**UNIT -1**

**2Marks**

1. **How to write a comment line in Python? Mention two types.**

Comments in Python are the lines in the code that the compiler ignores during the execution of the program. Comments enhance the readability of the code and help the programmers to understand the code very carefully. There are three types of comments in Python –

* Single line Comments
* Multiline Comments
* Docstring Comments

1. **What is Python Virtual Machine?**

Python Virtual Machine (PVM) is a program that provides a programming environment. The role of PVM is to convert the byte code instructions into machine code so the computer can execute those machine code instructions and display the output.

1. **List any four flavors of Python.**

* CPython
* Jython
* RubyPython
* Pythonxy

1. **Give 2 step process of Python program execution**

* **Compilation**: compile the program using python compiler.
* **Interpreter**: Python uses an interpreter called PVM (Python Virtual Machine), which understands the byte code and converts it into machine code.

1. **List any four standard datatypes supported by Python.**

* NoneType
* Numeric types
* Sequences
* Sets
* Mappings

1. **How to determine the data type of a variable? Give the syntax and example**

To know the datatype of a variable or object, we can use the type() function.

**For example, type(a) displays the datatype of the variable ‘a’.**

a=15

print(type(a))

<class ‘int’>

1. **What is the purpose of membership operators? Give example**

The membership operators are useful to test for membership in a sequence such as strings, lists, tuples, or dictionaries. For example, if an element is found in this sequence or not, can be asserted using these operators. There are 2 membership operators:

• in

• not in

**example:**

names = [‘Rani’,’Yamini’,’Ram’]

for name in names:

print (name)

1. **How to input data in Python? Give syntax and example**

Python provides input() function to accept input from the user. This function takes a value from the keyboard and returns it as a string. It is always better to display a message to the user so that the user can understand what to enter.

**Example**

str=input("Enter your name ")

print("Hello", str)

1. **List four type conversion functions.**

|  |  |
| --- | --- |
| **Function** | **Description** |
| str(y) | It converts y to a string. |
| tuple(y) | It converts y to a tuple. |
| list(y) | It converts y to a list. |
| set(y) | It converts y to a set. |

1. **List any four categories of Operators in Python.**

* Arithmetic operators.
* Assignment operators.
* Comparison operators.
* Logical operators.

1. **What is indentation? Why it is required?**

Indentation refers to the spaces at the beginning of a code line.Programs get structured through indentation.indentation from any program code, but in Python it is a requirement and not a matter of style. This principle makes the code look cleaner and easier to understand and read.Any statements written under another statement with the same indentation is interpreted to belong to the same code block. If there is a next statement with less indentation to the left, then it just means the end of the previous code block.

1. **Give syntax of if ..elif statement in Python**

**The syntax for if…elif…else statement is,**

if Boolean\_Expression\_1:

statement\_1

elif Boolean\_Expression\_2:

statement\_2

elif Boolean\_Expression\_3:

statement\_3

:

:

:

else:

statement\_last

1. **What is the purpose of else suit in Python loops? Give example**

Python allows the else keyword to be used with the for and while loops too. The else block appears after the body of the loop. The statements in the else block will be executed after all iterations are completed. The program exits the loop only after the else block is executed.

1. **What are the rules for naming identifiers?**

* Identifiers can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (\_). Names like myCountry, other\_1 and good\_morning, all are valid examples. A Python identifier can begin with an alphabet (A – Z and a – z and \_).
* An identifier cannot start with a digit but is allowed everywhere else. 1plus is invalid, but plus1 is perfectly fine.
* Keywords cannot be used as identifiers.
* One cannot use spaces and special symbols like !, @, #, $, % etc. as identifiers.
* Identifier can be of any length.

1. **What is an identifier? Give example**

An identifier is a name given to a variable, function, class or module.

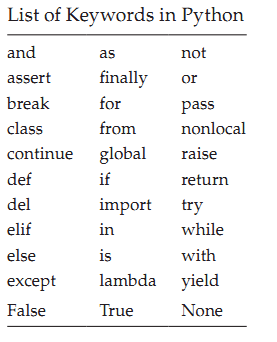
Example:

language = 'Python'

Here, language is a variable (an identifier) which holds the value 'Python'.

1. **What are python keywords? Give example**

Keywords are a list of reserved words that have predefined meaning. Keywords are special vocabulary and cannot be used by programmers as identifiers for variables, functions, constants or with any identifier name. **EXAMPLE**



1. **What are python Variables?**

Variable is a named placeholder to hold any type of data which the program can use to assign and modify during the course of execution.

**Syntax**: variable\_name = expression

**Example**:

language = 'Python'

Here, language is a variable (an identifier) which holds the value 'Python'.

1. **What are the rules for naming variables?**

* Variable names can consist of any number of letters, underscores and digits.
* Variable should not start with a number.
* Python Keywords are not allowed as variable names.
* Variable names are case-sensitive. For example, computer and Computer are different variables.

1. **Give syntax and example for assigning values to variables**

**Syntax**: variable\_name = expression

**Example:**

language = 'Python'

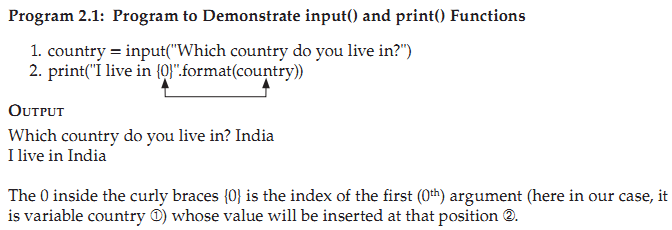
Here, language is a variable (an identifier) which holds the value 'Python'.

1. **Give the syntax and example for str.format() method**

Use str.format() method if you need to insert the value of a variable, expression or an object into another string and display it to the user as a single string. The format() method returns a new string with inserted values.

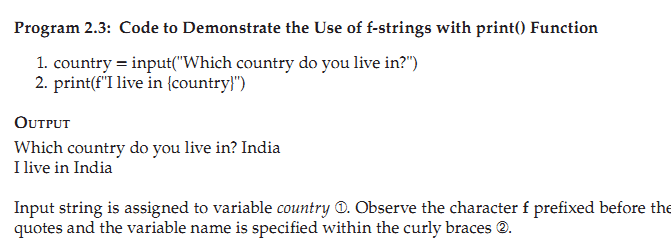
Syntax: **str.format(p0, p1, ..., k0= v0, k1= v1, ...)**

where p0, p1,... are called as positional arguments and, k0, k1,... are keyword arguments with their assigned values of v0, v1,... respectively.



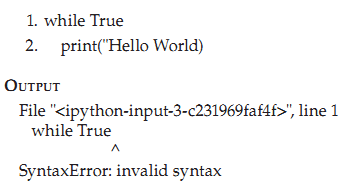
1. **What is f-string literal? Give an example**

A f-string is a string literal that is prefixed with “f”. These strings may contain replacement fields, which are expressions enclosed within curly braces {}. The expressions are replaced with their values. An f at the beginning of the string tells Python to allow any currently valid variable names within the string.



1. **What are syntax errors? Give example**

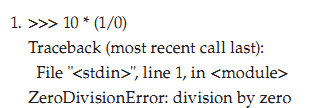
 A syntax error in computer science is **an error in the syntax of a coding or programming language, entered by a programmer**. Syntax errors are caught by a software program called a compiler, and the programmer must fix them before the program is compiled and then run.

****

#The error is caused by a missing colon (':')

1. **What are Exceptions? Give Example**

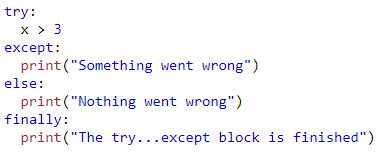
An exception is an unwanted event that interrupts the normal flow of the program. When an exception occurs in the program, execution gets terminated. **When the exceptions are not handled by programs it results in error messages as shown below.**

****

Exceptions come in different types, and the type is printed as part of the message: the types in the example are ZeroDivisionError ➀

1. **What is use of finally statement?**

A finally block is always executed before leaving the try statement, whether an exception has occurred or not.

****

**OUTPUT:**

**Something went wrong**

**The try...except block is finished**

1. **Differentiate scope and life time of a variable.**

The scope of variable is a region of the program where a variable is visible or accessible. Lifetime of a variable is the duration for which a variable exists in the memory. The existence and accessibility depend on the declaration of a variable in the program.

1. **List any four built in functions in Python.**

* len ()
* pow ()
* max ()
* min ()

1. **Give the Syntax of user defined function.**

The syntax for function definition

**def function\_name (parameter\_1, parameter\_2, …, parameter\_n):**

**statement(s)**

* A function definition consists of the def keyword, followed byThe name of the function. The function’s name has to adhere to the same naming rules as variables: use letters, numbers, or an underscore, but the name cannot start with a number. Also, you cannot use a keyword as a function name.
* A list of parameters to the function are enclosed in parentheses and separated by commas. Some functions do not have any parameters at all while others may have one or more parameters.
* A colon is required at the end of the function header. The first line of the function definition which includes the name of the function is called the function header.
* Block of statements that define the body of the function start at the next line of the function header and they must have the same indentation level.

1. **Give the syntax and example for range() function.**

**The syntax for range () function is,**

range ([start,] stop [, step])

Both start and step arguments are optional and the range argument value should always be an integer.

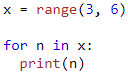
**start →** value indicates the beginning of the sequence. If the start argument is not specified, then the sequence of numbers start from zero by default.

**stop →** Generates numbers up to this value but not including the number itself.

**step →** indicates the difference between every two consecutive numbers in the sequence. The step value can be both negative and positive but not zero.

NOTE: The square brackets in the syntax indicate that these arguments are optional. You can leave them out.

**Example**

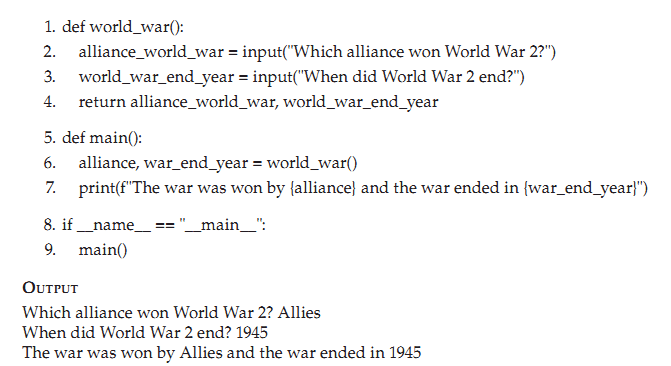
 **#O/P: 3 4 5**

1. **What is the meaning of \_\_name\_\_ == \_\_main\_\_?**

**\_\_name\_\_ == \_\_main\_\_ is a entry point of the program.** When a Python interpreter reads a Python file, it first sets a few special variables. Then it executes the code from the file. One of those variables is called \_\_name\_\_.When a Python module or package is imported, \_\_name\_\_ is set to the module’s name. Usually, this is the name of the Python file itself without the .py extension.

1. **Give example of function returning multiple values.**

Program to Demonstrate the Return of Multiple Values from a Function Definition

****

1. **What is keyword argument? Give example**

Keyword arguments are a list of arguments of type keyword = value, that can be accessed with the name of the argument inside curly braces like {keyword}.

def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):

print(f"This parrot wouldn't {action}, if you put {voltage}, volts through it.")

parrot(voltage=1000)

OUTPUT:

This parrot wouldn't voom, if you put 1000, volts through it.

1. **What is default argument? Give example**

 A default parameter is defined with a fallback value as a default argument. Such parameters are optional during a function call. If no argument is provided, the default value is used, and if an argument is provided, it will overwrite the default value.

**Example:**

def add\_numbers( a = 7, b = 8):

sum = a + b

print('Sum:', sum)

# function call with two arguments

add\_numbers(2, 3)

# function call with one argument

add\_numbers(a = 2)

# function call with no arguments

add\_numbers()

**OUTPUT:**

**Sum: 5**

**Sum: 10**

**Sum: 15**

Here, we have provided default values **7** and **8** for parameters a and b respectively

**33. Differentiate \*args and \*\*kwargs?**

|  |  |
| --- | --- |
| **\*args** | **\*\*kwargs** |
| \*args as parameter in function defi-nition allows you to pass a non-keyworded, variable length tuple argument list to the call-ing function. | \*\*kwargs as parameter in function definition allows you to pass keyworded,  variable length dictionary argument list to the calling function. |
| \*args must come after all  the positional parameters | \*\*kwargs must come right at the end |

**Long Answer Questions**

1. **Explain any five features of Python. (Any 5)**

* **Simple**: Python is a simple programming language. When we read a Python program, we feel like reading English sentences. It means more clarity and less stress on understanding the syntax of the language. Hence, developing and understanding programs will become easy.
* **Easy to learn**: Python uses very few keywords. Its programs use very simplestructure. So, developing programs in Python become easy. Also, Python resembles C language. Most of the language constructs in C are also available in Python. Hence,migrating from C to Python is easy for programmers.
* **Open source**: There is no need to pay for Python software. Python can be freelydownloaded from www.python.org website. Its source code can be read, modified and can be used in programs as desired by the programmers.
* **High level language**: Programming languages are of two types: low level and highlevel. A low level language uses machine code instructions to develop programs. These instructions directly interact with the CPU. Machine language and assembly language are called low level languages. High level languages use English words to develop programs. These are easy to learn and use. Like COBOL, PHP or Java, Python also uses English words in its programs and hence it is called high level programming language.
* **Platform independent**: When a Python program is compiled using a Python compiler, it generates byte code. Python's byte code represents a fixed set of instructions that run on all operating systems and hardware. Using a Python Virtual Machine (PVM), anybody can run these byte code instructions on any computer system. Hence, Python programs are not dependent on any specific operating system.
* **Open source**: There is no need to pay for Python software. It can be freely downloaded from www.python.org website. The source code can be read, modified and used as desired by the programmers.
* **Database connectivity**: Python provides interfaces to connect its programs to all major databases like Oracle, MySql, Sybase.

1. **Explain any five flavors of Python .**

Flavors of Python refer to different types of Python compilers. These flavors useful to integrate various programming languages into Python.

* **CPython**: This is the standard Python compiler implemented in C language. In this, any Python program is internally converted into byte code using C language functions. This byte code is run on the interpreter available in Python Virtual Machine (PVM) created in C language. The advantage is that it is possible to execute C and C++ functions and programs in CPython.
* **Jython:** This is earlier known as JPython. This is the implementation of Python programming language which is designed to run on Java platform. Jython compiler first compiles the Python program into Java byte code. This byte code is executed by Java Virtual Machine (JVM) to produce the output. Jython contains libraries which are useful for both Python and Java programmers.
* **RubyPython**: This is a bridge between the Ruby and Python interpreters. It encloses a Python interpreter inside Ruby applications.
* **Pythonxy:** This is pronounced as Python xy and written as Python(X,Y). This is the Python implementation that can be obtained after adding scientific and engineering related packages.
* **AnacondaPython:** When Python is redeveloped for handling large-scale data processing, predictive analytics and scientific computing, it is called Anaconda Python. This implementation mainly focuses on large scale of data.

1. **Explain various data types in Python.**

Data types specify the type of data like numbers and characters to be stored and manipulated within a program.Basic data types of Python are

• Numbers

• Boolean

• Strings

• None

* **Numeric Types**: In Python, numeric data type represents the data that has a numeric value. The numeric value can be an integer, floating number, or even complex number. These values are defined as int, float, and complex classes in Python.
* **int datatype** – This data type is represented with the help of int class. It consists of positive or negative whole numbers (without fraction or decimal). In Python, there is no limit for the size of an int datatype. It can store very large integer numbers conveniently. Example: a=-57
* **float datatype**: The float datatype represents floating point numbers. A floating point number is a number that contains a decimal point. Example nm=56.994456. Floating point numbers can also be written in scientific notations using ‘e’ or ‘E’ as x=22.55e3. The meaning is x=22.55× 103
* **complex datatype**: A complex number is a number that is written in the form a + bj or a + bJ. Here a represents the real part and b represents the imaginary part. The suffix j or J associated with b indicates that the square root value of -1. Example 3 + 3j, 5.5 + 8.0J.

**Example Program -1:**

#Python Program to add two complex numbers

c1= 2+ 3J

c2=3-2J

c3=c1+c2

print("Sum = ", c3) # Output: Sum = (5+1j)

* Boolean

Booleans may not seem very useful at first, but they are essential when you start using conditional statements. In fact, since a condition is really just a yes-or-no question, the answer to that question is a Boolean value, either True or False. The Boolean values, True and False are treated as reserved words.

* **strings**

A string consists of a sequence of one or more characters, which can include letters, numbers, and other types of characters. A string can also contain spaces. You can use single quotes or double quotes to represent strings and it is also called a string literal. Multiline strings can be denoted using triple quotes, ''' or """. These are fixed values, not variables that you literally provide in your script.

For example, 1. >>> s = 'This is single quote string'

1. >>> s = "This is double quote string"

3. >>> s = '''This is Multiline string'''

4. >>> s = "a"

In ➀ a string is defined using single quotes, in ➁ a string is defined using double quotes and a multiline string is defined in ➂, a single character is also treated as string ➃.

* **None Type:**  None is another special data type in Python. None is frequently used to represent the absence of a value.

For example,

1. >>> money = None

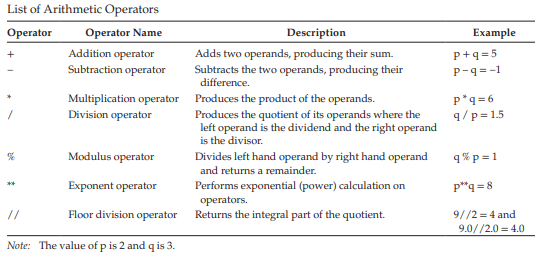
None value is assigned to variable money ➀.

**4. Explain the Arithmetic operators, logical operators and relational operators**

**with an example.**

**Arithmetic operators**

Arithmetic operators are used to execute arithmetic operations such as addition, subtraction, division, multiplication etc.



For example, 1. >>> 10+35 #o/p45

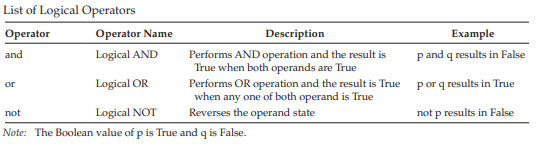
2. >>> −10+35 #O/P 25

3. >>> 4\*2 # O/P 8

4. >>> 4\*\*2 #O/P 16

**logical operators**

The logical operators are used for comparing or negating the logical values of their operands and to return the resulting logical value. The values of the operands on which the logical operators operate evaluate to either True or False. The result of the logical operator is always a Boolean value, True or False



EXAMPLE 1. >>> True and False #O/P False

2. >>> True or False #O/P True

3. >>> not(True) and False #O/P False

4. >>> not(True and False) #O/P True

**comparison operators(Relational operator)**

When the values of two operands are to be compared then comparison operators are used. The output of these comparison operators is always a Boolean value, either True or False.

