Python

Chapter 1

Introduction to Python

1. What is Python?

Python is a dynamically typed, General Purpose Programming Language that supports an object-oriented programming approach as well as a functional programming approach.
Python is also an interpreted and high-level programming language.
It was created by Guido Van Rossum in 1989.

2. Features of Python:

- Python is simple and easy to understand.
- It is Interpreted and platform-independent which makes debugging very easy.
- Python is an open-source programming language.

3. What is Python used for

- Python is a programming language that is used in AI and machine learning to emulate human behavior and learn from prior data without using hard coding.
- Python is used to create web applications.
- It is frequently used for data analysis and manipulation.
- It is sometimes used in game development, often with the help of libraries like Pygame.

Python First Program

```
print("Hello, World!")
```

Output

```
Hello World!
```

Python Comments

Comments are used to explain Python code and it can make the code more readable and understandable. Comments are completely ignored and not executed by code editors.

Types of Comments:

There are two types of comments.

- Single-Line Comments
- Multi-Line Comments

Single-Line Comments:

Single-line comments start with the hash symbol (#).

Example:

```
#This is a single line comment
print("Hello World!")
```

Multi-Line Comments:

To write multi-line comments you can use (#) at each line.

Example:

```
#This is a
#multi line
#comment
print("Hello World!")
```

Chapter 2

Python Variables

Variables are containers that store information that can be manipulated and referenced later by the programmer within the code.

Example:

```
name = "john"   #type str
age = 22        #type int
```

Rules for Naming Variables

- Variable name must start with a letter or the underscore character
- Variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variables are case sensitive.
- Variable name cannot start with a number.

Example:

```
Country = "india"           #valid variable name
country = "australia"       #valid variable name
_country = "japan"          #valid variable name

5country = "singapore"      #invalid variable name
$country = "russia"         #invalid variable name
```

Local Variable:

A local variable is defined inside a function and can only be utilized within that function.

Example:

```
def my_func():
    fruit = "Orange"
    print(fruit + " is a local variable.")

my_func()
```

Output

```
Orange is a local variable.
```

Global Variable:

A global variable is created in the main body of the code and can be used anywhere within the code.

Example:

```
fruit = "Orange"

def my_func():
    print(fruit + " is a global variable.")

my_func()
```

Output

```
Orange is a global variable.
```

Python Data Types

Data types in Python represent the types of values that a variable can hold. Python supports various built-in data types, including:

Numeric data Types

- **int:** 4, -6, 0
- **float:** 3.14, 2.0.
- **complex** 5 + 3i

Text data Type

- **str:** “Hello Python”

Boolean data Type

- Boolean data consists of True or False values.

Sequenced data Types

list: A list is an ordered collection of data elements separated by a comma and enclosed within square brackets.

Example:

```
list1 = ["Orange", "Mango", "Strawberry"]
print(list1)
```

Output:

```
['Orange', 'Mango', 'Strawberry']
```

tuple:A tuple is an ordered collection of data elements separated by a comma and enclosed within parentheses.

Example:

```
tuple1 = ("Microsoft", "Google", "Facebook")
print(tuple1)
```

Output:

```
('Microsoft', 'Google', 'Facebook')
```

Mapped data:

dict: A dictionary is an unordered collection of data containing a key:value pair.

Example:

```
dict1 = {"name":"Rahul", "age":22}
print(dict1)
```

Output:

```
{'name': 'Rahul', 'age': 22}
```

Set data type:

Set is an unordered collection of unique items. The elements of sets are separated by commas and enclosed in curly braces.

Example:

```
set1 = {4, 8, 12, 5.2}
print(set1)
```

Output:

```
{4, 8, 12, 5.2}
```

Python Numbers

In Python, numerical data types are classified into three types:

- int
- float
- complex

int

Integers are whole numbers, either positive or negative, with no decimal points.

```
x = 24
y = 100548

print(type(x))
print(type(y))
```

Output:

```
<class 'int'>
<class 'int'>
```

Float

Floating-point numbers are real numbers with a decimal point.

```
x = 2.34
y = 3.1

print(type(x))
print(type(y))
```

Output:

```
<class 'float'>
<class 'float'>
```

complex

Complex numbers are made up of both real and imaginary numbers.

```
x = 6j
y = -6j

print(type(x))
print(type(y))
```

Output:

```
<class 'complex'>
<class 'complex'>
```

Chapter 3
Python Operators

Python provides a variety of operators for performing operations on variables and values.

Here's a list of different types of Python operators:

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

Arithmetic Operators

Arithmetic Operators are used to perform mathematical operations.

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

Assignment Operators

Assignment operators are used to assign values to variables.

