INDEX

Unit –1 Beginning with Python, Data types, Operators, I/O and Control Statements 07 to 46

1.1 Introduction to Python

1.1.1 Features of Python

1.1.2 Viewing of Byte Code

1.1.3 Flavours of Python

1.1.4 Python Virtual Machine (PVM)

1.1.5 Memory Management in Python

1.1.6 Garbage collection in python

1.1.7 Comparisons between C-Java-Python

1.1.8 Writing first Python program

1.1.9 Execution of a Python program

1.2 Data types in Python

1.2.1 None Type

1.2.2 Numeric Types

1.2.3 Sequences in Python

1.2.4 str()

1.2.5 bytes

1.2.6 bytearray

1.2.7 Tuple

1.2.8 range()

1.2.9 Sets

1.2.10 Set() Constructor

1.2.11 Mapping types

1.2.12 Determining the datatype of a variable

1.2.13 Identifiers and reserved words

1.2.14 Naming conventions in Python

1.3 Operators, I/O and control statements

1.3.1 Python Arithmetic Operators

1.3.2 Python Assignment Operators
1.3.3 Python Comparison Operators
1.3.4 Python Logical Operators
1.3.5 Python Identity Operators
1.3.6 Python Membership Operators
1.3.7 Python Bitwise Operators
1.3.8 Input & Output statements
1.3.9 Command line arguments
1.3.10 A word on Indentation
1.3.11 Loops
1.3.12 The pass Statement
1.3.13 assert keyword
1.3.14 return keyword

Unit –2 Modules, Arrays, Functions, List, Tuples and Dictionaries **47 to 86**

2.1 Modules

2.1.1 Array
2.1.2 Slicing and indexing an Array
2.1.3 Returning result from a function
2.1.4 Different between function and methods
2.1.5 Anonymous Functions
2.1.6 Function Decorators

2.2 List

2.2.1 create a list
2.2.2 Create list using range() function
2.2.3 Updating the elements of the list
2.2.4 Concatenation of two lists
2.2.5 Membership in lists
2.2.6 Aliasing and Cloning lists
2.2.7 Using the list() method
2.2.8 List methods in python
2.2.9 Nested list
2.2.10 Tuples

2.2.11 Creating Tuples
2.2.12 Nesting of tuples

2.3 Introduction to Dictionaries

2.3.1 Creating a Dictionary
2.3.2 Operations on Dictionaries
2.3.3 Built-in Dictionary Methods
2.3.4 Converting List into Dictionary
2.3.5 Passing dictionaries to functions

Unit –3 Classes, Inheritance and Polymorphism **87 to 121**

3.1 Classes and Objects

3.1.1 Creating a class
3.1.2 The self-variable
3.1.3 Constructor
3.1.4 Types of variables
1.3.5 Types of Methods
1.3.6 Class Methods

3.2 Inheritance

3.2.1 Implementing Inheritance
3.2.2 Constructor in Inheritance
3.2.3 Overriding Super class constructors and methods
3.2.4 The Super() Method
3.2.5 Types of inheritance
3.2.6 Problems in Multiple Inheritances
3.2.7 Method Resolution Order (MRO)

3.3 Polymorphism

3.3.1 Introduction to polymorphism
3.3.2 Duck typing philosophy of Python
3.3.3 Operator Overloading
3.3.4 Method Overloading

Unit –4 Exception Handling, Standard Librabry, Creating Virtual Environment and Python Database Connectivity

122 to 152

4.1 Exception Handling and Standard Library

4.1.1 Exceptions

4.1.2 Exception handling

4.1.3 Types of exceptions

4.1.4 Operating System Interface

4.1.5 File wildcards

4.1.6 Command line arguments

4.1.7 String pattern matching

4.1.8 Mathematics

4.1.9 Internet access

4.1.10 Dates and time

4.1.11 Data compression

4.1.12 Performance measurement

4.2 Creating virtual environment

4.2.1 Introduction

4.2.2 Generating virtual environments

4.3.3 Managing packages with pip (Python Package Index)

4.3 Python and MySQL

4.3.1 Installing MySQL Connector

4.3.2 Verifying the Connector Installation

4.3.3 Using MySQL from Pyth

4.3.4 Retrieving all rows from a table

4.3.5 Inserting rows into a table

4.3.6 Deleting rows from table

4.3.7 Updating rows in a table

4.3.8 Creating database tables through Python

- **Self-Test Examination** — **153 To 161**
- **Paper 2019** — **162 To 164**

Unit –1 Beginning with Python, Datatypes, Operators, I/O and Control Statements

1.1 Introduction to Python:

Python is an interpreter, high-level, general-purpose programming language.

Python is a programming language that lets you work more quickly and integrate your systems more effectively.

1.1.1 Features of Python:

- **High-Level Language:** Python is a high-level language. When we write programs in python, we don't require remembering system architecture and managing the memory.
- **Free and Open Source:** Python language is freely available and it is an open source. Latest version can be download from python.org
- **Object-Oriented Language:** Python is an Object-Oriented programming language so it supports object-oriented concepts of classes, objects encapsulation, polymorphism and inheritance etc.
- **Python is Portable language:** Python language is also a portable language. It is supported in all the platforms like Linux, Unix, and Mac.
- **Easy to code:** Python is developer-friendly language and very easy to learn and write the code.
- **Python is integrated language:** Python is an integrated language and we can easily integrated python with other languages like c, c++, etc.
- **GUI Programming Support:** Python also supports the GUI and many GUI application and editors are available to write code and execute the applications.
- **Extensible feature:** Python is an **Extensible** language. We can write and compile python code into C or C++ language.
- **Interpreted Language:** Python is an Interpreted Language. Python code is executed line by line at a time. The source code of python is converted into an immediate form called **bytecode**.
- **Dynamically Typed Language:** Python is a dynamically-typed language. It means that data type (int, double, long etc.) for a variable is decided at run time not in advance so we don't require to specify the data type of variable.
- **Large Standard Library**: Python has a large standard library which provides a rich set of module and functions so you do not have to write your own code for every single thing. There are many libraries present in python for such as regular expressions, unit-testing, web browsers, etc.

1.1.2 Viewing of Byte Code:

Python is usually called an interpreted language. When we execute a source code (a file with a .py extension), Python first compiles it into a bytecode.

The bytecode is a low-level platform-independent representation of your source code. It is not the binary machine code and cannot be run by the target machine directly. In fact, it is a set of instructions for a virtual machine which is called the Python Virtual Machine (PVM).

After compilation, the bytecode is sent for execution to the PVM. The PVM is an interpreter that runs the bytecode and is part of the Python system.

In Python, the bytecode is stored in a .pyc file.

1.1.3 Flavours of Python:

Types of Python compilers are referred as flavors of Python. They help to integrate various types of programming languages in Python.

- **CPython:** It is a Python compiler that was implemented in C language. Even C++ code can be execute using CPython.
- **JPython:** It is enables Python implementation to be run on Java platform. It runs on JVM.
- **IronPython:** It is compiler designed for .NET framework, but it is written in C#. It can run on CLR (Common Language Run time).
- **PyPy:** It is a Python implemented by using Python language itself. It runs fast since JIT is incorporated to PVM.
- **Ruby Python:** It acts as a bridge from Ruby to Python interpreter. It embeds the Python interpreter inside the Ruby application
- **Pythonxy:** It is written in the form of Python (X, Y). It is designed by adding scientific and engineering related packages.
- **Anaconda Python:** The name Anaconda Python is obtained after redeveloping it to handle large scale data processing, predictive analytics and scientific computing. It handles huge amount of data.
- **Stackless Python:** Tasklets are the small tasks that are run independently. The communication is done with each by using channels. They schedule, control and suspend the Tasklets. Hundreds of tasklets can run by a thread. The thread and tasklets can be created in stackless python. It is a reimplementation of python.

1.1.4 Python Virtual Machine (PVM):

The PVM is the **runtime engine** of Python. It's always present as part of the Python system. It's the **component that truly runs our scripts**. Technically it's just **the last step** of what is called the **Python interpreter**.

The PVM is not a separate program. It need not be installed by itself. Actually, the PVM is just a big loop that iterates through our byte code instruction, one by one, to carry out their operations.

1.1.5 Memory Management in Python:

Memory management is the process by which applications read and write data. A memory manager determines where to put an application's data. Since there's a finite chunk of memory, like the pages in our book analogy, the manager has to find some free space and provide it to the application. This process of providing memory is generally called memory **allocation**.

Memory management is the process of efficiently allocating, de-allocating, and coordinating memory so that all the different processes run smoothly and can optimally access different system resources. Memory management also involves cleaning memory of objects that are no longer being accessed.

1.1.6 Garbage collection in python:

Python deletes objects that are no longer referenced in the program to free up memory space. This process in which Python frees blocks of memory that are no longer used is called Garbage Collection.

In Python, memory manager is responsible for periodically run to clean up, allocate, and manage the memory. Unlike C, Java, and other programming languages, Python manages objects by using reference counting. This means that the memory manager keeps track of the number of references to each object in the program. When an object's reference count drops to zero, which means the object is no longer being used, the garbage collector (part of the memory manager) automatically frees the memory from that particular object.

The user need not to worry about memory management as the process of allocation and de-allocation of memory is fully automatic. The reclaimed memory can be used by other objects.

1.1.7 Comparisons between C-Java-Python:

C	Python
Mainly used for hardware related applications.	General purpose programming language.
It follows an imperative programming model.	It follows object-oriented programming language
C supports the Pointers.	No pointers functionality available.

It uses the compiler.	It use the Interpreter.
It has limited number of built-in functions.	It has Large library of built-in functions.
In C Language, code execution is faster than python.	It is slower for the code execution compared to C.
It is compulsory to declare the variable type in C.	No need to declare a type of variable.
C program syntax is harder than python.	Python programs are easier to learn, write and read.

Parameter	Java	Python
Compilation	Java is a Compiled Language	Python is an Interpreted Language
Static or Dynamic	Java is statically typed	Python is dynamically typed
String operations	Offers limited string related functions.	It offers lots of string related functions.
Learning curve	Complex learning curve	Easy to learn and use
Multiple inheritances	Multiple inheritances is partially done through interfaces.	It offers both single and multiple inheritances.
Braces vs. Indentation	It uses curly braces to define the beginning and end of each function and class definition.	Python uses indentation to separate code into code blocks.
Speed	Java program runs slowly compared to Python.	Python programs run faster than Java.
Portability	Any computer or mobile device which is able to run the Java virtual machine can run a Java application	Python programs need an interpreter installed on the target machine to translate Python code. Compared to Java, Python is less portable.
Read file	Java takes 10 lines of code to read from a file in Java.	Python only needs 2 lines of code.
Architecture	Java Virtual Machine provides the runtime environment to execute the code and convert bytecode into machine language.	For Python, the interpreter translates source code into machine-independent bytecode.

Parameter	Java	Python
Backend Frameworks	Spring, Blade	Django, Flask
Best use for	Java is best for Desktop GUI apps, Embed Systems, Web application services, etc.	Python is excellent for scientific and numeric computing, Machine learning apps, more.
Database support	Java offers stable connectivity	Python offers weak connectivity.

Code example	class A {
	public static void main(String args[]){
	System.out.println("Hello World");
	}
	}
		Hello World in Python:
		print "hello world";

1.1.8 Writing first Python program:

To Install Python, download it from python.org

Step 1:

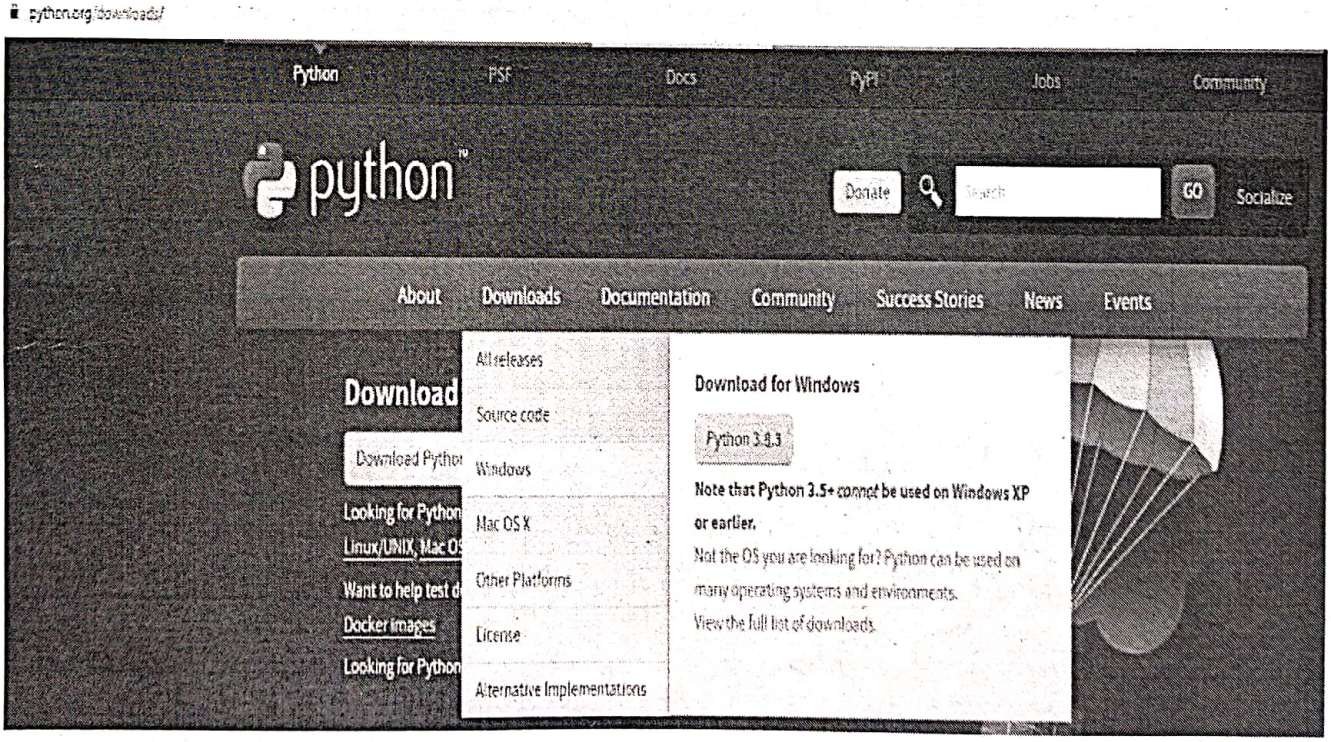


Fig.1.1 Download – Python

Step 2: After installation, click on IDLE (Python 3.8 32-bit):

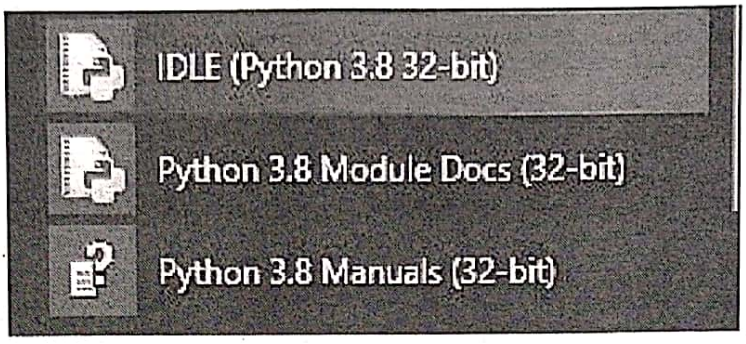


Fig.1.2 IDLE Python

Step 3: File → New

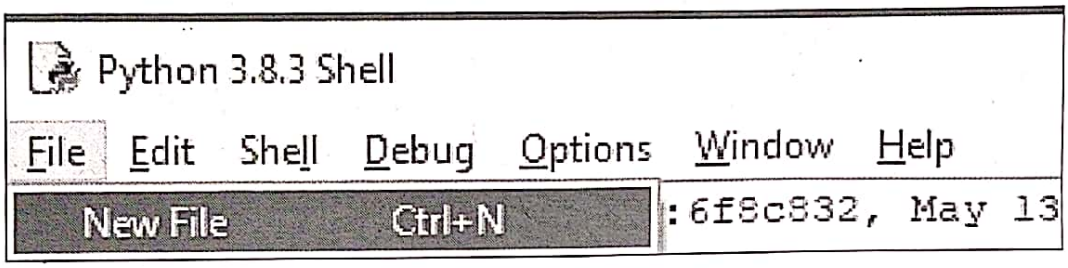


Fig.1.3 Python Menu

Step 4: Write the code and save it using .py extension.

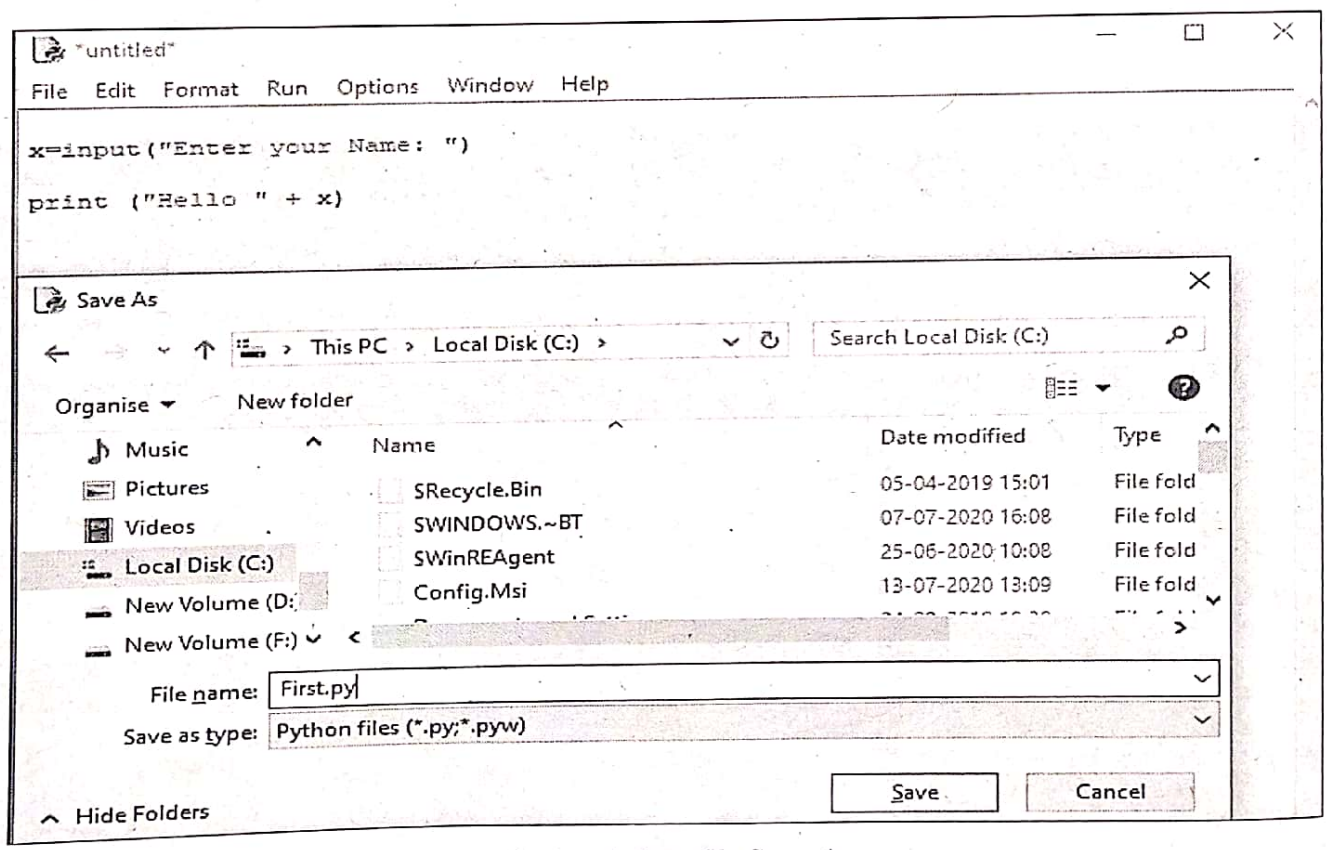


Fig.1.4 Python file Save As

In the above code, *input* is used accept the value from user and *print* is used to display the message and value.

1.1.9 Execution of a Python program:

Step 1: Execute the file using F5 or Run → Run Module

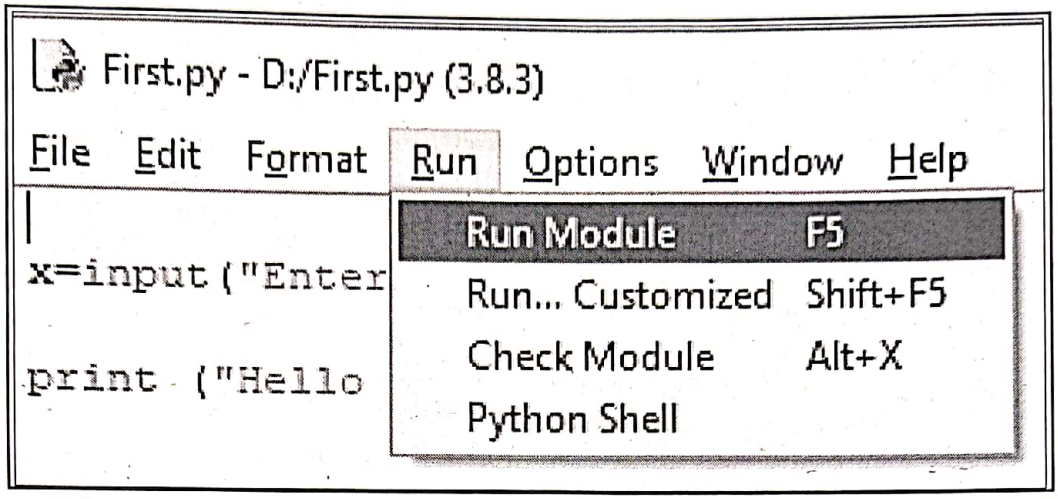


Fig.1.5 Run Module

Step 2: Enter the name

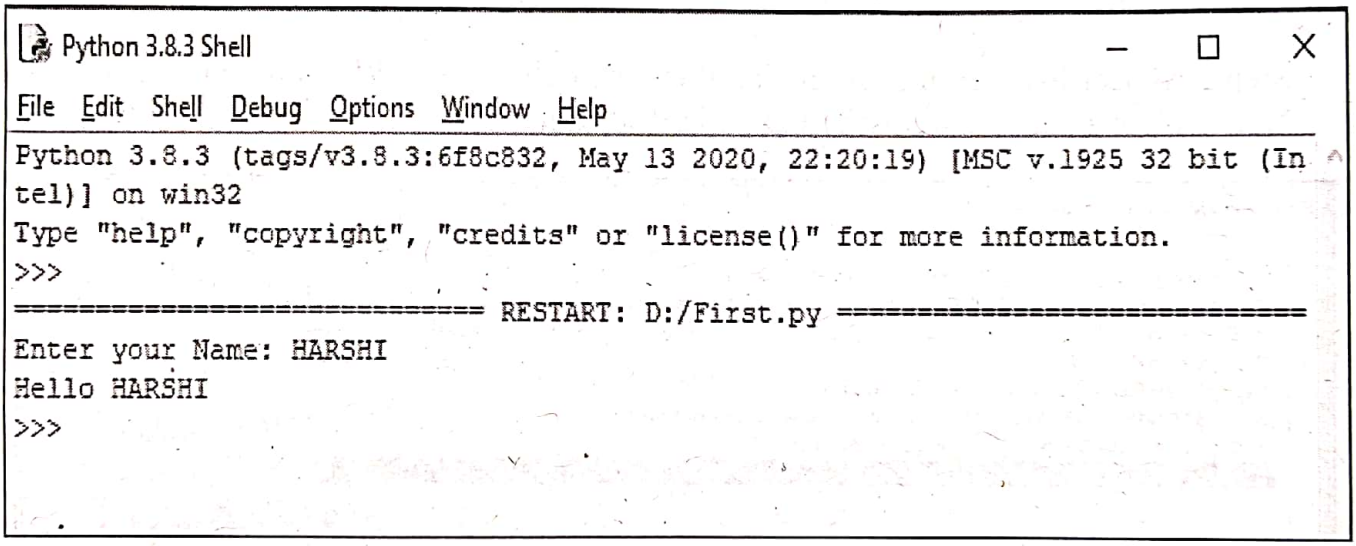


Fig.1.6 Run Module- Enter the name

Execution of a Python program through Command Prompt:

Step 1: Set Path using Control panel → System → Change settings

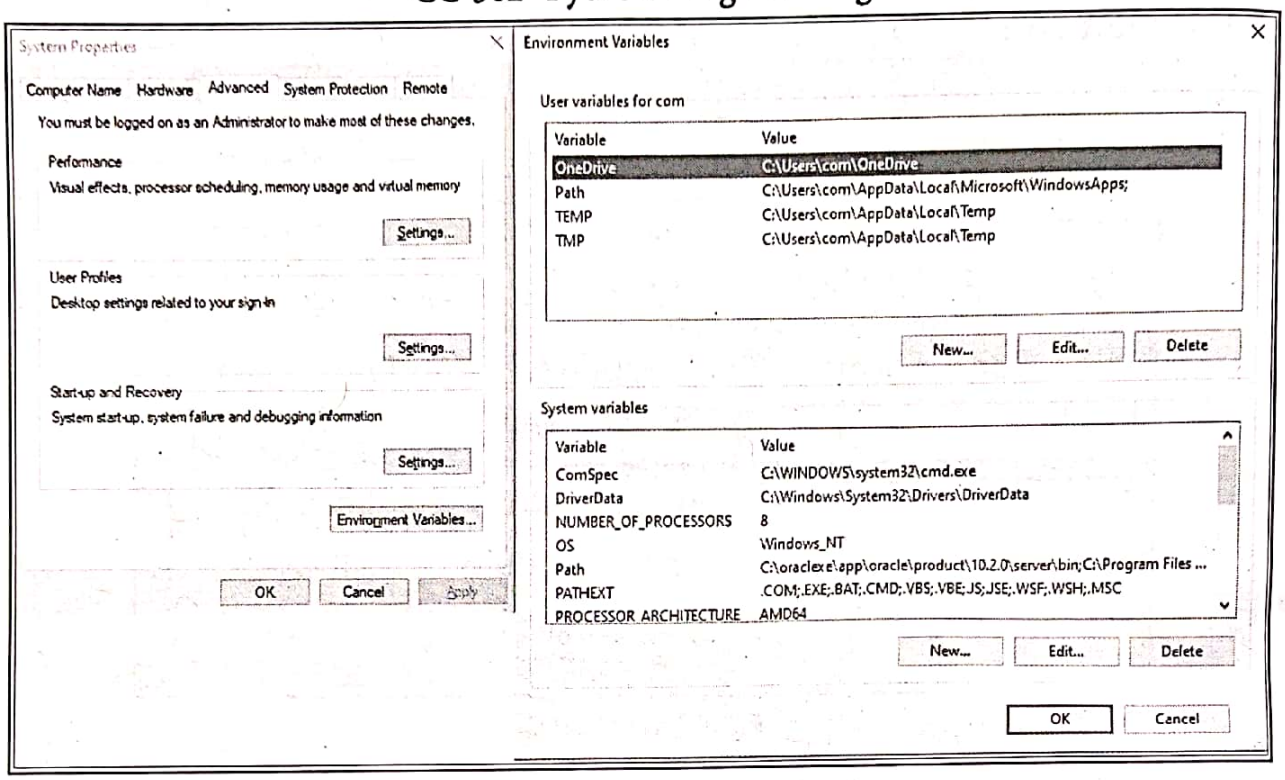


Fig.1.7 Change settings

Step 2: Select Path from System Variables then click on *EditButton* and add *New* path. New path is a path where python.exe file is located.

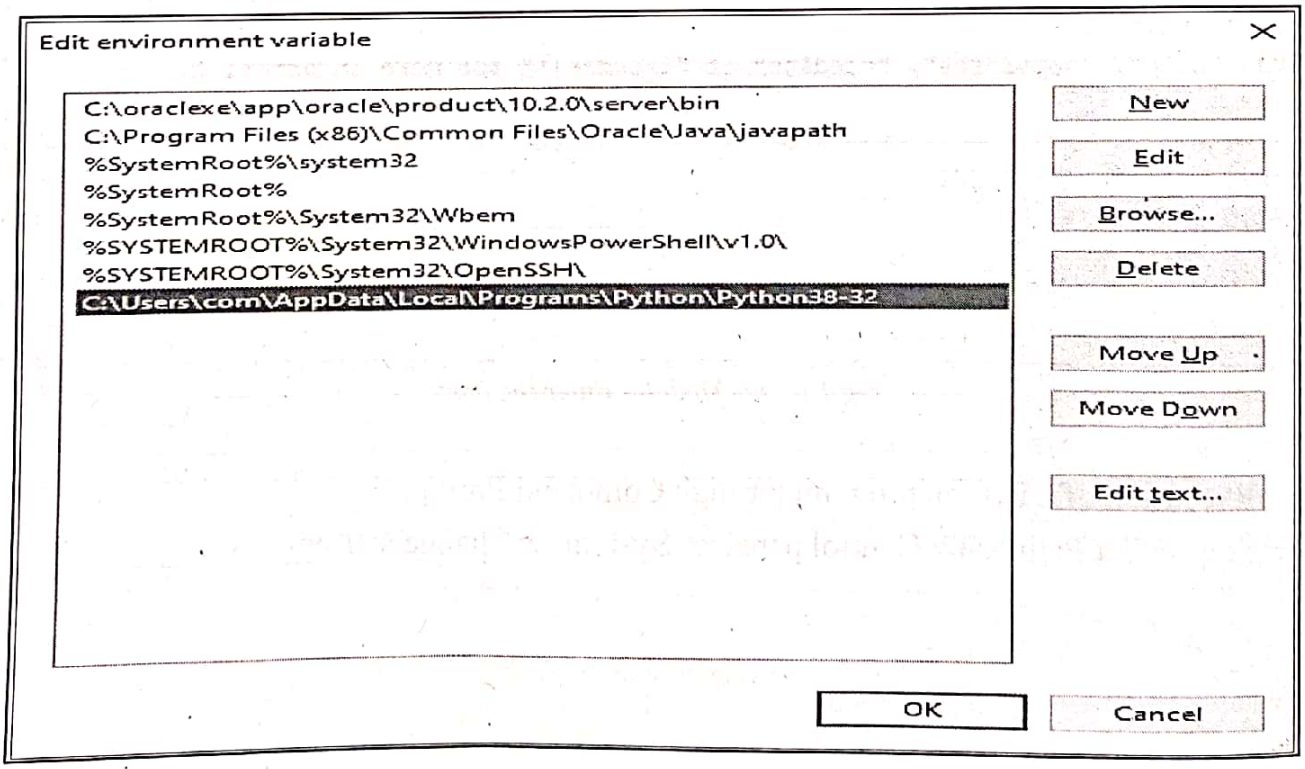


Fig.1.8 Edit Environment Variable Screen

Step 3: Open Command Prompt and execute the program.

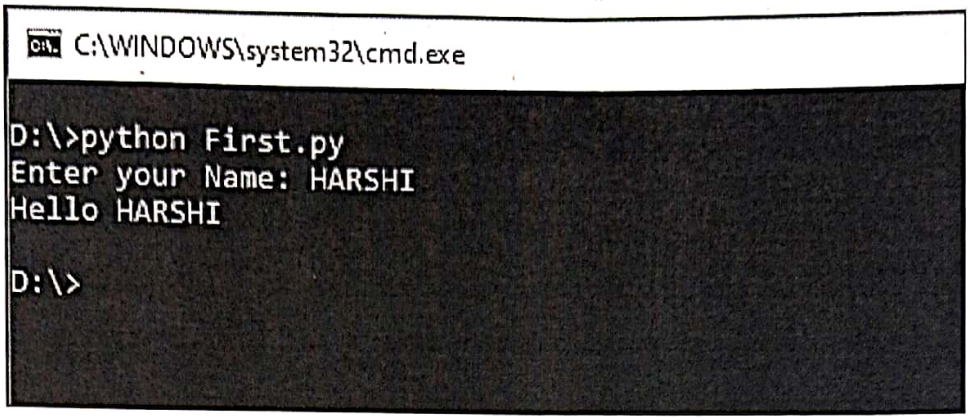


Fig.1.9 Open Command Prompt

1.2 Datatypes in Python:

Built-in datatypes:Every value in Python has a datatype. Since everything is an object in Python programming, data types are actually classes and variables are instance (object) of these classes.

There are various data types in Python. Some of the important types are listed below.

Python has the following data types built-in by default, in these categories:

Text Type:	str
Numeric Types:	int, float, complex
Sequence Types:	list, tuple, range
Mapping Type:	dict
Set Types:	set, frozenset
Boolean Type:	bool
Binary Types:	bytes, bytearray, memoryview

1.2.1 None Type:

The None **keyword** is used to define a null variable or an object.

In Python, None keyword is an object, and it is a data type of the class NoneType.

Example using is and == operator:

```
var = None
if var is None: # Checking if the variable is None
  print("None")
else:
  print("Not None")

# Declaring a None variable
var = None

if var == None: # Checking if the variable is None
  print("None")
else:
  print("Not None")
```

1.2.2 Numeric Types:

A numeric value is any representation of data which has a numeric value. Python identifies three types of numbers:

- **Integer:** Positive or negative whole numbers (without a fractional part) e.g. 12,0,-12.
- **Float:** Any real number with a floating point representation in which a fractional component is denoted by a decimal symbol or scientific notation. e.g. 1234.56, 3.142, -1.55, 0.23.
- **Complex number:** A complex number is a number with real and imaginary components. For example, 5 + 3j is a complex number where 5 is the real component and 3 multiplied by j is an imaginary component. e.g. 1+2j, 10-5.5j, 5.55+2.33j, 3.11e-6+4j

1.2.3 Sequences in Python:

In Python, **sequence** is the generic term for an ordered set. There are mainly three types of sequences.

Lists are the most versatile sequence type. The elements of a list can be any object, and lists are **mutable** - they can be changed. Elements can be reassigned or removed, and new elements can be inserted.

Tuples are like lists, but they are **immutable** - they can't be changed.

Strings are a special type of sequence that can only store characters, and they have a special notation. However, all of the sequence operations described below can also be used on strings.

Sequence Operations:

Sr.No.	Operation/Functions & Description
1	**x in seq** True, when x is found in the sequence seq, otherwise False
2	**x not in seq** False, when x is found in the sequence seq, otherwise True
3	**x + y** Concatenate two sequences x and y
4	**x * n or n * x** Add sequence x with itself n times
5	**seq[i]** ith item of the sequence.
6	**seq[i:j]** Slice sequence from index i to j
7	**seq[i:j:k]** Slice sequence from index i to j with step k
8	**len(seq)** Length or number of elements in the sequence
9	**min(seq)** Minimum element in the sequence
10	**max(seq)** Maximum element in the sequence
11	**seq.index(x[, i[, j]])** Index of the first occurrence of x (in the index range i and j)
12	**seq.count(x)**

	Count total number of elements in the sequence
13	**seq.append(x)** Add x at the end of the sequence
14	**seq.clear()** Clear the contents of the sequence
15	**seq.insert(i, x)** Insert x at the position i
16	**seq.pop([i])** Return the item at position i, and also remove it from sequence. Default is last element.
17	**seq.remove(x)** Remove first occurrence of item x
18	**seq.reverse()** Reverse the list

Example:

```
myList1 = [10, 20, 30, 40, 50]
myList2 = [60,70,80,90,100,90]

if 50 in myList1:
print('50 is present')

if 100 not in myList1:
print('100 is not present')

print(myList1 + myList2) #Concatenate lists
print(myList1 * 3) #Add myList1 three times with itself
print(max(myList2))
print(myList2.count(90)) #90 has two times in the list
```

```
print(myList2[2:5])
print(myList2[2:5:2])

myList1.append(60)
print(myList1)

myList2.insert(5, 17)
print(myList2)

myList2.pop(3)
print(myList2)
myList1.reverse()
print(myList1)

myList1.clear()
print(myList1)
```

Output:

50 is present
100 is not present
[10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 90]
[10, 20, 30, 40, 50, 10, 20, 30, 40, 50, 10, 20, 30, 40, 50]
100
2
[80, 90, 100]
[80, 100]
[10, 20, 30, 40, 50, 60]
[60, 70, 80, 90, 100, 17, 90]
[60, 70, 80, 100, 17, 90]
[60, 50, 40, 30, 20, 10]
[]

1.2.4 str():

The str() function converts the specified value into a string.

Syntax:

str(*object*, encoding=*encoding*, errors=*errors*)

Parameter	Description
object	Any object. Specifies the object to convert into a string
encoding	The encoding of the object. Default is UTF-8
errors	Specifies what to do if the decoding fails

Example:

Convert the number 3.5 into a string:

x = str(3.5)

Convert a string into an integer:

x = int("12")

1.2.5 bytes:

The `bytes()` function returns a bytes object.

It can convert objects into bytes objects, or create empty bytes object of the specified size. It is an immutable sequence of small integers in the range 0 <= x < 256 print as ASCII characters when displayed.

Example:

x = bytes(4)

Output: b'\x00\x00\x00\x00'

1.2.6 bytearray:

The `bytearray()` function returns a bytearray object.

It can convert objects into bytearray objects, or create empty bytearray object of the specified size.The bytearray type is a mutable (changeable) sequence of integers in the range 0 <= x < 256.

Example:

```
x = bytearray(4)
Output:bytearray(b'\x00\x00\x00\x00')
```

List:

A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

Example:

```
thislist = ["apple", "banana", "cherry", "orange", "kiwi"]
  print(thislist)
print(thislist[1])
```

```
print(thislist[-1])
print(thislist[1:3])# position 1 to 3 and 3 item is
  excluded.
thislist[1] = "Mango"
print(thislist)
```

Output:

['apple', 'banana', 'cherry', 'orange', 'kiwi']
Banana
kiwi
['banana', 'cherry']
['apple', 'Mango', 'banana', 'cherry', 'orange', 'kiwi']

1.2.7 Tuple:

Tuple is a collection which is ordered and unchangeable. Allows duplicate members.

Example:

```
thistuple = ("apple", "banana", "cherry")
print(thistuple)
print(thistuple[1])
print(thistuple[-1])
#to change into tuple, not directly possible like list
y = list(thistuple)
y[1] = "kiwi"
thistuple = tuple(y)
print(thistuple)
```

Output:

('apple', 'banana', 'cherry')
banana
cherry
('apple', 'kiwi', 'cherry')

1.2.8 range():

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

Syntax: range(start, stop, step)

Example

```
x = range(1, 10)
for n in x:
print(n)
y=range(11,20)
for n in y:
print(n)
```

Output:

```
1  2   3    4    5    6    7    8    9    10
11 13  15   17   19
```

1.2.9 Sets:

A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.

Example:

```
thisset = {"apple", "banana", "cherry"}
print(thisset)

for x in thisset:
print(x)

print("banana" in thisset)

thisset.add("orange")  # add method is used to adds an
  element.
print(thisset)

thisset.update(["orange", "mango", "grapes"]) #update(Add)
  multiple items in thisset

print(thisset)

print(len(thisset))  # Length of the set

thisset.remove("banana")  # Remove the element. If the
  item to remove does not exist, remove() will raise an
  error.
```

```
print(thisset)

thisset.discard("cherry")    #  Remove the element. If the
   item to remove does not exist, remove() will NOT raise
   an error.
print(thisset)

x = thisset.pop() #Remove the last item
print(x)

print(thisset)

thisset.clear() # Empties the set

print(thisset)

set1 = {"a", "b" , "c"}
set2 = {1, 2, 3}
set3 = set1.union(set2) # union combines the two sets and
   sent it to set3.
print(set3)

set1.update(set2) # update() method inserts the items in
   set2 into set1
print(set1)

del set3   # Delete the set
print(set3) # nothing will be print
```

Output:

{'banana', 'cherry', 'apple'}
banana
cherry
apple
True
{'banana', 'cherry', 'apple', 'orange'}

```
{'banana', 'apple', 'grapes', 'orange', 'mango', 'cherry'}
6
{'apple', 'grapes', 'orange', 'mango', 'cherry'}
{'apple', 'grapes', 'orange', 'mango'}
apple
{'grapes', 'orange', 'mango'}
set()
{1, 'c', 2, 3, 'a', 'b'}
{1, 'c', 2, 3, 'a', 'b'}
```

1.2.10 Set() Constructor:

It is also possible to use the set() constructor to make a set.

Example:

```
thisset = set(("apple", "banana", "cherry")) # double
   round-brackets
print(thisset)

set1 = set([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
print(set1)

set1.remove(5)
set1.remove(6)

print(set1)
```

Output:

```
{'apple', 'cherry', 'banana'}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}
{1, 2, 3, 4, 7, 8, 9, 10, 11, 12}
```

Frozenset: It Freeze the list and make it unchangeable.

Example:

```
mylist = ['apple', 'banana', 'cherry']
   x = frozenset(mylist)
x[1] = "Mango" # It gives an error message as change is
   not possible.
```

```
print(x)
```

Output: It gives an error message as change is not possible.

1.2.11 Mapping types:

The mapping objects are used to map hash table values to arbitrary objects. In python there is mapping type called **dictionary**. It is mutable (changeable).

The keys of the dictionary are arbitrary. As the value, we can use different kind of elements like lists, integers or any other mutable type objects.

Dictionaries are created by placing a comma-separated list of *key*: *value* pairs within braces.

Example:

```
myDict={'ten':10,'twenty':20,'thirty':30,'forty':40}
print(myDict)
print(list(myDict.keys()))  #print only keys ten, twenty
   .....
print(list(myDict.values())) # print values 10,20,30,40

#create items from the key-value pairs
print(list(myDict.items()))
myDict.update({'fifty':50}) # add new values
print(myDict)
```

Output:

{'ten': 10, 'twenty': 20, 'thirty': 30, 'forty': 40}

['ten', 'twenty', 'thirty', 'forty']

[10, 20, 30, 40]

[('ten', 10), ('twenty', 20), ('thirty', 30), ('forty', 40)]

{'ten': 10, 'twenty': 20, 'thirty': 30, 'forty': 40, 'fifty': 50}

1.2.12 Determining the datatype of a variable:

type() function is used to get the data type of any object.

Example:

```
x = 5
    print(type(x))
```

Output:<class 'int'>

Example	Output of print(type(x))
x = "Hello How r u"	Str
x=10	int
x=10.5	Float
x = 5j	complex
x = ["apple", "banana", "cherry"]	list
x = ("apple", "banana", "cherry")	tuple
x = range(6)	range
x = {"name" : "Jay", "age" : 26}	dict
x = {"apple", "banana", "cherry"}	set
x = frozenset({"apple", "banana", "cherry"})	frozenset
x = True	bool
x = b"Hi"	bytes
x = bytearray(7)	bytearray
x = memoryview(bytes(7))	memoryview

1.2.13 Identifiers and reserved words:

Keyword	Description
and	A logical operator
as	To create an alias
assert	For debugging
break	To break out of a loop
class	To define a class
continue	To continue to the next iteration of a loop
def	To define a function
del	To delete an object
elif	Used in conditional statements, same as else if
else	Used in conditional statements
except	Used with exceptions, what to do when an exception occurs
FALSE	Boolean value, result of comparison operations
finally	Used with exceptions, a block of code that will be executed no matter if there is an exception or not
for	To create a for loop
from	To import specific parts of a module
global	To declare a global variable

if	To make a conditional statement
import	To import a module
in	To check if a value is present in a list, tuple, etc.
is	To test if two variables are equal
lambda	To create an anonymous function
None	Represents a null value
nonlocal	To declare a non-local variable
not	A logical operator
or	A logical operator
pass	A null statement, a statement that will do nothing
raise	To raise an exception
return	To exit a function and return a value
TRUE	Boolean value, result of comparison operations
try	To make a try...except statement
while	To create a while loop
with	Used to simplify exception handling
yield	To end a function, returns a generator

1.2.14 Naming conventions in Python:

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

Rules for Python variables:

- A variable name must start with a letter or the underscore character
- A variable name cannot start with a number
- A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
- Variable names are case-sensitive (age, Age and AGE are three different variables)

Example:

```
#Legal variable names:
myvar.= "Harshi"
my_var = "Harshi"
_my_var = "Harshi"
myVar = "Harshi"
MYVAR = "Harshi"
```

```
myvar2 = "Harshi"

#Illegal variable names:
2myvar = "Harshi"
my-var = "Harshi"
myvar = "Harshi"
```

Explicit conversion of datatypes:

To convert from one type to another is possible through `int()`, `float()`, and `complex()` methods.

Example:

```
x = 10 # int
y = 12.8 # float
z = 1j # complex

#convert from int to float:
a = float(x)

#convert from float to int:
b = int(y)

#convert from int to complex:
c = complex(x)

print(a)
print(b)
print(c)

print(type(a))
print(type(b))
print(type(c))
```

Output:

```
1.0
 2
 (1+0j)
 <class 'float'><class 'int'><class 'complex'>
```

1.3 Operators, I/O and control statements:

Operators are used to perform operations on variables and values.

Python divides the operators in the following groups:

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

1.3.1 Python 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

1.3.2 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

1.3.3 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

1.3.4 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)

1.3.5 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

1.3.6 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

1.3.7 Python Bitwise Operators:

Bitwise operators are used to compare (binary) numbers:

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

1.3.8 Input & Output statements:

input ( ) : This function first takes the input from the user and then evaluates the expression, which means Python automatically identifies whether user entered a string or a number or list. If the input provided is not correct then either syntax error or exception is raised by python.

Example:

```
# Python program showing a use of input()
val = input("Enter your value: ")
print(val)
```

Output:

Enter your value: 10

10

Example:

```
# Program to check input type in Python
num = input ("Enter number :")
name1 = input("Enter name : ")
print(num)
print(name1)
```

```
# taking two inputs at a time

x, y = input("Enter two numbers: ").split()
print("First Number is : ", x)
print("Second Number is: ", y)

print( "Total Marks are:  " , int(x)+int(y))
# Printing type of input value
print ("type of number", type(num))
print ("type of name", type(name1))
```

Output :

Enter number :10
Enter name : HARSHI
10
HARSHI
Enter two numbers: 66 77
First Number is : 66
Second Number is: 77
Total Marks are: 143
type of number <class 'str'>
type of name <class 'str'>

1.3.9 Command line arguments:

The Python **sys** module provides access to any command-line arguments via the **sys.argv**. This serves two purposes –

- sys.argv is the list of command-line arguments.
- len(sys.argv) is the number of command-line arguments.

Here sys.argv[0] is the program ie. script name.

Example:

```
import sys
print 'Number of arguments:', len(sys.argv), 'arguments.'
print 'Argument List:', str(sys.argv)
```

```
Run in command prompt using arguments:
python test.py arg1 arg2 arg3
```

Output:

Number of arguments: 4 arguments.

Argument List: ['test.py', 'arg1', 'arg2', 'arg3']

1.3.10 A word on Indentation:

if-elif-else statement:

The **elif** statement allows you to check multiple expressions for TRUE and execute a block of code as soon as one of the conditions evaluates to TRUE.

Similar to the **else**, the **elif** statement is optional. However, unlike **else**, for which there can be at most one statement, there can be an arbitrary number of **elif** statements following an **if**.

Syntax

```
if expression1:
   statement(s)
elif expression2:
   statement(s)
elif expression3:
   statement(s)
else:
   statement(s)
```

Example:

```
import os
os.system("cls")
x=int(input("Enter the First number"))
y=int(input("Enter the Second number"))
if  x>y:
  print ("x is Max")
elif y>x:
  print("y is Max")
else:
  print("Both r Equal")
```

1.3.11 Loops:

Python programming language provides following types of loops to handle looping requirements.

Sr.No.	Loop Type & Description
1	while loop Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.
2	for loop Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.
3	nested loops You can use one or more loop inside any another while, for or do..while loop.