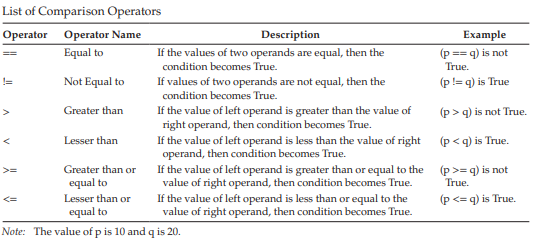
For example, 1. >>>10 == 12 #O/P False

2. >>>10 != 12 #O/P True

3. >>> "P" < "Q" #O/P True

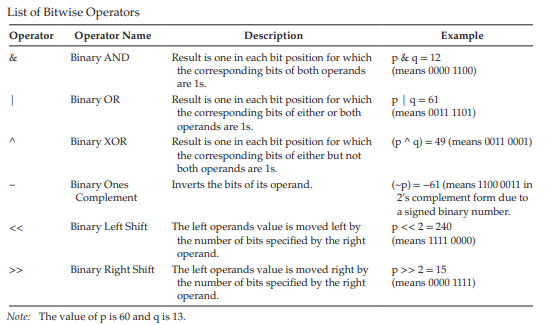
4. >>> "Aston" > "Asher" #O/P True

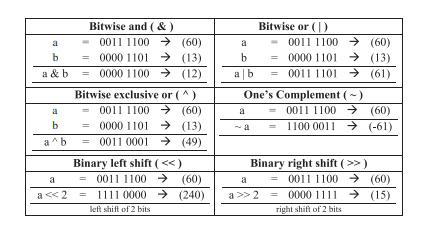
5. >>> True == True #O/P True



1. **Explain the bitwise operators with examples.**

**Bitwise operators**

Bitwise operators treat their operands as a sequence of bits (zeroes and ones) and perform bit by bit operation. Bitwise operators perform their operations on (1010) binary representations, but they return standard Python numerical values.



1. **How to read different types of input from the keyboard. Give examples**

Python provides input() function to accept input from the user. This function takes a value from the keyboard and returns it as a string. It is always better to display a message to the user so that the user can understand what to enter.

**Example**

str=input("Enter your name ")

print("Hello", str)

If the inputs need to be converted to a datatype other than string then the corresponding conversion function is to be used. For example int() for integers float() for floating point datatype and so on. Python accepts a character as string. If only one character is needed then indexing is to be used.

**Example:**

ch=input('Enter a character ')

print('You entered', ch[0])

Accepting Integers

**Output:**

Enter a character Asdf

You entered A

**Method-1**

**#Accepting integers**

x=int(input('Enter first number '))

y=int(input('Enter second number ‘))

y=int(input('Enter second number '))

print('X=', x, ‘Y=’, y, ‘Z=’,z)

**Method-2**

**#Accepting three numbers**

a, b, c=[int(x) for x in input("Enter three numbers ").split()]

print(a, b, c)

To accept more than one input in the same line a for loop can be used along with the input statement as mentioned in the above statement. The split() method by default splits the values where a space is found. Hence, when entering numbers, the user should separate them using a blank space. **The square brackets [] indicate that the input accepted will be in the form of elements of a list.**

lst=[x for x in input('Enter strings : ').split(',')] #accepts strings separated by comma

print('Your List', lst)

a,b,c=[int(x) for x in input('Enter three numbers ').split(',')]

sum=a+b+c

print('Sum=',sum)

**The eval() function**

The eval() function takes a string and evaluates the result of the string by taking it as a Python expression. Example “a+b-5” where a=10, b=15 will be evaluated as 20. If a string expression is passed to eval() the result of the expression passed will be returned

**Example**

#Use of eval() function

a, b= 5, 10

result=eval("a+b-5")

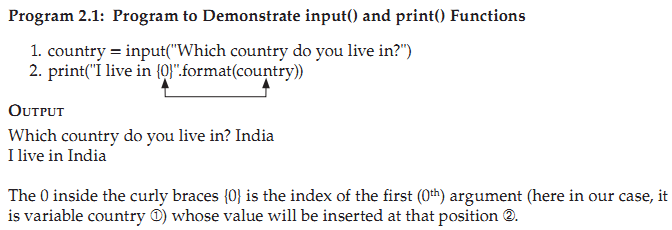
print("Result=", result)# will display 10 as Result

1. **Explain use of string.format and f-string with print() function.**

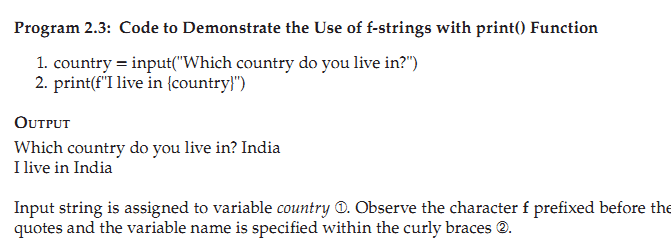
Use **str.format()** method if you need to insert the value of a variable, expression or an object into another string and display it to the user as a single string. The format() method returns a new string with inserted values.

**Syntax**: **str.format(p0, p1, ..., k0= v0, k1= v1, ...)**

where p0, p1,... are called as positional arguments and, k0, k1,... are keyword arguments with their assigned values of v0, v1,... respectively.



**A f-string** is a string literal that is prefixed with “f”. These strings may contain replacement fields, which are expressions enclosed within curly braces {}. The expressions are replaced with their values. An f at the beginning of the string tells Python to allow any currently valid variable names within the string.



1. **Explain any five type conversion functions with example.**

Explicitly convert, a variable from one type to another.

**int() function :**

To explicitly convert a float number or a string to an integer, cast the number using int() function.

1. >>>numerical\_value = input("Enter a number")

Enter a number 9

2. >>> numerical\_value '9'

3. >>> numerical\_value = int(input("Enter a number"))

Enter a number 9

4. >>> numerical\_value 9

* ➀–➁ User enters a value of 9 which gets assigned to variable numerical\_value and the value is treated as string type.
* In order to assign an integer value to the variable, you have to enclose the input() function within the int() function which converts the input string type to integer type ➂–➃.
* A string to integer conversion is possible only when the string value is inherently a number (like “1”) and not a character

**float() function :**

The float() function returns a floating point number constructed from a number or string.

1. int\_to\_float = float(4)

2. string\_to\_float = float("1") #number treated as string

3. print(f"After Integer to Float Casting the result is {int\_to\_float}")

4. print(f"After String to Float Casting the result is {string\_to\_float}")

**Output**

After Integer to Float Casting the result is 4.0

After String to Float Casting the result is 1.0

Convert integer and string values to float ➀–➁ and display the result ➂–➃

**str() function**

The str() function returns a string which is fairly human readable.

1. int\_to\_string = str(8)

2. float\_to\_string = str(3.5)

3. print(f"After Integer to String Casting the result is {int\_to\_string}")

4. print(f"After Float to String Casting the result is {float\_to\_string}")

**Output**

After Integer to String Casting the result is 8

After Float to String Casting the result is 3.5

Here, integer and float values are converted ➀–➁ to string using str() function and results are displayed ➂–➃.

**chr() function**

Convert an integer to a string of one character whose ASCII code is same as the integer using chr() function. The integer value should be in the range of 0–255.

1. ascii\_to\_char = chr(100)

2. print(f'Equivalent Character for ASCII value of 100 is {ascii\_to\_char}')

**Output**

Equivalent Character for ASCII value of 100 is d

An integer value corresponding to an ASCII code is converted ➀ to the character and printed ➁.

**complex() function**

Use complex() function to print a complex number with the value real + imag\*j or convert a string or number to a complex number.

1. complex\_with\_string = complex("1")

2. complex\_with\_number = complex(5, 8)

3. print(f"Result after using string in real part {complex\_with\_string}")

4. print(f"Result after using numbers in real and imaginary part {complex\_with\_ number}")

**Output**

Result after using string in real part (1+0j)

Result after using numbers in real and imaginary part (5+8j)

* The first argument is a string ➀. Hence you are not allowed to specify the second argument.
* In ➁ the first argument is an integer type, so you can specify the second argument which is also an integer. Results are printed out in ➂ and ➃.

1. **Explain while and for loops with syntax and example.**

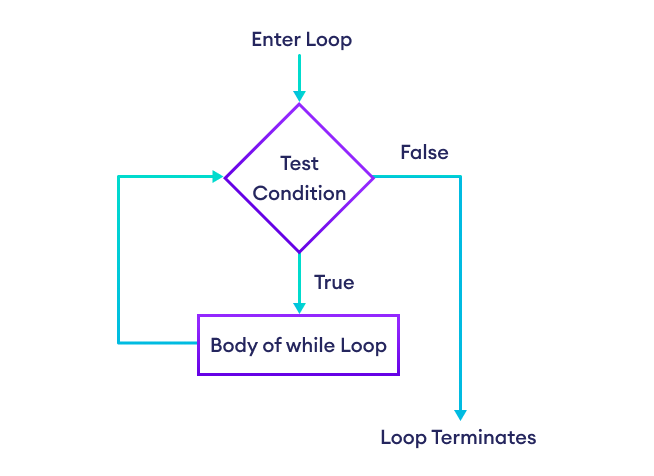
**while loop:** The while loop starts with the while keyword and ends with a colon. With a while statement, the first thing that happens is that the Boolean expression is evaluated before the statements in the while loop block is executed.

while Boolean\_Expression:

statement(s)

If the Boolean expression evaluates to False, then the statements in the while loop block are never executed. If the Boolean expression evaluates to True, then the while loop block is executed. After each iteration of the loop block, the Boolean expression is again checked, and if it is True, the loop is iterated again.

Each repetition of the loop block is called an iteration of the loop. This process continues until the Boolean expression evaluates to False and at this point the while statement exits. Execution then continues with the first statement after the while loop.



Flowchart of while Loop

Write Python Program to Display First 10 Numbers Using while Loop Starting from 0

1. i = 0

2. while i < 10:

3. print(f"Current value of i is {i}")

4. i = i + 1

**Output**

Current value of i is 0

Current value of i is 1

Current value of i is 2

Current value of i is 3

Current value of i is 4

Current value of i is 5

Current value of i is 6

Current value of i is 7

Current value of i is 8

Current value of i is 9

Variable i is assigned with 0 outside the loop ➀. The expression i < 10 is evaluated ➁. If the value of i is less than 10 (i.e., True) then the body of the loop is executed. Value of i is printed ➂ and i is incremented by 1 ➃. This continues until the expression in while loop becomes False.

**for loop:** The for loop starts with for keyword and ends with a colon.

The syntax for the for loop is,

for iteration\_variable in sequence:

statement(s)

The first item in the sequence gets assigned to the iteration variable iteration\_variable. Here, iteration\_variable can be any valid variable name. Then the statement block is executed. This process of assigning items from the sequence to the iteration\_variable and then executing the statement continues until all the items in the sequence are completed.

We take the liberty of introducing you to range() function which is a built-in function at this stage as it is useful in demonstrating for loop. The range() function generates a sequence of numbers which can be iterated through using for loop. The syntax for range() function is,

**range([start ,] stop [, step])**

Both start and step arguments are optional and the range argument value should always be an integer.

**start → value indicates the beginning of the sequence. If the start argument is not specified, then the sequence of numbers start from zero by default.**

**stop → Generates numbers up to this value but not including the number itself.**

**step → indicates the difference between every two consecutive numbers in the sequence. The step value can be both negative and positive but not zero.**

**NOTE: The square brackets in the syntax indicate that these arguments are optional. You can leave them out.**

Program 3.16: Write a Program to Find the Sum of All Odd and Even Numbers up to a Number Specified by the User.

1. number = int(input("Enter a number"))

2. even = 0

3. odd = 0

4. for i in range(number):

5. if i % 2 == 0:

6. even = even + i

7. else: 8. odd = odd + i

9. print(f"Sum of Even numbers are {even} and Odd numbers are {odd}")

**Output**

**Enter a number 10**

**Sum of Even numbers are 20 and Odd numbers are 25**

A range of numbers are generated using range() function ➃. The numbers are segregated as odd or even by using the modulus operator ➄. All the even numbers are added up and assigned to even variable and odd numbers are added up and assigned to odd variable ➅–➇ and print the result ➈.

1. **Explain Exception handling in Python with try…except… finally block**

**Try..except...finally:**

It is possible to write programs to handle exceptions by using try…except…finally statements.An exception is an unexpected event that occurs during program execution. For example,

**divide\_by\_zero = 7 / 0**

The above code causes an exception as it is not possible to divide a number by 0.

**The syntax for try…except…finally is,**

try: statement\_1

except Exception\_Name\_1:

statement\_2

except Exception\_Name\_2:

statement\_3 . . .

else: statement\_4

finally: statement\_5

* A try block consisting of one or more statements is used by Python programmers to partition code that might be affected by an exception.
* The associated except blocks are used to handle any resulting exceptions thrown in the try block. That is, you want the try block to succeed, and if it does not succeed, you want the control to pass to the catch block.
* If any statement within the try block throws an exception, control immediately shifts to the catch block. If no exception is thrown in the try block, the catch block is skipped.
* There can be one or more except blocks. Multiple except blocks with different exception names can be chained together.
* The except blocks are evaluated from top to bottom in your code, but only one except block is executed for each exception that is thrown.
* The first except block that specifies the exact exception name of the thrown exception is executed.
* If no except block specifies a matching exception name then an except block that does not have an exception name is selected, if one is present in the code.
* Instead of having multiple except blocks with multiple exception names for different exceptions, you can combine multiple exception names together separated by a comma (also called parenthesized tuples) in a single except block.

The syntax for combining multiple exception names in an except block is,

**except (Exception\_Name\_1, Exception\_Name\_2, Exception\_Name\_3):**

**statement(s)**

where Exception\_Name\_1, Exception\_Name\_2  and Exception\_Name\_3  are different exception names.

* **The try…except statement has an optional else block, which, when present, must follow all except blocks.**
* It is useful for code that must be executed if the try block does not raise an exception. The use of the else block is better than adding additional code to the try block because it avoids accidentally catching an exception that wasn’t raised by the code being protected by the try…except statement.
* The try statement has another optional block which is intended to define clean-up actions that must be executed under all circumstances.
* **A finally block is always executed before leaving the try statement, whether an exception has occurred or not. When an exception has occurred in the try block and has not been handled by an except block, it is re-raised after the finally block has been executed.**
* **The finally clause is also executed “on the way out” when any other clause of the try statement is left via a break, continue or return statement**.
* You can also leave out the name of the exception after the except keyword. This is generally not recommended as the code will now be catching different types of exceptions and handling them in the same way.
* This is not optimal as you will be handling a TypeError exception the same way as you would have handled a ZeroDivisionError exception.
* When handling exceptions, it is better to be as specific as possible and only catch what you can handle.

**Example**

try:

numerator = 10

denominator = 0

result = numerator/denominator

print(result)

except:

print("Error: Denominator cannot be 0.")

finally:

print("This is finally block.")

**OUTPUT:**

**Error: Denominator cannot be 0.**

**This is finally block.**

**11. Give the syntax of range function. Explain use range function in for loop**

**with examples.**

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

Syntax: **range(start, stop, step)**

**Start🡪 Optional. An integer number specifying at which position to start. Default is 0**

**Stop🡪 Required. An integer number specifying at which position to stop (not included).**

**Step🡪Optional. An integer number specifying the incrementation. Default is 1.**

Example: Write a Program to Find the Sum of All Odd and Even Numbers up to a Number Specified by the User.

1. number = int(input("Enter a number"))

2. even = 0

3. odd = 0

4. for i in range(number):

5. if i % 2 == 0:

6. even = even + i

7. else: 8. odd = odd + i

9. print(f"Sum of Even numbers are {even} and Odd numbers are {odd}")

**Output**

**Enter a number 10**

**Sum of Even numbers are 20 and Odd numbers are 25**

A range of numbers are generated using range() function ➃. The numbers are segregated as odd or even by using the modulus operator ➄. All the even numbers are added up and assigned to even variable and odd numbers are added up and assigned to odd variable ➅–➇ and print the result ➈.

**12. With syntax and example explain how to define and call a function in Python**

User-defined functions are reusable code blocks created by users to perform some specific task in the program

**The syntax** for function definition is

**def function\_name(parameter\_1, parameter\_2, …, parameter\_n):**

**statement(s)**

In Python, a function definition consists of the def keyword, followed by

1. The name of the function. The function’s name has to adhere to the same naming rules as variables: use letters, numbers, or an underscore, but the name cannot start with a number. Also, you cannot use a keyword as a function name.

2. A list of parameters to the function are enclosed in parentheses and separated by commas. Some functions do not have any parameters at all while others may have one or more parameters.

3. A colon is required at the end of the function header. The first line of the function definition which includes the name of the function is called the function header.

4. Block of statements that define the body of the function start at the next line of the function header and they must have the same indentation level.

Defining a function does not execute it. Defining a function simply names the function and specifies what to do when the function is called. Calling the function actually performs the specified actions with the indicated parameters.

**The syntax for function call or calling function is,**

**function\_name(argument\_1, argument\_2,…,argument\_n)**

Arguments are the actual value that is passed into the calling function. There must be a one to one correspondence between the formal parameters in the function definition and the actual arguments of the calling function.

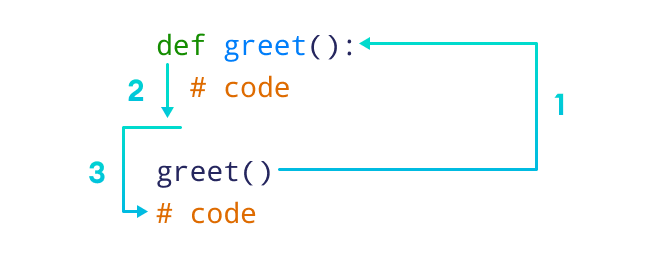
When a function is called, the formal parameters are temporarily “bound” to the arguments and their initial values are assigned through the calling function.

A function should be defined before it is called and the block of statements in the function definition are executed only after calling the function.

if \_\_name\_\_ == "\_\_main\_\_":

statement(s)

The special variable, \_\_name\_\_ with "\_\_main\_\_", is the entry point to your program. When Python interpreter reads the if statement and sees that \_\_name\_\_ does equal to "\_\_main\_\_", it will execute the block of statements present there.



Here,

1. When the function is called, the control of the program goes to the function definition.
2. All codes inside the function are executed.
3. The control of the program jumps to the next statement after the function call.