Operator	**Example**
=	a = 5;
+=	a += 5;

-=	a -= 3;
*=	a *= 2;
/=	a /= 2;
%=	a %= 2;
//=	a //= 2;

Comparison Operators

Comparison operators are used to compare two values.

Operator	Name	Example
==	Equal	a==b
!=	Not equal	a!=b
>	Greater than	a>b
>=	Greater than or equal to	a>=b
<	Less than	a<>b
<=	Less than or equal to	a<=b

Logical Operators

Logical operators perform logical operations and return a boolean value.

Operator	Name	Example
&&	AND	x && y
||	OR	x || y
!	NOT	!x

Bitwise Operators

Operator	Name	Example
&	Bitwise AND	a & b
|	Bitwise OR	a | b
~	Bitwise NOT	~a
<<	Left shift	b<<

Operator Precedence in Python

Name	Operator
Parenthesis	()
Exponential	**
Multiply, divide, modulus, floor division	*, /, %, //
Addition, subtraction	+, -
Left shift and right shift operators	<<, >>
Bitwise and	&
Bitwise or and xor	^, |
Comparison operators	<, >, >=, <=
Assignment operators	=, %=, /=, //=, -=, +=, *= , **=
Logical operators	and, or, not

Chapter 4

Python Strings

Python strings are a sequence of characters that are enclosed by double quotes ("") or single quotes (' ').

```
# Double Quotes
str1 = "Hello Python"

# Single quotes
str2 = 'Hello Python'
```

Python String Operations

Compare Two Strings

We use the == operator to compare two strings.

Example:

```
str1 = "Hello Python"
str2 = "I love Python"
str3 = "Hello Python"

print(str1 == str2)
print(str1 == str3)
```

Output:

```
False
True
```

String Concatenation

To concatenate, two or more strings you can use the + operator.

Example:

```
str1 = "Hello"
str2 = " World"

result = str1 + str2
print(result)
```

Output:

```
Hello World
```

String Length

To find the length of a string, use the len() function.

Example:

```
str1 = "Hello Python"
print(len(str1))
```

Output:

```
12
```

String Methods

JavaScript has several built-in methods for manipulating strings.

upper()

The upper() method converts a string to upper case.

Example:

```
str1 = "Hello Python"
print(str1.upper())
```

Output:

```
HELLO PYTHON
```

lower()

The lower() method converts a string to upper case.

Example:

```
str1 = "Hello Python"
print(str1.lower())
```

Output:

```
hello python
```

strip()

The strip() method removes all white spaces before and after the string.

Example:

```
str1 = " Hello Python "
print(str1.strip)
```

Output:

```
Hello Python
```

replace

The replace() method replaces a string with another string.

Example:

```
str1 = "Hello Python"
print(str1.replace("Python", "World"))
```

Output:

```
Hello World
```

Chapter 5

Python Lists

A list is an ordered collection of data elements separated by a comma and enclosed within square brackets. They store multiple items in a single variable.

Example:

```
list1 = [20, 40, 60]
print(list1)
```

Output

```
[20, 40, 60]
```

Add List Items

There are three ways to add items to a list: append(), insert(), extend().

append()

To add an item to the end of the list, use the append() method.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]

flowers.append("Blossom")

print(flowers)
```

Output:

```
['Rose', 'Sunflower', 'Lotus', 'Blossom']
```

To insert a list item at a specific index, use the insert() method.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]

flowers.insert(2, "Blossom")

print(flowers)
```

Output:

```
['Rose', 'Sunflower', 'Blossom', 'Lotus']
```

extend()

The extend() method adds an entire list to the existing list.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers2 = ["Blossom", "Tulip", "Jasmine"]

flowers.extend(flowers2)

print(flowers)
```

Output:

```
['Rose', 'Sunflower', 'Lotus', 'Blossom', 'Tulip', 'Jasmine']
```

Remove List Items

There are several ways to remove items from the list.

pop()

The `pop()` method removes the last item from the list if no index is specified. If an index is provided, the item at that specific index is removed.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers.pop()
print(flowers)
```

Output:

```
['Rose', 'Sunflower']
```

remove()

The `remove()` method removes specific item from the list.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers.remove("Sunflower")
print(flowers)
```

Output:

```
['Rose', 'Lotus']
```

List Methods

Python provides several built-in methods for dealing with lists.

sort()

The `sort()` method sorts the list in ascending order.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers.sort()
print(flowers)
```

Output:

```
['Lotus', 'Rose', 'Sunflower']
```

reverse()

The `reverse()` method reverses the order of the list.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
flowers.reverse()
print(flowers)
```

Output:

```
['Sunflower', 'Lotus', 'Rose']
```

index()

The `index()` method returns the index of the first occurrence of the list item.

Example:

```
flowers = ["Rose", "Sunflower", "Lotus"]
print(flowers.index("Sunflower"))
```

Output:

```
1
```

Chapter 6

Python Tuples

A tuple is an ordered collection of data elements separated by a comma and enclosed within parentheses. They store multiple items in a single variable. Tuples are unchangeable meaning we can not change them after creation.

Example:

```
colors = ("Red", "Blue", "White")
print(colors)
```

Output:

```
("Red", "Blue", "White")
```

Tuple Methods

Python offers two built-in methods for dealing with tuples.

count()

The `count()` method returns the number of times the specified items appears in the tuple.

Example:

```
colors = ("Red", "Blue", "White")
newtup = colors.count("White")
print(newtup)
```

Output:

```
1
```

index()

The `index()` method returns the index of the first occurrence of the tuple item.

Example:

```
colors = ("Red", "Blue", "White")
newtup = colors.index("White")
print(newtup)
```

Output:

```
2
```

Chapter 7

Python Sets

Sets are unordered collection of data items. They store multiple items in a single variable. Sets items are separated by commas and enclosed within curly braces`{}`.

Example:

```
set1 = {2, 6, 14}
print(set1)
```

Output:

```
{2, 6, 14}
```

Add set Items

To add a single item to a set use the `add()` method.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits.add("Banana")
print(fruits)
```

Output:

```
{'Banana', 'Orange', 'Mango', 'Apple'}
```

Remove items from set

To remove an item from a set, use the `remove()` method.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits.remove("Mango")
print(fruits)
```

Output:

```
{'Orange', 'Apple'}
```

Set Methods

Python provides several built-in methods for dealing with sets.

isdisjoint()

The `isdisjoint()` method checks if items of given set are present in another set.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits2 = {"Apple", "Orange", "Mango"}
print(fruits.isdisjoint(fruits2))
```

Output:

```
False
```

issuperset()

The `issuperset()` method checks if all the items of a specified set are present in the original set.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits2 = {"Apple", "Mango"}
print(fruits.issuperset(fruits2))
```

Output:

```
True
```

issubset()

The `issubset()` method checks if all the items of the original set are present in the specified set.

Example:

```
fruits = {"Apple", "Orange", "Mango"}
fruits2 = {"Orange", "Mango"}
print(fruits2.issubset(fruits))
```

Output:

```
True
```

Chapter 8

Python Dictionaries

Dictionaries are ordered collection of data items. Dictionaries items are key-value pairs that are separated by commas and enclosed within curly brackets `{}`.

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
print(details)
```

Output:

```
{'name': 'Rahul', 'age': 22, 'canVote': True}
```

Add Items to a Dictionary

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
details["DOB"] = 2003
print(details)
```

Output:

```
{'name': 'Rahul', 'age': 22, 'canVote': True, 'DOB': 2003}
```

Remove Dictionary Items

There are several methods to remove items from a dictionary.

pop()

The `pop()` method removes the item with the provided key name.

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
details.pop("canVote")
print(details)
```

Output:

```
{'name': 'Rahul', 'age': 22}
```

clear()

The `clear()` method removes all the items from the dictionary.

Example:

```
details = {
  "name":"Rahul",
  "age":22,
  "canVote":True
}
details.clear()
print(details)
```

Output:

```
{}
```

Chapter 9

Python Conditional Statements

There are four types of conditional statements in Python:

- The if statement
- The if-else statement
- The if…elif…else Statement
- The nested-if statement

If Statement

The if statement is used to execute a block of code if a given condition is true.

Syntax:

```
if condition:
  # block of code to be executed if the condition is true
```

Example:

```
number = 8
if (number > 5):
   print("Number is greater than 5")
```

Output:

```
Number is greater than 5
```

If...else statement

The If...else statement is used to execute a block of code if a specified condition is true and another block of code if the condition is false.

Syntax:

```
if condition:
  # block of code to be executed if the condition is true
else:
  # block of code to be executed if the condition is false
```

Example:

```
number = 8
if (number > 5):
   print("Number is greater than 5")
else:
   print("Number is not greater than 5")
```

Output:

```
Number is greater than 5
```

if…elif…else Statement

Python's if-elif-else statement executes a block of code among multiple possibilities.

Syntax:

```
if (condition1):
  # block of code to be executed if condition1 is true
elif (condition2):
  # block of code to be executed if the condition1 is false and
condition2 is true
else:
  # block of code to be executed if the condition1 is false and
condition2 is false
```

Example:

```
number = 10
if (number > 15):
   print("Number is greater than 15")
elif (number > 10):
   print("Number is greater than 10 but less than or equal to 15")
else:
   print("Number is equal to 10")
```

Output:

```
x is equal to 10
```

Chapter 10

Python for & while Loop

for Loop

A for loop in Python is used to iterate over a sequence (e.g., a list, tuple, or string) or any other iterable object. .