Example of While loop:

```
import os
os.system("cls")
x=1
sum=0
while(x!=6):
   y=int(input("Enter the number:"))
   sum=sum+y
   x=x+1
print("Sum is:",sum)
```

Example of For loop using range():

Example:

```
# print 1 to 10
import os
os.system("cls")

for i in range(1,11):
   print(i)
```

Example:

```
# Multiplication Table
import os
os.system("cls")
```

```
x=int(input("Enter the Number:"));
for i in range(1,11):
   print(x,"*",i,"=",x*i)
```

Example:

```
# Reverse Loop
import os
os.system("cls")
for i in range(11,1,-1):
   print(i)
```

Example of Nested loop:

```
adj = ["red", "big", "tasty"]
 fruits = ["apple", "banana", "cherry"]

 for x in adj:
   for y in fruits:
     print(x, y)
```

Output:

red apple
red banana
red cherry
big apple
big banana
big cherry
tasty apple
tasty banana
tasty cherry

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Python supports the following control statements. Click the following links to check their detail.

Let us go through the loop control statements briefly

Sr.No.	Control Statement & Description
1	break statement Terminates the loop statement and transfers execution to the statement

	immediately following the loop.
2	continue statement Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
3	pass statement The pass statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.

Example of Break Statement:

```
fruits = ["apple", "banana", "cherry"]
 for x in fruits:
   print(x)
   if x == "banana":
     break
```

Example of Continue Statement:

```
fruits = ["apple", "banana", "cherry"]
 for x in fruits:
   if x == "banana":
     continue
   print(x)
```

1.3.12 The pass Statement:

for loops cannot be empty, but if you for some reason have a for loop with no content, put in the pass statement to avoid getting an error.

```
for x in [0, 1, 2]:
pass
```

1.3.13 assert keyword:

The assert keyword is used when debugging code.

The assert keyword test condition in your code and returns true if it is satisfied or raise an AssertionError.

Example:

```
x = "hello"
#if condition returns True, then nothing happens:
assert x == "hello"
```

```
#if condition returns False, AssertionError is raised:
assert x == "goodbye"
```

Output:

```
Traceback (most recent call last):
  File "demo_ref_keyword_assert.py", line 5, in <module>
    assert x == "goodbye"
AssertionError
```

1.3.14 return keyword:

The return keyword is to exit a function and return a value.

Example:

```
def myfunction():
  return 3+3
print(myfunction())
```

Output: 6

Example:

```
import os
os.system("cls")

a=int(input("Enter a:"))
b=int(input("Enter b:"))
c=max(a,b)
print(c)

def max(x,y):
   if x>y:
       return x
   else:
       return y
```

Output:

```
Enter a:20
Enter b:30
30
```

☺ ☺ ☺ ☺ ☺

Exercises

Fill in the Blanks

(1) Python is a _____ Language.

(2) PVM stands for _____.

(3) _____ is the extension of bytecode file.

(4) The _____ function converts the specified value into a string.

(5) The _____ function returns a bytes object.

(6) The _____ function returns a sequence of numbers

(7) The _____ keyword is to exit a function and return a value.

(8) The _____keyword is used when debugging code

(9) _____ is the list of command-line arguments.

(10) _____ function is used to get the data type of any object..

Answers:

Programming / interpreted
bytes()
sys.argv
Intersect
range()
type()
.pyc
return
str()
assert

Descriptive Questions:

1. What are the features of Python? Explain any four in detail.
2. What is bytecode? Give the name of any four flavours of python.
3. What is PVM? Explain the Garbage collection in python.
4. Compare C and Python.
5. Compare Java and Python.
6. Explain list, tuple and range sequence types with example.
7. What are the rules for naming conventions in python?
8. Explain the Explicit conversion of datatype using two example.
9. Explain any three string function with example.
10. Explain python logical operator with example.
11. Explain any three Python bitwise operator with example.
12. What is the difference between in and not in operator.
13. Explain break, continue and pass statement.

CC-305 Unit – 1 Python Programming Practicals

1. Write a program to swap two numbers without taking a temporary variable.

```
a=int(input("Enter value of first variable: "))
b=int(input("Enter value of second variable: "))
a=a+b
b=a-b
a=a-b
print("a is:",a," b is:",b)
```

Output:

Enter value of first variable: 10 Enter value of second variable: 20

a is: 20 b is: 10

2. Write a program to display sum of two complex numbers.

```
print("Enter two complex numbers in the form a+bj:")
n1 = complex(input())
n2 = complex(input())
print("sum =", n1 + n2)
```

Output:

Enter two complex numbers in the form a+bj:

5+3j

5+6j

sum = (10+

3. Write a program to create a byte type array, read, modify, and display the elements of the array

```
number = 6
arr = bytes(number)
print(arr)
arr = bytes(3)
print(arr)
lis = [1, 2, 3, 4, 5]
arr = bytes(lis)
```

```
print(arr)
```

Output:

```
b'\x00\x00\x00\x00\x00\x00'                    b'\x00\x00\x00'
                b'\x01\x02\x03\x04\x05'
```

4. Create a sequence of numbers using range datatype to display 1 to 30, with an increment of 2.

```
y=range(1,30,2)
for n in y:
print(n)
```

Output: 1,3,5,7,9,11,13,15,17,19,21,23,25,27,29

5. Write a program to find out and display the common and the non common elements in the list using membership operators.

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];
if (a in list):
   print("Line 1 a is available in the given list")
else:
   print("Line 1 - a is not available in the given list")

if (b not in list):
   print("Line 2 - b is not available in the given list")
else:
   print ("Line 2 - b is available in the given list")
a = 2
if (a in list):
   print("Line 3 - a is available in the given list")
else:
   print("Line 3 - a is not available in the given list")
```

Output:

Line 1 - a is not available in the given list
Line 2 -b is not available in the given list
Line 3 - a is available in the given list

6. Create a program to display memory locations of two variables using id() function, and then use identity operators two compare whether two objects are same or not.

```
a = 20
b = 20
if (a is b):
   print("Line 1 - a and b have same identity")
else:
   print("Line 1 - a and b do not have same identity")
if (id(a) == id(b)):
   print("Line 2 - a and b have same identity")
else:
   print("Line 2 - a and b do not have same identity")
b = 30
if (a is b):
   print("Line 3 - a and b have same identity")
else:
   print("Line 3 - a and b do not have same identity")
if (a is not b):
   print("Line 4 - a and b do not have same identity")
else:
   print("Line 4 - a and b have same identity")
```

Output:

```
Line 1 - a and b have same identity
Line 2 - a and b have same identity
Line 3 - a and b do not have same identity
Line 4 - a and b do not have same identity
```

7. Write a program that evaluates an expression given by the user at run time using eval() function.

```
Example:
Enter and expression: 10+8-9*2-(10*2)
Result: -20
x=input("Enter any expression:")
print(eval(x))
```

Output:

Enter any expression: 10+8-9*2-(10*2)

-20

8. Write a python program to find the sum of even numbers using command line arguments.

```
minimum = int(input(" Please Enter the Minimum Value : "))
maximum = int(input(" Please Enter the Maximum Value : "))
even_total = 0
odd_total = 0
for number in range(minimum, maximum + 1):
    if(number % 2 == 0):
        even_total = even_total + number
    else:
        odd_total = odd_total + number
print("The Sum of Even Numbers from 1 to {0} =
  {1}".format(number, even_total))
print("The Sum of Odd Numbers from 1 to {0} =
  {1}".format(number, odd_total))
```

Output:

Please Enter the Minimum Value : 1

Please Enter the Maximum Value : 5

The Sum of Even Numbers from 1 to 5 = 6

The Sum of Odd Numbers from 1 to 5 = 9

9. Write a menu driven python program which perform the following:

Find area of circle

Find area of triangle

Find area of square and rectangle

Find Simple Interest

Exit.(Hint: Use infinite while loop for Menu)

```
choice=0
while(choice!=5):
    print("1. Find area of circle")
    print("2. Find area of triangle")
    print("3. area of square and rectangle")
```

```
print("4. Find Simple Interest")
print("5. Exit")
choice=int(input("Enter your choice"))
    if choice==1:
    PI = 3.14
    radius = float(input(' Please Enter the radius of
a circle: '))
    area = PI * radius * radius
    circumference = 2 * PI * radius
    print(" Area Of a Circle = %.2f" %area)
    print(" Circumference Of a Circle = %.2f"
%circumference)

    elif choice==2:
    a = float(input('Enter first side: '))
    b = float(input('Enter second side: '))
    c = float(input('Enter third side: '))

 # calculate the semi-perimeter
    s = (a + b + c) / 2
    # calculate the area
    area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
    print('The area of the triangle is %0.2f' %area)
elif choice==3:
    side = int(input("Enter side length of Square: "))
    area_square = side*side
    print("Area of Square =",area_square)
    width = float(input('Please Enter the Width of a
Rectangle: '))
    height = float(input('Please Enter the Height of a
Rectangle: '))
    # calculate the area
    Area = width * height
    # calculate the Perimeter
    Perimeter = 2 * (width + height)
    print("\n Area of a Rectangle is: %.2f" %Area)
```

```
        print("\n Perimeter is: %.2f" %Area)
    elif choice==4:
        p = int(input("Enter P: "))
        r = int(input("Enter R: "))
        n = int(input("Enter N: "))
        i=(p*r*n)/100
        print(i)
    elif choice==5:
        exit()
    else:
        print("Bye Bye")
```

Output:

1. Find area of circle 2. Find area
3. area of square and rectangle 4. Find Simple Interest
5. Exit

Enter your choice :4

Enter P: 10000 Enter R: 10

Enter N: 1 1000.0

10 Write a program to assert the user enters a number greater than zero.

```
num=int(input('Enter a number: '))
assert num>=0, "Only positive numbers accepted."
print('You entered: ', num)
```

Output:

Enter a number: -11

Traceback (most recent call last):

File "main.py", line 2, in <module> ssert num>=0, "Only positive numbers accepted."

AssertionError: Only

11. Write a program to search an element in the list using for loop and also demonstrate the use of "else" with for loop.

```
def search(list,n):
    for i in range(len(list)):
        if list[i] == n:
            return True
```

```
    return False

list = [1, 2, 'sachin', 4,'Python', 6]

n = 'Python'
  if search(list, n):
    print("Found")
else:
    print("Not Found")
```

Output: Found

12. Write a python program that asks the user to enter a length in centimeters. If the user enters a negative length, the program should tell the user that the entry is invalid. Otherwise, the program should convert the length to inches and print out the result. (2.54 = 1 inch).

```
cm = int(input("Enter length in cm : "))
if cm < 0:
    print("invalid entry")
else:
    print(cm/2.54, "inches")
```

Output:

Enter length in cm : -11
invalid entry
Enter length in cm : 5
1.968503937007874 inches

☺ ☺ ☺ ☺ ☺

Unit –2 Modules, Arrays, Functions, List, Tuples and Dictionaries

2.1 Modules:

A module is a python file that contains definitions and statements or a module is a collection of functions, classes and variables. It also can include runnable code.

Basically, two types of modules available in python

1. Inbuild module
2. User define module

1. **Inbuild module:** There are several built-in-modules in python, which you can import whenever you like. In this module code pre-define that you can use

   Example:

```
Import math
Print(math.sqrt(5))
```

   Output: 25

2. **User-define module:** in this module user can create own module with own rule's and also can import in another file and use it.

   Example:

```
# A simple module, calc.py
def add(x, y):
return (x+y)
def subtract(x, y):
return (x-y)
```

The *import* statement:

We can use any Python source file as a module by executing an import statement in some other Python source file.

```
# importing module calc.py
import calc
print add(10, 2)
```

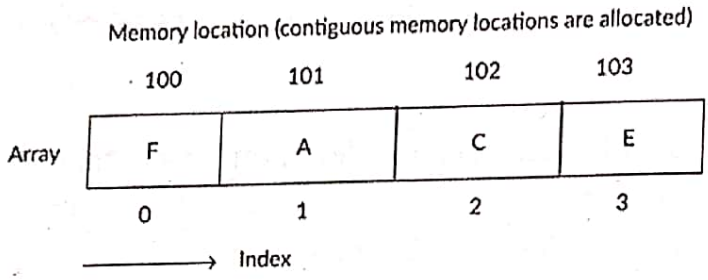
Output: 12

2.1.1 Array:

➢ **Advantages of array:**

- **Arrays** represent multiple data items of the same type using a single name.
- An **arrays**, the elements can be accessed randomly by using the index number.

- **Arrays** allocate memory in contiguous memory locations for all its elements.
- Using arrays, other data structures like linked lists, stacks, queues, trees, graphs etc. can be implemented.



➢ **Creating an array:**

- Python doesn't use array but we can use NumPy array same as array
- **NumPy** is a **python** library used for working with arrays.
- NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.
- Using NumPy we can create 1D, 2D, and 3D array
- 1D -> 1 dimensional ,2D -> 2 dimensional and so on …..
- Importing array:

```
import numpy
```

➢ **Let's Create 1-D Array**

```
import numpy
arr = numpy.array([1, 2, 3, 4, 5])
print(arr)
```

➢ Create array using list

```
List1=[1,2,3,4,5,6,7]
arr = numpy.array(list1)
```

Also, we can create an array using tuple

➢ 2-D Array

```
import numpy as np
arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr)
```

➢ 3-D Array

```
import numpy as np
arr = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]])
print(arr)
```

2.1.2 Slicing and indexing an Array:

Using slicing we can access particular element from an array and also get range of element from array

Example like

```
arr = numpy.array([1, 2, 3, 4, 5])

print(arr[0]) // indexing
                print(arr[3])

                print(arr[1:5])  //slicing
```

Output: 1
2
2,3,4

➔ **Functions:**

A function is a block of code which only runs when it is called.
You can pass data, known as parameters, into a function.
A function can return data as a result.

- **Creating a Function:**

Using def keyword we can create function in python
Example:

```
def my_function():
                print("Hello from a function")
```

- **Calling a Function:**

to call a function, use the function name followed by parenthesis
Example:

```
def my_function():
   print("Hello from a function")
my_function()
```

- **Arguments :**

Information can be passed into functions as arguments.
Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
Example 1:

```
def my_function(fname):
          print(fname + " Hello")

my_function("Email")
my_function("Windows")
my_function("Linux")
```

Example 2:

```
def my_function(fname, lname):
        print(fname + " " + lname)
my_function("Emil", "Hello")
```

Arbitrary arguments, *args

If you do not know how many arguments that will be passed into your function, add a * before the parameter name in the function definition.

Example:

```
def my_function(*kids):
        print("The youngest child is " + kids[2])

my_function("Email", "Windows", "Linux")
```

Keywords arguments

You can also send arguments with the key = value syntax.

Example:

```
def my_function(child3, child2, child1):
        print("The youngest child is " + child3)

my_function(child1 = "Email", child2 = "Windows", child3
= "Linux")
```

Default parameter value

the following example shows how to use a default parameter value.

Example:

```
def my_function(country = "India"):
    print("I am from " + country)

my_function("Sweden")
my_function("USA")
my_function()
my_function("Brazil")
```

2.1.3 Returning result from a function:

Python supports single and multi-value return

Let's Understand with an example:

```
def func(a,b):
    return a+b
        print(func(4,5))    // single-value return
        def func(a,b):
            return a*a,b*b
        print(func(4,5))    // multi-value return
```

2.1.4 Different between function and methods:

- **Method:**

1. Method is called by its name, but it is **associated to an object** (dependent).
2. A method is **implicitly passed the object** on which it is invoked.
3. It **may or may not return any data.**
4. A method **can operate on the data (instance variables) that is contained by the corresponding class**

Example:

```
class ABC:
def method_abc (self):
print("I am in method_abc of ABC class. ")
class_ref = ABC()
class_ref.method_abc()
```

- **Function:**

1. Function is block of code that is also **called by its name**. (independent)
2. The function can have different parameters or may not have any at all. If **any data (parameters)** are passed, they are **passed explicitly**.
3. It **may or may not return any data.**
4. Function does not deal with Class and its instance concept.

Example:

```
    def Subtract (a, b):
                        return (a-b)
print( Subtract(10, 12) ) # prints -2
print( Subtract(15, 6) ) # prints 9
```

2.1.5 Anonymous Functions:

An **anonymous function** is a python **function** that is defined without a name.

While normal **functions** are defined using the **def** keyword in **Python**, **anonymous functions** are defined using the **lambda** keyword. Hence, **anonymous functions** are also called **lambda functions**.

This function can have any number of arguments but only one expression, which is evaluated and returned.

Let's Understand with an example:

```
def cube(y):
                        return y*y*y;
print(cube(5)) // with def() function
                cube = lambda x: x*x*x
```

```
print(cube(7)) // with lambda() function
```

So, above example you can understand different between lambda and def function

Now let's use lambda function with filter, map, reduce

- **Filter**: The filter() function in Python takes in a function and a list as arguments. This offers an elegant way to filter out all the elements of a sequence "sequence", for which the function returns True. Here is a small program that returns the odd numbers from an input list:

```
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(filter(lambda x: (x%2 != 0) , li))
print(final_list)
```

Output: [5, 7, 97, 77, 23, 73, 61]

- **map**: The map() function in Python takes in a function and a list as argument. The function is called with a lambda function and a list and a new list is returned which contains all the lambda modified items returned by that function for each item.

Example:

```
# map() with lambda()
# to get double of a list.
li = [5, 7, 22, 97, 54, 62, 77, 23, 73, 61]
final_list = list(map(lambda x: x*2 , li))
print(final_list)
```

Output: [10, 14, 44, 194, 108, 124, 154, 46, 146, 122]

- **reduce**: The reduce() function in Python takes in a function and a list as argument. The function is called with a lambda function and a list and a new reduced result is returned. This performs a repetitive operation over the pairs of the list. This is a part of fun tools module. Example:

```
# reduce() with lambda()
# to get sum of a list
from functools import reduce
li = [5, 8, 10, 20, 50, 100]
sum = reduce((lambda x, y: x + y), li)
print (sum)
```

Output: 193

2.1.6 Function Decorators:

Functions can be defined inside another function and can also be passed as argument to another function.

Functions are objects; they can be referenced to, passed to a variable and returned from other functions as well.

Decorators are very powerful and useful tool in Python since it allows programmers to modify the behavior of function or class. Decorators allow us to wrap another function in order to extend the behavior of wrapped function, without permanently modifying it.

Example:

```
# defining a decorator
def hello_decorator(func):

    def inner1():
    print("Hello, this is before function execution")
    func()
    print("This is after function execution")
    return inner1
def function_to_be_used():
print("This is inside the function !!")
function_to_be_used =
hello_decorator(function_to_be_used)
function_to_be_used()
```

Output:

Hello, this is before function execution

This is inside the function!!

This is after function execution

2.2 List:

List is nothing but collection of elements where elements can be integer, float, string and complex number.

List is mutable means its allows to modified after created list and its also contains duplicate members

List is a collection which is ordered and changeable.

2.2.1 create a list:

```
thislist = ["Ram", "Shyam", 10]
print(thislist)
```

- **Access items from list:**
  Print the second item of the list:

```
thislist = [10, 12, "cherry"]
print(thislist[1])
```

output: 12

- **Negative indexing:**

  Print the last item of the list:

```
thislist = [10, 12, "cherry"]
print(thislist[-1])
```

  output: cherry

- **Slicing of list:**

  Return the third, fourth, and fifth item:

```
thislist =
["apple", "banana", "cherry", "orange", "kiwi", "melon", "
mango"]
print(thislist[2:5])
```

  output: ["cherry", "orange", "kiwi"]

2.2.2 Create list using range() function:

```
My_list = [range(10, 20, 1)]
# Print the list
print(My_list)
here my range start from 10 and end with 20(n-1)(19) with
differentiate is 1
```

output:

[10,11,12,13,14,15,16,17,18,19]

2.2.3 Updating the elements of the list:

```
mylist=[1,12,33,45,57,68,77]
mylist[3]=20 // 20 value update at index=3
Print(mylist)
```

Output: [1,12,33,20,57,68,77]

2.2.4 Concatenation of two lists:

Let's see how to concatenate two lists using different methods in Python. This operation is useful when we have numbers of lists of elements which needs to be processed in a similar manner.

Method #1 : Using + operator

```
test_list3 = [1, 4, 5, 6, 5]
test_list4 = [3, 5, 7, 2, 5]
test_list3 = test_list3 + test_list4
print ("Concatenated list using + : ",test_list3)
```

output: Concatenated list using + : [1, 4, 5, 6, 5, 3, 5, 7, 2, 5]

Method #2 : Using Naive Method

In this method, we traverse the second list and keep appending elements in the first list, so that first list would have all the elements in both lists and hence would perform the append.

```
test_list1 = [1, 4, 5, 6, 5]
test_list2 = [3, 5, 7, 2, 5]
for i in test_list2 :
 test_list1.append(i)
print ("Concatenated list using naive method :
",test_list1)
```

output: Concatenated list using naive method : [1, 4, 5, 6, 5, 3, 5, 7, 2, 5]

2.2.5 Membership in lists:

Membership operators are operators used to validate the membership of a value. It tests for membership in a sequence, such as strings, lists, or tuples.

1. **in operator :** The 'in' operator is used to check if a value exists in a sequence or not. Evaluates to true if it finds a variable in the specified sequence and false otherwise.

Finding common member in list

```
list1=[1,2,3,4,5]
list2=[6,7,8,9]
for item in list1:
 if item in list2:
      print("overlapping")
else:
 print("not overlapping")
```

output: not overlapping

2. **'not in' operator-** Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.

```
# not 'in' operator
x = 24
y = 20
list = [10, 20, 30, 40, 50 ]

if ( x not in list ):
print("x is NOT present in given list")
```

```
else:
print("x is present in given list")
if ( y in list ):
print("y is present in given list")
else:
print("y is NOT present in given list")
```

2.2.6 Aliasing and Cloning lists:

1. Using slicing technique:

This is the easiest and the fastest way to clone a list. This method is considered when we want to modify a list and also keep a copy of the original. In this we make a copy of the list itself, along with the reference. This process is also called cloning.

```
def Cloning(li1):
 li_copy = li1[:]
 return li_copy
li1 = [4, 8, 2, 10, 15, 18]
li2 = Cloning(li1)
print("Original List:", li1)
print("After Cloning:", li2)
```

output: Original List: [4, 8, 2, 10, 15, 18]
After Cloning: [4, 8, 2, 10, 15, 18]

2. Using the extend() method:

The lists can be copied into a new list by using the extend() function. This appends each element of the iterable object to the end of the new list.

```
def Cloning(li1):
 li_copy = []
 li_copy.extend(li1)
 return li_copy
li1 = [4, 8, 2, 10, 15, 18]
li2 = Cloning(li1)
print("Original List:", li1)
print("After Cloning:", li2)
```

output: Original List: [4, 8, 2, 10, 15, 18]
After Cloning: [4, 8, 2, 10, 15, 18]

2.2.7 Using the list() method:

This is the simplest method of cloning a list by using the built-in function list().

```
def Cloning(li1):
 li_copy = list(li1)
 return li_copy
li1 = [4, 8, 2, 10, 15, 18]
li2 = Cloning(li1)
print("Original List:", li1)
print("After Cloning:", li2)
```

output: Original List: [4, 8, 2, 10, 15, 18]
After Cloning: [4, 8, 2, 10, 15, 18]

2.2.8 List methods in python:

1. **append():** Used for appending and adding elements to List. It is used to add elements to the last position of List.

Example:

```
List = ['Mathematics', 'chemistry', 1997, 2000]
List.append(20544)
print(List)
```

2. **insert():** Inserts an elements at specified position.

Example:

```
List = ['Mathematics', 'chemistry', 1997, 2000]
# Insert at index 2 value 10087
List.insert(2,10087)
print(List)
```

3. **extend():** Adds contents to List2 to the end of List1

Example:

```
List1 = [1, 2, 3]
List2 = [2, 3, 4, 5]
# Add List2 to List1
List1.extend(List2)
print(List1)

# Add List1 to List2 now
List2.extend(List1)
print(List2)
```

4. **sum() :** Calculates sum of all the elements of List.

Example:

```
List = [1, 2, 3, 4, 5]
print(sum(List))
```

5. **count():**Calculates total occurrence of given element of List

Example:

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(List.count(1))
```

6. **length:**Calculates total length of List.

