. Write a program to print third, fifth and seventh element from a list using enumerate function.

3. Write a list comprehension to print a list which contains the multiplication table of a user entered number.

4. Write a program to display a/b where a and b are integers. If b=0, display infinite by handling the 'ZeroDivisionError'.

5. Store the multiplication tables generated in problem 3 in a file named Tables.txt.

CHAPTER 13 – ADVANCED PYTHON 2

VIRTUAL ENVIRIONMENT

An environment which is same as the system interpreter but is isolated from the other Python environments on the system.

INSTALLATION

To use virtual environments, we write:

```
pip install virtualenv  # Install the package
```

We create a new environment using:

```
virtualenv myprojectenv # Creates a new venv
```

The next step after creating the virtual environment is to activate it.

We can now use this virtual environment as a separate Python installation.

PIP FREEZE COMMAND

'pip freeze' returns all the package installed in a given python environment along with the versions.

```
pip freeze > requirements .txt
```

The above command creates a file named 'requirements.txt' in the same directory containing the output of 'pip freeze'.

We can distribute this file to other users, and they can recreate the same environment using:

```
pip install -r requirements.txt
```

LAMBDA FUNCTIONS

Function created using an expression using 'lambda' keyword.

Syntax:

```
lambda arguments:expressions

# can be used as a normal function
```

Example:

```
square = lambda x:x*x
square(6)  # returns 36
sum = lambda a,b,c:a+b+c
```

```
sum(1,2,3) # returns 6
```

JOIN METHOD (STRINGS)

Creates a string from iterable objects.

```
l = ["apple", "mango", "banana"]
result = ", and, ".join(l)
print(result)
```

The above line will return “apple,and,mango,and,banana”.

FORMAT METHOD (STRINGS)

Formats the values inside the string into a desired output.

```
template.format(p1,p2...)
```

Syntax:

```
"{} is a good {}".format("harry", "boy")   #1.
"{} is a good {o}".format("harry", "boy")  #2.

# output for 1:
# harry is a good boy

# output for 2:
# boy is a good harry
```

MAP, FILTER & REDUCE

Map applies a function to all the items in an input_list.

Syntax.

```
map(function, input_list)
              # the function can be lambda function
```

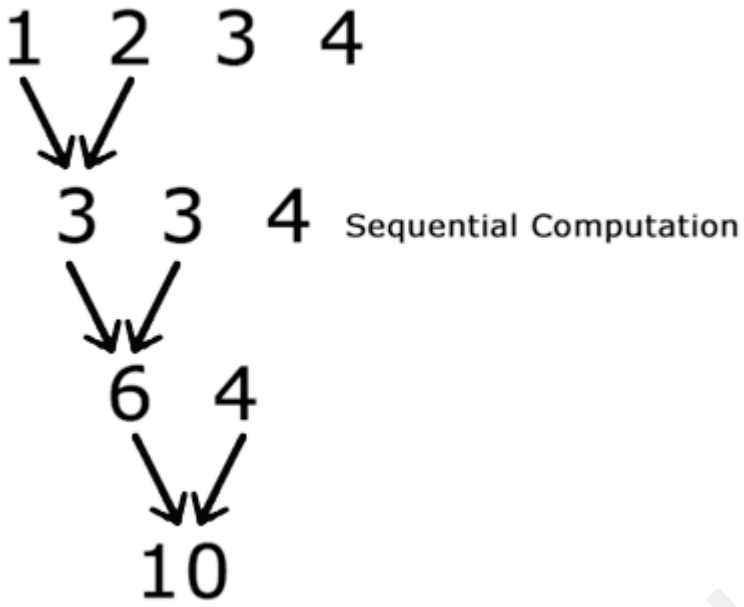
Filter creates a list of items for which the function returns true.

```
list(filter(function))
              # the function can be lambda function
```

Reduce applies a rolling computation to sequential pair of elements.

```
from functools import reduce
val=reduce (function, list1)
              # the function can be lambda function
```

If the function computes sum of two numbers and the list is [1,2,3,4]



CHAPTER 13- PRACTICE SET

1. Create two virtual environments, install few packages in the first one. How do you create a similar environment in the second one?

2. Write a program to input name, marks and phone number of a student and format it using the format function like below:

“The name of the student is Harry, his marks are 72 and phone number is 99999888”

3. A list contains the multiplication table of 7. write a program to convert it to vertical string of same numbers.

4. Write a program to filter a list of numbers which are divisible by 5.

5. Write a program to find the maximum of the numbers in a list using the reduce function.

6. Run pip freeze for the system interpreter. Take the contents and create a similar virtualenv.

7. Explore the ‘Flask’ module and create a web server using Flask & Python.

MEGA PROJECT 1: JARVIS - VOICE-ACTIVATED VIRTUAL ASSISTANT

Jarvis is a voice-activated virtual assistant designed to perform tasks such as web browsing, playing music, fetching news, and responding to user queries using OpenAI's GPT-3.5-turbo model.