**Example:**

Program to Find the Area of Trapezium Using the Formula Area = (1/2) \* (a + b) \* h Where a and b Are the 2 Bases of Trapezium and h Is the Height

1. def area\_trapezium(a, b, h):

2. area = 0.5 \* (a + b) \* h

3. print(f"Area of a Trapezium is {area}")

4. def main():

5. area\_trapezium(10, 15, 20)

6. if \_\_name\_\_ == "\_\_main\_\_":

7. main()

**Output**

**Area of a Trapezium is 250.0**

Here the function definition area\_trapezium(a, b, h) uses three formal parameters a, b, h➀ to stand for the actual values passed by the user in the calling function area\_trapezium(10, 15, 20) ➄. The arguments in the calling function are numbers. The variables a, b and h are assigned with values of 10, 15 and 20 respectively. The area of a trapezium is calculated using the formula 0.5 \* (a + b) \* h and the result is assigned to the variable area ➁ and the same is displayed ➂.

**13. Explain \*args and \*\*kwargs with example**

**\*args**

\*args as parameter in function defi-nition allows you to pass a non-keyworded, variable length tuple argument list to the call-ing function. \*args must come after all the positional parameters.

**Example: 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)

**OUTPUT:**

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.

**\*\*kwargs**

\*\*kwargs as parameter in function definition allows you to pass keyworded, variable length dictionary argument list to the calling function. \*\*kwargs must come right at the end

**Example: 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="John", Lastname="Wood", Email="johnwood@nomail.com", Country="Wakanda", Age=25, Phone=9876543210)

**OUTPUT**

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 John

Lastname is Wood

Email is johnwood@nomail.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.

**The advantages of \*args and \*\*kwargs with examples.**

* The syntax \*args allows to pass a variable number of arguments to the calling function
* The syntax \*\*kwargs allows you to pass keyworded, variable length dictionary  
  arguments to the calling function
* \*args and \*\*kwargs are mostly used as parameters in function definitions. \*args and  
  \*\*kwargs allows you to pass a variable number of arguments to the calling function.
* Here variable number of arguments means that the user does not know in advance about how many arguments will be passed to the calling function.
* \*args as parameter in function definition allows you to pass a non-keyworded, variable length tuple argument list to the calling function.
* \*\*kwargs as parameter in function efinition allows you to pass keyworded, variable length dictionary argument list to the calling function.
* \*args must come after all the positional parameters and \*\*kwargs must come right at the end.

**14. With example explain keyword arguments and default arguments to the function.**

**Keyword arguments** are a list of arguments of type keyword = value, that can be accessed with the name of the argument inside curly braces like {keyword}.

def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):

print(f"This parrot wouldn't {action}, if you put {voltage}, volts through it.")

parrot(voltage=1000)

OUTPUT:

This parrot wouldn't voom, if you put 1000, volts through it.

A **default parameter** is defined with a fallback value as a default argument. Such parameters are optional during a function call. If no argument is provided, the default value is used, and if an argument is provided, it will overwrite the default value.

**Example:**

def add\_numbers( a = 7, b = 8):

sum = a + b

print('Sum:', sum)

# function call with two arguments

add\_numbers(2, 3)

# function call with one argument

add\_numbers(a = 2)

# function call with no arguments

add\_numbers()

**OUTPUT:**

**Sum: 5**

**Sum: 10**

**Sum: 15**

Here, we have provided default values **7** and **8** for parameters a and b respectively

**15. With example explain how command line arguments are passed to python program.**

A Python program can accept any number of arguments from the command line.

Command line arguments is a methodology in which user will give inputs to the program

through the console using commands. You need to import sys module to access command

line arguments. All the command line arguments in Python can be printed as a list of

string by executing **sys.argv**.

**Example Program : Program to Demonstrate Command Line Arguments in Python**

1. import sys

2. def main():

3. print(f"sys.argv prints all the arguments at the command line including file name {sys.argv}")

4. print(f"len(sys.argv) prints the total number of command line arguments including file name {len(sys.argv)}")

5. print("You can use for loop to traverse through sys.argv")

6. for arg in sys.argv:

7. print(arg)

8. if \_\_name\_\_ == "\_\_main\_\_":

9. main()

**OUTPUT**

**C:\Introduction\_To\_Python\_Programming\Chapter\_4>python Program\_4.12.py arg\_1**

**arg\_2 arg\_3**

**sys.argv prints all the arguments at the command line including file name ['Program\_4.12.**

**p y ', 'a r g \_ 1', 'a r g \_ 2 ', 'a r g \_ 3 ' ]**

**len(sys.argv) prints the total number of command line arguments including file name 4**

**You can use for loop to traverse through sys.argv**

**Program\_4.12.py**

**arg\_1**

**arg\_2**

**arg\_3**

To execute a command line argument program, you need to navigate to the directory where your program is saved. Then issue a command in the format python file\_name argument\_1 argu-ment\_2argument\_3……argument\_n. Here argument\_1  argument\_2  and so on can be any argument and should be separated by a space. Import sys module ➀ to access command line arguments. Print all the arguments including file name using sys.argv ➂ but excluding the python command. The number of arguments passed at the command line can be obtained by len(sys.argv) ➃. A for loop can be used to traverse through each of the arguments in sys.argv ➄–➆.

**16. Write a program to check for ‘ValueError’ exception**

1. while True:

2. try:

3. number = int(input("Please enter a number: "))

4. print(f"The number you have entered is {number}")

5. break

6. except ValueError:

7. print("Oops! That was no valid number. Try again…")

**Output**

Please enter a number: g

Oops! That was no valid number. Try again…

Please enter a number: 4

The number you have entered is 4

First, the try block (the statement(s) between the try and except keywords) is executed ➁ –➄ inside the while loop ➀. If no exception occurs, the except block is skipped and execution of the try statement is finished. If an exception occurs during execution of the try block statements, the rest of the statements in the try block is skipped. Then if its type matches the exception named after the except keyword, the except block is executed ➅ –➆, and then execution continues after the try statement. When a variable receives an inappropriate value then it leads to ValueError exception.

**17. Write a program to check for ZeroDivisionError Exception**

1. x = int(input("Enter value for x: "))

2. y = int(input("Enter value for y: "))

3. try:

4. result = x / y

5. except ZeroDivisionError:

6. print("Division by zero!")

7. else:

8. print(f"Result is {result}")

9. fi na l ly:

10. print("Executing finally clause")

**Output**

**Case 1**

Enter value for x: 8

Enter value for y: 0

Division by zero!

Executing finally clause

**Case 2**

Enter value for x: p

Enter value for y: q

Executing finally clause

ValueError Traceback (most recent call last)

<ipython-input-16-271d1f4e02e8> in <module>()

ValueError: invalid literal for int() with base 10: 'p'

In the above example divide by zero exception is handled. In line ➀ and ➁, the user entered val-ues are assigned to x and y variables. Line ➃ is enclosed within line ➂ which is a try clause. If the statements enclosed within the try clause raise an exception then the control is transferred to line ➄ and divide by zero error is printed ➅. As can be seen in Case 1, ZeroDivisionError occurs when the second argument of a division or modulo operation is zero. If no exception occurs, the except block is skipped and result is printed out ➆–➇. In Case 2, as you can see, the finally clause is executed in any event ➈– ➉. The ValueError raised by dividing two strings is not handled by the except clause and therefore re-raised after the finally clause has been executed.

**18. How to return multiple values from a function definition? Explain with an example**

**return statement**

The function to perform its specified task to calculate a valueand return the value to the calling function so that it can be stored in a variable and usedlater. This can be achieved using the optional return statement in the function definition.  
**The syntax for return statement is,**

**return [expression\_list]**

The return statement terminates the execution of the function definition in which it appearsand returns control to the calling function. It can also return an optional value to its callingfunction. it makes sense to return multiple values if they are related to each other. Ifso, return the multiple values separated by a comma which by default is constructed asa tuple by Python. When calling function receives a tuple from the function definition, it is common to assign the result to multiple variables by specifying the same number of variables on the left-hand side of the assignment as there were returned from the function definition. This is called tuple unpacking.

**EXAMPLE: Program to Demonstrate the Return of Multiple Values from a Function Definition**

1. def world\_war():

2. alliance\_world\_war = input("Which alliance won World War 2?")

3. world\_war\_end\_year = input("When did World War 2 end?")

4. return alliance\_world\_war, world\_war\_end\_year

5. def main():

6. alliance, war\_end\_year = world\_war()

7. print(f"The war was won by {alliance} and the war ended in {war\_end\_year}")

8. if \_\_name\_\_ == "\_\_main\_\_":

9. ma i n()

**Output**

Which alliance won World War 2? Allies

When did World War 2 end? 1945

The war was won by Allies and the war ended in 1945

Line ➇ is the entry point of the program and you call the function main() ➈. In the func-tion definition main() you call another function world\_war() ➅. The block of statements in function definition world\_war() ➀ includes statements to get input from the user and a return statement ➁ –➃. The return statement returns multiple values separated by a comma. Once the execution of function definition is completed, the values from function definition world\_war() are returned to the calling function. At the calling function on the left-hand side of the assignment ➅ there should be matching number of variables to store the values returned from the return statement. At ➆ returned values are displayed.

**UNIT II**

**Two Mark questions**

1. **Give two methods of creating strings in Python.**

Strings are another basic data type available in Python. They consist of one or more char-acters surrounded by matching quotation marks. To create a string, put the sequence of characters inside either single quotes, double quotes, or triple quotes and then assign it to a variable.

Example:

>>> single\_quote\_character = 'a'

>>> print(single\_quote\_character)

a

>>> double\_quote\_character ="b"

>>>print(double\_quote\_character)

b

>>> triple\_quote\_example ="""this is a sentence written in triple quotes"""

>>>print(type(triple\_quote\_example))

<class'str'>

1. **List any two built in functions used with python strings. Mention their use.**
2. **islower()**
3. **isupper()**
4. **islower()**

**string\_name.islower()**

The method islower() returns Boolean True if all characters in the string are lowercase, else it returns Boolean False.

1. **isupper()**

**string\_name.isupper()**

The method isupper() returns Boolean True if all cased characters in the string are uppercase and there is at least one cased character, else it returns Boolean False.

1. **Why strings are called immutable?**

As strings are immutable, it cannot be modified. The characters in a string cannot be changed once a string value is assigned to string variable.

**1. >>> immutable = "dollar"**

**2. >>> immutable[0] = "c"**

**Traceback (most recent call last):**

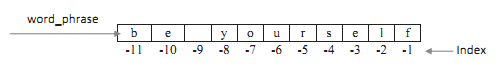
**File "<stdin>", line 1, in <module>**

**TypeError: 'str' object does not support item assignment**

The string value "dollar" is assigned to string variable immutable ➀. If you try to change the string by assigning a character in place of existing character through indexing, then it results in an error as the string is immutable ➁.

1. **What is use of negative indexing? Give example.**

You can also access individual characters in a string using negative indexing. If you have a long string and want to access end characters in the string, then you can count backward from the end of the string starting from an index number of −1.

****

**1. >>> word\_phrase[-1]**

**'f'**

**2. >>> word\_phrase[-2]**

**'l'**

By using negative index number of −1, you can print the character ‘f’ ➀, the negative index number of −2 prints the character ‘l’ ➁.

**5. Give the output of the following Python code:**

str1 = 'This is Python'

print( "Slice of String : ", str1[1 : 4 : 1] )

print ("Slice of String : ", str1[0 : -1 : 2] )

**OUTPUT**

Slice of String : his

Slice of String : Ti sPto

**6. Give the output of following Python code**

newspaper = "new york times"

print(newspaper[0:12:4])

print (newspaper[::4])

**OUTPUT**

**ny**

**ny e**

**7. Give the syntax and example for split function.**

The split() method returns a list of string items by breaking up the string using the delim-iter string. The syntax of split() method is

string\_name.split([separator [, maxsplit]])

Here separator is the delimiter string and is optional. A given string is split into list of strings based on the specified separator. If the separator is not specified then whitespace is considered as the delimiter string to separate the strings. If maxsplit is given, at most max-split splits are done (thus, the list will have at most maxsplit + 1 items). If maxsplit is not specified or −1, then there is no limit on the number of splits.

Example:

1. >>> inventors = "edison, tesla, marconi, newton"

2. >>> inventors.split(",")

['edison', ' tesla', ' marconi', ' newton']

3. >>> watches = "rolex hublot cartier omega"

4. >>> watches.split()

['rolex', 'hublot', 'cartier', 'omega']

**8. Write Python code to print each character in a string.**

1. string = "characters";
3. #Displays individual characters from given string
4. **print**("Individual characters from given string:");
6. #Iterate through the string and display individual character
7. **for** i **in** range(0, len(string)):
8. **print**(string[i], end="  ");

**OUTPUT**

Individual characters from given string:

c h a r a c t e r s

1. **What is list? How to create list?**

Lists are used to store multiple items in a single variable. Lists are constructed using square brackets [ ] wherein you can include a list of items separated by commas. A list can have integer, string or float elements. The syntax for creating list is:

List\_name=[item1,item2,….item N]

Example:

1. >>> superstore = ["metro", "tesco", "walmart", "kmart", "carrefour"]

2. >>> superstore

['metro', 'tesco', 'walmart', 'kmart', 'carrefour']

1. **List any four built-in functions used on list.**

* **len ()**
* **sum()**
* **all()**
* **sorted()**
* **any()**

**11. Write the output of the given python code :**

aList = [123, 'xyz', 'zara', 'abc'];

aList.insert (3,2009)

print ("Final List:", aList)

**OUTPUT:**

Final List: [123, 'xyz', 'zara', 2009, 'abc']

**12. Give the syntax and example for list slicing**

Slicing of lists is allowed in Python wherein a part of the list can be extracted by

specifying index range along with the colon (:) operator which itself is a list.

**Syntax** list\_name[start:stop[:step]]

**Example**

1. >>> fruits = ["grapefruit", "pineapple", "blueberries", "mango", "banana"]

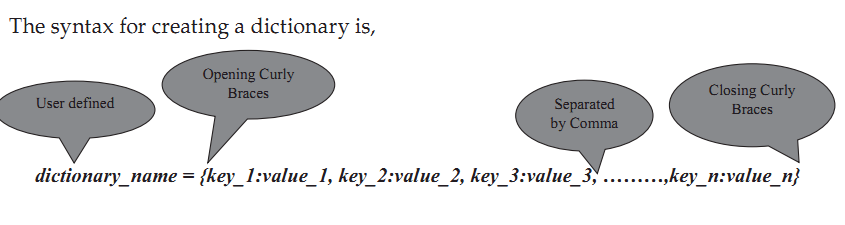
2. >>> fruits[1:3]

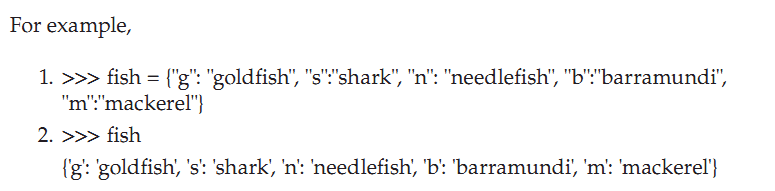
['pineapple', 'blueberries']

**13. What is dictionary? Give example**

A dictionary is a collection of an unordered set of key:value pairs, with the requirement that the keys are unique within a dictionary. Dictionaries are constructed using curly braces {}, wherein you include a list of key:value pairs separated by commas.

**EXAMPLE:**





**14. How to access and modify key value pairs of dictionary ?**

Each individual key:value pair in a dictionary can be accessed through keys by specifying it inside square brackets. The key provided within the square brackets indicates the key:value pair being accessed.

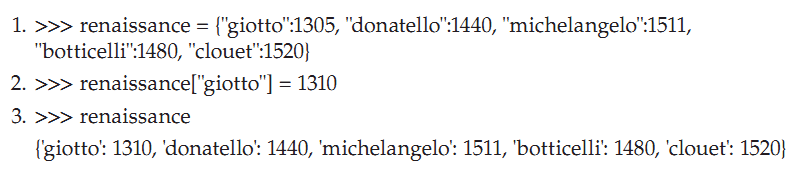
**The syntax for accessing the value for a key in the dictionary is,**

dictionary\_name[key]

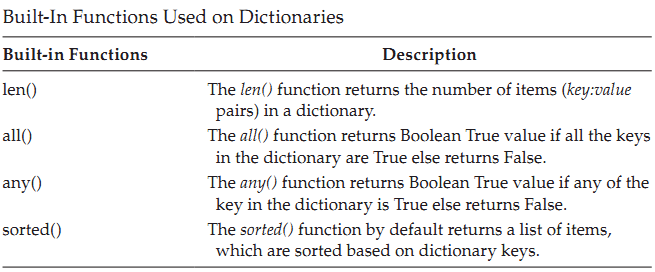
**The syntax for modifying the value of an existing key or for adding a new key:value pair to a dictionary is,**

dictionary\_name[key] = value

**For example,**

****

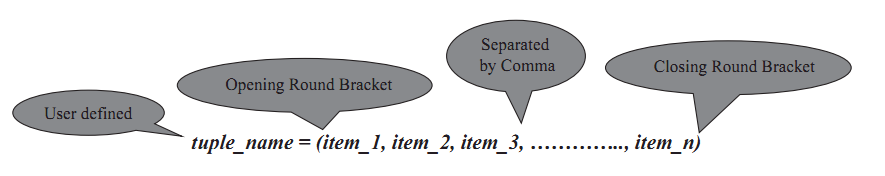
**15. List any four built-in functions used on dictionary.**



**16. What is tuple? How it is created in Python**

* A tuple is a finite ordered list of values of possibly different types which is used to bundle related values together without having to create a specific type to hold them.
* Tuples are immutable. Once a tuple is created, you cannot change its values. A tuple is defined by putting a comma-separated list of values inside parentheses ( ).
* Each value inside a tuple is called an item.

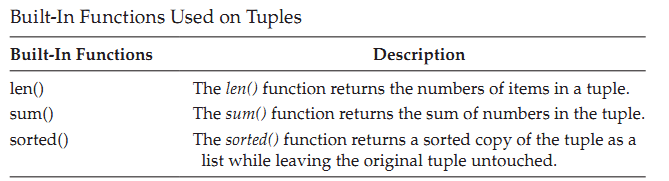
The syntax for creating tuples is,

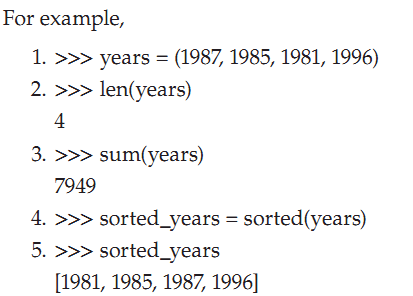


**17. What is the output of print (tuple[1:3]) if tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )?**

It will print elements starting from 2nd till 3rd. Output would be (786, 2.23)

**18. Give syntax and purpose of two built-in functions used on tuples**





**19. How to convert tuple in to List? Give example**

You can convert a tuple to a list by passing the tuple name to the list() function.