Example:

```
List = [1, 2, 3, 1, 2, 1, 2, 3, 2, 1]
print(len(List))
```

So, here many more methods available in list that you can use

2.2.9 Nested list:

O. Nested list means one list inside another list.

Let's take one example

Example:

```
List1=[1,2,3,4,5,['A'.'B','C']]
      Or
List2=[12,13,14,List1]
```

2.2.10 Tuples:

A Tuple is a collection of Python objects separated by commas. In some ways a tuple is similar to a list in terms of indexing, nested objects and repetition but a tuple is immutable unlike lists which are mutable.

2.2.11 Creating Tuples:

```
# An empty tuple
empty_tuple = ()
print (empty_tuple)
```

output: ()

```
# Creating non-empty tuples
# One way of creation
tup = 'python', 'hello'
print(tup)
# Another for doing the same
tup = ('python', 'hello')
```

```
print(tup)
```

output:

('python', 'hello')

('python', 'hello')

- **Concatenation two tuples :**

```
tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'hello')
print(tuple1 + tuple2)
```

output: (0, 1, 2, 3, 'python', 'hello')

2.2.12 Nesting of tuples:

```
tuple1 = (0, 1, 2, 3)
tuple2 = ('python', 'hello')
tuple3 = (tuple1, tuple2)
print(tuple3)
```

Output: ((0, 1, 2, 3), ('python', 'hello'))

- **Repetition in tuples:**

```
tuple3 = ('python',) *3
print(tuple3)
```

Output: ('python', 'python', 'python')

- **Immutable tuples**

#code to test that tuples are immutable

```
tuple1 = (0, 1, 2, 3)
tuple1[0] = 4
print(tuple1)
```

Output: TypeError: 'tuple' object does not support item assignment

- **Slicing in tuples**

```
tuple1 = (0 ,1, 2, 3)
print(tuple1[1:])
print(tuple1[::-1])
print(tuple1[2:4])
```

Output:

(1, 2, 3)

(3, 2, 1, 0)

(2, 3)

- **Deleting a tuple:**
  Using **del** keyword we can delete any tuples

```
tuple3 = ( 0, 1)
del tuple3
print(tuple3)
```

  Output: NameError: name 'tuple3' is not defined

- **Covert list to tuple:**
  one can also convert list to tuple using tuple () method

```
List1=[1,2,3,4,5]
tup=tuple(List1)
print(tup)
```

- **Covert tuple to list**
  we can also convert tuple to list using list() method

```
tup1=(1,2,3,4,5)
list1=list(tup1)
print(list1)
```

2.3 Introduction to Dictionaries:

Python have some variety of built-in data structures, capable of storing different types of data. It is a type of data structure that can store data in the form of key-value pairs. The values in a dictionary can be accessed using the keys.

2.3.1 Creating a Dictionary:

For create a Python dictionary, we need to pass a sequence of different types of data inside curly braces {}, and separate them using a comma (,). Each item has a key and a value expressed as a "key:value" pair.

The values can belong to any data type and they repeat also, but the keys must remain unique.

The following examples demonstrate how to create dictionaries:

Creating an empty dictionary:

dict_sample = {}

Creating a dictionary with integer keys:

dict_sample = {1:'mango',2:'banana'}

Creating a dictionary with mixed keys:

dict_sample = {'fruit' : 'mango' , 1:[2,4,6]}

We can also create a dictionary by explicitly calling the method dict();

dict_sample = dict({1:'mango',2:'banana'})

or it can also be created from a sequence

dict_sample = dict([(1,'mango'),(2,'banana')])

Dictionaries can also be nested, which means that we can create a dictionary inside another dictionary. Example like:

dict_sample = {1:{'fruit1':'banana','fruit2':'mango','fruit3':'Apple'},2:{'vegetable1':'Potato','vegetable2':'Tomato','vegetable3':'Bringle'}}

For printing the dictionary contents, we can use the function print() and pass the dictionary name as the argument to the function. For example:

print(dict_sample) gives

output : - 1:'mango',2:'banana'

2.3.2 Operations on Dictionaries:

- **Accessing Elements from dictionaries:**

  To access dictionary items, pass the key inside square brackets [].

  For example:

```
dict_sample={"type":"Petrol","Model":"Prime","year" : 2005}
X = dict_sample["Model"]
```

  dictionary named dict_sample having variable named X was then created and its value is set to be the value for the key "Model" in the dictionary.

  The dictionary object also provides the function name get(), which can be used to access dictionary elements as well. We append the function with the dictionary name using the dot(.) operator and then pass the name of the key as the argument to the function. For example:

```
dict_sample ={"type":"Petrol","Model":"Prime","year" :2005}
x = dict_sample.get("Model")
print(X)
```

  Output : Prime

  In the next section we'll discuss how to add new elements to an already existing dictionary.

- **Adding Elements to the existing dictionary:**

  There are numerous ways to add new elements to a dictionary. We can use a new index key and assign a value to it. For example:

```
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
dict_sample["Name "] ="HondaCity"
print(dict_sample)
```

Output: {'Name':'HondaCity','type':'Petrol','Model':'Prime','year' :2005}

The new element has "Name" as the key and 'HondaCity' as its corresponding value. It has been added as the first element of the dictionary.

The dictionary returns nothing as it has nothing stored yet. Let us add some elements to it, one at a time:

```
MyDictionary[0] = 'Apples'
MyDictionary[2] = 'Mangoes'
MyDictionary[3] = 20
print("\n 3 New elements have been added: ")
print(MyDictionary)
```

Output:

3 New elements have been added:

{0: 'Apples', 2: 'Mangoes', 3: 20}

To add the elements, we specified keys as well as the corresponding values. For example:

```
MyDictionary[0] = 'Apples'
```

In the above example, 0 is the key while "Apples" is the value.

It is even possible for us to add a set of values to one key. For example:

```
MyDictionary['Values'] = 1, "with pair", 4
print("\n3 elements have been added: ")
print(MyDictionary)
```

Output:

3 elements have been added:

{'Values': (1, 'with pair', 4)}

In the above example, the name of the key is "Values" while everything after the = sign are the actual values for that key, stored as a Set. Other than adding new elements to a dictionary, dictionary elements can also be updated/changed.

- **Updating Elements:**

  After adding a value to a dictionary, we can then change the existing dictionary element. You use the key of the element to change the corresponding value. For example:

```
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
dict_sample["year "] ="2010"
print(dict_sample)
```

Output:

{'year': 2010, 'type':'Petrol','Model':'Prime'}

In this example you can see that we have updated the value for the key "year" from the old value of 2005 to a new value of 2010.

- **Removing Elements from Dictionary:**

The removal of an element from a dictionary can be done in several ways, which we'll discuss one-by-one in this section:

The del keyword can be used to remove the element with the specified key.

For example:

```
dict_sample = {
  "Company": "Toyota",
  "model": "Premio",
  "year": 2012
}
del dict_sample["year"]
print(dict_sample)
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
dict_sample["year "] ="2010"
print(dict_sample)
```

Output:

{'year': 2010, 'type':'Petrol','Model':'Prime'}

In this example you can see that we have updated the value for the key "year" from the old value of 2005 to a new value of 2010.

- **Removing Elements from Dictionary**

The removal of an element from a dictionary can be done in several ways, which we'll discuss one-by-one in this section:

The del keyword can be used to remove the element with the specified key.

For example:

```
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
del dict_sample["year"]
```

```
print(dict_sample)
```

Output:

{'type': 'Toyota', 'model': 'Premio'}

We called the del keyword followed by the dictionary name. Inside the square brackets that follow the dictionary name, we passed the key of the element we need to delete from the dictionary, which in this example was "year". The entry for "year" in the dictionary was then deleted.

Another way to delete a key-value pair is to use the pop() function and pass the key of the entry to be deleted as the argument to the function.

For example:

```
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
dict_sample.pop("year")
print(dict_sample)
```

Output:

{'type': 'Toyota', 'model': 'Premio'}

We invoked the pop() function by appending it with the dictionary name. Again, in this example the entry for "year" in the dictionary will be deleted.

The popitem() function removes the last item inserted into the dictionary, without needing to specify the key. Take a look at the following example:

```
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
dict_sample.popitem()
print(dict_sample)
```

Output:

{'type': 'Toyota', 'model': 'Premio'}

The last entry into the dictionary was "year". It has been removed after calling the popitem() function.

But what if you want to delete the entire dictionary? It would be difficult to use one of these methods on every single key. Instead, you can use the del keyword to delete the entire dictionary. For example:

```
dict_sample =
{"type":"Petrol","Model":"Prime","year":"2005"}
del dict_sample
print(dict_sample)
```

Output:

NameError: name 'dict_sample' is not defined

The code returns error because we are trying to print that dictionary element which is not exist.

However, your use-case may require you to just remove all dictionary elements and be left with an empty dictionary. This can be achieved by calling the clear() function on the dictionary:

```
dict_sample = {"type":"Petrol","Model":"Prime","year"
:2005}
dict_sample.clear()
print(dict_sample)
```

Output:

{}

The code returns an empty dictionary since all the dictionary elements have been removed.

2.3.3 Built-in Dictionary Methods:

There are several built-in methods that can be invoked on dictionaries. In fact, in some of cases, the list and dictionary methods share the same name. (In the discussion on object-oriented programming, you will see that it is perfectly acceptable for different types to have methods with the same name.)

The following is an overview of methods that apply to dictionaries:

- **d.clear() : -** To clears a dictionary. It empties all key-value pairs of dictionary.

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> d.clear()
>>> d
{}
```

- **d.get(key,default=none) :-** Which returns the value for a key if it exists in the dictionary.

The Python dictionary. get() method provides a convenient way of getting the value of a key from a dictionary without checking ahead of time whether the key exists, and without occurring an error.

`d.get(<key>)` searches dictionary `d` for `<key>` and returns the associated value if it is found. If `<key>` is not found, it returns `None`:

```
>>>
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> print(d.get('b'))
```

```
20
>>> print(d.get('z'))
None
```

If `<key>` is not found and the optional `<default>` argument is specified, that value is returned instead of `None`

```
>>> print(d.get('z', -1))
-1
```

- **d.items() :-** Which returns a list of key-value pairs in a dictionary. The first item in each tuple is the key, and the second item is its value:

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> list(d.items())
[('a', 10), ('b', 20), ('c', 30)]
>>> list(d.items())[1][0]
'b'
>>> list(d.items())[1][1]
20
```

- **d.keys() :-** Which returns a list/all of keys in a dictionary.

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> list(d.keys())
['a', 'b', 'c']
```

- **d.values() :-** Which returns a list of all values in a dictionary.duplicate values also may occur as many times.

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d
{'a': 10, 'b': 20, 'c': 30}
>>> list(d.values())
[10, 20, 30]
>>> d = {'a': 10, 'b': 10, 'c': 10}
>>> d
{'a': 10, 'b': 10, 'c': 10}
>>> list(d.values())
```

[10, 10, 10]

- **d.pop(<key>[,<default>]) :-** Which removes a key from a dictionary, if it is present, and returns its value.

If <key> is present in d, d.pop(<key>) removes <key> and returns its associated value: d.pop(<key>) raises a error if <key> is not in dictionary.

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d.pop('b')
20
>>> d
{'a': 10, 'c': 30}
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d.pop('z')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
      d.pop('z')
```

- **KeyError:** 'z'

If <key> is not in d, and the optional <default> argument is specified, then that value is returned, and no exception is raised:

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d.pop('z', -1)
-1
>>> d
{'a': 10, 'b': 20, 'c': 30}
```

- **d.popitem() :-** Which removes a last key-value pair added from d and returns it as a tuple from a dictionary. If d is empty than it gives a key Error exception.

```
>>> d = {'a': 10, 'b': 20, 'c': 30}
>>> d.popitem()
('c', 30)
>>> d
{'a': 10, 'b': 20}
>>> d.popitem()
('b', 20)
>>> d
{'a': 10}
>>> d = {}
```

```
>>> d.popitem()
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    d.popitem()
```

- **KeyError: 'popitem():** dictionary is empty'

d.update(<obj>): - Which merges a dictionary with another dictionary or with an iterable of key-value pairs.

If `<obj>` is a dictionary, `d.update(<obj>)` merges the entries from `<obj>` into `d`.

For each key in `<obj>`:

- If the key is not present in d, the key-value pair from <obj> is added to d.
- If the key is already present in d, the corresponding value in d for that key is updated to the value from <obj>.

Here is an example showing two dictionaries merged together:

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d2 = {'b': 200, 'd': 400}
>>> d1.update(d2)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

In this example, key `'b'` already exists in `d1`, so the value updated from 20 to 200, the value for that key from `d2`. However, there is no key `'d'` in `d1`, so that key-value pair is added from `d2`.

`<obj>` may also be a sequence of key-value pairs, similar to when the `dict()` function is used to define a dictionary. For example, `<obj>` can be specified as a list of tuples:

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d1.update([('b', 200), ('d', 400)])
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

Or the values to merge can be specified as a list of keyword parameter:

```
>>> d1 = {'a': 10, 'b': 20, 'c': 30}
>>> d1.update(b=200, d=400)
>>> d1
{'a': 10, 'b': 200, 'c': 30, 'd': 400}
```

2.3.4 Converting List into Dictionary:

Lists and **dictionaries** are two most frequently used types of python. As you have seen, they have several similarities, but differ in accessing the elements. Lists elements are accessed by numerical index based on order, and dictionary elements are accessed by key.

Because of this difference, lists and dictionaries tend to be appropriate for different conditions. You should now have a good feel for which, if either, would be best for a given situation.

If you have worked for a while with Python, at the time of conversion of lists into dictionaries or vice versa. It wouldn't be too hard to write a function doing this. If we have a dictionary

{"list":"Liste", "dictionary":"Wörterbuch", "function":"Funktion"}

we could turn this into a list with two or more than two-tuples:

[("list","Liste"), ("dictionary","dictionary"), ("function","Funktion")]

Now we will turn show, how to turn lists into dictionaries, but it if these lists satisfy certain conditions.

We have two lists, one containing the dishes and the other one the corresponding countries:

```
>>> dishes = ["pizza", "sauerkraut", "paella", "Hamburger"]
>>> countries = ["Italy", "Germany", "Spain", "USA"]
```

Now we will create a dictionary, which assigns a dish to a country, of course according to the common prospective. For this purpose we need the function zip(). The name zip was well chosen, because the two lists get combined like a zipper.

```
>>> country_specialities = zip(countries, dishes)
>>> print country_specialities
[('Italy', 'pizza'), ('Germany', 'sauerkraut'), ('Spain', 'paella'), ('USA', 'Hamburger')]
>>>
```

The variable country specialities contains now the "dictionary" in the 2-tuple list form. This form can be easily transformed into a real dictionary with the function dict().

```
>>> country_specialities_dict = dict(country_specialities)
>>> print country_specialities_dict
{'Germany': 'sauerkraut', 'Spain': 'paella', 'Italy': 'pizza', 'USA': 'Hamburger'}
```

There is one problem concerning with function zip(). What happened if one of the two arguments lists contain more elements that the other one. The answer is simple that we are not used the extra elements.

```
>>> countries = ["Italy", "Germany", "Spain", "USA", "Switzerland"]
```

```
>>> dishes = ["pizza", "sauerkraut", "paella", "Hamburger"]
>>> country_specialities = zip(countries,dishes)
>>> print country_specialities
```

So, in this course, we will not answer the question, that what the national dish of Switzerland is.

2.3.5 Passing dictionaries to functions:

In the programming, For the reusability of code we call a function with the specific value, which is called a function argument in python.

Example 1: Function to add 3 numbers

```
def adder(x,y,z):
    print("sum:",x+y+z)
adder(10,12,13)
output : - sum: 35
```

In above program we have adder() function with three arguments x, y and z. When we pass three values while calling adder() function, we get sum of the 3 numbers as the output.

Lets see what happens when we pass more than 3 arguments in the adder() function.

```
def adder(x,y,z):
  print("sum:",x+y+z)
adder(5,10,15,20,25)
```

When we run the above program, the output will be

TypeError: adder() takes 3 positional arguments but 5 were given

In the above program, we passed 5 arguments to the adder() function instead of 3 arguments due to which we got TypeError

Now , Overcome from this problem we will use *args and **kwargs in python.

Introduction to *args and **kwargs in Python

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols.

*args (Non Keyword Arguments)

**kwargs (Keyword Arguments)

We use *args and **kwargs as an argument when we are not sure about the number of arguments to pass in the functions.

Python *args

1. As in the above example we are not sure about the number of arguments that can be passed to a function. Python has *args which allow us to pass the variable number of non keyword arguments to function.
2. In the function, we should use an asterisk * before the parameter name to pass variable length arguments.The arguments are passed as a tuple and these passed arguments make tuple inside the function with same name as the parameter excluding asterisk *.

Example 2: Using *args to pass the variable length arguments to the function

```
def adder(*num):
      sum = 0
           for n in num:
          sum = sum + n
           print("Sum:",sum)
       adder(3,5)
 adder(4,5,6,7)
 adder(1,2,3,5,6)
```

the output will be

```
Sum: 8
Sum: 22
Sum: 17
```

In the above program, we used *num as a parameter which allows us to pass variable length argument list to the adder() function. Inside the function, we have a loop which adds the passed argument and prints the result. We passed 3 different tuples with variable length as an argument to the function.

Python **kwargs

Python passes variable length non keyword argument to function using *args but we cannot use this to pass keyword argument. For this problem Python has got a solution called **kwargs, it allows us to pass the variable length of keyword arguments to the function.

In the function, we use the double asterisk ** before the parameter name to denote this type of argument. The arguments are passed as a dictionary and these arguments make a dictionary inside function with name same as the parameter excluding double asterisk **.

Example 3: Using **kwargs to pass the variable keyword arguments to the function

```
def intro(**data):
    print("\nData type of argument:",type(data))
```

```
        for key, value in data.items():
            print("{} is {}".format(key,value))
    intro(Firstname="Sita",  Lastname="Sharma",  Age=22,
Phone=1234567890)
    intro(Firstname="Ram",Lastname="Sharma",Email="ram@rmail.
    com", Country="Egypt", Age=25, Phone=9876543210)
```

the output will be

Data type of argument: <class 'dict'>
Firstname is Sita
Lastname is Sharma
Age is 22
Phone is 1234567890
Data type of argument: <class 'dict'>
Firstname is Ram
Lastname is Sharma
Email is ram@rmail.com
Country is Wakanda
Age is 25
Phone is 9876543210

In the above program, we have a function intro() with **data as a parameter. We passed two dictionaries with variable argument length to the intro() function. We have for loop inside intro() function which works on the data of passed dictionary and prints the value of the dictionary.

☺ ☺ ☺ ☺ ☺

Exercises

- **MCQ Question /Answer:**

1. What will be the output of the following code :

   print type(type(int))

   a) type 'int'
   b) type 'type'
   c) Error
   d) 0

2. What is the output of the following code :

   L = ['a','b','c','d']

print "".join(L)

a) Error
b) None
c) abcd
d) ['a','b','c','d']

3. What is the output of the following segment :
chr(ord('A'))

a) A
b) B
c) a
d) Error

4. What is called when a function is defined inside a class?

a) Module
b) Class
c) Another Function
d) Method

5. Which of the following is the use of id() function in python?

a) Id returns the identity of the object
b) Every object doesn't have a unique id
c) All of the mentioned
d) None of the mentioned

6. Suppose list1 is [3, 4, 5, 20, 5, 25, 1, 3], what is list1 after list1.pop(1)?

a) .[3, 4, 5, 20, 5, 25, 1, 3]
b) [1, 3, 3, 4, 5, 5, 20, 25]
c) [3, 5, 20, 5, 25, 1, 3]
d) [1, 3, 4, 5, 20, 5, 25]

7. time.time() returns _____

a) the current time
b) the current time in milliseconds since midnight, January 1, 1970 GMT (the Unix time)
c) the current time in milliseconds since midnight
d) the current time in milliseconds since midnight, January 1, 1970

8. Which of these definitions correctly describes a module?

a) Denoted by triple quotes for providing the specification of certain program elements
b) Design and implementation of specific functionality to be incorporated into a program

c) Defines the specification of how it is to be used
d) Any program that reuses code

9. Program code making use of a given module is called a _____ of the module.
a) Client
b) Interface
c) Modularity
d) None of above

10. Which of the following is true about top-down design process?
a) The details of a program design are addressed before the overall design
b) Only the details of the program are addressed
c) The overall design of the program is addressed before the details
d) Only the design of the program is addressed

11. Which of the following isn't true about main modules?
a) When a python file is directly executed, it is considered main module of a program.
b) Main modules may import any number of modules
c) Special name given to main modules is: __main__
d) Other main modules can import main modules

12. What will be the output of the following Python code?
```
from math import factorial
print(math.factorial(5))
```
a) 120
b) Nothing is printed
c) Error, the statement should be: print(factorial(5))
d) Error, method factorial doesn't exist in math module

13. Suppose listExample is ['h','e','l','l','o'], what is len(listExample)?
a) 5
b) 4
c) None
d) Error

14. Suppose list1 is [1, 5, 9], what is sum(list1)?
a) 1
b) 9
c) 15
d) Error

15. Which of the following is a Python tuple?

a) [1, 2, 3]
b) (1, 2, 3)
c) {1, 2, 3}
d) {}

16. Suppose t = (1, 2, 4, 3), which of the following is incorrect?
a) print(t[3])
b) print(len(t))
c) print(max(t))
d) t[3] = 45

17. What will be the output of the following Python code?
>>>t = (1, 2, 4, 3, 8, 9)
>>>[t[i] for i in range(0, len(t), 2)]
a) [2, 3, 9]
b) [1, 2, 4, 3, 8, 9]
c) [1, 4, 8]
d) (1, 4, 8)

18. What type of data is: a=[(1,1),(2,4),(3,9)]?
a) Array of tuples
b) List of tuples
c) Invalid type
d) Tuples of lists

19. Is the following Python code valid?
>>> a,b=1,2,3
a) Yes, this is an example of tuple unpacking. a=1 and b=2
b) No, too many values to unpack
c) Yes, this is an example of tuple unpacking. a=(1,2) and b=3
d) Yes, this is an example of tuple unpacking. a=1 and b=(2,3)

20. Suppose d = {"abc":10, "pqr":20}. To obtain the number of entries in dictionary which command do we use?
a) d.size()
b) len(d)
c) size(d)
d) d.len()

21. Which of these about a dictionary is false?
a) The keys of a dictionary can be accessed using values
b) The values of a dictionary can be accessed using keys

c) Dictionaries aren't ordered
d) Dictionaries are mutable

22. Which of the following isn't true about dictionary keys?
a) Keys must be integers
b) More than one key isn't allowed
c) Keys must be immutable
d) When duplicate keys encountered, the last assignment wins

23. What will be the output of the following Python code?

```
a={1:5,2:3,3:4}
print(a.pop(4,9))
```

a) 3
b) 9
c) Too many arguments for pop() method
d) 4

Question :-1 Answer the Following Questions (7 marks):

1) What is dictionary in Python? Explain with an example.
2) What is lambda function in python? Explain lambda function with example. Explain filter(), map() and reduce() functions in brief.
3) What is Decorators? What is Generators? Explain both with example
4) Give a comparison between List, Tuple and Dictionaries in Python.
5) What do you know about the <list> and <dict> comprehensions? Explain with an example.
6) What is lambda function in python? Support your answer with appropriate code.
7) Explain different ways of importing module in python.
8) What do you mean by Anonymous function? Explain map() function with reference to lambda function.
9) Explain different types of arguments in python function

Question : - 2 Answer the following questions (3 Marks):

1) How to create an Array and write advantages of it.
2) How to assign indexing and slicing in Array? Explain with example.
3) Explain with example Anonymous function.
4) State the difference between List and Tuples.
5) Write the different functions apply on tuples with program code.
6) Write all dictionary methods.
7) How to pass dictionaries to functions.