Example:

```
companies = ["Google", "Facebook", "Microsoft"]
for i in companies:
    print(i)
```

Output:

```
Google
Facebook
Microsoft
```

while Loop

While loops in Python are used to execute a block of code several times as long as a condition is true.

Example:

```
number = 1
while (number <= 5):
    print(number)
    number = number + 1
```

Output:

```
1
2
3
4
5
```

Chapter 11

Python Functions

A function is a block of code that executes a specific task when called. They are defined with the def keyword followed by the function name, parentheses `()`, and a colon.

Example:

```
def my_func():
    print("Hello World")
```

Types of functions

There are two types of functions:

- built-in functions
- user-defined functions

built-in functions

These functions are pre-defined in python. Some examples of built-in functions are:
`len(), sum(), type(), range(), dict(), list(), tuple(), set(), print(), etc.`

user-defined functions

These are functions defined by the user to perform specific tasks.

Example:

```
def my_func(parameters):
    # block of code
```

Call a function

To call a function, use the function name followed by parenthesis

Example:

```
def my_func():
    print("Hello World")
my_func()
```

Output:

```
Hello World
```

Function Arguments

Arguments are the inputs passed to the function.

Example:

```
def my_func(fname, lname):
    print("Hello", fname, lname)
my_func("John", "Doe")
```

Output:

```
Hello John Doe
```

Recursion

Recursion is a programming method that involves calling a function itself to solve a problem.

Example:

```
def fibonacci(n):
    if n == 1 or n == 2:
      return 1
    else:
      return fibonacci(n - 1) + fibonacci(n - 2)
print(fibonacci(10))
```

Output:

```
55
```

Chapter 12

Python OOPS

OOPS stand for Object Oriented Programming System. It is a programming paradigm that uses objects and classes in programming.

Class

A class is a blueprint for creating objects. It can be defined using the class keyword, followed by the class name and a colon.

Example:

```
class Student:
    name = "Arka"
    age = 22
```

Objects

An object is an instance of a class.

Example:

```
class Student:
    name = "Arka"
    age = 22

obj1 = Student()
print(obj1.name)
```

Output:

```
Arka
```

__init__ method

The `__init__` method in Python is used to initialize objects of a class.

Example:

```
class Person:

    def __init__(self, name):

        self.name = name

    def greet(self):

        print('Hello, my name is', self.name)

p = Person('Sayan')

p.greet()
```

Output:

```
Hello, my name is Sayan
```

self method

The self parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

Example:

```
class Details:

    name = "John"

    age = 25

    def desc(self):

        print("Hello my name is", self.name)

obj1 = Details()

obj1.desc()
```

Output:

```
Hello my name is John
```

Chapter 13

Python Modules

Python modules are python files that contain python code that we can use within our python files.

Here are some popular python built-in modules:

`datetime, json, math, random, statistics, tkinter, turtle, etc.`

Math Module

Math Module consists of mathematical functions and constants. It is a built-in module made for mathematical tasks.

Example:

```
import math

print(math.floor(0.6))
print(math.floor(1.4))
print(math.floor(5.3))
print(math.floor(-5.3))
```

Output:

```
0
1
5
-6
```

Chapter 14

Python File Handling

File handling is a powerful tool that can be used to perform a wide range of operations. Python supports file handling and allows users to handle files to read and write and modify files.

Python File Open

Before performing any operation on the file like reading or writing, we need to open the file.

There are various modes in which we can open files.

read (r): This mode opens the file for reading only.

write (w): This mode opens the file for writing only.

append (a): This mode opens the file for appending only.

create (x): This mode creates a file.

Example:

```
f = open(filename, mode)
```

Creating a file is done using the create (x) mode.

Example:

```
file = open("myfile.txt", "x")
```

Output:

```
A new empty file is created.
```

Write onto a File

This method writes content onto a file.

Example:

```
file = open("demofile.txt", "w")

file.write("This is an example of file creation.")

file.close
```

Output:

```
This is an example of file creation.
```

Read a File

This method allows you to read the contents of the file.

Example:

```
file = open("demofile2.txt", "r")

print(file.read())

file.close
```

Output:

```
Hello,  Welcome to this tutorial.
```

Append a File:

This method appends content into a file.

Example:

```
file = open("newFile.txt", "a")

file.write("This is an example of file appending.")

file.close
```

Output:

```
This is an example of file appending.
```