1.>>> coral\_reef = ("great\_barrier", "ningaloo\_coast", "amazon\_reef", "pickles\_reef")

2.>>> coral\_reef\_list = list(coral\_reef)

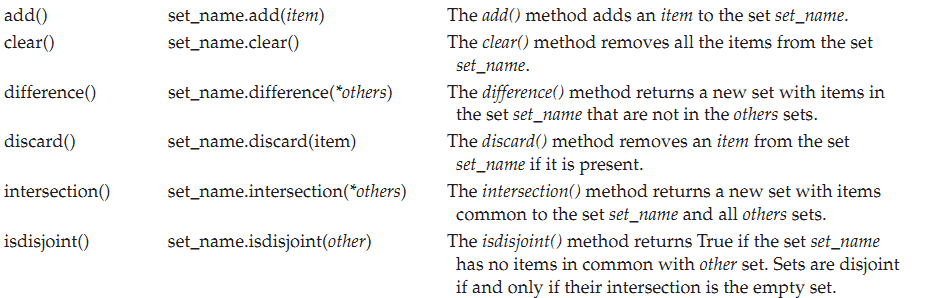
3.>>> coral\_reef\_list

['great\_barrier', 'ningaloo\_coast', 'amazon\_reef', 'pickles\_reef']

20. Differentiate tuple and set datatype.

|  |  |
| --- | --- |
| **tuple** | **Set** |
| A tuple also refers to a data structure of the non-homogenous type that functions to store various elements in columns, multiple rows, and single rows. | A set also refers to a data structure of the non-homogenous type, but it stores various elements in a single row. |
| ordered sequences of elements but their values can't be changed once they're made | Unordered sequences of elements |
| allows duplicates | does not allow duplicates |
| Tuples can be created using **tuple()** | Set can be created using **set()** |
| Example:  **my\_tuple = (10, 20, "codes", "cracker")** | Example:  my\_set = {10, 20, "codes", "cracker"} |

21. List any two set methods and mention purpose of each method.



**Long Answer Questions**

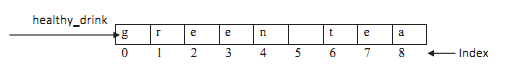
1. **What is string slicing? Explain with examples**

The "slice" syntax is a handy way to refer to sub-parts of sequence of characters within an original string. **The syntax for string slicing is,**

**string\_name[start:end[:step]]**

With string slicing, you can access a sequence of characters by specifying a range of index numbers separated by a colon. String slicing returns a sequence of characters beginning at start and extending up to but not including end. The start and end indexing values have to be integers. String slicing can be done using either positive or negative indexing.

**Positive index slicing**



**Example:**

1. >>> healthy\_drink = "green tea"

2. >>> healthy\_drink[0:3]

'g r e'

3. >>> healthy\_drink[:5]

'g r e e n'

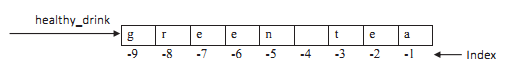
4. >>> healthy\_drink[6:]

'tea'

5. >>> healthy\_drink[:]

'green tea'

**Negative index slicing**



**Example:**

1. >>> healthy\_drink = "green tea"

2. >>> healthy\_drink[-3:-1]

'te'

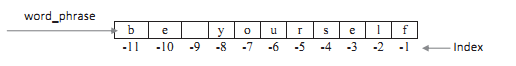
3. >>> healthy\_drink[6:-1]

'te'

1. **Write a note on negative indexing and slicing strings using negative indexing.**

**Negative Indexing**

You can also access individual characters in a string using negative indexing. If you have a long string and want to access end characters in the string, then you can count backward from the end of the string starting from an index number of −1.



1. >>> word\_phrase[-1]

'f'

2. >>> word\_phrase[-2]

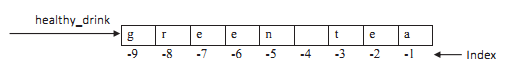
'l'

By using negative index number of −1, you can print the character ‘f’ ➀, the negative index

number of −2 prints the character ‘l’ ➁. You can benefit from using negative indexing when

you want to access characters at the end of a long string.

**Negative index slicing**



**Example:**

1. >>> healthy\_drink = "green tea"

2. >>> healthy\_drink[-3:-1]

'te'

3. >>> healthy\_drink[6:-1]

'te'

You need to specify the lowest negative integer number in the start index position when using negative index numbers as it occurs earlier in the string ➀. You can also combine positive and negative indexing numbers ➁.

1. **Explain split and join methods of string with example.**

**The split()**

The split() method returns a list of string items by breaking up the string using the delim-iter string.

**The syntax of split() method is,**

string\_name.split([separator [, maxsplit]])

**Example:**

1. >>> inventors = "edison, tesla, marconi, newton"

2. >>> inventors.split(",")

['edison', ' tesla', ' marconi', ' newton']

Here comma splited the items.

**join() string**

Strings can be joined with the join() string. The join() method provides a flexible way to concatenate strings. The syntax of join() method is,

**string\_name.join(sequence)**

**Example:**

1. >>> date\_of\_birth = ["17", "09", "1950"]

2. >>> ":".join(date\_of\_birth)

'17:09:1950'

Here colon is joines with date

1. **Explain concatenation , repetition and membership operations on string.**

**concatenation:strings can also be concatenated using + sign**

**Example:**

1. >>> string\_1 = "face"

2. >>> string\_2 = "book"

3. >>> concatenated\_string = string\_1 + string\_2

4. >>> concatenated\_string

'facebook'

Two string variables are assigned with "face"➀ and "book" ➁ string values. The string\_1 and string\_2  are concatenated using + operator to form a new string. The new string concatenated\_string ➃ has the values of both the strings ➂.

**repetition : \* operator is used to create a repeated sequence of strings.**

**Example**

5. >>> repetition\_of\_string = "wow" \* 5

6. >>> repetition\_of\_string

'wowwowwowwowwow'

You can use the multiplication operator \* on a string 5. It repeats the string the number of times you specify and the string value “wow” is repeated five times 6.

**member-ship operators(in and not in)**

You can check for the presence of a string in another string using in and not in member-ship operators. It returns either a Boolean Tr u e or False. The in operator evaluates to Tr u e if the string value in the left operand appears in the sequence of characters of string value in right operand. The not in operator evaluates to Tr u e if the string value in the left operand does not appear in the sequence of characters of string value in right operand.

**Example:**

1. >>> fruit\_string = "apple is a fruit"

2. >>> fruit\_sub\_string = "apple"

3. >>> fruit\_sub\_string in fruit\_string

True

4. >>> another\_fruit\_string = "orange"

5. >>> another\_fruit\_string not in fruit\_string

True

Statement ➂ returns True because the string "apple" is present in the string "apple is a fruit". The not in operator evaluates to True as the string "orange" is not present in "apple is a fruit" string ➄.

1. **Explain any five string functions with syntax and example.** 
   1. **count():**

The method count(), returns the number of non-overlapping occurrences of substring in the range [start, end]. Optional arguments start and end are interpreted as in slice notation.

**Syntax:** string\_name.count(substring [,start [, end]])

**Example:**

>>>>>> warriors = "ancient gladiators were vegetarians"

>>> warriors.count("a")

5

* 1. **upper():**

The method upper() converts lowercase letters in string to uppercase.

**Syntax:** string\_name.upper()

**Example:**

**>>> "galapagos".upper()**

**' G A L A PAG O S '**

* 1. **lower():**

The method lower() converts uppercase letters in string to lowercase**.**

**Syntax:** String\_name.lower()

**Example:**

**>>> "Tortoises".lower()**

**'tortoises'**

* 1. **capitalize():**

The capitalize() method returns a copy of the string with its first character capitalized and the rest lowercased.

**Syntax:** string\_name.capitalize()

**Example:**

>>> species = "charles darwin discovered galapagos tortoises"

>>> species.capitalize()

'Charles darwin discovered galapagos tortoises'

* 1. **find():**

Checks if substring appears in string\_name or if substring appears in string\_name specified by starting index start and ending index end. Return position of the first character of the first instance of string substring in string\_name, otherwise return –1 if substring not found in string\_name.

**Syntax:** string\_name. find(substring[,start[, end]])

**Example:**

>>> "cucumber".find("cu")

0

>>> "cucumber".find("um")

3

>>> "cucumber".find("xyz")

-1

**6.Write a note on indexing and slicing lists.**

As an ordered sequence of elements, each item in a list can be called individually, through  
indexing. The expression inside the bracket is called the index. Lists use square brackets  
[ ] to access individual items, with the first item at index 0, the second item at index 1 and  
so on. The index provided within the square brackets indicates the value being accessed.

**The syntax for accessing an item in a list is,  
list\_name[index]**

**Positive Index**where index should always be an integer value and indicates the item to be selected. For  
the list superstore, the index breakdown is shown below.

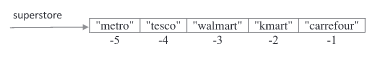
****

**Example:**

**>>> superstore = ["metro", "tesco", "walmart", "kmart", "carrefour"]  
>>> superstore[0]  
'metro'  
>>> superstore[1]  
'tesco'**

**Negative index**

In addition to positive index numbers, you can also access items from the list with a negative  
index number, by counting backwards from the end of the list, starting at −1. Negative index-  
ing is useful if you have a long list and you want to locate an item towards the end of a list.

****

>>> superstore[-3]  
'walmart'  
If you would like to print out the item 'walmart' by using its negative index number, you  
can do so as in ➀

**7. Explain any five list methods with syntax.**

**1. append():**

The append() method adds a single item to the end of the list. This method does not return new list and it just modifies the original.

**Syntax:**

**list.append(item)**

**Example:**

>>> cities.append('brussels')

>>> cities

['washington', 'paris', 'seattle', 'london', 'washington', 'delhi', 'oslo', 'brussels']

**2.count():**

The count() method counts the number of times the item has occurred in the list and returns it.

**Syntax:**

**list.count(item)**

**Example:**

>>> cities = ["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]

>>> cities.count('seattle')

1

1. **remove():**

The remove() method searches for the first instance of the given item in the list and removes it. If the item is not present in the list then ValueError is thrown by this method.

**Syntax:**

**list.remove(item)**

**Example:**

>>>cities=['brussels', 'delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'brussels', 'copenhagen']

>>> cities.remove("brussels")

>>> c it i e s

['delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'brussels', 'copenhagen']

1. **sort():**

The sort() method sorts the items in place in the list. This method modifies the original list and it does not return a new list.

**Syntax:**

**list.sort()**

**Example:**

>>> cities=['washington', 'paris', 'seattle', 'london', 'washington', 'delhi', 'oslo', 'brussels']

>>> cities.sort()

>>> c it i e s

['brussels', 'delhi', 'london', 'oslo', 'paris', 'seattle', 'washington', 'washington']

1. **reverse():**

The reverse() method reverses the items in place in the list. This method modifies the original list and it does not return a new list.

**Syntax:**

**list.reverse()**

**Example:**

>>> cities = ["oslo", "delhi", "washington", "london", "seattle", "paris", "washington"]

>>> cities.reverse()

>>> cities

['washington', 'paris', 'seattle', 'london', 'washington', 'delhi', 'oslo']

**8. Write a Python code to implement stack operations using lists**

1. stack = []

2. stack\_size = 3

3. def display\_stack\_items():

4. print("Current stack items are: ")

5. for item in stack:

6. print(item)

7. def push\_item\_to\_stack(item):

8. print(f"Push an item to stack {item}")

9. if len(stack) < stack\_size:

10. stack.append(item)

11. else:

12. print("Stack is full!")

13. def pop\_item\_from\_stack():

14. if len(stack) > 0:

15. print(f"Pop an item from stack {stack.pop()}")

16. else:

17. print("Stack is empty.")

18. def main():

19. push\_item\_to\_stack(1)

20. push\_item\_to\_stack(2)

21. push\_item\_to\_stack(3)

22. display\_stack\_items()

23. push\_item\_to\_stack(4)

24. pop\_item\_from\_stack()

25. display\_stack\_items()

26. pop\_item\_from\_stack()

27. pop\_item\_from\_stack()

28. pop\_item\_from\_stack()

29. if \_\_name\_\_ == "\_\_main\_\_":

30. main()

**OUTPUT:**

Push an item to stack 1

Push an item to stack 2

Push an item to stack 3

Current stack items are:

1

2

3

Push an item to stack 4

Stack is full!

Pop an item from stack 3

Current stack items are:

1

2

Pop an item from stack 2

Pop an item from stack 1

Stack is empty.

1. **Write a Python code to implement queue operations using lists**

1. from collections import deque

2. def queue\_operations():

3. queue = deque(["Eric", "John", "Michael"])

4. print(f"Queue items are {queue}")

5. print("Adding few items to Queue")

6. queue.append("Terry")

7. queue.append("Graham")

8. print(f"Queue items are {queue}")

9. print(f"Removed item from Queue is {queue.popleft()}")

10. print(f"Removed item from Queue is {queue.popleft()}")

11. print(f"Queue items are {queue}")

12. def main():

13. queue\_operations()

14. if \_\_name\_\_ == "\_\_main\_\_":

15. main()

**OUTPUT:**

Queue items are deque(['Eric', 'John', 'Michael'])

Adding few items to Queue

Queue items are deque(['Eric', 'John', 'Michael', 'Terry', 'Graham'])

Removed item from Queue is Eric

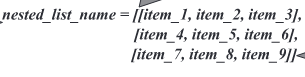
Removed item from Queue is John

Queue items are deque(['Michael', 'Terry', 'Graham'])

1. **Write a note on nested lists.**

A list inside another list is called a nested list.

**The syntax for nested lists is,**



**Each list inside another list is separated by a comma. For example,**1. >>> asia = [["India", "Japan", "Korea"],  
["Srilanka", "Myanmar", "Thailand"],  
["Cambodia", "Vietnam", "Israel"]]  
2. >>> asia[0]  
['India', 'Japan', 'Korea']  
3. >>> asia[0][1]  
'Japan'  
4. >>> asia[1][2] = "Philippines"  
5. >>> asia  
[['India', 'Japan', 'Korea'], ['Srilanka', 'Myanmar', 'Philippines'], ['Cambodia', 'Vietnam','Israel']]

You can access an item inside a list that is itself inside another list by chaining two sets ofsquare brackets together. For example, in the above list variable asia you have three lists ➀which represent a 3 × 3 matrix. If you want to display the items of the first list then specifythe list variable followed by the index of the list which you need to access within the  
brackets, like asia[0] ➁. If you want to access "Japan" item inside the list then you need tospecify the index of the list within the list and followed by the index of the item in the list, like asia[0][1] ➂. You can evenmodify the contents of the list within the list. For example,to replace "Thailand" with "Philippines" use the code in ➃.

1. **With example explain how to Access and Modify key: value Pairs in Dictionaries?**

Accessing and Modifying key: value Pairs in Dictionaries:

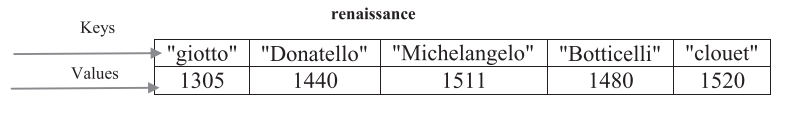
* Each individual key: value pair in a dictionary can be **accessed through keys by specifying it inside square brackets**. The key provided within the square brackets indicates the key: value pair being accessed.
* **The syntax for accessing the value for a key in the dictionary is,**

**dictionary\_name[key]**

* **The syntax for modifying the value of an existing key or for adding a new key:value pair to a dictionary is,**

**dictionary\_name[key] = value**

* If the key is already present in the dictionary, then the key gets updated with the new value. If the key is not present then the new key:value pair gets added to the dictionary.

****

1. >>> renaissance = {"giotto":1305, "donatello":1440, "michelangelo":1511,"botticelli":1480, "clouet":1520}

2. >>> renaissance["giotto"] = 1310

**# Modified value of giotto**

3. >>> renaissance

{'giotto': 1310, 'donatello': 1440, 'michelangelo': 1511, 'botticelli': 1480, 'clouet': 1520}

**# Added leonardo**

4. >>> renaissance["leonardo"] = 1503

5. >>> renaissance

{'giotto': 1310, 'donatello': 1440, 'michelangelo': 1511, 'botticelli': 1480, 'clouet': 1520,

'leonardo': 1503}

* **You can add a new key:value pair by specifying the name of the dictionary followed by a bracket within where you specify the name of the key and assign a value to it .**
* **If you try to access a non-existing key then it results in KeyError.**

1. **Explain any five dictionary methods with syntax.**

**1. fromkeys():**

The fromkeys() method creates a new dictionary from the given sequence of elements with a value provided by the user.

**Syntax: dictionary\_name.fromkeys(seq [, value])**

**Example:**

>>> box\_office\_billion = {"avatar":2009, "titanic":1997, "starwars":2015, "harry-potter":2011, "avengers":2012}**#original dictionery**

.>>>box\_office\_billion\_fromkeys = box\_office\_billion.fromkeys(box\_office\_ billion, "billion\_dollar")

>>> box\_office\_billion\_fromkeys

{'avatar': 'billion\_dollar', 'titanic': 'billion\_dollar', 'starwars': 'billion\_dollar', 'harry-potter': 'billion\_dollar', 'avengers': 'billion\_dollar'}

**2. Keys():** The keys() method returns a new view consisting of all the keys in the dictionary**.**

**Syntax: dictionary\_name.keys()**

**Example:**

>>> box\_office\_billion = {"avatar":2009, "titanic":1997, "starwars":2015, "harry-potter":2011, "avengers":2012} **#original dictionery**

>>> box\_office\_billion.keys()

dict\_keys(['avatar', 'titanic', 'starwars', 'harrypotter', 'avengers'])

1. **Values():** The values() method returns a new view consisting of all the values in the dictionary.

**Syntax dictionary\_name.values()**

**Example:**

>>> box\_office\_billion = {"avatar":2009, "titanic":1997, "starwars":2015, "harry-potter":2011, "avengers":2012} **#original dictionery**

>>> box\_office\_billion.values()

dict\_values([2009, 1997, 2015, 2011, 2012])

**4. Update(): The update() method updates the dictionary with the key:value pairs from other dictionary object and it returns None.**

**Syntax:** dictionary\_name.update([other])

**Example**:

>>> box\_office\_billion = {"avatar":2009, "titanic":1997, "starwars":2015, "harry-potter":2011, "avengers":2012} **#original dictionery**

>>> box\_office\_billion.update({"frozen":2013})

>>> box\_office\_billion

{'avatar': 2009, 'titanic': 1997, 'starwars': 2015, 'harrypotter': 2011, 'avengers': 2012, ' frozen': 2013}

**5. Setdefault():** The setdefault() method returns a value for the key present in the dictionary. If the key is not present, then insert the key into the dictionary with a default value and return the default value. If key is present, default defaults to None, so that this method never raises a KeyError.