☺ ☺ ☺ ☺ ☺

CC-305 Unit – 2 Python Programming Practicals

1) Write a program to create one array from another array.

```
x = [24, 27, 30, 29, 18, 14]
print("Original array:")
print(x)
y = []
y[:] = x
print("New Array:")
print(y)
```

Output : -

Original array:
[24, 27, 30, 29, 18, 14]
New Array:
[24, 27, 30, 29, 18, 14]

2) Create a program to retrieve, display and update only a range of elements from an array using indexing and slicing in arrays.

```
import array as arr
a=arr.array('i',[1,2,3,4,5])
# Display array
print(a)
#update array using indexing
a[0]=7
print(a)
#uodate    array using slicing
a[0:2]= arr.array('i',[8,9])
print(a)
```

Output : -

array('i', [1, 2, 3, 4, 5])
array('i', [7, 2, 3, 4, 5])
array('i', [8, 9, 3, 4, 5])

3) Write a program to understand various methods of array class mentioned: append, insert, remove, pop, index, tolist and count.

```
list1=[1,2,3,4,5,6,7]
list1.append(10)
list1.insert(2,20)
print(list1)
list1.remove(2)
print(list1)
list1.pop()
print(list1)
print(list1.index(3))
print(list1.count(4))
import numpy as np
arr = np.array([1, 2, 3, 4, 5])
print(arr.tolist())
```

Output : -

[1, 2, 20, 3, 4, 5, 6, 7, 10]
[1, 20, 3, 4, 5, 6, 7, 10]
[1, 20, 3, 4, 5, 6, 7]
2
1

4) Write a program to sort the array elements using bubble sort technique.

```
def bubbleSort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
arr = [64, 34, 25, 12, 22, 11, 90]
bubbleSort(arr)
print ("Sorted array is:")
for i in arr:
    print (i),
```

Output : -

Sorted array is:
11

12
22
25
34
64
90

5) Create a program to search the position of an element in an array using index() method of array class.

```
list1=['hello','A','B','C','D']
print(list1)
position=list1.index(input("Enter Element from above list:
  "))
print(position)
```

Output : -

['hello', 'A', 'B', 'C', 'D']
Enter Element from above list: A
1

6) Write a program to generate prime numbers with the help of a function to test prime or not.

```
def test_prime(num):
    if num > 1:
        for i in range(2,num):
            if (num % i) == 0:
                print(num,"is not a prime number")
                break
        else:
            print(num,"is a prime number")
    else:
        print(num,"is not a prime number")
num=int(input("Enter Number to check prime or not:"))
test_prime(num)
```

Output : -

Enter Number to check prime or not:11
11 is a prime number

7) Write a python program that removes any repeated items from a list so that each item appears at most once. For instance, the list [1,1,2,3,4,3,0,0] would become [1,2,3,4,0].

```
li=[1,2,3,3,4,2,5,5,5,6,6,7,7,12,12]
li2=[]
for i in li:
    if i not in li2:
        li2.append(i)
print(li2)
```

Output : -

[1, 2, 3, 4, 5, 6, 7, 12]

8) Write a program to pass a list to a function and display it.

```
def display_list(list1):
    for i in list1:
        print(i)
list1=[1,2,3,4,5,6,7,8,9]
display_list(list1)
```

Output : -

1
2
3
4
5
6
7
8
9

9) Write a program to demonstrate the use of Positional argument, keyword argument and default arguments.

```
# positional arguement
def positional(ram,shyam,harish):
    print(ram,shyam,harish)
a='Ram'
b='Shyam'
c='Harish'
```

```
positional(a,b,c)
# Keyword Arguement
def Keyword(ram,shyam,harish):
    print(ram,shyam,harish)
a='Ram'
b='Shyam'
c='Harish'
Keyword(ram=a,harish=c,shyam=b)
# Default arguement
def Keyword(ram,shyam,harish='harish'):
    print(ram,shyam,harish)
a='Ram'
b='Shyam'
Keyword(ram=a,shyam=b)
```

Output : -

Ram Shyam Harish
Ram Shyam Harish
Ram Shyam harish

10) Write a program to show variable length argument and its use.

```
def variable_length(*args):
    for value in range(0,len(args)):
        print(args[value])

variable_length("A","B","C","D","E")
```

Output : -

A
B
C
D
E

11) Write a lambda/Anonymous function to find bigger number in two given numbers.

```
val=lambda x,y:max(x,y)
print(val(120,123))
```

Output : -

123

12) Create a decorator function to increase the value of a function by 3.

```
def plus_one(number):
    def add_one(number):
        return number + 3
    result = add_one(number)
    return result
```

Output : -

```
>>> plus_one(3)
6
```

13) Create a program name "employee.py" and implement the functions DA, HRA, PF, and ITAX. Create another program that uses the function of employee module and calculates gross and net salaries of an employee.

```
import employee
print("SALARY PROGRAM")
name= str(input("Enter name of employee:"))
basic=float(input("Enter Basic Salary :"))
netpay=employee.netPay(basic)
print(f'Net Salary: {netpay}')
grospay=employee.grossPay(basic,netpay)
print(f'gross Salary: {grosspay}')
```

Output : -

SALARY PROGRAM
Enter name of employee:xyz
Enter Basic Salary :12345
Net Salary: 18208.875
gross Salary: 16357.125

14) Write a program to create a list using range functions and perform append, update and delete elements operations in it.

```
list1=[i for i in range(0,9)]
print(list1)
list1.append(12)
```

```
print(list1)
#update
list1.insert(1,15)
print(list1)
#delete
list1.remove(7)
print(list1)
```

Output : -

[0, 1, 2, 3, 4, 5, 6, 7, 8]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 12]
[0, 15, 1, 2, 3, 4, 5, 6, 7, 8, 12]
[0, 15, 1, 2, 3, 4, 5, 6, 8, 12]

15) Write a program to combine two List, perform repetition of lists and create cloning of lists.

```
list1=[ i for i in range(1,6)]
list2=[ i for i in range(7,11)]
print(list1)
print(list2)
list3=list1+list2
print(list3)
#repetition of list
new_list=[i for i in list3 for x in (0, 1)]
print(new_list)
#cloning of list
list_cloning=list3[:]
print(list_cloning)
```

Output : -

[1, 2, 3, 4, 5]
[7, 8, 9, 10]
[1, 2, 3, 4, 5, 7, 8, 9, 10]
[1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 7, 7, 8, 8, 9, 9, 10, 10]
[1, 2, 3, 4, 5, 7, 8, 9, 10]

16) Create a sample list of 7 elements and implement the List methods mentioned: append, insert, copy, extend, count, remove, pop, sort, reverse and clear.

```
list1=[i for i in range(1,8)]
print(list1)
list1.append(8)
print(list1)
list1.insert(2,12)
print(list1)
list2=list1.copy()
print(list2)
another_list=['A','B','C']
list1.extend(another_list)
print(list1)
print(list1.count(3))
list1.remove('C')
print(list1)
print(list1.pop())
list2=[4,8,3,4,6,4,5,68,33,46,5,0]
list2.sort()
print(list2)
list1.reverse()
print(list1)
list2.clear()
print(list2)
```

Output : -

[1, 2, 3, 4, 5, 6, 7]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 12, 3, 4, 5, 6, 7, 8]
[1, 2, 12, 3, 4, 5, 6, 7, 8]
[1, 2, 12, 3, 4, 5, 6, 7, 8, 'A', 'B', 'C']
1
[1, 2, 12, 3, 4, 5, 6, 7, 8, 'A', 'B']
B
[0, 3, 4, 4, 4, 5, 5, 6, 8, 33, 46, 68]
['A', 8, 7, 6, 5, 4, 3, 12, 2, 1]
[]

17) Write a program to create nested list and display its elements.

```
list1=[[1, 2, 3], [4, 5, 6], [7, 8, 9, 10]]
for x in list1:
    print(" ".join(map(str, x)))
```

Output : -

1 2 3
4 5 6
7 8 9 10

18) Write a program to accept elements in the form of a tuple and display its minimum, maximum, sum and average.

```
tup=(1,2,3,4,5,6,7,8)
print("Max:",max(tup))
print("Min:",min(tup))
print("Sum:",sum(tup))
print("Avg:",sum(tup)/len(tup))
```

Output : -

Max: 8
Min: 1
Sum: 36
Avg: 4.5

19) Create a program to sort tuple with nested tuples

```
tup = [(1,(2,3)),(3,(2,1)),(2,(2,1))]
sort_tup=sorted(tup, key=lambda t: (t[1][1],t[0]))
print(sort_tup)
```

Output : -

[(2, (2, 1)), (3, (2, 1)), (1, (2, 3))]

20) Write a program to create a dictionary from the user and display the elements.

```
>>> my_dict = {'name': 'Jack', 'age': 26}
>>> print(my_dict['name'])
Jack
>>> print(my_dict.get('age'))
26
>>> print(my_dict.get('address'))
None
```

21) Write a program to convert the elements of two lists into key-value pairs of a dictionary.

```
>>> keys = ['Cricket', 'BasketBall', 'Hockey']
>>> values = ['11','5', '11']
>>> combine_dictionary = dict(zip(keys, values))
>>> print(combine_dictionary)
{'Cricket': '11', 'BasketBall': '5', 'Hockey': '11'}
```

22) Create a python function to accept python function as a dictionary and display its elements.

```
def func(d):
        for key in d:
        print("key:", key, "Value:", d[key])
D = {'a':1, 'b':2, 'c':3}
func(D)
```

Output : -

key: a Value: 1
key: b Value: 2
key: c Value: 3

☺ ☺ ☺ ☺ ☺

Unit – 3 Classes, Inheritance and Polymorphism

3.1 Classes and Objects:

Python is an object oriented programming language. A class is a model to create objects. An object is the instance of a class. An object is simply a collection of data (variables) and methods (functions) that act on those data. Similarly, a class is a blueprint for that object. A class is written with attributes and methods. The variables that are created inside a class are called Instance variable. A method is defined as a function written inside a class.

3.1.1 Creating a class:

A class is created with the keyword class followed by class name. Attributes are the variables that belong to class. Attributes are always public and can be accessed using dot (.) operator. E.g.: classname.attribute. The general format of declaring a class is as follows:

```
Class classname(Object)  #Object is the super class of all
the classes
""""" Document string describing the class"""
attributes
def __init__(self)
def method1()
def method2()
```

- Object is the super class for all the Python classes. It is not compulsory to write 'Object' , as it is implied.
- The document string is enclosed in triple single quotes or triple double quotes. It is also optional.
- Attributes are the variables that contains data
- def __init__(self) is a special method or constructor to initialize variables.
- Method1() and Method2() are methods intended to process data.

Example 1 : Python program to create a class Player and its object and a method to display detail of player.

```
class player:
  def __init__(self):
        self.name="Sachin"
      self.age=25
```

```
        self.runs=79
    #instance method
        def display(self):
        print("Name :",self.name)
        print("Age:",self.age)
        print("Runs Scored",self.runs)
    #create an instance to player class
    P1 = player()
    #call the method
    P1.display()
```

Output

Name : Sachin

Age: 25

Runs Scored 79

Note : the display() method is called using the object of the class followed the dot operator followed by the method name. E.g.: P1.display()

3.1.2 The self variable:

self represents the instance of the class. By using the "self" keyword we can access the attributes and methods of the class in python. It binds the attributes with the given arguments. There are two main uses of self variable

1. It is used as first parameter in the constructor, to refer to the instance variables inside the constructor. It is used as follows.

   def __init__(self)

2. It is used as first parameter in the instance method , as it acts on instance variables. It is used as follows

   def method1(self)

3.1.3 Constructor:

A constructor is a special method that is used to initialize the instance variables of a class. The first parameter of constructor will be self variable that contains the memory address of the instance. The task of constructors is to initialize(assign values) to the data members of the class when an object of class is created. In Python the __init__() method is called the constructor and is always called when an object is created.

Syntax of constructor declaration :

```
def __init__(self):
    # body of the constructor
```

Types of constructors :

- **default constructor** :The default constructor is simple constructor which doesn't accept any arguments. Its definition has only one argument 'self', which is a reference to the instance being constructed. Example 1 shows the use of default constructor.
- **parameterized constructor** :constructor with parameters is known as parameterized constructor. The parameterized constructor take its first argument as self and the rest of the arguments are to be provided.

Example 2 : Python program to show use of parameterized constructor.

```
class player:
 #parameterized constructor
    def __init__(self,n,a,r):
        self.name=n
        self.age=a
        self.runs=r
#instance method
    def display(self):
        print("Name :",self.name)
        print("Age:",self.age)
        print("Runs Scored",self.runs)
#create an instance to player class
P1 = player("Kapil",34,50)
#call the method
P1.display()
```

Output:

Name : Kapil

Age: 34

Runs Scored 50

3.1.4 Types of variables:

There are two types of variables in Python

1. **Instance Variable** — Declared inside the constructor method of class (the __init__ method). They are tied to the particular object instance of the class, hence the contents of an instance variable are completely independent from one object instance to the other.

Example 3 :Python Program to understand instance variables

```
class player:
 #parameterized constructor
    def __init__(self,n,a):
```

```
        self.name=n    #instance variable
        self.age=a     # instance variable
#instance method
    def display(self):
        print("Name :",self.name)
        print("Age:",self.age)
        #create an instance to player class
P1 = player("Virat Kohli",34)
#call the method
P1.display()
```

Output:

Name : Virat Kohli

Age: 34

2. **Class Variables or Static variables** — Declared inside the class definition (but outside any of the instance methods). They are not tied to any particular object of the class, hence shared across all the objects of the class. Modifying a class variable affects all objects instance at the same time.

Example 4: Python program to understand class variable

```
# class variable
class player:
 # class variable
  name = "Sachin"
  age = 30
 #class method
  @classmethod
  def modify(md):
md.name = "Sachin Tendulkar"
#create an instance to player class
P1 = player()
print("Name :",P1.name)
print("Age:",P1.age)
#call the method
P1.modify()
print("Name :",P1.name)
print("Age:",P1.age)
```

Output:

Name : Sachin

Age: 30

Name : Sachin Tendulkar

Age: 30

3.1.5 Types of Methods:

A Method is defined as a function written inside a class. There are 3 types of methods.

1. Instance method
    a. Accessor Method
    b. Mutator Method
2. Class method
3. Static method

1. Instance method

Instance methods are the methods which act upon the instance variables of the class.. they are bound to the object of the class and therefore called as instancename.method().Instance methods must have **self** as a parameter, but you don't need to pass this in every time

Example 5 : Python program to understand instance method

```
class addition:
      def __init__(self, a, b):
      self.a = a
      self.b = b
      def avg(self):
      return (self.a + self.b) / 2
s1 = addition(10, 20)
print( s1.avg() )
```

Output:

15.0

Accessor Methods and Mutator Methods

Accessor methods are used to access or read data of the variable. They do not modify the data in the variable. They are also called getter methods, as they are used to get data and written in form of getXX().

Mutator methods are used to read as well as modify data of the variable. They are known as setter methods and are generally written in form of setXX()

Example 6 Python program to understand accessor and mutator methods

```
class players:
```

```
    #mutator method
    def setname(self,name):
        self.name = name
    #accessor method
    def getname(self):
        return self.name
    #mutator method
    def setruns(self,runs):
        self.runs = runs
    #accessor method
    def getruns(self):
        return self.runs
n = int(input("how many players ?"))
i=0
while(i<n):
    p=players()
    name=input("Enter Name ")
    p.setname(name)
    runs=int(input("Enter runs "))
    p.setruns(runs)
    print("HI ",p.getname())
    print("You Scored : ",p.getruns()," runs")
    i+=1
    print("-----------------------------------")
```

Output:

how many players ?2
Enter Name sachin
Enter runs 44
HI sachin
You Scored : 44 runs

Enter Name Rohit
Enter runs 78
HI Rohit
You Scored : 78 runs

2. Class Methods:

A class method is a method that is bound to a class rather than its object. It doesn't require creation of a class instance, much like static method.

Example 7 : Program showing use of class method

```
class Student:
    name = 'Student'
    def __init__(self, a, b):
        self.a = a
        self.b = b
        @classmethod
    def info(cls):
        return cls.name

print(Student.info())
```

Output:

Student

3. Static Method

Static methods are used when processing is related to the class but do not need the class or its instances to perform any work. A static method can be called without an object for that class, using the class name directly.

Example 8 : Program showing use to static method.

```
class Player:
    name = 'Student'
    def __init__(self, a, b):
        self.a = a
        self.b = b
        @staticmethod
    def info():
        return "This is a player class"
print(Player.info())
```

Output

This is a player class

Passing members of one class to another.

Accessing attributes and methods of one class in another class is done by passing the object of one class to another.

Example 9 : Program to create student class and make all the members of student class available to user class

```
#passing members of one class to another
class Student:
    def __init__(self,id,name,marks):
        self.id = id
        self.name = name
        self.marks = marks
    def display(self):
        print("Id = ",self.id)
        print("Name = ",self.name)
        print("marks = ",self.marks)
class user:
    @staticmethod
    def show(s):
        s.marks += 10 # incrementing marks by 10
        s.display()
s=Student(1,"Siddharta",60)
user.show(s)
```

Output

Id = 1

Name = Siddharta

marks = 70

3.2 Inheritance:

Inheritance is an important feature of Object oriented programming. It provides reusability of code

3.2.1 Implementing Inheritance:

Inheritance allows us to define a class that inherits all the methods and properties from another class. Parent class is the class being inherited from, also called base class or super class. Child class is the class that inherits from another class, also called derived class or subclass.

Example 10 : Program demonstrating Inheritance in Python

```
#Parent class
class Parent:
            # Constructor
    def __init__(self, name):
```

```
        self.name = name
    # instance method to  get name
    def getName(self):
        return self.name
    # To check if this person is a child or not
    def ischild(self):
        return False

# child class or Subclass
class Child(Parent):
    # Here we return true
    def ischild(self):
        return True
p = Parent("Mr Kumar")  # An Object of Parent class
print("Is ",p.getName(),"a child ?" , p.ischild())
p = Child("Mohit") # An Object of Child class
print("Is ",p.getName(),"a child ?" , p.ischild())
```

Output:

Is Mr Kumar a child ? False

Is Mohit a child ? True

3.2.2 Constructor in Inheritance:

In Python, constructor of class used to create an object (instance), and assign the value for the attributes. Just like variables and methods, the constructors in the super class are also available to the sub class object by default.

Example 11 : Python program to access the base class constructor from the derived class.

```
class Teacher:
    def __init__(self,schoolname):
        self.schoolname = schoolname
    def display(self):
        print("School Name = ",self.schoolname)
        class Student(Teacher):
    pass # nothing to write in sub class
        s=Student("Karnavati School")
s.display()
```

Output

School Name = Karnavati School

3.2.3 Overriding Super class constructors and methods:

Constructor can be written in sub class also. Where a constructor is written in sub class, the super class constructor is not more available to the sub class. In this case, the sub class constructor overrides the super class constructor. This is known as constructor overriding.

Similarly, if there is a method in the sub class with the same name as that of super class, then the sub class method overrides the method of super class. This is known as methods overriding.

Example 12 : Program to override super class constructor t and method in sub class.

```
class Teacher:
    def __init__(self,schoolname):
        self.schoolname = schoolname
        def display(self):
        print("School Name = ",self.schoolname)
        class Student(Teacher):
    def __init__(self,name,std):
        self.name = name
        self.std = std
        def display(self):
        print("Student Name = ",self.name)
        print("Class = ",self.std)
    s=Teacher("Karnavati School")
s.display()
s=Student("Ram",8)
s.display()
```

Output:

School Name = Karnavati School

Student Name = Ram

Class = 8

3.2.4 The Super() Method:

The **super() method** in Python makes class inheritance more manageable and extensible. The method returns a temporary object that allows reference to a parent class by the keyword *super*.super() provides the access to those methods of the super-

class (parent class) which have been overridden in a sub-class (child class) that inherits from it.

Example 13: A Python program to access base class constructor and method in the sub class using the super() method

```
class Parent:
  def __init__(self, txt):
    self.message = txt
  def printmessage(self):
    print(self.message)
    print("From parent class")
class Child(Parent):
  def __init__(self, txt):
    super().__init__(txt)
  def printmessage(self):
      super().printmessage()
      print("From child class")
x = Child("Hello, and welcome!")
x.printmessage()
```

Output:

Hello, and welcome!

From parent class

From child class

3.2.5 Types of inheritance:

There are two types of inheritance available in python

1. Single inheritance
2. Multiple inheritance

1. **Single inheritance**:

Deriving one or more subclasses from as single super class is called "single inheritance". Thus enables code reusability of parent class and adding new features to a class makes code more readable, elegant and less redundant.

Example 14 : Program showing single inheritance in which two subclasses are derived from a single base class

```
class Parent(object): # object is superclass for Parent
  class also.
```

```
    def display(self):
        print("Parent class")
class child_boy(Parent):
   def display(self):
        super().display()
        print("Boy child")
class child_girl(Parent):
       def display(self):
        print("Girl Child")
a =child_boy()
a.display()
s =child_girl()
s.display()
```

Output

Parent class

Boy child

Girl Child

2. Multiple Inheritance:

Deriving subclasses from more than one super class, is called multiple inheritance.

Example 15 : Program showing multiple inheritance using two base classes

```
#Multiple inheritance
#1st base class
class Cal1:
    def Sum(self,a,b):
        return a+b;
#2nd base class
class Cal2:
    def Mul(self,a,b):
        return a*b;
# derived class
class Display(Cal1,Cal2):
    def Divide(self,a,b):
        return a/b;
d = Display()
```

```
print(d.Sum(10,20))
print(d.Mul(10,20))
print(d.Divide(10,20))
```

Output

30

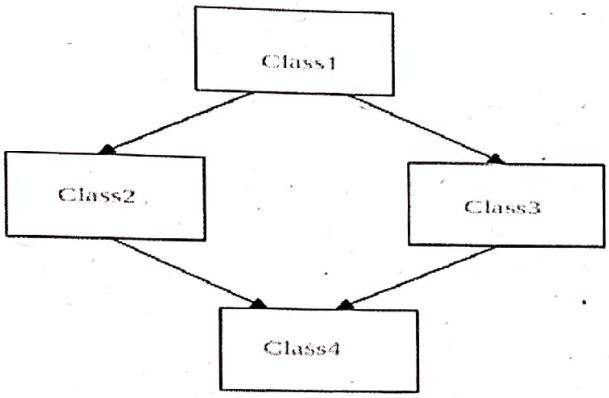
200

0.5

3.2.6 Problems in Multiple Inheritances:

If you allow multiple inheritance then it is possible that you might inherit the same class more than once.

For example, if class 2 and class 3 inherit from Class 1 and Class 4 inherits from Class 2 and Class 3 then it potentially gets two copies of class class 1:



The problem arises that class 4 is unable to access constructors of both the super classes. It means Class 4 cannot access all the instance variables of both its super classes. If it wants to access instance variables of both its super classes, then the solution is to user super();. init () in every class. The super function comes to a conclusion, on which method to call with the help of the **method resolution order (MRO).**

3.2.7 Method Resolution Order (MRO):

In the case of multiple inheritances a given attribute is first searched in the current class if it's not found then it's searched in the parent classes. The parent classes are searched in a depth-first, left-right fashion and each class is searched once. Searching in this way is called **Method Resolution Order (MRO).**

There are 3 principles followed by MRO

1. To search for the sub class before going for its base classes. Thus if class B is inherited from class A, it will search B first and then goes to A.
2. When a base class is inherited from multiple super classes, it searches in the order from left to right in the base classes. For example if class c is inherited from A and

B as class C (A,B), them it will first search in A and then in B.

3. It will not visit any class more than once. That means a class in the inheritance hierarchy is traversed only once exactly.

Example 16: Write a program to understand the order of execution of methods in several base classes according to method resolution order (MRO).

```
class A(object):
    def method(self):
        print("A class method")
        super().method()
class B(object):
    def method(self):
        print("B class method")
        super().method()
class C(object):
    def method(self):
        print("C class method")
        super().method()
class X(A,B):
    def method(self):
        print("X class method")
        super().method()
class Y(B,C):
    def method(self):
        print("Y class method")
        super().method()
class P(X,Y,C):
    def method(self):
        print("P class method")
        super().method()
newp = P()
print(P.mro())
newp.method()
```

Output

[<class '__main__.P'>, <class '__main__.X'>, <class '__main__.A'>, <class '__main__.Y'>, <class '__main__.B'>, <class '__main__.C'>, <class 'object'>]

P class method
X class method
A class method
Y class method
B class method
C class method

To understand the sequence of execution MRO is to be applied. The sub class at the bottom most level is P. So, first P class method is executed. P is derived from X, Y and C. Hence from left to right, first comes X, which in turn in derived from A and B. hence, A is searched first, but as it does not have any user defined super class the search comes back to Y. Y is derived from B and C. Left to right B is searched first. Even B does not have a user defined super class, at last it will search C.