FEATURES

- Voice Recognition
- Utilizes the speech_recognition library to listen for and recognize voice commands.
- Activates upon detecting the wake word "Jarvis."
- Text-to-Speech
- Converts text to speech using pyttsx3 for local conversion.
- Uses gTTS (Google Text-to-Speech) and pygame for playback.
- Web Browsing.
- Opens websites like Google, Facebook, YouTube, and LinkedIn based on voice commands.
- Music Playback
- Interfaces with a musicLibrary module to play songs via web links.
- News Fetching
- Fetches and reads the latest news headlines using NewsAPI.
- OpenAI Integration
- Handles complex queries and generates responses using OpenAI's GPT-3.5-turbo.
- Acts as a general virtual assistant similar to Alexa or Google Assistant.
- Activates upon detecting the wake word "Jarvis."
- Text-to-Speech

WORKFLOW

1. Initialization
2. Greets the user with "Initializing Jarvis...."
3. Wake Word Detection
4. Listens for the wake word "Jarvis."
5. Acknowledges activation by saying "Ya."
6. Command Processing.
7. Processes commands to determine actions such as opening a website, playing music, fetching news, or generating a response via OpenAI.
8. Speech Output.
9. Provides responses using speak function with either pyttsx3 or gTTS.
10. Greets the user with "Initializing Jarvis...."
11. Wake Word Detection
12. Acknowledges activation by saying "Ya."

13. Processes commands to determine actions such as opening a website, playing music, fetching news, or generating a response via OpenAI.

LIBRARIES USED

- speech_recognition
- webbrowser
- pyttsx3
- musicLibrary
- requests
- openai
- gTTS
- pygame
- os

MEGA PROJECT 2: AUTO-REPLY AI CHATBOT

DESCRIPTION

This project automates the process of interacting with a chat application, specifically designed to analyze chat history and generate humorous responses using OpenAI's GPT-3.5-turbo model. The virtual assistant, named Naruto, is a character that roasts people in a funny way, based on the chat history.

FEATURES

14. Automated Chat Interaction
15. Uses pyautogui to perform mouse and keyboard operations, interacting with the chat application without manual intervention.
16. Chat History Analysis
17. Copies chat history from the chat application and analyzes it to determine if the last message was sent by a specific user (e.g., "Rohan Das").
18. Humorous Response Generation
19. Integrates with OpenAI's GPT-3.5-turbo model to generate funny, roast-style responses based on the analyzed chat history.
20. Clipboard Operations
21. Utilizes pyperclip to copy and paste text, facilitating the retrieval and insertion of chat messages.
22. Uses pyautogui to perform mouse and keyboard operations, interacting with the chat application without manual intervention.
23. Copies chat history from the chat application and analyzes it to determine if the last message was sent by a specific user (e.g., "Rohan Das").
24. Humorous Response Generation
25. Integrates with OpenAI's GPT-3.5-turbo model to generate funny, roast-style responses based on the analyzed chat history.

WORKFLOW

- Initialization and Setup
- Click on the Chrome icon to open the chat application.
- Wait for a brief period to ensure the application is open and ready for interaction.
- Chat History Retrieval
- Periodically select and copy chat history by dragging the mouse over the chat area and using the copy shortcut.
- Retrieve the copied text from the clipboard.
- Message Analysis

- Analyze the copied chat history to check if the last message is from a specific user (e.g., "Rohan Das").
- If the last message is from the target user, send the chat history to OpenAI's GPT-3.5-turbo to generate a humorous response.
- Copy the generated response to the clipboard.
- Send Response
- Click on the chat input area and paste the generated response.
- Press 'Enter' to send the response.
- Wait for a brief period to ensure the application is open and ready for interaction.
- Chat History Retrieval
- Retrieve the copied text from the clipboard.
- Message Analysis
- Analyze the copied chat history to check if the last message is from a specific user (e.g., "Rohan Das").
- Generate Response
- Copy the generated response to the clipboard.
- Send Response

LIBRARIES USED

1. pyautogui: For automating mouse and keyboard interactions.
2. time: For adding delays between operations.
3. pyperclip: For clipboard operations.
4. openai: For interacting with OpenAI's GPT-3.5-turbo model.