**Syntax:** dictionary\_name.setdefault(key[, default])

**Example:**

>>> box\_office\_billion = {"avatar":2009, "titanic":1997, "starwars":2015, "harry-potter":2011, "avengers":2012} **#original dictionery**

>>> box\_office\_billion.setdefault("minions")

>>> box\_office\_billion

{'avatar': 2009, 'titanic': 1997, 'starwars': 2015, 'harrypotter': 2011, 'avengers': 2012, ' frozen': 2013, 'minions': None}

**13. Write a Python Program to Dynamically Build dictionary using User Input as a List**

1. def main():

2. print("Method 1: Building Dictionaries")

3. build\_dictionary = {}

4. for i in range(0, 2):

5. dic\_key = input("Enter key ")

6. dic\_val = input("Enter val ")

7. build\_dictionary.update({dic\_key: dic\_val})

8. print(f"Dictionary is {build\_dictionary}")

9. print("Method 2: Building Dictionaries")

10. build\_dictionary = {}

11. for i in range(0, 2):

12. dic\_key = input("Enter key ")

13. dic\_val = input("Enter val ")

14. build\_dictionary[dic\_key] = dic\_val

15. print(f"Dictionary is {build\_dictionary}")

16. print("Method 3: Building Dictionaries")

17. build\_dictionary = {}

18. i = 0

19. while i < 2:

20. dict\_key = input("Enter key ")

21. dict\_val = input("Enter val ")

22. build\_dictionary.update({dict\_key: dict\_val})

23. i = i + 1

24. print(f"Dictionary is {build\_dictionary}")

25. if \_\_name\_\_ == "\_\_main\_\_":

26. main()

**OUTPUT**

Method 1: Building Dictionaries

Enter key microsoft

Enter val windows

Enter key canonical

Enter val ubuntu

Dictionary is {'microsoft': 'windows', 'canonical': 'ubuntu'}

Method 2: Building Dictionaries

Enter key apple

Enter val macos

Enter key canonical

Enter val ubuntu

Dictionary is {'apple': 'macos', 'canonical': 'ubuntu'}

Method 3: Building Dictionaries

Enter key microsoft

Enter val windows

Enter key apple

Enter val macos

Dictionary is {'microsoft': 'windows', 'apple': 'macos'}

1. **Explain with example how to traverse dictionary using key: value pair**

A for loop can be used to iterate over keys or values or key:value pairs in dictionaries. If you iterate over a dictionary using a for loop, then, by default, you will iterate over the keys.If you want to iterate over the values, use values() method and for iterating over the key:value pairs, specify the dictionary’s items() method explicitly. The dict\_keys, dict\_values, and dict\_items data types returned by dictionary methods can be used in for loops to iterate over the keys or values or key:value pairs.

**Program to Illustrate Traversing of key:value Pairs in Dictionaries Using for Loop**

1. currency = {"India": "Rupee", "USA": "Do l l a r ", "Russia": "Ruble", "Japan": "Ye n",

"Germany": "Euro"}

2. def main():

3. print("List of Countries")

4. for key in currency.keys():

5. print(key)

6. print("List of Currencies in different Countries")

7. for value in currency.values():

8. print(value)

9. for key, value in currency.items():

10. print(f"'{key}' has a currency of type '{value}'")

11. if \_\_name\_\_ == "\_\_main\_\_":

12. main()

**Output**

List of Countries

India

USA

Russia

Japan

Germany

List of Currencies in different Countries

Rupee

Dollar

Ruble

Yen

Euro

'India' has a currency of type 'Rupee'

'USA' has a currency of type 'Dollar'

'Russia' has a currency of type 'Ruble'

'Japan' has a currency of type 'Yen'

'Germany' has a currency of type 'Euro'

Using the keys() ➃–➄, values() ➆–➇, and items() ➈–➉ methods, a for loop can iterate over the keys, values, or key:value pairs in a dictionary, respectively. By default, a for loop iterates over the keys. So the statement for key in currency.keys(): results in the same output as for key in currency:. When looping through dictionaries, the key and its corresponding value can be retrieved simultaneously using the items() method. The values in the dict\_items type returned by the items() method are tuples where the first element in the tuple is the key and the second element is the value. You can use multiple iterating variables in a for loop to unpack the two parts of each tuple in the items() method by assigning the key and value to separate variables ➈

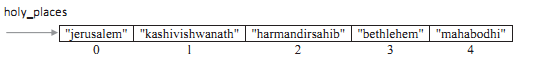
**15. Write a note on indexing and slicing tuples**

Each item in a tuple can be called individually through indexing. The expression inside the bracket is called the index. Square brackets [ ] are used by tuples to access individual items, with the first item at index 0, the second item at index 1 and so on. The index pro-vided within the square brackets indicates the value being accessed.

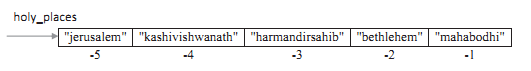
**The syntax for accessing an item in a tuple is,**

tuple\_name[index]

where index should always be an integer value and indicates the item to be selected.



you can also access tuple items using a negative index number, by counting backwards from the end of the tuple, starting at −1. Negative indexing is useful if you have a large number of items in the tuple and you want to locate an item towards the end of a tuple.



**Example:**

>>>holy\_places = ("jerusalem", "kashivishwanath", "harmandirsahib", "bethlehem", "mahabodhi" )

>>> holy\_places[0]

'jerusalem'

>>> holy\_places[2]

'harmandirsahib'

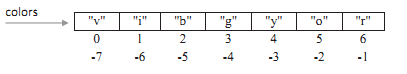
>>> holy\_places[-2]

'beth lehem'

**Slicing** of tuples is allowed in Python wherein a part of the tuple can be extracted by specifying an index range along with the colon (:) operator, which itself results as tuple type. **Syntax:**

tuple\_name[start:stop[:step]]

where both start and stop are integer values (positive or negative values). Tuple slicing returns a part of the tuple from the start index value to stop index value, which includes the start index value but excludes the stop index value. The step specifies the increment value to slice by and it is optional.



>>> colors = ("v", "i", "b", "g", "y", "o", "r")

>>> colors[1:4]

( ' i', ' b', 'g' )

>>> colors[-5:-2]

( ' b', 'g', ' y ' )

**16. Write a Python program to populate tuple with user input data.**

1. tuple\_items = ()

2. total\_items = int(input("Enter the total number of items: "))

3. for i in range(total\_items):

4. user\_input = int(input("Enter a number: "))

5. tuple\_items += (user\_input,)

6. print(f"Items added to tuple are {tuple\_items}")

7. list\_items = []

8. total\_items = int(input("Enter the total number of items: "))

9. for i in range(total\_items):

10. item = input("Enter an item to add: ")

11. list\_items.append(item)

12. items\_of\_tuple = tuple(list\_items)

13. print(f"Tuple items are {items\_of\_tuple}")

**OUTPUT**

Enter the total number of items: 4

Enter a number: 4

Enter a number: 3

Enter a number: 2

Enter a number: 1

Items added to tuple are (4, 3, 2, 1)

Enter the total number of items: 4

Enter an item to add: 1

Enter an item to add: 2

Enter an item to add: 3

Enter an item to add: 4

Tuple items are ('1', '2', '3', '4')

**17. Explain any Five set methods.**

1. **add():** The add() method adds an item to the set set\_name.

**Syntax:** set\_name.add(item)

Example:

1. >>> european\_flowers = {"sunflowers", "roses", "lavender", "tulips", "goldcrest"}

2. >>> american\_flowers = {"roses", "tulips", "lilies", "daisies"}

3. >>> american\_flowers.add("orchids")

4. >>> american\_flowers.difference(european\_flowers)

{'lilies', 'orchids', 'daisies'}

1. **clear():** The clear() method removes all the items from the setset\_name.

**Syntax:** set\_name.clear()

**Example:**

1 >>>american\_flowers = {"roses", "tulips", "lilies", "daisies"}

2>>>american\_flowers.clear()

3>>> american\_flowers

set()

1. **pop():**The method pop() removes and returns an arbitrary item from the set set\_name. It raises KeyError if the set is empty.

**Syntax**: set\_name.pop()

Example:

1>>>european\_flowers = {"sunflowers", "roses", "lavender", "tulips"}

2>>>european\_flowers.pop()

'tulips'

1. **issubset()**: The issubset() method returns True if every item in the set set\_name is in other set.

**Syntax:** set\_name.issubset(other)

Example:

1>>>european\_flowers = {"sunflowers", "roses", "lavender", "tulips"}

2>>> american\_flowers.issubset(european\_flowers)

False

1. **issuperset()**

**Syntax:** set\_name.issuperset(other)

Example:

1>>>european\_flowers = {"sunflowers", "roses", "lavender", "tulips"}

2>>> american\_flowers.issuperset(european\_flowers)

False

**UNIT III**

**2Marks**

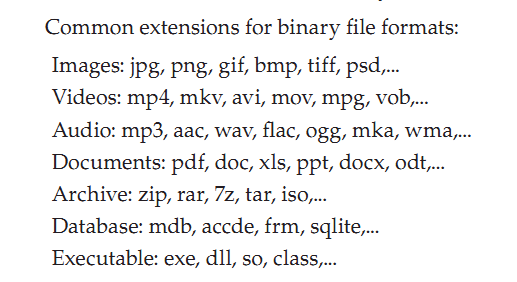
1. **List file types supported by Python. Give example for each?**

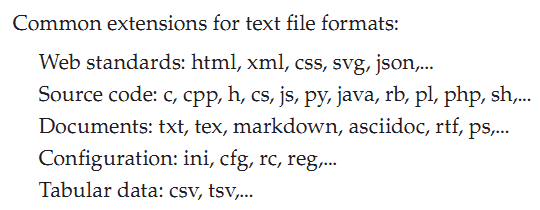
Python supports two types of files –

* **text files**
* **binary files.**

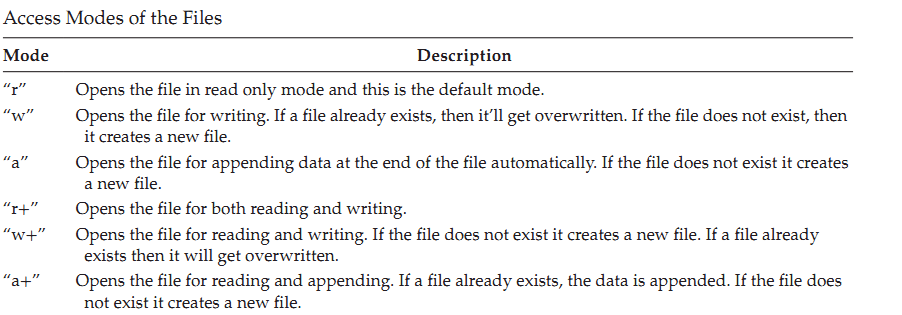
binary and text files contain data stored as a series of bits (binary values of 1s and 0s), the bits in text files represent characters, while the bits in binary files represent custom data.

EXAMPLE:



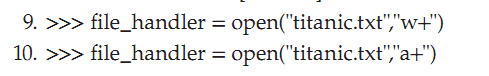


1. **List any four file access modes in Python.**

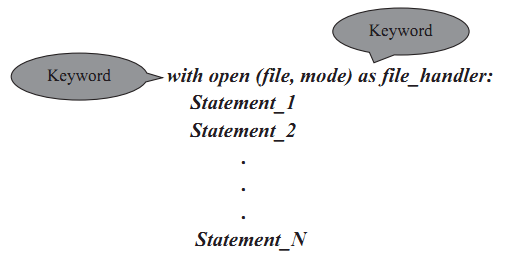


Example:



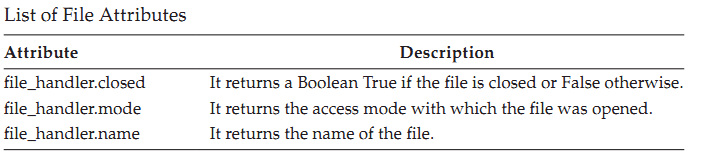


1. **Give the syntax of with statement to open the file.**



* The words with and as are keywords and the with keyword is followed by the open() function and ends with a colon.
* The as keyword acts like an alias and is used to assign the returning object from the open() function to a new variable file\_handler.
* The with statement creates a context manager and it will automatically close the file handler object for you when you are done with it.

1. **List any two file object attributes and mention its purpose.**



1. **What is use of seek() and tell() methods.**

**tell () :**

* This method returns an integer giving the file handler’s current position within the file, measured in bytes from the beginning of the file.

**Syntax: file\_handler.tell ()**

**seek ():**

* This method is used to change the file handler’s position. The position is computed from adding offset to a reference point.
* The reference point is selected by the from\_what argument.
* A from\_what value of 0 measures from the beginning of the file, 1 uses the current file position, and 2 uses the end of the file as the reference point.
* If the from\_what argument is omitted, then a default value of 0 is used, indicating that the beginning of the file itself is the reference point.

**Syntax: file\_handler.seek(offset, from\_what)**

1. **Give syntax of constructor definition in Python.**

Python uses a special method called a constructor method. Python allows you to define only one constructor per class. Also known as the \_\_init\_\_() method, it will be the first method definition of a class.

**syntax is,**

def \_ \_init\_ \_(self, parameter\_1, parameter\_2, …., parameter\_n):

statement(s)

* The \_\_init\_\_() method defines and initializes the instance variables.
* It is invoked as soon as an object of a class is instantiated.
* Class methods that begin with a double underscore (\_\_) are called special methods as they have special meaning.

1. **What is self-variable ?**

The self variable is used to represent the instance of the class which is often used in object-oriented programming. It works as a reference to the object.

When an instance to the class is created, the instance name contains the memory location of the instance. This memory location is

internally passed to 'self'.

**Example:**

def \_ \_init\_ \_(self, parameter\_1, parameter\_2, …., parameter\_n):

statement(s)

1. **How to return object from a method? Give example**

* **Python allows return objects from a method**

**# This function returns a dictionary**

def foo():

d = dict();

d['str'] = "Tutorialspoint"

d['x'] = 50

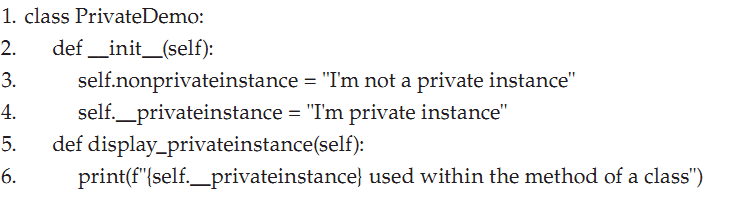
return d

print foo()

**OUTPUT:** {'x': 50, 'str': 'Tutorialspoint'}

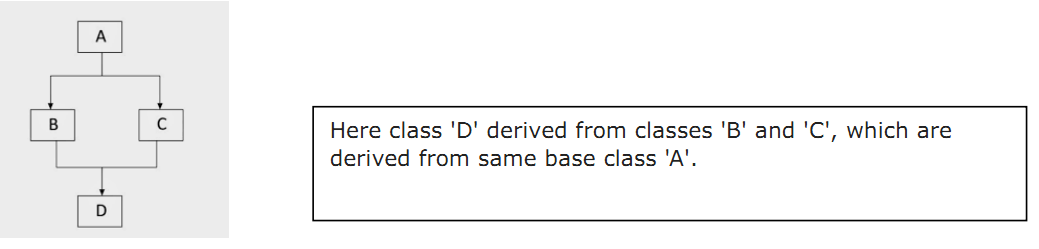
1. **How to define private instance variables and methods in Python.**

Instance variables or methods, which can be accessed within the same class and can’t be seen outside, are called private instance variables or private methods.



1. **What is multipath inheritance ?**

**When a class is derived from two or more classes which are derived from the same base class then such type of inheritance is called multipath inheritance.**

****

**Here's the syntax of the multipath inheritance,**

class ClassA:

# Super class code here

class ClassB(ClassA):

# Derived class B code here

class ClassC(ClassA):

# Derived class C code here

class ClassD(ClassB, ClassC):

# Derived class D code here

**11. What is purpose of super() method in inheritance ?**

super() function can be used to refer to base classeswithout naming them explicitly, thus making the code more maintainable.. The super() function returns an object that represents the parent class.

**Syntax:**

super().\_\_init\_\_(base\_class\_parameter(s))

**12.Give the general syntax of multiple inheritance.**

Class Base\_1()

<statement-1>

………

<statement-N>

Class Base\_2()

<statement-1>

………

<statement-N>

Class Base\_3()

<statement-1>

………

<statement-N>

.

.

class DerivedClassName(Base\_1, Base\_2, Base\_3,.,.):

<statement-1>

.

.

.

<statement-N>

1. **What is operator Overloading ?**

Operator overloading is where we can change the way operators work for user-defined types.For example, the + operator will perform arithmetic addition on two numbers, merge two lists, or concatenate two strings.This feature in Python that allows the same operator to have different meaning according to the context is called operator overloading.

Example:

print (14 + 32)   **O/P: 46**

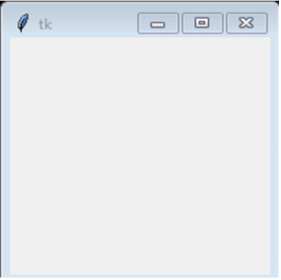
# Now, we will concatenate the two strings

print ("Java" + "Tpoint")   **O/P: JavaTpoint**

1. **What is root window? How it is created in Python?**

The Root Window:

To display the graphical output, we need space on the screen. This space that is initially allocated to every GUI program is called top level window or root window. We can say that the root window is the highest level GUI component in any tkinter application. We can reach this root window by creating an object to Tk class.

**Code to create root window**

#import all components from tkinter

from tkinter import \*

#create the root window

root = Tk()

#wait and watch for any events that may take place

#in the root window

root.mainloop()

1. **What is Canvas? How it is created in Python?**

A canvas is a rectangular area which can be used for drawing pictures like lines, circles, polygons, arcs, etc.

To create a canvas, we should create an object to Canvas class as:

**C = Canvas (root, bg="blue", height=500, width=600, cursor='pencil')**

* c is the Canvas class object, root is the name of the parent window.
* bg represents background color,
* height and width represent the height and width of the canvas in pixels.
* The option 'cursor' represents the shape of the cursor in the canvas.

1. **Differentiate Canvas and frame.**

|  |  |
| --- | --- |
| **Canvas** | **Frames** |
| This is a container that is generally used to draw shapes like lines, curves, arcs and circles. | This is a container that is generally used to display widgets like buttons, check buttons or menus. |
| **C = Canvas (root, bg="blue", height=500, width=600, cursor='pencil')**   * c is the Canvas class object, root is the name of the parent window. * bg represents background color, * height and width represent the height and width of the canvas in pixels. * The option 'cursor' represents the shape of the cursor in the canvas. | **frame= Frame (root, height=n, width=m, bg="color", cursor="cursortype")**   * f is the object of Frame class. The frame is created as a child of root' window. * The options 'height' and width' represent the height and width of the frame in pixels. * Bg represents the back ground color to be displayed . * 'cursor' indicates the type of the cursor to be displayed in the frame. |

1. **How to add a scrollbar to a Text widget?**

it is useful to add scroll bars to the Text widget. A scroll bar is a bar that is

useful to scroll the text either horizontally or vertically. For example, we can create a vertical scroll bar by **creating an object to Scroll bar** class as:

**s = Scrollbar (root, orient=VERTICAL, Command= t.yview)**

**s=Scrollbar (root, orient=HORIZONTAL, command=t.xview)**

* **'orient'** indicates whether it is a vertical scroll bar or horizontal scroll bar.
* The**„command‟** option specifies to which widget this scroll bar should be connected.
* **t.yview** represents that the scroll bar is connected to „t‟, i.e. Text widget and **„yview‟ is for vertical scrolling**.(**t.xview for horizontal**).