3.3 Polymorphism:

3.3.1 Introduction to polymorphism:

Polymorphism is derived from Greek words, poly means many and morphos means forms. Polymorphism means many forms. In programming, polymorphism means same function name (but different signatures) being uses for different types. If a variable, object or method exhibits different behavior in different contexts, it is called polymorphism. Python has built-in polymorphism. The following are the examples of Polymorphism in python.

- Duck typing philosophy of Python
- Operator overloading
- Method overloading
- Method overriding

3.3.2 Duck typing philosophy of Python:

Duck typing, this term comes from the saying

"If it walks like a duck, and it talks like a duck, then it must be a duck."

This is called Duck Typing (There are other variations to this).

Duck typing is a concept related to **dynamic typing**, where the type or the class of an object is less important than the methods it defines. When you use duck typing, you do not check types at all. Instead, you check for the presence of a given method or attribute. In duck typing, an object suitability is determined by the presence of certain methods and properties rather than type of the object itself. If we store integer into that variable, its type is taken as 'int' and if we store a string into that variable, its type is taken as 'str'. To check the type of a variable or object, we can use type() function.

```
x = 5   # store integer into x
print(type(x))   # display type of x
<class 'int'>

x = 'Hello'   # store string into x
print(type(x))   # display type of x
<class 'str'>
```

Python variables are names or tags that point to memory locations where data is stored. So, if 'x' is a variable, we can make it refer to an integer or a string. We can conclude two points from this discussion:

1. Python's type system is 'strong' because every variable or object has a type that we can check with the type() function.
2. Python's type system is 'dynamic' since the type of a variable is not explicitly declared, but it changes with the content being stored.

Similarly, if we want to call a method on an object, we do not need to check the type of the object and we do not need to check whether that method really belongs to that object or not.

For example, take a method call_talk() that accepts an object (or instance).

def call_talk(obj):

obj.talk()

Example 17: A Python program to invoke a method on an object without knowing the type (or class) of the object.

```
class Duck:
    def talk(self):
        print('Quack, quack!')
        #Human class contains talk() method
class Human:
    def talk(self):
        print('Good Morning!')
        #this method accepts an object and calls talk()
method
def call_talk(obj):
        obj.talk()
        #call call_talk() method and pass an object
        #depending on type of object, talk() method is
executed
x = Duck()
```

```
call_talk(x)
x=Human()
call_talk(x)
```

Output:

Quack, quack!

Good Morning!

3.3.3 Operator Overloading:

Operator overloading in Python is the ability of a single operator to perform more than one operation based on the class (type) of operands.

For example, the + operator can be used to add two numbers, concatenate two strings or merge two lists. This is possible because the + operator is overloaded with int and str classes.

Similarly, you can define additional methods for these operators to extend their functionality to various new classes and this process is called Operator overloading.

Example 18 : Simple Python program to demonstrate operator overloading.

```
# Python program to show use of
# + operator for different purposes.
  print(1 + 2)
  # concatenate two strings
print("Good"+"Morning")
  # Product two numbers
print(3 * 4)
  # Repeat the String
print("Five"*4)
```

Output:

3

GoodMorning

12

FiveFiveFiveFive

In Python, when any operator is used, a special function is internally invoked by the compiler for that particular operator.Python methods that have double underscores before and after their names are called **Magic methods or Special functions**.By changing this magic methods code, we can extend the functionality of the operator. Following table summarizes important operators and their corresponding internal methods that can be overridden to act on objects. These methods are called magic methods.

Example 19 : Python Program to dem

Operator	Magic method
+	object.__add__(self, other)
-	object.__sub__(self, other)
*	object.__mul__(self, other)
/	object.__div__(self, other)
//	object.__floordiv__(self, other)
%	object.__mod__(self, other)
**	object.__pow__(self, other[, modulo])
+=	object.__iadd__(self, other)
-=	object.__isub__(self, other)
*=	object.__imul__(self, other)
/=	object.__idiv__(self, other)
//=	object.__ifloordiv__(self, other)
%=	object.__imod__(self, other[, modulo])
**=	object.__ipow__(self, other)
<	object.__lt__(self, other)
<=	object.__le__(self, other)
>	object.__gt__(self, other)
>=	object.__ge__(self, other)
==	object.__eq__(self, other)
!=	object.__ne__(self, other)

onstrate how to overload an binary + operator

```
  class A:
    def __init__(self, a):
        self.a = a
      # adding two objects
    def __add__(self, o):
        return self.a + o.a
ob1 = A(1)
ob2 = A(2)
ob3 = A("Good")
ob4 = A("Morning")
  print(ob1 + ob2)
print(ob3 + ob4)
```

Output:

3

GoodMorning

3.3.4 Method Overloading:

In Python you can define a method in such a way that there are multiple ways to call it.Given a single method or function, we can specify the number of parameters ourself. Depending on the function definition, it can be called with zero, one, two or more parameters. If a method is written such that it can perform more than one task, it is known as *method overloading*

Example 20 : A Python program to show method overloading.

```
# We create a class with one method display().
#The first parameter of this method is set to None, this
  gives us the option to call it with or without a
  #parameter.
#An object is created based on the class, and we call its
  method using zero and one parameter.
class Human:
    def display(self, name=None):
            if name is not None:
            print('Hello ' + name)
        else:
            print('Hello ')
        # Create instance
obj = Human()
# Call the method
obj.display()
# Call the method with a parameter
obj.display('President')
```

Output:

Hello

Hello President

Method Overriding:

When there is a method in the super class, writing the same method in the sub class so that it replaces the super class method is called "Method Overriding". The programmer overrides the super class methods when he does not want to use them in the sub class. Instead, he wants a new functionality to the same method in the sub class.

☺ ☺ ☺ ☺ ☺

Exercises

Multiple Choice Questions

1. **A _____ is a model or plan to create objects**
   (a) Class
   (b) Method
   (c) Function
   (d) Data field
2. **A _____ is a special method that is useful to declare and initialize the instance variables**
   (a) Method
   (b) Class
   (c) Function
   (d) Constructor
3. **Which of the following keywords mark the beginning of the class definition?**
   (a) def
   (b) return
   (c) class
   (d) all of above
4. **what will be the output of the following code?**

```
class test:
    def __init__(self,a):
        self.a=a

    def display(self):
        print(self.a)
obj=test()
obj.display()
```

   (a) Runs normally, doesn't display anything
   (b) Displays 0, which is the automatic default value
   (c) Error as one argument is required while creating the object
   (d) Error as display function requires additional argument
5. **Which of the following best describes inheritance?**
   (a) Ability of a class to derive members of another class as a part of its own definition
   (b) Means of bundling instance variables and methods in order to restrict access

to certain class members\

(c) Focuses on variables and passing of variables to functions

(d) Allows for implementation of elegant software that is well designed and easily modified

6. **When a constructor is written in sub class, the super class constructor is not available to the sub class.**

(a) True

(b) False

7. **What will be the output of the following code?**

```
class A:
    def one(self):
        return self.two()

    def two(self):
        return 'A'

class B(A):
    def two(self):
        return 'B'
obj1=A()
obj2=B()
print(obj1.two(),obj2.two())
```

(a) A A

(b) A B

(c) B B

(d) An exception is thrown

8. **Which of the following is not a type of inheritance?**

(a) Single

(b) Multiple

(c) Multi-level

(d) Double-level

9. **Which of the following best describes polymorphism?**

(a) Ability of a class to derive members of another class as a part of its own definition

(b) Means of bundling instance variables and methods in order to restrict access to certain class members

(c) Focuses on variables and passing of variables to functions

(d) Allows for objects of different types and behaviour to be treated as the same general type

10. Which of the following statements is true?

a) A non-private method in a superclass can be overridden
b) A subclass method can be overridden by the superclass
c) A private method in a superclass can be overridden
d) Overriding isn't possible in Python

Answers:

1. (a) 2.(d) 3.(c) 4. (c) 5. (a) 6. (a) 7. (b) 8. (d) 9.(d) 10. (a)

Explain Answer in detail :

1. How can we create a class in Python? Explain giving example.
2. Discuss : Self variable
3. Discuss the different types of variables in Python.
4. Discuss different types of methods in Python
5. How is inheritance implemented in Python? Explain giving suitable example
6. How is super class method and constructor overridden in subclass? Explain giving example
7. Write a note on super() method.
8. Discuss : Different types of Inheritance in Python.
9. Write a detailed note on MRO.
10. What is Polymorphism in Python? Explain the following in detail
    - Duck typing
    - Operator overloading
    - Method overloading
    - Method overriding

☺ ☺ ☺ ☺ ☺

CC-305 Unit – 3 Python Programming Practicals

21. Write a program to create a Student class with name, age and marks as data members. Also create a method named display() to view the student details. Create an object to Student class and call the method using the object.

```
class student:
    def __init__(self):
        self.name="Sachin"
        self.age=23
        self.marks=79
#instance method
    def display(self):
        print("Name :",self.name)
        print("Age:",self.age)
        print("Marks",self.marks)
s = student()    #creating an object(or instance) to
student class
s.display()  # call to display() method
```

Output:

Name : Sachin

Age: 23

Marks 79

22. Write a program to create Student class with a constructor having more than one parameter.

```
class student:
 def __init__(self,nm='.',ag=15,m=0):
     self.name=nm
     self.age=ag
     self.marks=m
#instance method
 def display(self):
     print("Name :",self.name)
     print("Age:",self.age)
```

```
        print("Marks",self.marks)
s = student("Nisha",18,46)      #creating an object(or
instance) to student class
s.display()  # call to display() method
s1 = student("Shama") # here it will take default values
s1.display()
```

Output:

Name : Nisha
Age: 18
Marks 46
Name : Shama
Age: 15
Marks 0

3. **Write a program to demonstrate the use of instance and class/static variables.**

```
#program demonstrating use of class variable
class sample:
    var = 10   # this is a class variable
    #this is  class method
    # using built-in decorator @classmethod, to mark this
method as class method
    @classmethod
    def new_modify(cls):
        cls.var+=1
s1 = sample()
s2 = sample()
print("X in s1 ", s1.var)
print("X in s2 ", s2.var)
#modify x in s1....showing use of class variable and
method
s1.new_modify()
print("X in s1 ", s1.var)
print("X in s2 ", s2.var)
```

Output:

X in s1 10
X in s2 10

```
X in s1 11
X in s2 11
```

```
#program demonstrating use of instance variable
class sample:
    def __init__(self):
        self.x = 10 # this is instance variable
        # this is instance method
    def modify(self):
        self.x+=1
s1 = sample()
s2 = sample()
print("X in s1 ", s1.x)
print("X in s2 ", s2.x)
# modify x in s1...showing use of instance variable and
method
s1.modify()
print("X in s1 ", s1.x)
print("X in s2 ", s2.x)
```

Output:

```
X in s1 10
X in s2 10
X in s1 11
X in s2 10
```

4. **Write a program to store data into instances using mutator methods and to retrieve data from the instances using accessor methods.**

```
class student:
    #mutator method
    def setname(self,name):
        self.name = name
    #accessor method
    def getname(self):
        return self.name
    #mutator method
    def setmarks(self,marks):
        self.marks = marks
    #accessor method
    def getmarks(self):
```

```
        return self.marks
n = int(input("how many students ?"))
i=0
while(i<n):
    s=student()
    name=input("Enter Name")
    s.setname(name)
    marks=int(input("Enter marks"))
    s.setmarks(marks)
    print("HI ",s.getname())
    print("Your Marks :",s.getmarks())
    i+=1
    print("-----------------------------------")
```

Output:

```
how many students ?2
Enter Name vinay
Enter marks 23
HI  vinay
Your Marks : 23
-----------------------------------
Enter Name Reena
Enter marks 55
HI  Reena
Your Marks : 55
-----------------------------------
```

5. Write a program to use class method to handle the common features of all the instance of Student class.

```
class student:
    marks=10
    @classmethod
    def modify(cls,name):
        print(" {} scored {}
marks".format(name,cls.marks))
student.modify("Sanjay")
student.modify("Ajay")
```

Output:

```
Sanjay scored 10 marks
Ajay scored 10 marks
```

6. Write a program to create a static method that counts the number of instances created for a class.

```
class myclass:
    .n=0
    def __init__(self):
        myclass.n+=1
    # static method
    @staticmethod
    def no_of_objects():
        print("No of instances created are", myclass.n)
obj1 = myclass()
obj2 = myclass()
obj3 = myclass()
myclass.no_of_objects()
```

Output:

No of instances created are 3

7. Create a Bank class with two variables name and balance. Implement a constructor to initialize the variables. Also implement deposit and withdrawals using instance methods.

```
import sys
class bank(object):
    def __init__(self,name,balance=0.0):
        self.name = name
        self.balance = balance
    def deposit(self,amount):
        self.balance += amount
        return self.balance
    def withdrawals(self,amount):
        if amount > self.balance:
            print("Low Balance, Cannot withdraw")
        else:
            self.balance -= amount
        return self.balance
#creates an account with given name and balance 0.0
name = input("Enter name ")
```

```
b = bank(name)
    while(True):
    print("d/D -deposit, w/W -withdrawal, e/E -exit")
    choice=input("Enter your choice ")
    if choice == "e" or choice=="E":
        sys.exit()
    amount = float(input("Enter amount "))
    if choice == "d" or choice=="D":
        print("Balance after deposit
",b.deposit(amount))
    elif choice == "w" or choice=="W":
        print("Balance after withdrawals
",b.withdrawals(amount))
```

Output:

Enter name Bank of Ahmedabad
d/D -deposit, w/W -withdrawal, e/E -exit
Enter your choice d
Enter amount 10000
Balance after deposit 10000.0
d/D -deposit, w/W -withdrawal, e/E -exit
Enter your choice w
Enter amount 4000
Balance after withdrawals 6000.0
d/D -deposit, w/W -withdrawal, e/E -exit
Enter your choice e

18. Write a program to create a Emp class and make all the members of the Emp class available to another class (Myclass). [By passing members of one class to another]

```
class Emp:
    def __init__(self,id,name,sal):
        self.id = id
        self.name = name
        self.salary = sal
    def display(self):
        print("Id = ",self.id)
        print("Name = ",self.name)
        print("Salary = ",self.salary)
```

```
class myclass:
    @staticmethod
    def mymethod(e):
        e.salary += 1000 # incrementing salary by 1000
        e.display()
e=Emp(1001,"Raj Kumar",12000.75)
myclass.mymethod(e)
```

Output:

Id = 1001

Name = Raj Kumar

Salary = 13000.75

59. **Create a Student class to with the methods set_id, get_id, set_name, get_name, set_marks and get_marks where the method name starting with set are used to assign the values and method name starting with get are returning the values. Save the program by student.py. Create another program to use the Student class which is already available in student.py.**

Note: This program contains two .py files

1) student.py 2) newstudent.py

```
#student.py
class student:
    def setid(self,id):
        self.id=id
    def getid(self):
        return self.id
    def setname(self,name):
        self.name=name
    def getname(self):
        return self.name
    def setaddress(self,address):
        self.address=address
    def getaddress(self):
        return self.address
    def setmarks(self,marks):
        self.marks=marks
    def getmarks(self):
```

```
        return self.marks
#newstudent.py
from student_9  import student
s= student()
s.setid(100)
s.setname("Rakesh Sharma")
s.setaddress("City College, satellite,Ahmedabad")
s.setmarks(79)
print("Student ID = ",s.getid())
print("Student Name = ",s.getname())
print("Student Address = ",s.getaddress())
print("Student Marks = ",s.getmarks())
```

Output:

Student ID = 100
Student Name = Rakesh Sharma
Student Address = City College, satellite,Ahmedabad
Student Marks = 79

10. Write a program to access the base class constructor from a sub class by using super() method and also without using super() method.

```
class square:
    def __init__(self,x):
        self.x = x
    def area(self):
        print("Area of square ", self.x * self.x)
class rectangle(square):
    def __init__(self,x,y):
        super().__init__(x)
        self.y=y
    def area(self):
        super().area()
        print("Area of Rectangle ",self.x * self.y)
a,b = [float(x) for x in input("Enter length and breadth
:").split()]
r = rectangle(a,b)
```

```
r.area()
```

Output:

```
Enter length and breadth :3 4
Area of square  9.0
Area of Rectangle  12.0
```

11. Write a program to override super class constructor and method in sub class.

```
class teacher:
    def __init__(self):
        self.id = 1001
    def display(self):
        print("Teachers id ", self.id)
class student(teacher):
    def __init__(self):
        self.id = 1
    def display(self):
        print("Students id ", self.id)
s = student()
s.display()
```

Output:

```
Students id  1
```

12. Write a program to implement single inheritance in which two sub classes are derived from a single base class.

```
class Bank(object): # object is superclass for Bank class
also.
    cash = 1000000
    @classmethod
    def chk_cash(cls):  # prints avialable cash
        print(cls.cash)
class Andhrabank(Bank): #first subclass
    pass
class Statebank(Bank):#second subclass
    cash = 200000
    @classmethod
    def chk_cash(cls):
```

```
        print(cls.cash+Bank.cash)
a =Andhrabank()
a.chk_cash()
s =Statebank()
s.chk_cash()
```

Output:

1000000

1200000

13. Write a program to implement multiple inheritance using two base classes.

```
class Father:
        def height(self):
        print("Height is 6.0 foot")
class Mother:
        def complexion(self):
        print("Complexion is Fair")
class child(Father,Mother):
    pass
    c = child()
print ("Child inherited qualities :")
c.height()
c.complexion()
```

Output:

Child inherited qualities :

Height is 6.0 foot

Complexion is Fair

14. Write a program to understand the order of execution of methods in several base classes according to method resolution order (MRO).

```
class A(object):
    def method(self):
        print("A class method")
        super().method()
class B(object):
    def method(self):
```

```
        print("B class method")
        super().method()
class C(object):
    def method(self):
        print("C class method")
        super().method()
class X(A,B):
    def method(self):
        print("X class method")
        super().method()
class Y(B,C):
    def method(self):
        print("Y class method")
        super().method()
class P(X,Y,C):
    def method(self):
        print("P class method")
        super().method()
newp = P()
print(P.mro())
newp.method()
```

Output:

[<class '__main__.P'>, <class '__main__.X'>, <class '__main__.A'>, <class '__main__.Y'>, <class '__main__.B'>, <class '__main__.C'>, <class 'object'>]

P class method

X class method

A class method

Y class method

B class method

C class method

15. Write a program to check the object type to know whether the method exists in the object or not.

```
class Dog:
    def bark(self):
        print("Bow Wow")
class Duck:
```

```
    def talk(self):
        print("Quack Quack")
class Human:
    def talk(self):
        print("Hello, Hi")
#this method accepts an object and calls talk() method
def call_talk(obj):
    if hasattr(obj,"talk"):
        obj.talk()
    elif hasattr(obj,"bark"):
        obj.bark()
    else:
        print("Wrong object passed")
x = Duck()
call_talk(x)
x = Human()
call_talk(x)
x = Dog()
call_talk(x)
```

Output:

Quack Quack
Hello, Hi
Bow Wow

10/10 **16. Write a program to overload the addition operator (+) to make it act on the class objects.**

```
class bookx:
    def __init__(self,pages):
        self.pages = pages
    def __add__(self,other):
        return self.pages + other.pages
class booky:
    def __init__(self,pages):
        self.pages = pages
b1 = bookx(100)
b2 = booky(150)
print("Total pages = ",(b1+b2))
```

Output:

Total pages = 250

17. Write a program to show method overloading to find sum of two or three numbers.

```
class myclass:
    def sum(self,a=None,b=None,c=None):
        if a!= None and b != None and c != None:
            print("Sum of three numbers = ",(a+b+c))
        elif a!= None and b!= None:
            print("sum of two numbers =",(a+b))
        else:
            print(" please enter 2 or 3 arguments")
m = myclass()
m.sum(10,15,20)
m.sum(30,50)
m.sum(100)
```

Output:

Sum of three numbers = 45

sum of two numbers = 80

please enter 2 or 3 arguments

18. Write a program to override the super class method in subclass.

```
import math
class square:
    def area(self,x):
        print("Area of square = ",(x*x))
class circle(square):
    def area(self,x):
        print("Area of circle = %.2f"%(math.pi*x*x))
c= circle()
c.area(4)
```

Output:

Area of circle = 50.27

☺ ☺ ☺ ☺ ☺

Unit – 4 Exception Handling, Standard Librabry, Creating Virtual Environment and Python Database Connectivity

4.1 EXCEPTION HANDLING AND STANDARD LIBRARY:

4.1.1 Exceptions:

An error that occurs at runtime is called an exception. Another way to state it can be logical error detected during execution of program is an exception.

4.1.2 Exception handling:

Exception handling enables a program to deal with exceptions and continue its normal execution.

For example, if you try to run a program by entering a nonexistent filename, the program would report this IOError: The lengthy error message is called a stack traceback or traceback. The traceback gives information on the statement that caused the error by tracing back to the function calls that led to this statement. The line numbers of the function calls are displayed in the error message for tracing the errors.

To deal with an exception, so that the program can catch the error and prompt the user to enter a correct filename can be done using Python's exception handling syntax.

The syntax for exception handling is to wrap the code that might raise (or throw) an exception in a try clause, as follows:

```
try:
    <body>
except <ExceptionType>:

    <handler>
```

Here, <body> contains the code that may raise an exception. When an exception occurs, the rest of the code in <body> is skipped. If the exception matches an exception type, the corresponding handler is executed. <handler> is the code that processes the exception.

A try statement can have more than one except clause to handle different exceptions ; try statement can also have an optional else and/or finally statement, in a syntax like this:

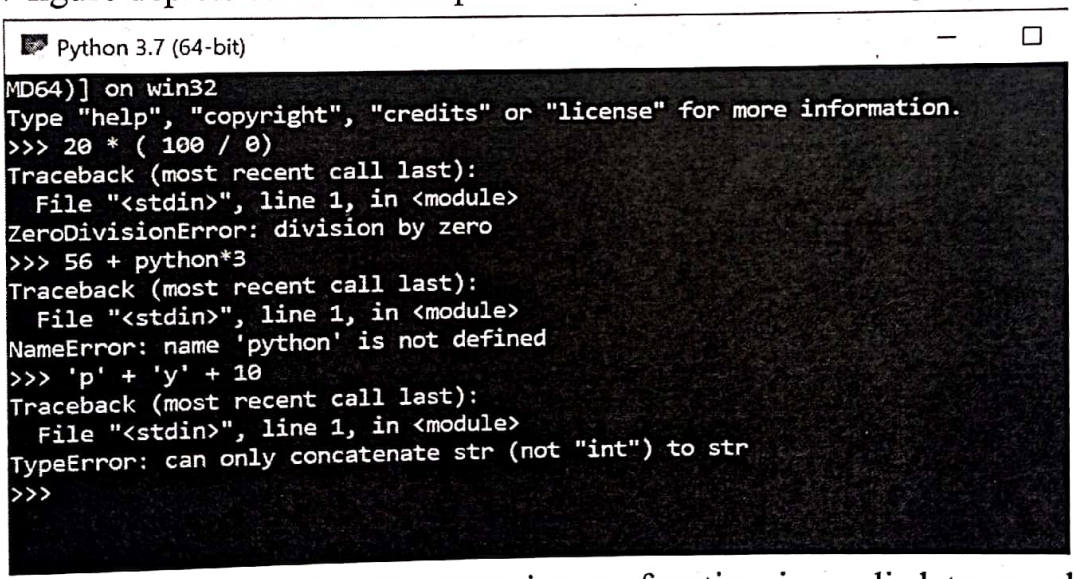
```
    <body>
except                    <ExceptionType1>:
    <handler1>
...
except                    <ExceptionTypeN>:
    <handlerN>
except:
    <handlerExcept>
else:
<process_else>
finally:
    <process_finally>
```

The multiple excepts are similar to elif's. When an exception occurs, it is checked to match an exception in an except clause after the try clause sequentially. If a match is found, the handler for the matching case is executed and the rest of the except clauses are skipped.