After creating the scroll bar, it should be attached to the widget like Text widget or Listbox as:

**t.configure (xscrollcommand=h.set)**

Here**, t'indicates Text widget**. xscrollcommand calls the **set() method** of horizontal scroll bar.

In the same way, we can attach vertical scroll bar as:

**t.configure(yscrollcommand=v.set)**

1. **Differentiate Label and Text Widget.**

|  |  |
| --- | --- |
| **Label** | **Text** |
| A label represents constant text that is displayed in the frame or container. A label can display one or more lines of text that cannot be modified. | Text widget is same as a label or message. But Text widget has several options and can  display multiple lines of text in different colors and fonts. It is possible to insert text into atext widget, modify it or delete it. We can also display images in the Text widget. |
| **A label is created as an object of Label class as:**  lbl = Label(f, text=”text to be displayed”, width=m, height=n, font=('font name', size,„style‟), fg=‟color‟, bg=‟‟color‟)   * f represents the frame object to which the label is created as a child. * 'text' representsthe text to be displayed. * „width‟ represents the width of the label in number of characters   and 'height' represents the height of the label in number of lines.   * „font‟ represents a tuple   that contains font name, size and style.   * fg and 'bg' represents the foreground and   background colors for the text. | **create a Text widget by creating an object to Text class as:**  t = Text (root, width=m, height=n, font=('fontname', size, 'fontstyle'), fg='color'  bg='color', wrap=WORD)   * t represents the object of Text class. * 'root' represents an object of root window or frame. * width' represents the width of the Text widget in characters. * ‘height’ represents The height of the widget in lines. * The option wrap' specifies where to cut the line. |

**19. What is an entry widget? How it is created?**

Entry widget is useful to create a rectangular box that can be used to enter or display one line of text. For example, we can display names, passwords or credit card numbers using Entry widgets. An Entry widget can be created as an object of Entry class as:

**el=Entry(f, width=m, fg='color', bg='color', font=('fontname'. 14),show=‟\*‟)**

* **el**' is the Entry class object.
* **f** indicates the frame which is the parent component for the Entry widget width' represents the size of the widget in number of characters.
* **'fg'** indicates the fore ground color in which the text in the widget is displayed,
* **'bg'** represents the back ground color in the widget.
* **'font'** represents a tuple that contains ſont family name, size and style.
* **'show'** represents a character that replaces the originally typed characters in the Entry widget. For example, show" is useful when the user wants to hide his password by displaying stars in the place of characters.

**20. What is a spin box widget? How it is created?**

**A spin box widget allows the user to select values from a given set of values.** The values may be one of numbers of a set of strings The spin box appears as a long rectangle attached with arrow heads pointing towards up and down. The user can click on the arrowheads to see the next value or previous value The user can also edit the value being displayed in the spin box just like he can do in case of an Entry widget

**A spin box is created as an object of Spinbox class.**

**s1= Spinbox(f, from\_=x to=y, textvariable= val1,width= 15,fg='blue' ,bg='yellow', font=('fontname', size, „fontface‟))**

* **f** represents the parent widget.
* **from\_**' indicates the starting value and „to‟ indicatesthe ending value in the spin box.
* **„textvariable**' shows the control variable, i.e. Val1 that is created as an object of the IntVar class.



**21. List the values that can be assigned to selectmode property of listbox**

* BROWSE
* SINGLE
* MULTIPLE
* EXTENDED

**Long Answer**

1. **List and explain various file opening modes with examples.**
2. **“r”** - Opens the file in read only mode and this is the default mode

Example: >>> file\_handler = open("moon.txt","r")

**#The text file moon.txt present in the current directory is opened in read mode .**

1. **“w”** - Opens the file for writing. If a file already exists, then it’ll get overwritten. If the file does not exist, then it creates a new file.

Example: >>> file\_handler = open("moon.txt","w")

# **The text file moon.txt present in the current directory is opened in write mode .If the file does not exist it create the file moon.txt.**

1. **“a”** - Opens the file for appending data at the end of the file automatically. If the file does not exist it creates a new file.

Example: >>> file\_handler = open("titanic.txt","a+")

**# Opens the file titanic.txt for appending data at the end**

1. **“r+”** - Opens the file for both reading and writing.

**“w+”** - Opens the file for reading and writing. If the file does not exist it creates a new file. If a file already exists then it will get overwritten.

Example: >>> file\_handler = open("titanic.txt","w+")

**# The file is opened in w+ mode for reading and writing.**

1. **“a+”** - Opens the file for reading and appending. If a file already exists, the data is appended. If the file does not exist it creates a new file.

**“x”** - Creates a new file. If the file already exists, the operation fails.

**Example: >>> file\_handler = open("titanic.txt","a+")**

If the file does not exist, then the file name titanic.txt is created. The file is opened in a+ mode for reading, writing, and appending. If the file exists, then the content or data is appended. If the file does not exist, then the file is created.

**Example: >>> file\_handler = open("example.txt","x")**

The mode is "x". The file named example.txt is created if it is not present. If the file already exists, then the operation fails.

**2. With program example explain how ‘with’ statement is used to open and**

**close files**

we can use a with statement in Python such that you do not have to close the file handler object.**The syntax of the with statement for the file I/O is,**

**with open (file, mode) as file\_handler:**

**Statement\_1**

**Statement\_2**

**.**

**.**

**Statement\_N**

the words with and as are keywords and the with keyword is followed by the open() function and ends with a colon. The as keyword acts like an alias and is used to assign the returning object from the open() function to a new variable file\_handler. The with statement creates a context manager and it will automatically close the file handler object.

In the below example, The with statement automatically closes the file after executing its block ofstatements ➂. You can read the contents of the file japan.txt line-by-line using a for loop without running out of memory ➃. This is both efficient and fast.

**EXAMPLE:**

1. def read\_file():

2. print("Printing each line in text file")

3. with open("japan.txt") as file\_handler:

4. for each\_line in file\_handler:

5. print(each\_line, end="")

6. def main():

7. r e a d \_ fi l e ( )

8. if \_\_name\_\_ == "\_\_main\_\_":

9. ma i n()

**OUTPUT:**

Printing each line in text file

National Treasures of Japan are the most precious of Japan's Tangible Cultural Properties.A Tangible Cultural Property is considered to be of historic or artistic value, classified either as"buildings and structures", or as "fine arts and crafts".

**3. With code example explain any two methods to read data from the file.**

**read():**

This method is used to read the contents of a file up to a size and return it as a string. The argument size is optional, and, if it is not specified, then the entire contents of the file will be read and returned.

**Syntax: file\_handler.read([size])**

**Example:**

1. def main():

2. with open("rome.txt") as file\_handler:

3. print("Print entire file contents")

4. print(file\_handler.read(), end=" ")

5. if \_\_name\_\_ == "\_\_main\_\_":

6. main()

**OUTPUT**

Print entire file contents

Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century BC.

The Roman Emperors were monarchial rulers of the Roman State.

The Emperor was supreme ruler of Rome.

Rome remained a republic.

**readlines():**

This method is used to read all the lines of a file as list items.

**Syntax: file\_handler.readlines()**

**Example:**

1. def main():

2. with open("rome.txt") as file\_handler:

3. print("Print file contents as a list")

4. print(file\_handler.readlines())

5. if \_\_name\_\_ == "\_\_main\_\_":

6. main()

**OUTPUT**

['Ancient Rome was a civilization which began on the Italian Peninsula in the 8th century

BC.\n', 'The Roman Emperors were monarchial rulers of the Roman State.\n', 'The Emperor

was supreme ruler of Rome.\n', 'Rome remained a republic.']

**4. With Code example explain any two methods to write data to the file.**

**write() :** This method will write the contents of the string to the file, returning

the number of characters written. If you want to start a new line, you must include the new line character.

**Syntax:** file\_handler.write(string)

**Example:**

f = open("demofile2.txt", "a")

f.write("See you soon!")

f.close()

**#open and read the file after the appending:**

f = open("demofile2.txt", "r")

print(f.read())

**OUTPUT:**

**See you soon!**

**writelines():** This method will write a sequence of strings to the file.

**Syntax:** file\_handler.writelines(sequence)

**Example:**

f = open("demofile3.txt", "a")

f.writelines(["See you soon!", "Over and out."])

f.close()

**#open and read the file after the appending:**

f = open("demofile3.txt", "r")

print(f.read())

**OUTPUT:**

See you soon!Over and out

**5. Write Python Program to Count the Occurrences of Each Word and Also**

**Count the Number of Words in a text File.**

**Data.tat(text file)**

Am python easy but not lazy  
Am very particular in indentation

Source Code:

file = open("data.txt", "rt")

c = dict()

data = file.read()

words = data.split()

for t in words:

if t in c:

c[t] += 1

else:

c[t] = 1

print('Number of words in text file :', len(words))

print('Occurrences of Each Word :',c)

**OUTPUT**

Number of words in text file : 11

Occurrences of Each Word : {'Am': 2, 'python': 1, 'easy': 1, 'but': 1, 'not': 1, 'lazy': 1, 'very': 1, 'particular': 1, 'in': 1, 'indentation': 1}

**6. Explain declaring a class, defining an object and constructor with syntax and example.**

* A **class** is a blueprint from which individual objects are created.
* An **object** is a bundle of related state (variables) and behavior (methods).
* Objects contain variables, which rep-resents the state of information about the thing you are trying to model, and the methods represent the behavior or functionality that you want it to have.

**Syntax of Class**

**class ClassName:**

**<statement-1>**

**.**

**.**

**<statement-N>**

* Classes are defined by using the class keyword, followed by the Class Name and a colon. Class definitions must be executed before they have any effect. Methods are a special kind of function that is defined within a class.
* **Object** refers to a particular instance of a class where the object contains variables and methods defined in the class. Class objects support two kinds of operations: attribute references and instantiation.

**Syntax of Object:**

**object\_name = ClassName (argument\_1, argument\_2, ….., argument n)**

* Class instantiation uses function notation, wherein the class name is followed by parentheses ()
* Python uses a special method called a constructor method. Python allows you to define only one constructor per class. Also known as the \_\_init\_\_ () method, it will be the first method definition of a class.

**Syntax of Constructor**

**def \_\_init\_\_(self, parameter\_1, parameter\_2, …., parameter\_n):**

**statement(s)**

* The **\_\_init\_\_ ()** method defines and initializes the instance variables. It is invoked as soon as an object of a class is instantiated.
* The first parameter in each of these methods is the word self. When self is used, it is just a variable name to which the object that was created based on a class is assigned.

**Example:**

class Mobile: **#declaring class**

def \_\_init\_\_(self): **#Constructor**

print("This message is from Constructor Method")

def receive\_message(self):

print("Receive message using Mobile")

def send\_message(self):

print("Send message using Mobile")

def main():

nokia = Mobile() **#defining a object**

nokia.receive\_message()

nokia.send\_message()

if \_\_name\_\_ == "\_\_main\_\_":

main()

Output

**This message is from Constructor Method**

**Receive message using Mobile**

**Send message using Mobile**

**7. What is inheritance? How to implement inheritance in Python? Give an**

**example**

* Inheritance enables new classes to receive or inherit variables and methods of existing classes.
* A class that is used as the basis for inheritance is called a superclass or base class. A class that inherits from a base class is called a subclass or derived class.
* The terms parent class and child class are also acceptable terms to use respectively.
* A derived class inherits variables and methods from its base class while adding additional variables and methods of its own. Inheritance easily enables reusing of existing code.

**The syntax for a derived class definition looks like this:**

class DerivedClassName(BaseClassName):

<statement-1>

.

.

<statement-N>

* To create a derived class, you add a BaseClassName after the DerivedClassName within the parenthesis followed by a colon. The derived class is said to directly inherit from the listed base class.
* Example:

**class**Parent():

**def**first(self):

           print('first function')

**class**Child(Parent):

**def**second(self):

          print('second function')

ob **=**Child()

ob.first()

ob.second()

**Output:**

first function

second function

# In the above program, you can access the parent class function using the child class object.

1. **Explain with example overriding superclass constructor and method**

* The built-in super () function can be used to refer to base classes without naming them explicitly, thus making the code more maintainable.
* If you need to access the data attributes from the base class in addition to the data attributes being specified in the derived class’s \_\_init\_\_() method, then you must explicitly call the base class \_\_init\_\_() method using super() yourself, since that will not happen automatically.
* The syntax for using super() in derived class \_\_init\_\_() method definition looks like this:

**super().\_\_init\_\_(base\_class\_parameter(s))**

* **super() is a built-in method in Python** that contains the history of super class methods.Hence, we can use **super() to refer to super class constructor and methods from a sub class** .

super().\_\_init\_\_() **#call super class constructor** super().\_\_init\_\_(arguments) **#call super class constructor and pass** arguments super().method() **#call super class method**

* Example: **#Accessing base class constructor and method in the sub class**

class Square:

def \_\_init\_\_(self, x): **#constructor of Base Class**

self.x = x

def area(self):

print('Area of square=', self.x\*self.x)

class Rectangle(Square):

def \_\_init\_\_(self, x, y): **#constructor of derived Class**

**super().\_\_init\_\_(x) #super class constructor**

self.y = y

def area(self):

**super().area()** **#super class method**

print('Area of rectangle=', self.x\*self.y)

**#find areas of square and rectangle**

a,b = [float(x) for x in input("Enter two measurements: ").split()]

r = Rectangle(a,b)

r.area()

#Output

Enter two measurements:

2 2

Area of square= 4.0

Area of rectangle= 4.0

1. **Explain multi-level inheritance with example.**

* Multi-Level inheritance is possible in python like other object-oriented languages. Multi-level inheritance is archived when a derived class inherits another derived class.
* There is no limit on the number of levels up to which, the multi-level inheritance is archived in python.
* Syntax:

**class base:**

**# Members and Functions**

**class derived1(base):**

**# Members and functions of both base and derived1 classes**

**class derived2(derived1):**

**# Members and functions of base class, derived1 class, and derived2 class.**

* In the above syntax base class has derived class (derived class1) which act as a base class for derived class2 and derived class2 can access the property of derived class1 and base class as well.
* Example: **The following code of python shows multilevel inheritance:**

class SuperClass:

def super\_method(self):

print("Super Class method called")

**# define class that derive from SuperClass**

class DerivedClass1(SuperClass):

def derived1\_method(self):

print("Derived class 1 method called")

**# define class that derive from DerivedClass1**

class DerivedClass2(DerivedClass1):

def derived2\_method(self):

print("Derived class 2 method called")

**# create an object of DerivedClass2**

d2 = DerivedClass2()

d2.super\_method() **# Output: "Super Class method called"**

d2.derived1\_method() **# Output: "Derived class 1 method called"**

d2.derived2\_method() **# Output: "Derived class 2 method called"**

**Output**

Super Class method called

Derived class 1 method called

Derived class 2 method called

* In the above example, DerivedClass2 is derived from DerivedClass1, which is derived from SuperClass.
* It means that DerivedClass2 inherits all the attributes and methods of both DerivedClass1 and SuperClass.
* Hence, we are using d2 (object of DerivedClass2) to call methods from SuperClass, DerivedClass1, and DerivedClass2.

1. **Explain multiple inheritance in Python with an example.**

* A [class](https://www.programiz.com/python-programming/class) can be derived from more than one superclass in Python. This is called multiple [inheritance](https://www.programiz.com/python-programming/inheritance).
* Python supports **multiple-class inheritance** and can be defined as an inheritance where a subclass or child class inherits from more than one superclass.
* Syntax:

class DerivedClassName(Base\_1, Base\_2, Base\_3):

<statement-1>

.

.

.

<statement-N>

* Derived class DerivedClassName is inherited from multiple base classes, Base\_1, Base\_2, Base\_3.
* You can call the base class method directly using Base Class name itself without using the super() function. The syntax is,

**BaseClassName.methodname(self, arguments)**

* Use issubclass() function to check class inheritance. The syntax is,

**issubclass(DerivedClassName, BaseClassName)**

* This function returns Boolean True if DerivedClassName is a derived class of base class BaseClassName.
* The DerivedClassName class is considered a subclass of itself. BaseClassName may be a tuple of classes, in which case every entry in BaseClassName will be checked.
* In any other case, a TypeError exception is raised.
* Example: **Example programmultiple inheritance using issubclass**

class Baseclass1: **#Base Class1**

def display(self):

print(“hai am baseclass1”)

class Baseclass2: **#Base Class2**

def show(self):

print(“hai am baseclass2”)

class Derived(Baseclass1,Baseclass2): **#Derived Class**

def view(self):

print(“this is derived class”)

d\_obj = Derived() **#creating object using derived class**

print(issubclass(Derived,Baseclass2)) **#usage of issubclass**

print(issubclass(Baseclass1, Baseclass2))

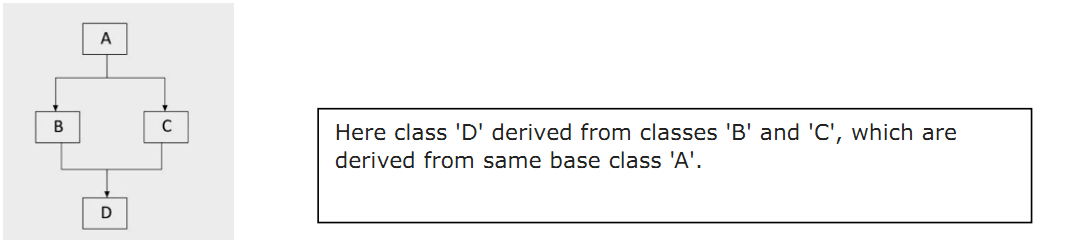
OUTPUT:

**True #Derived class is subclass of Baseclass2**

**False # Baseclass1 & Baseclass2 both are base class.**

1. **Explain multipath inheritance with example.**

When a class is derived from two or more classes which are derived from the same base class then such type of inheritance is called multipath inheritance.



**Here's the syntax of the multipath inheritance,**

class ClassA:

# Super class code here

class ClassB(ClassA):

# Derived class B code here

class ClassC(ClassA):

# Derived class C code here

class ClassD(ClassB, ClassC):

# Derived class D code here

Example:

**#Program to demonstrate Multipath Inheritance**

class University:

def \_\_init\_\_(self):

print("Constructor of the Base class")

def display(self):

print(f"The University Class display method")

class Course(University):

def \_\_init\_\_(self):

print("Constructor of the Child Class 1 of Class University")

super().\_\_init\_\_()

def display(self):

print(f"The Course Class display method")

super().display()

class Branch(University):

def \_\_init\_\_(self):

print("Constructor of the Child Class 2 of Class University")

super().\_\_init\_\_()

def display(self):

print(f"The Branch Class display method ")

super().display()

class Student(Course, Branch):

def \_\_init\_\_(self):

print("Constructor of Child class of Course and Branch is called")

super().\_\_init\_\_()

def display(self):

print(f"The Student Class display method")

super().display()

**# Object Instantiation:**

ob = Student() **# Object named ob of the class Student**.

print()

ob.display() **# Calling the display method of Student class.**

**Output:**

Constructor of Child class of Course and Branch is called

Constructor of the Child Class 1 of Class University

Constructor of the Child Class 2 of Class University

Constructor of the Base class

The Student Class display method

The Course Class display method

The Branch Class display method

The University Class display method

1. **Explain method overloading and overriding with example**

* It is a form of Compile time polymorphism. In the case of method overloading, more than a single method belonging to a single class can share a similar method name while having different signatures.

**Python Method Overloading**

If a method is written such that it can perform more than one task, it is called method overloading.For example, we call a method as:

sum(10, 15)

sum(10, 15, 20)

In the first call, we are passing two arguments and in the second call, we are passing three arguments. It means, the sum() method is performing two distinct operations: finding sum of two numbers or sum of three numbers.This is called method overloading.

**A Python program to show method overloading to find sum of two or three numbers**.

**#method overloading**

class Myclass:

def sum(self, a=None, b=None, c=None):

if a!=None and b!=None and c!=None:

print('Sum of three=', a+b+c)

elif a!=None and b!=None:

print('Sum of two=', a+b)

else:

print('Please enter two or three arguments')

**#call sum() using object**

m = Myclass()

m.sum(10, 15, 20)

m.sum(10.5, 25.55)

m.sum(100)

* Here the sum () method is calculating sum of two or three numbers and hence it is performing more than one task. Hence it is an overloaded method.
* In this way, overloaded methods achieve polymorphism. 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 sub class. Instead, he wants a new functionality to the same method in the sub class.

**A Python program to override the super class method in sub class.**

**#method overriding**

import math

class Square:

def area(self, x):

print('Square area= %.4f'% x\*x)

class Circle(Square):

def area(self, x):

print('Circle area= %.4f'% (math.pi\*x\*x))

c = Circle()

c.area(15)

**13. Explain the steps involved in creating a GUI application in Python with a suitable example.**

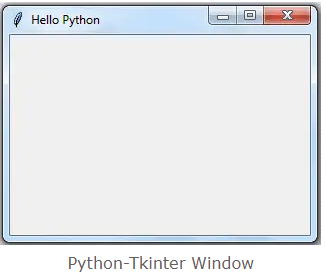
**The following are the general steps involved in basic GUI programs:**

1. First of all, **we should create the root window**. The root window is the top level window that provides rectangular space on the screen where we can display text colors, images, components, etc.

2. **In the root window, we have to allocate space for our use**. This is done by creating a canvas or frame: So, canvas and frame are child windows in the root window.

3Generally, **we use canvas for displaying drawings like lines, arcs, circles,shapes, etc** We use frame for the purpose of displaying components like push buttons, check buttons, menus, etc. These components are also called ‘widgets’.

4. **When the user clicks on a widget like push button, we have to handle that event**. It means we have to respond to the events by performing the desired tasks.



**Example:**

from tkinter import \*

window=Tk()

# add widgets here

window.title('Hello Python')

window.geometry("300x200+10+20")

window.mainloop()

**14. How to create a button widget and bind it to the event handler? Explain with example.**

A push button is a component that performs some action when clicked. These buttons are created as objects of Button class as:

**b = Button(f, text='My Button', width=15, height=2, bg='yellow', fg='blue', activebackground='green', activeforeground='red')**

* Here, „**b‟** is the object of Button class. **‟ f'**‟ represents the frame for which the button is created as a child. It means the button is shown in the frame.
* The **'text**' option represents the text to be displayed on the button.
* **„width‟** represents the width of the button in characters. If an image is displayed on the button instead of text, then width represents the width in pixels.
* **'height'** represents the height of the button in textual lines. If an image is displayed on the button, then 'height' represents the height of the button in pixels.
* **bg** represents the **back groundcolor** color and '**fg**' represents the **foreground** of the button.
* **'activebackground'** represents the background color when the button is clicked.
* **'activeforeground'** represents the foreground color when the button is clicked.

We create a push button with some options and add the button to the frame. Then we link the mouse left button with the **buttonClick()** method using **bind()** method as:

**b.bind(‘<Button-1>’,buttonClick)**

<Button-1> is represents the mouse left button that is linked with buttonClick() method. It means when the mouse left button is clicked, the buttonClick() method is called. This method is called event handler.

Program 9: A Python program to create a push button and bind it with an event handler function.

from tkinter import \*

**# method to be called when the button is clicked**

def buttonclick(self):

print ('You have clicked me)

**# create root window**

root = TkO

**# create frame as child to root window**

f= Frame (root, height-200, width=300)

**# let the frame wil1 not shrink**

f-propagate (0)

**# attach the frame to root w1ndow**

f. pack()

**# create a push button as child to frame**

b= Button(f, text='My Button', width=15, height=2, bg='yellow' fg= ‘blue’, activebackground= ‘green’, activeforeground=’red')

**# attach button to the frame**

b.packO

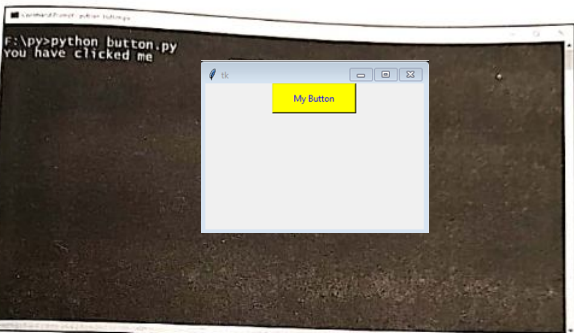
**# bind the left mouse button with the method to be called**

b.bind("<Button-1>", buttonClick)

**# the root window handles the mouse click event**

root.mainloop()

**OUTPUT:**



**15. Write a note on arranging Widgets in a frame using layout managers.**

Arranging Widgets in the Frame Once we create widgets or components, we can arrange them in the frame in a particular manner. Arranging the widgets in the frame is called layout management.

**There are three types of layout managers**

* Pack layout manager
* Grid Layout manager
* Place layout manager

1. **Pack layout manager**

* Pack layout manager uses **Pack()** method. This method is useful to associate a widget with its parent component While using the pack() method, we can mention the position of the widget using ‘**fill**’ or ‘**side’** options

**b.pack(fill=x)**

**b.pack(fill=x)**

* The ‘fill’ option can take the values: X, Y, BOTH, NONE. The value **X** represents that the **widget should occupy the frame horizontally** and the value **Y** represents that the **widget should occupy vertically**.
* **BOTH** represents that the **widget should occupy in both the directions**. **NONE** represents that the **widget should be displayed as it is**. The default value is NONE.
* Along with **‘fill’** option, we can use **padx** and **pady** options that represent how much space should be left around the component horizontally and vertically.

**For example:**

#occupy vertically, space on x-axis 10 px, on Y-axis 15 px

b1.pack(fill=Y, padx=10, pady=15)

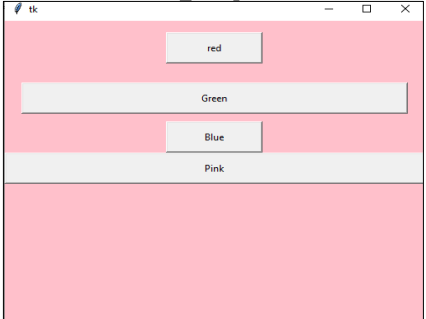
# occupy horizontally. Space on x-axis 10 px, on Y-axis 15 px

b2.pack(fill=X, padu-10, pady-15

b3.pack(fill=X) # occupy horizontally, No space outside the widget

b4.pack(fill=X) # occupy horizontally, No space outside the widge

**OUTPUT:**



* The **pack()** method can take **another option ‘side’** which is used to place the widgets side by side. „side‟ can take the values LEFT, RIGHT, TOP or BOTTOM. The default value is TOP.

**For example,**

**# align towards left with 10 px and 15 px spaces**

b1.pack(side=LEFT, padx=10, pady=15)

**# align towards bottom with 10 px and 15 px spaces**

b2.pack(side=BOTTOM, padx=10, pady=15)

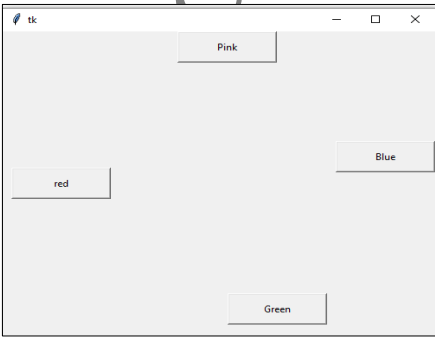
**# align towards right with 0 px space around the widget**

b3.pack(side=RIGHT)

**# align towards top with 0 px space around the widget**

b4.pack()

**Output : Arrangement of buttons using pack() method with 'side' option**



1. **Grid layout manager**

* Grid layout manager uses the grid() method to arrange the widgets in a two dimensional table that contains rows and columns.
* The horizontal arrangement of data is called „row‟ and vertical arrangement is called „column‟. The position of a widget is defined by a row and a column number.
* The size of the table is determined by the grid layout manager depending on the widgets size.

**Example**

**# display in 0th row, 0th column with spaces around**

bl.grid(row=0, column=0, padx=10, pady=15)

**# display in Oth row, 1st column with spaces around**

b2.grid(row=0, column=1, padx=10, pady=15

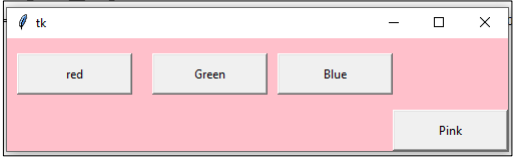
**# display in Oth row, 2nd column without spaces around**

b3.grid(row=0, column=2)

**# display in 1st row, 3rd column without spaces around**

b4.grid(row=1, column=3)

**Output:**



1. **Place layout manager**

* Place layout manager uses the place() method to arrange the widgets.
* The place() method takes x and y coordinates of the widget along with width and height of the window where the widget has to be displayed.

For example:

**# display at (20, 30) coordinates in the window 100 pxwidth and 50 pxheight**

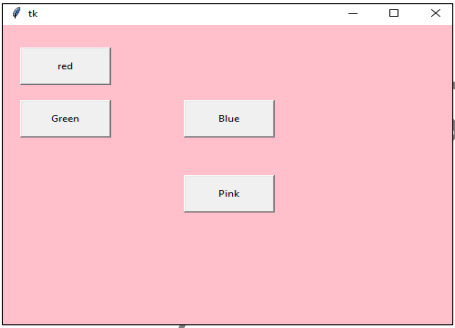
bl.place(x=20, y=30, width=100, height=50)

b2.place(x=20, y=100, width=100, height=50) **# display at (20, 100)**

b3.place(x=200, y=100, width=100,height=50**)# display at (200, 100)**

b4.place(x=200, y=200, width=100, height=50) **# display at (200, 100)**

**OUTPUT:**



**16. Explain the process of creating a Listbox widget with a suitable example. Also, explain different values associated with selectmode option.**

* A list box is useful to display a list of items in a box so that the user can select 1 or more items. To create a list box, we have to create an object of Listbox class, as:

**lb = Listbox(f, font="Arial 12 bold", fg='blue', bg='yellow', height=8, width=24, activestyle='underline', selectmode=MULTIPLE)**

* Here, lb is the list box object. The option 'height' represents the number of lines shown in the list box. width represents the width of the list box in terms of number of characters and the default is 20 characters.
* The option 'activestyle' indicates the appearance of the selected item. It may be ‘underline', 'dotbox' or 'none'. The default value is underline'.
* **The option 'selectmode' may take any of the following values:**
* **BROWSE**: Normally, we can select one item (or line) out of a list box. If we click on an item and then drag to a different item, the selection will follow the mouse. This is the default value of 'selectmode' option.
* **SINGLE**: This represents that we can select only one item( or line) from all available list of items.
* **MULTIPLE**: We can select 1 or more number of items at once by clicking on the items. If an item is already selected, clicking second time on the item will un-select it.
* **EXTENDED**: We can select any adjacent group of items at once by clicking on the first item and dragging to the last item.

**17. Write a note on**

**i) Text Widget ii) Entry Widget**

**Text Widget**

* Text widget has several options and can display multiple lines of text in different colors and fonts. It is possible to insert text into a text widget, modify it or delete it.
* We can also display images in the Text widget. One can create a Text widget by creating an object to Text class as:

****

t = Text (root, width=20, height=10, font=('Verdana', 14, 'bold'), fg='blue' bg='yellow', wrap=WORD)

* Here, t represents the object of Text class. 'root' represents an object of root window or frame. width' represents the width of the Text widget in characters.
* 'height' represents the height of the widget in lines. The option wrap' specifies where to cut the line.
* wrap=CHAR represents that any line that is too long will be broken at any character. wrap=WORD will eak the line in the widget after the last word that fits in the line.
* wrap=NONE will not wrap the lines. In this case, it is better to provide a horizontal scroll bar to view the lines properly in the Text widget.
* Once the Text widget is created, we can insert any text using the insert() method as:
* **t.insert(END, „Text widget \nThis text is inserted into the text widget. \n This is second line\n and this is third line\n‟)** Here, the first argument END represents that the text is added at the end of the previous text.
* We can also use CURRENT to represent that the text is added at the current cursor position. The second argument is the text that is added to the Text widget.

from tkinter import\*

class MyText:

**#constructor**

def \_\_init\_\_(self, root):

**# create a Text widget with 20 chars width and 10 lines height**

self.t= Text (root, width=20, height=10, font=(„Verdana‟, 14,„bold‟), fg=‟blue‟, bg=‟yellow‟,

wrap=WORD)

**#insert some text into the Text widget**

self.t.insert(END, “Text widget this text is inserted into the Text widget. \n This is second

line widget \n and this is third line \n”)

**# attach Text to the root**

self.t.pack(side=LEFT)

**OUTPUT:**

****

**Entry Widget**

Entry widget is useful to create a rectangular box that can be used to enter or display one line of text. For example, we can display names, passwords or credit card numbers using Entry widgets. An Entry widget can be created as an object of Entry class as:

**el=Entry(f, width=m, fg='color', bg='color', font=('fontname'. 14),show=‟\*‟)**

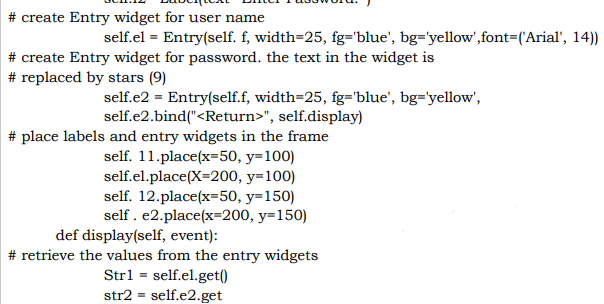
* **el**' is the Entry class object.
* **f** indicates the frame which is the parent component for the Entry widget width' represents the size of the widget in number of characters.
* **'fg'** indicates the fore ground color in which the text in the widget is displayed,
* **'bg'** represents the back ground color in the widget.
* **'font'** represents a tuple that contains ſont family name, size and style.
* **'show'** represents a character that replaces the originally typed characters in the Entry widget. For example, show" is useful when the user wants to hide his password by displaying stars in the place of characters.
* After typing text in the Entry widget, the user presses the Enter button. Such an event should be linked with the Entry widget using bind() method as:

**el.bind("<Return") ,self.display)**

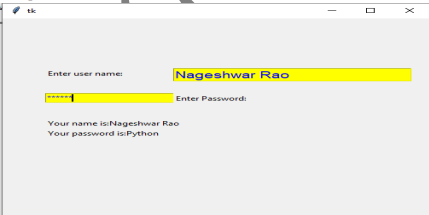
* When the user presses Enter (or Return) button, the event is passed to display() method, Hence, we are supposed to catch the event in the display method, using the following statement:

**def display (self, event):**

* As seen in the preceding code, we are catching the event through an argument 'event' in the display() method. This argument is never used inside the method. The method consists of the code that is to be executed when the user pressed Enter button.



**OUTPUT**



**Unit4**

**2Marks**

**1. How to connect to the SQLite database ? Give example**

**Connect ():**

To use SQLite3 in Python, first we have to import the sqlite3 module and then create a connection object. Connection object allows to connect to the database and will let us execute the SQL statements.

**Creating Connection object using the connect () function:**

import sqlite3

con = sqlite3.connect('mydatabase.db')

**#This will create a new file with the name ‘mydatabase.db’.**

**2. Write SQL code to create table in SQLite database.**

import sqlite3

con = sqlite3.connect('mydatabase.db')

cursorObj = con.cursor()

cursorObj.execute('''CREATE TABLE movie(title text, year int, score real)''')

con.commit()

con.close()

**# In the above code, it establishes a connection and creates a cursor object to execute the create table statement.**

**3. Write SQL code to insert data in SQLite table.**

import sqlite3

con = sqlite3.connect('mydatabase.db')

cursorObj = con.cursor()

cursorObj.execute('''CREATE TABLE movie(title text, year int, score real)''')

cursorObj.execute('''INSERT INTO movie VALUES ("Titanic",1997, 9.5)''')

con.commit()

con.close()

**4. Write SQL code to update SQLite table.**

import sqlite3

con = sqlite3.connect('mydatabase.db')

cursorObj = con.cursor()

cursorObj.execute("UPDATE MOVIE SET SCORE=10 WHERE TITLE='Dil' ")

con.commit()

con.close()

**5. What is NumPy in Python? Give any two uses of NumPy.**

NumPy is the fundamental package for scientific computing with Python. It stands for

“Numerical Python.” It supports:

• N-dimensional array object

• Broadcasting functions

• Tools for integrating C/C++ and Fortran code

• Useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as a multi-dimensional con-tainer to store generic data. Arbitrary data types can also be defined.

**6. Give Python code to create NumPy array using array function.**

We can create a NumPy array from a regular Python list or tuple using the np.array() function.

1. >>> import numpy as np

2. >>> int\_number\_array = np.array([1,2,3,4])

3. >>> int\_number\_array

array([1, 2, 3, 4])

**7. How to create two-dimensional arrays using NumPy.**

To create a 2D (2 dimensional) array in Python using NumPy library using the np.array() function, we can use any of the following methods.