4.1.3 Types of exceptions:

A number of built-in exceptions are defined in the Python library. Let's see some common error types.

Below figure depicts common exceptions occurred while exceuting a program:



TypeError is thrown when an operation or function is applied to an object of an inappropriate type.

ZeroDivisionError is thrown when the second operator in the division is zero.

NameError is thrown when an object could not be found.

IndexError is thrown when trying to access an item at an invalid index.

```
Python 3.7 (64-bit)
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
>>> l=['a','b','c']
>>> l[3]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: list index out of range
>>>
```

ModuleNotFoundError is thrown when a module could not be found.

```
Python 3.7 (64-bit)
>>> import nonexistmodule
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ModuleNotFoundError: No module named 'nonexistmodule'
>>>
```

KeyError is thrown when a key is not found.

```
Python 3.7 (64-bit)
>>> d={1:'abc',2:'cd',3:'ef'}
>>> d[4]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 4
>>>
```

4.1.4 Operating System Interface:

The os module includes hundreds of interface features with the operating system:

```
Python 3.7 (64-bit)
>>> import os
>>> os.getcwd()
'C:\\Users\\91963\\AppData\\Local\\Programs\\Python\\Python37'
>>> os.listdir()
['DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt', 'NEWS.txt', 'python.exe', 'python3.dll',
 'python37.dll', 'pythonw.exe', 'Scripts', 'tcl', 'Tools', 'vcruntime140.dll']
>>> os.makedirs("C:\\XYZ")
>>> os.removedirs("C:\\XYZ")
>>>
```

The module offers a long list of functions and attributes that help the programmer achieve many OS-related tasks. Above mentioned figure depicts usage of getcwd(), listdir() etc. functions explained below:

getcwd () - This function returns the current working directory.

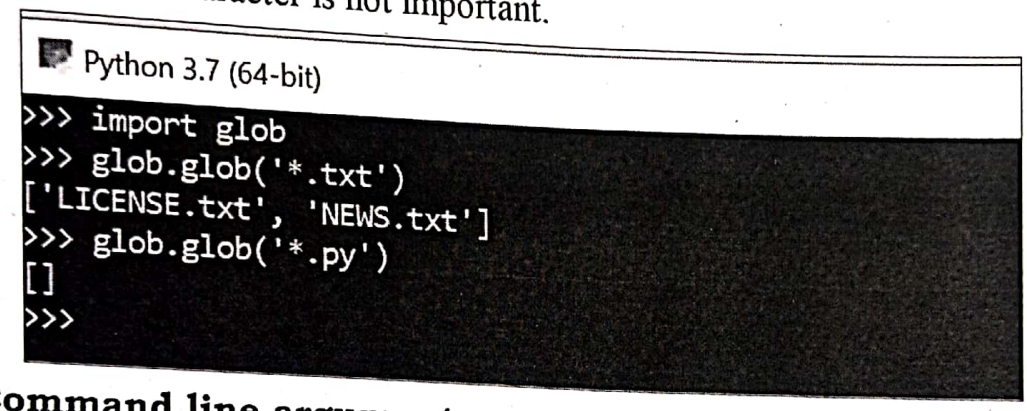
listdir () - This function lists the contents of a Directory. If none specified, the contents of the current directory are listed.

makedirs () -This function creates the directory specified by the passed string argument.
removedir () -This function deletes the directory specified as argument.
rename(), system() , getlogin() are also commonly used

4.1.5 File wildcards:

Linux and Unix systems and shells also support and provide glob

It provides a function for making file lists from directory wildcard searches. Wildcard is important glob operator for glob operations. Wildcard or asterisk is used to match zero or more characters. Wildcard specified that there may be zero character or multiple character where character is not important.



4.1.6 Command line arguments:

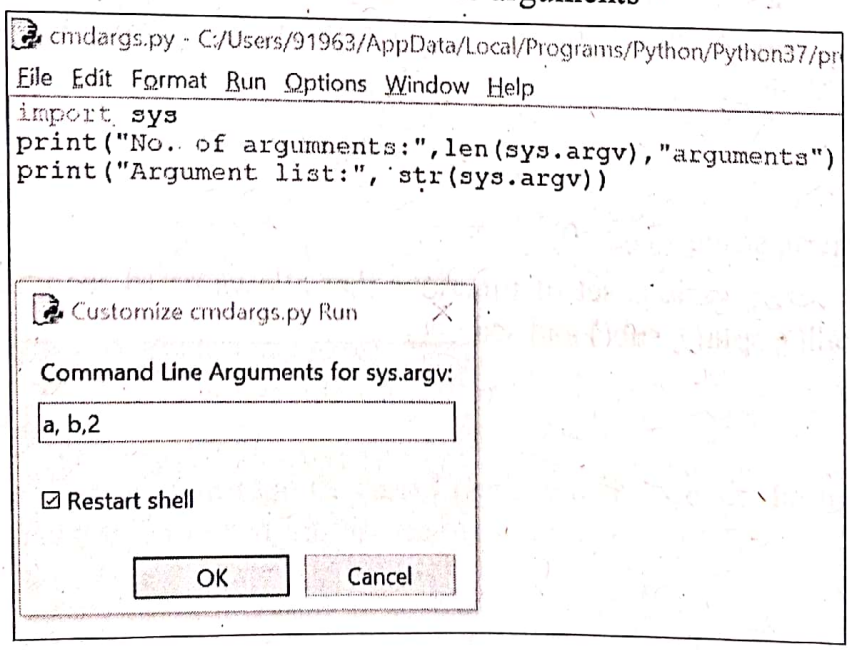
The arguments given in the operating system's command line shell after the program name are known as Command Line Arguments. The three most common to deal with these types of arguments are:

- **Using sys.argv:**

The python sys module provides access to any command – line through the sys.argv.

sys.argv is the list of command line arguments

len(sys.argv) is the number of command line arguments



```
Python 3.7.9 Shell
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/91963/AppData/Local/Programs/Python/Python37/programs/cmdarg
s.py
No. of argumnents: 3 arguments
Argument list: ['C:/Users/91963/AppData/Local/Programs/Python/Python37/programs/
cmdargs.py', 'a,', 'b,2']
>>>
```

➢ **Using getopt module:**

Unlike sys module getopt module extends the separation of the input string by parameter validation. It allows both short, and long options including a value assignment.

Syntax:

getopt.getopt(args,options,[long options])

➢ **Using argparse module:**

The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

4.1.7 String pattern matching:

A *regular expression* is a special sequence of characters that helps you match or find other strings or sets of strings, using a specialized syntax held in a pattern. The Python module **re** provides full support for functioning of regular expressions in Python. The re module raises the exception re.error if an error occurs while compiling or using a regular expression.

Syntax:

re.search(pattern, string, flags=0)

're' modules offers various set of functions that allows us to search a string for given text like findall(), split(), sub() and search()

```
import re

#Check if the string starts with "The" and ends with "world":

txt = "The elephant is largest animal in world"
x = re.search("^The.*world$",txt)

if x:
  print("YES! We have a match!")
else:
  print("No match")
```

```
Python 3.7.9 Shell
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
YES! We have a match!
>>>
```

4.1.8 Mathematics:

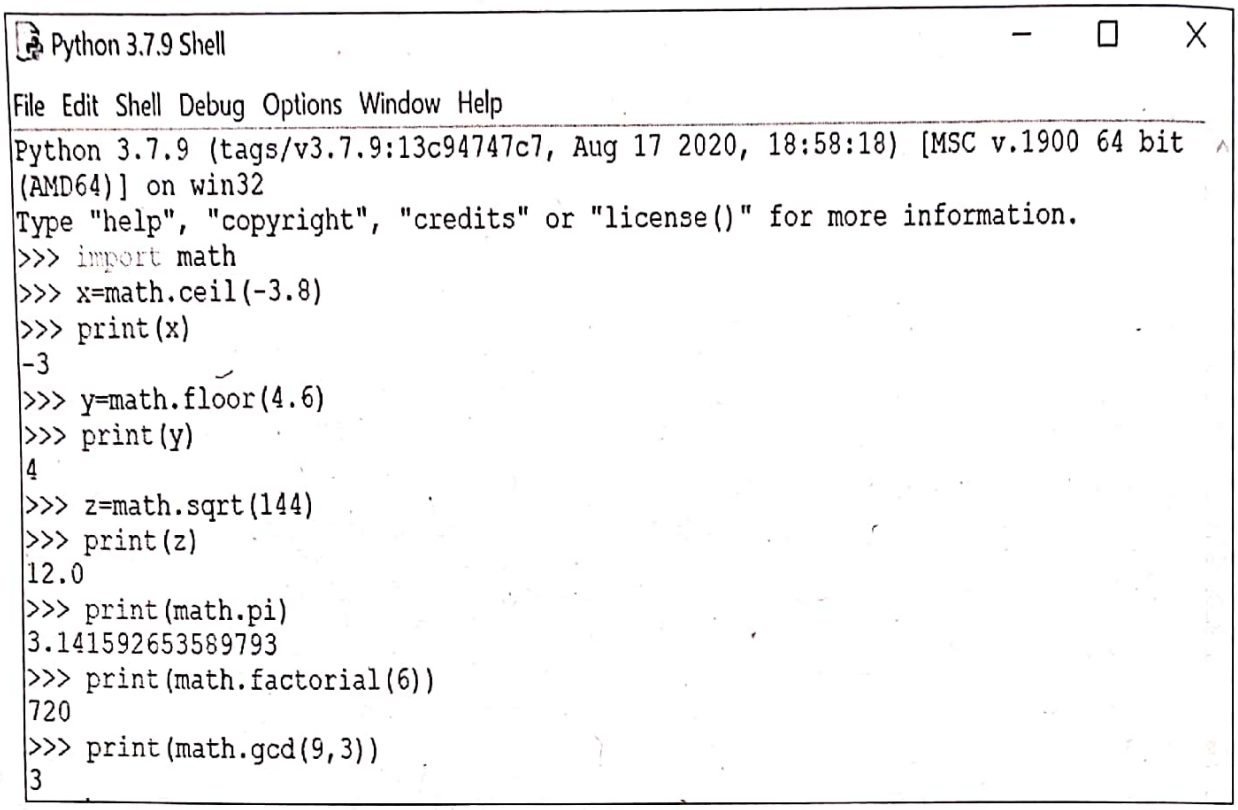
Python has a set of built-in math functions, including an extensive math module, that allows you to perform mathematical tasks on numbers.

- Built in functions : python supports basic mathematical functions like min(), max(), abs(),pow() etc. as shown below.

```
Python 3.7.9 Shell
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x=min(2,6,0)
>>> print(x)
0
>>> x=abs(-20.5)
>>> print(x)
20.5
>>> x=pow(4,2)
>>> print(x)
16
```

The Math Module

Python has also a built-in module called math, which extends the list of mathematical functions. To use it, you must import the math module.



4.1.9 Internet access:

'urllib' is a Python module that can be used for opening URLs. It defines functions and classes to help in URL actions.One can also access and retrieve data from the internet like XML, HTML, JSON, etc.

- How to Open URL using Urllib:

Before we run the code to connect to Internet data, we need to import statement for URL library module or "urllib".

```
>>> import urllib
>>> import urllib.request
>>> sampleurl=urllib.request.urlopen('https://www.amazon.com')
>>> print("the result:"+ str(sampleurl.getcode()))

the result:200
```

The above result 200 indicates that internet is working correctly (i.e. HTTP request is processed successfully)

Urllib is a package contains several modules for working with URLs, such as:

urllib.request -for opening and reading.

urllib.parse -for parsing URLs

urllib.error -for the exceptions raised

urllib.robotparser- for parsing robot.txt files

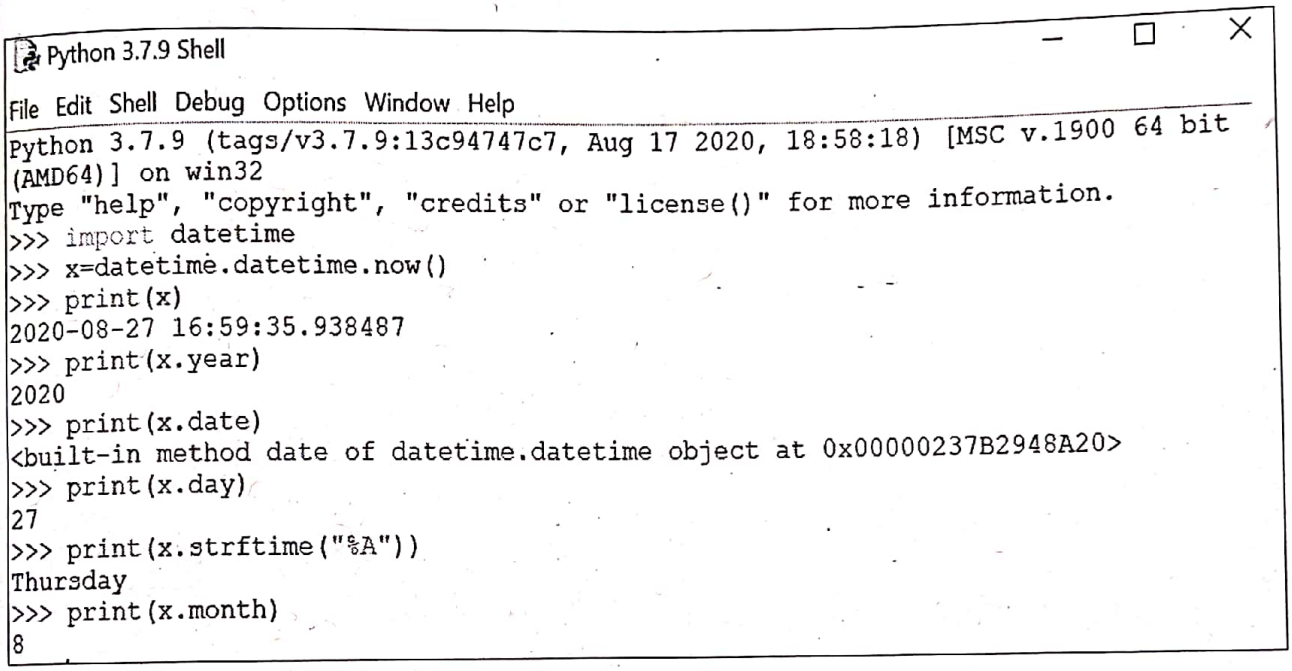
4.1.10 Dates and time:

A date in Python is not a data type of its own, but we can import a module named datetime to work with dates as date objects.

Always import datetime module before using any methods corresponding to it.

datetime.now() displays current date , containing year, month, day, hour, minute, second, and microsecond.

Below given sample describes proper information of datetime module



To create a date, we can use the datetime() class (constructor) of the datetime module.

The datetime() class requires three parameters to create a date: year, month, day.

The datetime object has a method for formatting date objects into readable strings.

The method is called strftime(), and takes one parameter, format, to specify the format of the returned string:

%a - Weekday, short version

%A -Weekday, full version

%w -Weekday as a number 0-6, 0 is Sunday

%j -Day number of year 001-366 etc.

```
Python 3.7.9 Shell
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58
MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for m
formation.
>>> import datetime
>>> x=datetime.datetime(2020,10,12)
>>> print(x)
2020-10-12 00:00:00
>>> print(x.strftime("%B"))
October
>>> print(x.strftime("%m"))
10
>>> print(x.strftime("%j"))
286
>>> print(x.strftime("%w"))
1
>>> print(x.strftime("%a"))
Mon
>>> print(x.strftime("%A"))
Monday
>>> print(x.strftime("%d"))
12
```

4.1.11 Data compression:

In python, the data can be archived, compressed using the modules like zlib, gzip, bz2,lzma,zipfile and tarfile. To use the respective module, you need to import the module first.

- With the help of *zlib.compress(s)* method, we can get compress the bytes of string. In below mentioned example we can see that initially we cannot compress string so it is converted to byte format and then by using zlib.compress(s) method, we are able to compress the string in the byte format.

Python 3.7.9 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18
:58:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" fo
r more information.
>>> import zlib
>>> s="It's fun to learn python"
>>> len(s)
24
>>> t=zlib.compress(s)
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    t=zlib.compress(s)
TypeError: a bytes-like object is required, not 'str'
>>> s=b'It"s fun to learn python'
>>> len(s)
24
>>> t=zlib.compress(s)
>>> len(t)
32
>>> zlib.decompress(t)
b'It"s fun to learn python'
```

➢ The *gzip* module provides function and class definitions that make it easy to handle simple Gzip files. These allow you to open a compressed file and read it as if it were already decompressed.

gzip.open(*filename* , *mode* , *level* , *fileobj*) → gzip.GzipFile

Open the file named *filename* with the given *mode* ('r', 'w' or 'a'). The compression *level* is an integer that provides a preference for speed versus size. As an alternative, you can open a file or socket separately and provide a *fileobj*, for example, f= open('somefile','r'); zf= gzip.open(fileobj=f).

Once this file is open, it can perform ordinary read, readline, readlines, write, writeline, and writelines operations.

Below mentioned example describes the similar working of gzip file alike to zlib.

```
Python 3.7.9 Shell
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58
:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for m
ore information.
>>> import gzip
>>> s = b'This is python author, and students so read it
carefully.'
>>> print
<built-in function print>
>>> print(len(s))
57
>>> t = gzip.compress(s)
>>> print(len (t))
74
>>> z=gzip.decompress(t)
>>> print(z)
b'This is python author, and students so read it careful
ly.'
```

The *bz2* compress/decompress works in similar way.

a .tar file is often compressed using *GZip* to create *a .tar.gz* file, sometimes called a *.tgz* file. In the case of the ZIP file archive, the compression algorithms are already part of the zipfile module.

These modules are very flexible and can be used in a variety of ways by an application program. The interface to zlib and bz2 are very similar. The interface to gzip is greatly simplified.

4.1.12 Performance measurement:

Time is precious. As programmers, we have to write programs which take less time for their execution. Python provides the *timeit* module for measuring the execution time of small code snippits. This can be called from the command line, or by importing it into an exisiting program. In the below given example we will understand the difference of calculating the running time of a program in python.

time() - used to get the current time.

timeit() - used to calculate the execution time of any program in python.

Following example shows the steps to calculate the running time of a program using

- time() function of time module.

- Store the starting time before the first line of the program executes.
- Store the ending time after the last line of the program executes.
- The difference between *ending time* and *starting time* will be the running time of the program.
- timeit() method accepts four arguments:
  - *setup*, which takes the code which runs before the execution of the main program, the default value is *pass*
  - *stmt*, is a statement which we want to execute.
  - *timer*, is a timeit.Timer object, we don't have to pass anything to this argument.
  - *number*, which is the number of times the statement will run.

*time.py - C:\Users\91963\AppData\Local\Programs\Python\Pyt...

File Edit Format Run Options Window Help

```
#Using time module
import time

# starting time
start = time.time()
# program body starts
for i in range(5):
    print(i)
# sleeping for 1 sec to get 10 sec runtime
time.sleep(1)
# program body ends
# end time
end = time.time()
# total time taken
print(f"Runtime of the program is {end - start}")



# Using timeit module
import timeit

setup_code = "from math import factorial"
statement = """for i in range(5):
    factorial(i)
"""

print(f"Execution time is: {timeit.timeit(setup = setup_code,
stmt = statement, number = 10000000)}")
```

```
Python 3.7.9 Shell
File Edit Shell Debug Options Window Help
Python 3.7.9 (tags/v3.7.9:13c94747c7, Aug 17 2020, 18:58:18) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
0
1
2
3
4
Runtime of the program is 1.0200505256652832
Execution time is: 4.3847128
```

4.2 Creating virtual environment:

4.2.1 Introduction:

A virtual environment is a tool that helps to keep dependencies required by different projects separate by creating isolated python virtual environments for them.

Imagine a scenario where you are working on two web based python projects and both of them uses different versions of Django. In such situations virtual environment can be really useful to maintain dependencies of both the projects. This is a real problem for Python since it can't differentiate between versions in the "site-packages" directory. So both versions would reside in the same directory with the same name. This is where virtual environments come into play. To solve this problem, we just need to create two separate virtual environments for both the projects.

4.2.2 Generating virtual environments:

The module used to create and manage virtual environments is called venv. It will usually install the most recent version of Python that you have available. To create a virtual environment, decide upon a directory where you want to place it, and run the venv module as a script with the directory path:

python3 -m venv my-env

This will create the my-env directory if it doesn't exist, and also create directories inside it containing a copy of the Python interpreter, the standard library, and various supporting files. This directory will contain all the necessary executables to use the packages that a python project would need. As shown above, one may specify python interpreter of their choice.

Once you've created a virtual environment, you may activate it:

my-env\Scripts\activate.bat

Once the virtual environment is activated, the name of your virtual environment will appear and let you know that which virtual environment is currently active.

4.3.3 Managing packages with pip (Python Package Index):

pip is the de-facto package manager in the Python world. It can install packages from many sources, but PyPI is the primary package source where it's used (These files are stored in a large "on-line repository" termed as Python Package Index PyPI).

pip uses PyPI as the default source for packages and their dependencies.

pip install package_name

by running above statement pip will look for that package on PyPI and if found, it will download and install the package on your local system. You can install the latest version of a package by specifying a package's name pip has a number of subcommands: "search", "install", "uninstall", "freeze", etc. pip can also be downloaded and installed using command-line .

4.3 Python and MySQL:

Python Database API (Application Program Interface) is the Database interface for the standard Python. here are various Database servers supported by Python Database such as MySQL, GadFly, mSQL, PostgreSQL, Microsoft SQL Server 2000, Informix, Interbase, Oracle, Sybase etc.

- **What is MySQL?**
  - MySQL is an open-source relational database management system (RDBMS)
  - MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses.
- **Why MySQL?**
  - MySQL is released under an open-source license. So you have nothing to pay to use it.
  - MySQL uses a standard form of the well-known SQL data language.
  - Cross-platform support

python needs a MySQL driver to access the MySQL database.

4.3.1 Installing MySQL Connector:

To install Mysql connector, you may download it through cmd:

>>>pip3 install mysql-connector

Or, you may download the Mysql connector for python from online source. (Oracle website recommended: https://dev.mysql.com/downloads/connector/python/)

4.3.2 Verifying the Connector Installation:

To test if the installation was successful, or if you already have "MySQL Connector" installed, write statement on python shell

>>> import mysql.connector

if this is executed with no errors, you have the "mysql.connector" module installed.

4.3.3 Using MySQL from Python:

Start by creating a connection to the database to create a database named "mydatabase".

```
import mysql.connector

mydb = mysql.connector.connect(
  host="localhost",
  user="myusername",
  password="mypassword"
)

mycursor = mydb.cursor()

mycursor.execute("CREATE DATABASE mydatabase")
```

If the above code was executed with no errors, you have successfully created a database. If the database does not exist, you will get an error.

One can also use an alias foy mysqlconnector:

>>>import mysql.connector as m

Note: Ensure that once cursor connection object is established all the sql queries can be written inside execute method of cursor connection object (in our example mycursor is cursor connection object)

4.3.4 Retrieving all rows from a table:

Ensure that you have created a table having specific rows. To check for its existence:

```
mycursor.execute("SHOW TABLES")

for x in mycursor:
    print(x)
```

The above code will return the table name.

Try to enter few records using simple sql rules inside execute().

To select a specific record or return all records generally fetchone ()/ fetchall() is used.

Using Fetchall():

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchall()

for x in myresult:
  print(x)
```

Using Fetchone():

```
mycursor = mydb.cursor()

mycursor.execute("SELECT * FROM customers")

myresult = mycursor.fetchone()

print(myresult)
```

To select only some of the columns in a table, use the "SELECT" statement followed by the column name(s):

>>> mycursor.execute("SELECT name, address FROM customers")

4.3.5 Inserting rows into a table:

To fill a table in MySQL, use the "INSERT INTO" statement.

>>>sql = "INSERT INTO customers (name, address) VALUES (%s, %s)"

>>>val = ("Paul", "Ahmedabad")

>>>mycursor.execute(sql, val)

>>>mydb.commit()

Ensure that mydb.commit() is required to make the changes, otherwise no changes are made to the table.one can also enter multiple entries by using exceutemany() instead of execute().

4.3.6 Deleting rows from table:

One can delete records from an existing table by using the "DELETE FROM" statement:

>>> sql = "DELETE FROM customers WHERE address = 'Vadodra'"

Ensure that WHERE clause specifies which record(s) that should be deleted. If you omit the WHERE clause, all records will be deleted!

4.3.7 Updating rows in a table:

One can update existing records in a table by using the "UPDATE" statement:

>>> sql = "UPDATE customers SET address = 'Jamnagar' WHERE address = 'Ahmedabad'"

4.3.8 Creating database tables through Python:

The above operations were carried out only after checking whether table exists or not. If table does not exist create it immediately after creating cursor connection object 'mycursor'.

>>>mycursor.execute ("CREATE TABLE customers (name VARCHAR(255), address VARCHAR(255))")

One may set the primary key specifications accordingly or alter the table if required.

In a nutshell below are the steps for working with MySql through Python:

- Installing MySQL
- Import MySQL in Python
- Connect with MySQL (i.e making database connection object)
- Create cursor(i.e making cursor connection object_
- Create Database (i.e. create table)
- Use created database
- Ready to perform database operations on test database(execution of sql queries through execute)

❖ TRY IT YOURSELF (SELF EXERCISES)

1. Create new database for storing student information using SQLite.
   - Create table to store student id (sid) using python script
   - Insert record in student info (sid, student name and branch) using python script
   - Fetch one record using python script
   - Update record
   - Delete record
   - Display student info records
2. Create a class Person which has attributes name and age. Person has a method to verify his age for voting. A method will throw exception if he is under-aged (i.e. less than 18) for Voting. In such case, application class must handle the exception. If age is more than 18, the program will run smoothly.
3. Write a program that prompts the user to enter an integer for today's day of the week (Sunday is 0, Monday is 1, ..., and Saturday is 6). Also prompt the user to enter the number of days after today for a future day and display the future day of the week. Sample Run is shown below:

Enter today's day: 1

Enter the number of days elapsed since today: 3

Today is Monday and the future day is Thursday.

☺ ☺ ☺ ☺ ☺

CC-305 Unit – 4 Python Programming Practicals

1) Write a program to handle some built in exceptions like ZeroDivisionError.

```
a =int(input("Enter a dividend"))
b =int(input("Enter a divisor"))
try:
    Ans= a/b
except ZeroDivisionError:
    print ("zero division error")
else:
    print ("Answer =" , ans )
```

Output:

Enter a dividend:10
Enter a divisor:0
zero division error

2) Write a program to handle multiple exceptions like SyntaxError and TypeError

```
try:
    a =eval(input("Enter a dividend"))
    b =eval(input("Enter a divisor"))
    Ans= a/b
except (TypeError, SyntaxError):
    print("Error")
else:
    print ("Answer =" , ans )
```

Output:

Enter a dividend:10
Enter a divisor:1
>>>Ans=1.0
Enter a dividend:10
Enter a divisor: k
>>>Error

3) Write a program to import "os" module and to print the current working directory and returns a list of all module functions.

```
import os
# Return the absolute path
print(os.getcwd( ))

# Return the list of functions of OS
print(dir(os))
```

Output:

```
E:\SEM 5\Sqlite
['DirEntry', 'F_OK', 'MutableMapping', 'O_APPEND', 'O_BINARY', 'O_CREAT', 'O_EXCL', 'O_NOINHERIT', 'O_RANDOM', 'O_RDONLY', 'O_RDWR', 'O_SEQUENTIAL', 'O_SHORT_LIVED', '
O_TEMPORARY', 'O_TEXT', 'O_TRUNC', 'O_WRONLY', 'P_DETACH', 'P_NOWAIT', 'P_NOWAITO', 'P_OVERLAY', 'P_WAIT', 'PathLike', 'R_OK', 'SEEK_CUR', 'SEEK_END', 'SEEK_SET', 'TMP
_MAX', 'W_OK', 'X_OK', '_AddedDllDirectory', '_Environ', '__all__', '__builtins__', '__cached__', '__doc__', '__file__', '__loader__', '__name__', '__package__', '__sp
ec__', '_check_methods', '_execvpe', '_exists', '_exit', '_fspath', '_get_exports_list', '_putenv', '_unsetenv', '_wrap_close', 'abc', 'abort', 'access', 'add_dll_dire
ctory', 'altsep', 'chdir', 'chmod', 'close', 'closerange', 'cpu_count', 'curdir', 'defpath', 'device_encoding', 'devnull', 'dup', 'dup2', 'environ', 'error', 'execl',
'execle', 'execlp', 'execlpe', 'execv', 'execve', 'execvp', 'execvpe', 'extsep', 'fdopen', 'fsdecode', 'fsencode', 'fspath', 'fstat', 'fsync', 'ftruncate', 'get_exec_p
ath', 'get_handle_inheritable', 'get_inheritable', 'get_terminal_size', 'getcwd', 'getcwdb', 'getenv', 'getlogin', 'getpid', 'getppid', 'isatty', 'kill', 'linesep', 'l
ink', 'listdir', 'lseek', 'lstat', 'makedirs', 'mkdir', 'name', 'open', 'pardir', 'path', 'pathsep', 'pipe', 'popen', 'putenv', 'read', 'readlink', 'remove', 'removedi
rs', 'rename', 'renames', 'replace', 'rmdir', 'scandir', 'sep', 'set_handle_inheritable', 'set_inheritable', 'spawnl', 'spawnle', 'spawnv', 'spawnve', 'st', 'startfile
', 'stat', 'stat_result', 'statvfs_result', 'strerror', 'supports_bytes_environ', 'supports_dir_fd', 'supports_effective_ids', 'supports_fd', 'supports_follow_symlinks
', 'symlink', 'sys', 'system', 'terminal_size', 'times', 'times_result', 'truncate', 'umask', 'uname_result', 'unlink', 'urandom', 'utime', 'waitpid', 'walk', 'write']
>>>
```

4) Write a program to provide a function for making file lists from directory wildcard searches:

```
import os
def findfiles(filename, searchpath):
    result= []
#walking topdown from root
    for root, dir, files in os.walk(searchpath):
        if filename in files:

        result.append(os.path.join(root,filename))
        if result == []:
            return("No such file found")
        else:
            return result
```

```
name= input("Enter filename with extension for
file creation")
f= open (name,'w')
f.close()

file=input("Enter filename with extension for
searching")
print(findfiles(file,os.getcwd())
```

Output:

Enter filename with extension for file creation Temp.txt
Enter filename with extension for searching tem
No such file found
Enter filename with extension for file creation Temp.txt
Enter filename with extension for searching Temp.txt
['E:\\Sem 5\\Sqlite\\Temp.txt']

15) Write a program to import datetime module and format the date as required. Also use the same module to calculate the difference between your birthday and today in days.

```
import datetime
year =int (input("Enter your birth year")
month=int (input("Enter your birth month")
day =int (input("Enter your birth date")
birthday=datetime.datetime.now()
Difference= now-birthday
print("Difference is", Diffrerence.days,"days")
```

Output:

Enter your birth year 2000
Enter your birth month 11
Enter your birth date 15
Difference is 7216 days

16) Write a program to create a database named "Sample_DB" in MySQL(). [First ensure connection is made or not and then check if the database Sample_DB already exists or not, if yes then print appropriate message].

Initially

```
('ad',)
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('srms',)
('test',)
>>>
```

```
import mysql.connector
mydb=mysql.connector.connect(
 host = "localhost",
 user ="root" ,
 password = "",
)
print (mydb)
try:
      cursor =mydb.cursor()
      cursor.execute("Create database Sample_DB")
      cursor.execute("show databases")
      my = cursor.fetchall()
      print("Database Created \n").
      for i in my:
           print(i)
      except error as e :
           print(e)
```

After

```
Database Created

('ad',)
('information_schema',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('sample_db',)
('srms',)
('test',)
>>>
```

Write a program to retrieve and display all the rows in the employee table. [First create an employee table in the Sample_DB with the fields as eid, name, sal. Also enter some valid records]

Creation of Table

```
import mysql.connector
try:
    mydb=mysql.connector.connect(
    host="localhost",
    user="root",
    password=""
    database="Sample_DB"
    )
    if mydb is None:
        print("Database not Connected")
    else:
        print("Database is Connected")
    cursor = mydb.cursor()
cursor.execute("create table employee (id varchar(10)
primary key,name varchar(10),sal numeric(15))")
    print("Employee table created")

except error as e :
    print(e)
```

Output:

Database is Connected
Employee table created

INSERT DATA MANUALLY

```
import mysql.connector
try:
    mydb=mysql.connector.connect(
    host="localhost",
    user="root",
    password=""
```

```
        database="Sample_DB"
        )
        if mydb is None:
            print("Database not Connected")
        else:
            print("Database is Connected")
        cursor = mydb.cursor()
        cursor.execute("insert into employee values('EMP01'
,'ketan' , '150000')")
        cursor.execute("insert into employee values('EMP02'
,'chetan' , '180000')")
        cursor.execute("insert into employee values('EMP03'
,'shalvi' , '110000')")
        cursor.execute("insert into employee values('EMP04'
,'varun' , '210000')")
        print("New table created")
        mydb.commit()
    except error as e :
        print(e)
```

Output:

Database connected
Records Inserted

RETRIVE DATA

```
import mysql.connector
try:
    mydb=mysql.connector.connect(
    host="localhost",
    user="root",
    password=""
    database="Sample_DB"
    )
    if mydb is None:
        print("Database not Connected")
    else:
```

```
        print("Database is Connected")
    cursor = mydb.cursor()
    cursor.execute("select * from employee")
    rows = cursor.fetchall()
    #print (rows)
    #To display all records
    for i in rows:
        print(i)
    mydb.commit()

    except error as e :
    print(e)
```

Output:

Database is Connected
('EMP01' ,'ketan' , '150000')
('EMP02' ,'chetan' , '180000')
('EMP03' ,'shalvi' , '110000')
('EMP04' ,'varun' , '210000')

8) Write a program to insert several rows into employee table from the keyboard.

```
import mysql.connector
try:
    mydb=mysql.connector.connect(
    host="localhost",
    user="root",
    password=""
    database="Sample_DB"
    )
    if mydb is None:
        print("Database not Connected")
    else:
        print("Database is Connected")
    cursor = mydb.cursor()
```

```
#insert data dynamically
while('True') :
      choice =input("Would you like to enter record")
      if (choice =="Yes" or choice=="yes"):
            empid=input("Enter Employee id")
            name = input ("Enter Employee name")
            salary = int(input("Enter Employee Salary"))
            sql= "INSERT INTO employee (id,name,sal) VALUES
(%s,%s,%s)"
            val = (empid,name,salary)
            cursor.execute(sql,val)
            mydb.commit()
      elif(choice =="No" or choice="no"):
            break
      else:
            print(:Wrong Input")
except error as e :
      print(e)
```

Output:

Database is Connected
Would you like to enter record Yes
Enter Employee id EMP05
Enter Employee name Amitabh
Enter Employee Salary 300000
Would you like to enter record no

9) Write a program to delete a row from an employee table by accepting the employee identity number (eid) from the user.

```
import mysql.connector
try:
      mydb=mysql.connector.connect(
      host="localhost",
      user="root",
      password=""
      database="Sample_DB"
      )
```

```
        if mydb is None:
            print("Database not Connected")
        else:
            print("Database is Connected")
        cursor = mydb.cursor()
    #delete data dynamically
        empid =input('Enter Employee id')
        sql="delete from employee where id =%s"
        val = (empid)
        cursor.exceute(sql,val)
        print("Record Deleted")
        mydb.commit()
    except error as e :
        print(e)
```

Output:

```
Database is Connected
('EMP01', 'Ketan', 150000)
('EMP02', 'Chirayu', 170000)
('EMP04', 'Raj', 130000)
('EMP05', 'Amitabh', 300000)
>>>
==============================
Database is Connected
('EMP01', 'Ketan', 150000)
('EMP02', 'Chirayu', 170000)
('EMP03', 'Payal', 100000)
('EMP04', 'Raj', 130000)
('EMP05', 'Amitabh', 300000)
>>>
==============================
Database is Connected
Enter Employee Id EMP03
Record Deleted
>>>
==============================
Database is Connected
('EMP01', 'Ketan', 150000)
('EMP02', 'Chirayu', 170000)
('EMP04', 'Raj', 130000)
('EMP05', 'Amitabh', 300000)
>>>
```

10) Write a program to increase the salary (sal) of an employee in the employee table by accepting the employee identity number (eid) from the user.

```
import mysql.connector
try:
      mydb=mysql.connector.connect(
      host="localhost",
      user="root",
      password=""
      database="Sample_DB"
      )
      if mydb is None:
            print("Database not Connected")
      else:
            print("Database is Connected")
      cursor = mydb.cursor()
#delete data dynamically
      empid =input('Enter Employee id')
      increment = int(input("Enter Increment Amount"))
      sql="update employee set sal=sal +%s where id = %s"
      val= (increment,empid, )
      cursor.execute(sql,val)
      print ("Record Updated")
      mydb.commit()
except error as e :
      print (e)
```

Output:

```
==========================================
Database is Connected
('EMP01', 'Ketan', 150000)
('EMP02', 'Chirayu', 170000)
('EMP04', 'Raj', 130000)
('EMP05', 'Amitabh', 300000)
>>>
==========================================
Database is Connected
Enter Employee Id EMP01
Enter Increment Amount 50000
Record Updated
>>>
==========================================
Database is Connected
('EMP01', 'Ketan', 200000)
('EMP02', 'Chirayu', 170000)
('EMP04', 'Raj', 130000)
('EMP05', 'Amitabh', 300000)
>>>
```

11) Write a program to create a table named new_employee_tbl with the fields eno, ename, gender and salary in Sample_DB database. The datatypes of the fields are eno-int, enamechar(30), gender-char(1) and salary-float.

```
try:
    mydb=mysql.connector.connect(
    host="localhost",
    user="root",
    password=""
    database="Sample_DB"
    )
    #check database is connected or not
```

```
        if mydb is None:
            print("Database not Connected")
        else:
            print("Database is Connected")
        cursor = mydb.cursor()
        cursor.execute("create table new_employee_tbl(eno
int,ename char(30), gender char(1) , salary float)")
        print("New table created")
        mydb.commit()
    except error as e:
        print(e)
```

Output:

Database is Connected

New table created

☺ ☺ ☺ ☺ ☺

Exercises

1) What is an exception? How python handles exception? Briefly discuss in-built exception through an example.

2) What is displayed when the following program is run?

```
try:
    list = 10 * [0]
    x = list[10]
    print("Done ")
except IndexError:
    print("Index out of bound")
```

3) What is displayed when the following program is run?

```
def main():
    try:
        f()
        print("After the function call")
    except IndexError:
        print("Index out of bound")
    except:
        print("Exception in main")

def f():
    try:
        s ="abc"
        print(s[3])
```

4) How do you raise an exception in a function and define a custom exception class.

5) How an user creates virtual environment in python? Demonstrate the usage of PIP.

6) What is MySQL? How does it differ from other SQLite?

7) Write a program for creation of new database for storing student information and perform following manipulations:

- Create table to store student info (sinfo) using python script
- Insert record in student info using python script

- Fetch one record using python script
- Update record
- Delete record
- Display studentinfo records
- Fetch and show all records at end.

8) Explain the wild card feature in python

9) How can your script recover from an exception?

10) Name two ways to specify actions to be run at termination time, whether an exception occurs or not.

11) Name three things that exception processing is good for:

12) What are the two common variations of the try statement?

13) What are the technical features of MySQL?

14) Briefly explain time() module in python for performance measurement.

15) How URL's can be accessed using python?

☺ ☺ ☺ ☺ ☺

Self-Test Examination

1. What will be the output of the following Python code?

```
>>> a={1:"A",2:"B",3:"C"}
>>> a.items()
```

(A) dict_items([(1, 'A'), (2, 'B'), (3, 'C')])
(B) Syntax error
(C) dict_items([('A'), ('B'), ('C')])
(D) dict_items([(1,2,3)])

2. If a is a dictionary with some key-value pairs, what does a.popitem() do?

(A)Removes all the key-value pairs
(B)Removes an arbitrary element
(C) Removes the key-value pair for the key given as an argument.
(D) Invalid method for dictionary

3. Which of the following will run without errors?

(A) round(45.8)
(B) round(6352.898,2,5)
(C) round()
(D) round(7463.123,2,1)

4. What datatype is the object below?

L = [1, 23, 'hello', 1].

(A) list
(B) dictionary
(C) array
(D) tuple

5. What is the maximum possible length of an identifier?

(A) 31 characters
(B) 63 characters
(C) 79 characters
(D) none of the mentioned

6. All keywords in Python are in

(A) lower case
(B) UPPER CASE
(C) Capitalized
(D) None of the mentioned

7. Which of these in not a core datatype?

(A) Lists
(B) Dictionary
(C) Tuples
(D) Class

8. What is the output of the code shown below?

```
'The {} side {1} {2}'.format('bright', 'of', 'life')
```

(A) Error
(B) 'The bright side of life'

(C) 'The {bright} side {of} {life}'
(D) No output

9. What is the result of the expression shown below if x=56.236?

```
print("%.2f"%x)
```

(A) 56.00
(B) 56.24
(C) 56.23
(D) 0056.236

10. Which of the following formatting options can be used in order to add 'n' blank spaces after a given string 'S'?

(A) print("-ns"%S)
(B) print("-ns"%S)
(C) print("%ns"%S)
(D) print("%-ns"%S)

11. The output of the code shown below is:

```
s='{0}, {1}, and {2}'
s.format('hello', 'good', 'morning')
```

(A) 'hello good and morning'
(B) 'hello, good, morning'
(C) 'hello, good, and morning'
(D) error

12. What is the output of the code shown below?

```
def d(f):
def n(*args):
return '$' + str(f(*args))
return n
@d
def p(a, t):
return a + a*t
 print(p(100,0))
```

(A) 100
(B) $100
(C) $0
(D) 0

13. What is the output when following code is executed?

```
print r"\nhello"
The output is
```

(A) a new line and hello
(B) \nhello
(C) the letter r and then hello
(D) Error

14. What is the output of the following code?

```
example = "snow world"
example[3] = 's'
print example
```

(A) snow
(B) snow world
(C) Error
(D) snos world

15. Suppose i is 5 and j is 4, i + j is same as

(A) i.__add(j)
(B) i.__add__(j)
(C) i.__Add(j)
(D) i.__ADD(j)

16. What is the output of the following?

```
print("Hello {1} and {0}".format('bin', 'foo'))
```

(A) Hello foo and bin
(B) Hello bin and foo
(B) Error
(D) None of the mentioned

17. What is the output of the following?

```
print('ab'.isalpha())
```

(A) True
(B) False
(C) None
(D) Error

18. What is the output of the following?

```
print('1.1'.isnumeric())
```

(A) True
(B) False
(C) None
(D) Error

19. What is the output of the following?

```
print('abcd'.translate({'a': '1', 'b': '2', 'c': '3', 'd': '4'}))
```

(A) abcd
(B) 1234
(C) error
(D) none of the mentioned

20. What is the output of the following code ?

```
>>>max("what are you")
```

(A) Error
(B) u
(C) t
(D) y

21. What is the output of the following?

```
x = [12, 34]
print(len(''.join(list(map(str, x)))))
```

(A) 4
(B) 5
(C) 6
(D) Error

22. Is the output of the function abs() the same as that of the function math.fabs()?

(A) Sometimes
(B) always
(C) never
(D) none of the mentioned

23. What does os.getlogin() return?

(A) name of the current user logged in

(B) name of the superuser

(C) gets a form to login as a different user

(D) all of the above

24. Which of the following is true about top-down design process?

(A) The details of a program design are addressed before the overall design

(B) Only the details of the program are addressed

(C) The overall design of the program is addressed before the details

(D) Only the design of the program is addressed

25. What is the output of the function shown below (random module has already been imported)?

```
random.choice('sun')
```

(A) sun

(B) u

(C) either s, u or n

(D) error

26. What is the output of this code?

```
import sys
eval(sys.stdin.readline())
"India"
```

(A) India5

(B) India

(C) 'India\n'

(D) 'India'

27. Which of the following functions does not accept any arguments?

(A) position

(B) fillcolor

(C) goto

(D) setheading()

28. What is the output of the following code?

```
import turtle
t=turtle.Pen()
t.right(90)
t.forward(100)
t.heading()
```

(A) 0.0

(B) 90.0

(C) 270.0

(D) 360.0

29. Python is said to be easily

(A) Readable language
(B) Writable language
(C) bug-able language
(D) Script-able language

30. Extensible programming language that can be extended through classes and programming interfaces is

(A) Python
(B) Perkl
(C) PHP
(D) Ada

31. Python was released publicly in

(A) 1967
(B) 1971
(C) 1981
(D) 1991

32. Which of these in not a core datatype?

(A) Lists
(B) Dictionary
(C) Tuples
(D) Class

33. Given a function that does not return any value, What value is thrown by it by default when executed in shell.

(A) int
(B) bool
(C) void
(D) None

34. Following set of commands are executed in shell, what will be the output?

```
>>>str="hello"
>>>str[:2]
>>>str
```

(A) he
(B) lo
(C) olleh
(D) hello

35. Which of the following will run without errors (multiple answers possible)?

(A) round(45.8)
(B) round(6352.898,2)
(C) round()
(D) round(7463.123,2,1)

36. What is the return type of function id?

(A) int
(B) float
(C) bool
(D) dict

37. In python we do not specify types, it is directly interpreted by the compiler, so consider the following operation to be performed.

```
>>>x = 13 ? 2
```

objective is to make sure x has a integer value, select all that apply (python 3.xx)

(A) x = 13 // 2
(B) x = int(13 / 2)
(C) x = 13 / 2
(D) x = 13 % 2

38. What error occurs when you execute?

```
apple = mango
```

(A) SyntaxError
(B) NameError
(C) ValueError
(D) TypeError

39. Carefully observe the code and give the answer.

```
def example(a):
a = a + '2'
a = a*2
return a
>>>example("hello")
```

(A) indentation Error
(B) cannot perform mathematical operation on strings
(C) hello2
(D) hello2hello2

40. What datatype is the object below?

```
L = [1, 23, 'hello', 1]
```

(A) List
(B) dictionary
(C) array
(D) tuple

41. In order to store values in terms of key and value we use what core datatype.

(A) List
(B) tuple
(C) class
(D) dictionary

42. Which of the following results in a SyntaxError (Multiple answers possible)?

(A)'"Once upon a time...", she said.'
(B) "He said, "Yes!""
(C) '3\'
(D) "'That's okay"'

43. What is the average value of the code that is executed below?

```
>>>grade1 = 80
>>>grade2 = 90
>>>average = (grade1 + grade2) / 2
```

(A) 85
(B) 85.0
(C) 95
(D) 95.0

44. Suppose t = (1, 2, 4, 3), which of the following is incorrect?

(A) print(t[3])
(B) t[3] = 45
(C) print(max(t))
(D) print(len(t))

45. Select all options that print

```
hello-how-are-you
```

(A) print('hello', 'how', 'are', 'you')
(B) print('hello', 'how', 'are', 'you' + '-' * 4)
(C) print('hello-' + 'how-are-you')
(D) print('hello' + '-' + 'how' + '-' + 'are' + '-' + 'you')

46. What is the return value of trunc() ?

(A) int
(B) bool
(C) float
(D) None

47. What is the output when following statement is executed?

```
>>>"a"+"bc"
```

(A) a
(B) bc
(C) bca
(D) abc

48. What is the output when following statement is executed?

```
>>>"abcd"[2:]
```

(A) a
(B) ab
(C) cd
(D) dc

49. What is the output when following code is executed?

```
>>>str1="helloworld"
>>>str1[::-1]
```

(A) dlrowolleh
(B) hello
(C) world
(D) helloworld

50. What is the output of the following code?

```
>>>example = "helle"
>>>example.find("e")
```

(A) Error
(B) -1
(C) 1
(D) 0