* **numpy.array()** – Creates array from given values.
* **numpy.zeros()** – Creates array of zeros.
* **numpy.ones()** – Creates array of ones.
* **numpy.empty()** – Creates an empty array.

1 >>> two\_dimensional\_array\_list = np.array([[1,2,3], [4,5,6]])

2 >>> two\_dimensional\_array\_list

array([[1, 2, 3],

[4, 5, 6]])

**8. List any four NumPy array attributes.**

* **ndarray.ndim**: Gives the number of axes or dimensions in the array
* **ndarray.size**: Gives the total number of elements of the array.
* **ndarray.itemsize** : Gives the size of each element of the array in bytes.
* **ndarray.shape** : Gives the dimensions of the array. For an array with n rows and m columns, shape will be a tuple of integers (n, m).

**9. Give syntax and example for NumPy arrange() function**

**np.arange():** Returns evenly spaced values within a given interval where start (a number and optional) is the start of interval and its default value is zero, stop (a number) is the end of interval, and step (a number and is optional) is the spacing between the values and dtype is the type of output array.

**The syntax for arange() is,**

np.arange([start,]stop, [step,][dtype=None])

**Example:**

>>>np.arange(0, 2, 0.3)

array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8])

**10. What is Pandas Library?**

pandas is a Python library that provides fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data both easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python.

**11. What is Padas Series ? Give example.**

Series is a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.

**Pandas Series is created using series() method and its syntax is,**

**s = pd.Series(data, index= None)**

s is the Pandas Series, data can be a Python dict, a ndarray, or a scalar value (like 5). The passed index is a list of axis labels. Both integer and label-based indexing are sup-ported. If the index is not provided, then the index will default to range(n) where n is the length of data

1. >>> import numpy as np

2. >>> import pandas as pd

3. >>> s = pd.Series(np.random.randn(5), index=[ 'a', ' b', 'c ', 'd', 'e' ])

4. >>> t y pe(s)

<class 'pandas.core.series.Series'>

5. >>> s

a -0.367740 **#O/P**

b 0.855453

c -0.518004

d -0.060861

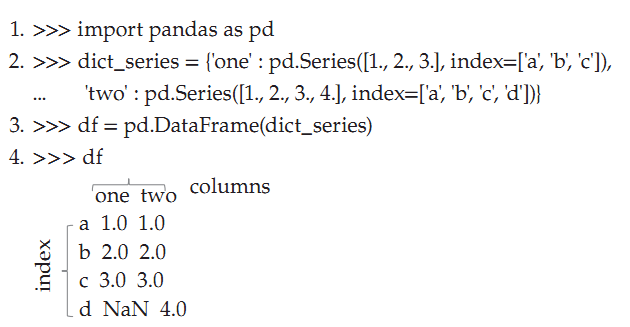
e -0.277982

index

dtype: float64

* Import NumPy and pandas libraries ➀–➁ .
* Create a series using ndarray which is NumPy’s array class using Series() method ➂ which returns a Pandas Series type s ➃.
* You can also specify axis labels for index, i.e., index= ['a', 'b', 'c', 'd', 'e'] ➂ .
* When data is a ndarray, the index must be the same length as data. In series s ➄ , by default the type of values of all the elements is dtype: float64.

**12. Write Python code to create Dataframe from a dictionary and display its contents.**



**13. Write Python code to create Dataframe from a tuple and display its contents.**

**# Importing pandas package**

import pandas as pd

**# Creating two list of tuples**

data = [

('Ram', 'APPLE', 23),

('Shyam', 'GOOGLE', 25),

('Seeta', 'GOOGLE', 22),

('Geeta', 'MICROSOFT', 24),

('Raman', 'GOOGLE', 23),

('Sahil', 'SAMSUNG', 23)

]

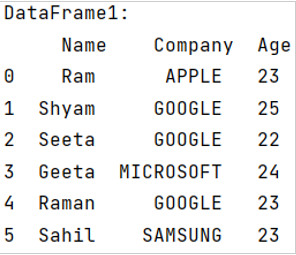
**# Creating a DataFrame**

df = pd.DataFrame(data,columns=['Name','Company','Age'])

**# Display DataFrame**

print("DataFrame1:\n",df,"\n")

**OUTPUT**

****

**14. What is Pandas DataFame ? How it is created ?**

**DataFrame is a two-dimensional, labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or database table, or a dict of Series objects**. (A Pandas DataFrame is a 2-dimensional data structure, like a 2-dimensional array, or a table with rows and columns). It is generally the most commonly used pandas object. DataFrame accepts many different kinds of input like Dict of one-dimensional ndarrays, lists, dicts, or Series, two-dimensional ndarrays, structured or record ndarray, a dictionary of Series, or another DataFrame.

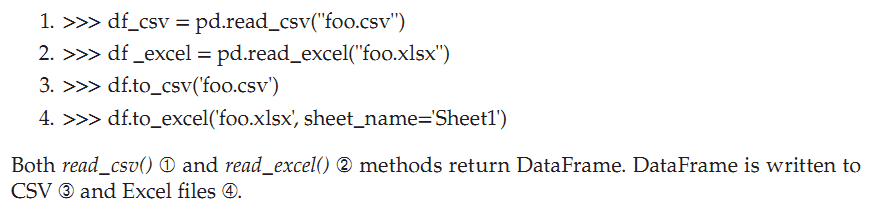
**df = pd.DataFrame(data=None, index=None, columns=None)**

**df** is the DataFrame and **data** can be NumPy ndarray, dict, or DataFrame. Along with the data, you can optionally pass an **index** (row labels) and **columns** (column labels) attributes as arguments.

**15. Give the Python code to create dataframe from .csv file**

You can read from CSV and Excel files using read\_csv() and read\_excel() methods. Also, you can write to CSV and Excel files using to\_csv() and to\_excel() methods.

Example:



**16. How to add new column to dataframe?**

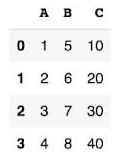
Pandas DataFrame presents data in tabular rows and columns. Adding new columns is an important task in data analysis. By using insert() method we can add new column to dataframe.

1>>>import numpy as np **#O/P**

2>>>import pandas as pd

3>>>df = pd.DataFrame({"A": [1, 2, 3, 4],

"B": [5, 6, 7, 8]})

4>>>df

**ADDING COLUMNS ON THE END #O/P**

5>>>df["C"] = [10, 20, 30, 40]

6>>>df

**ADD COLUMNS AT A SPECIFIC INDEX(insert()) #O/P**

7>>>df.insert(1, "D", 5)

8>>>df

**ADD COLUMNS WITH LOC #O/P**

9>>>df.loc[:, "E"] = list("abcd")

10>>>df

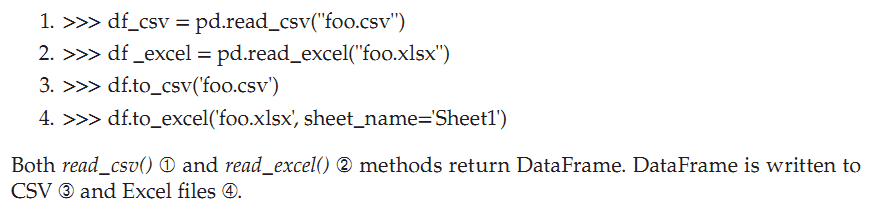
**ADD COLUMNS WITH THE ASSIGN FUNCTION O/P**

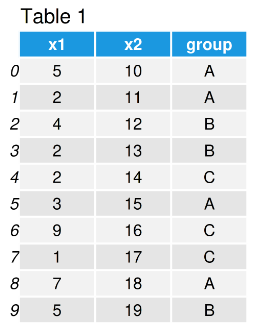
11>>>df = df.assign(F = df.C \* 10)

12>>>df

**17. Give the Python code to create datafram from Excel file.**

You can read from CSV and Excel files using read\_csv() and read\_excel() methods. Also, you can write to CSV and Excel files using to\_csv() and to\_excel() methods.



**18. Give Python code to find maximum and minimum values for particular column of dataframe.**

* **min()** Compute minimum of group values
* **max()** Compute maximum of group values

1>>>import pandas as pd **# Import pandas library**

**# Create pandas DataFrame**

2>>>data = pd.DataFrame({'x1':[5, 2, 4, 2, 2, 3, 9, 1, 7, 5], 'x2':range(10, 20),

'group':['A', 'A', 'B', 'B', 'C', 'A', 'C', 'C', 'A', 'B']})

3>>>print(data) **# Print pandas DataFrame**

4>>>print(data['x1'].max()) **# Get max of one column**

# 9

5>>>print(data['x1'].min()) **# Get min of one column**

# 1

**19. What is Data Visualization ?**

When data is shown in the form of pictures,it becomes easy for the user to understand it.Representing the data in the form of pictures or graphs is called data visualization.The importance of data visualization is simple: it helps people see, interactwith, and better understand data. Whether simple or complex, the rightvisualization can bring everyone on the same page, regardless of their level ofexpertise.

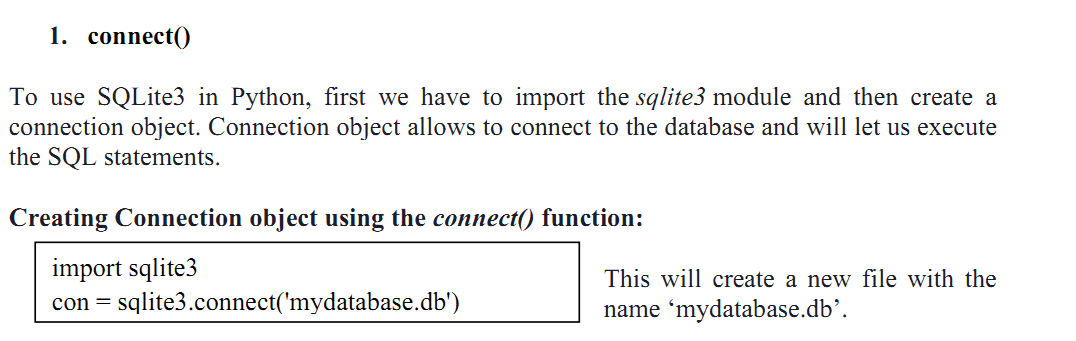
**20. What is matplotlib and pyplot ?**

**Pyplot is an API (Application Programming Interface) for Python’s matplotlib that effectively makes matplotlib a viable open source alternative to MATLAB.**

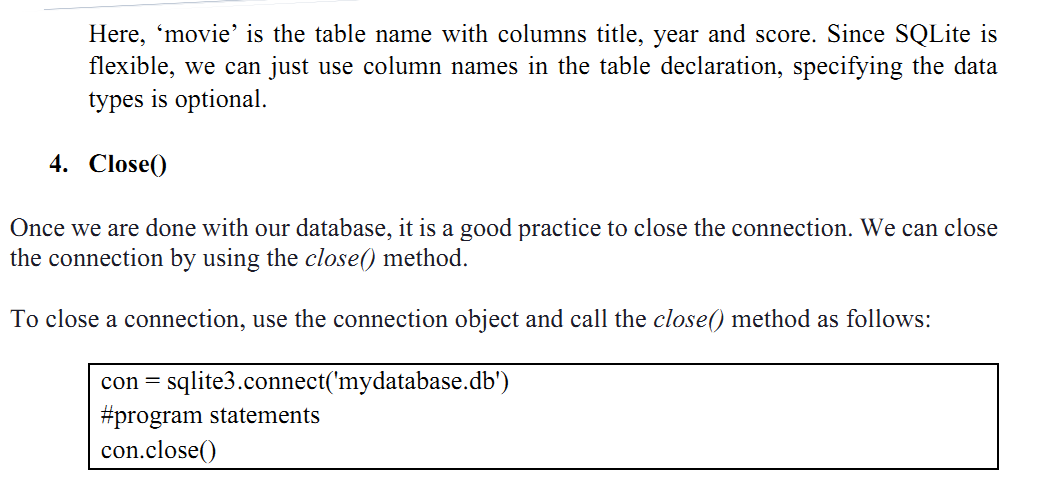
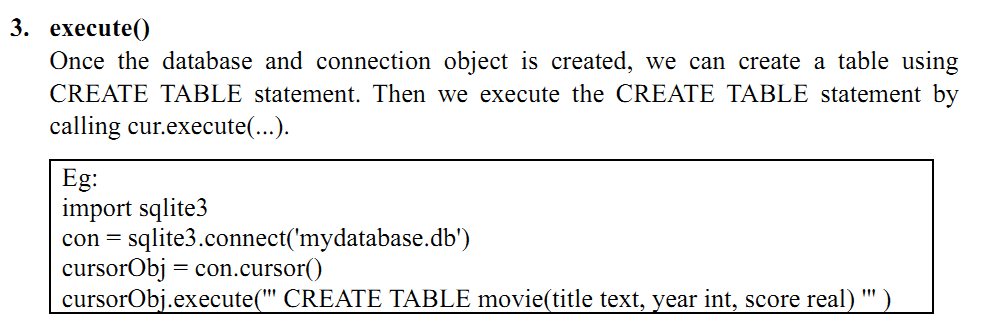
matplotlib is another important package in Python that is useful to produce good quality 2D graphics. It is mostly used for the purpose of showing data in the form of graphs and also for designing electronic circuits, machinery parts, etc. To download and install this package, we should go to System prompt and then use 'pip' (Python Installation of Packages) command as shown :C:\> pip install matplotlib.

**LONG Answer**

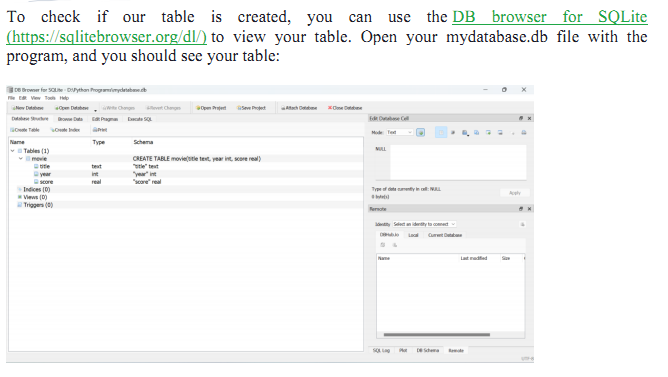
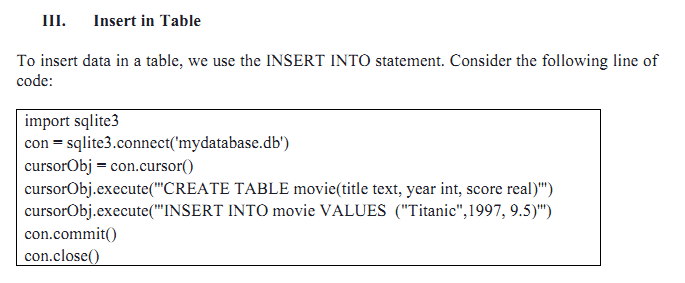
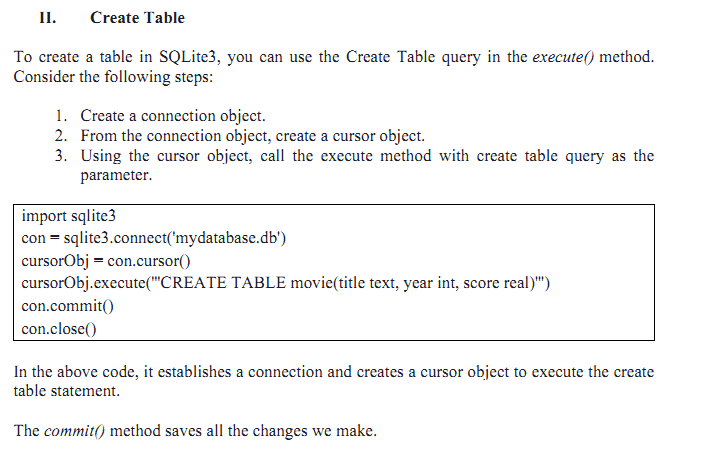
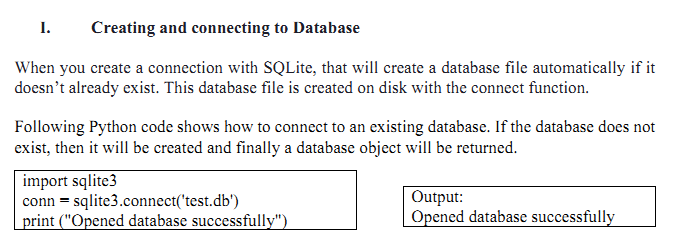
1. **Explain four SQLite module methods required to use SQLite database.**

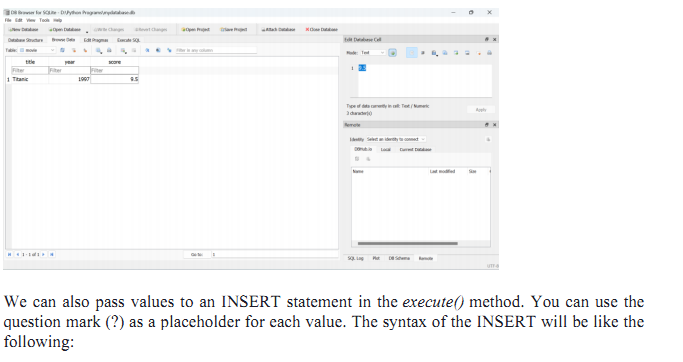


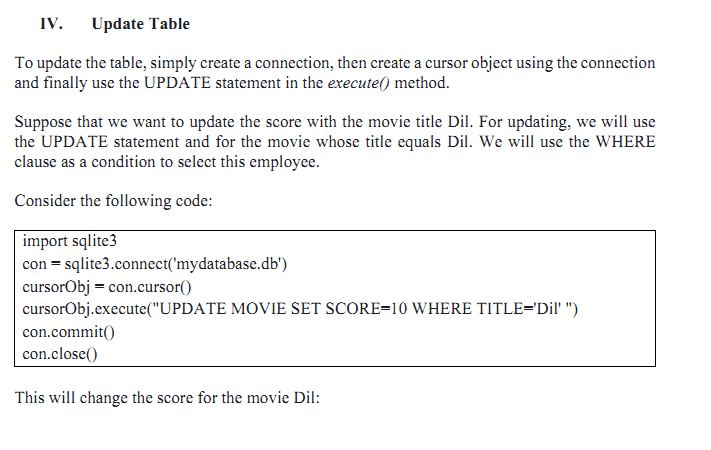
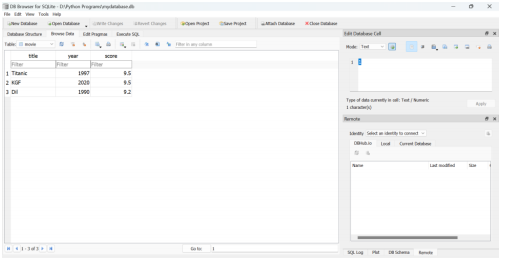
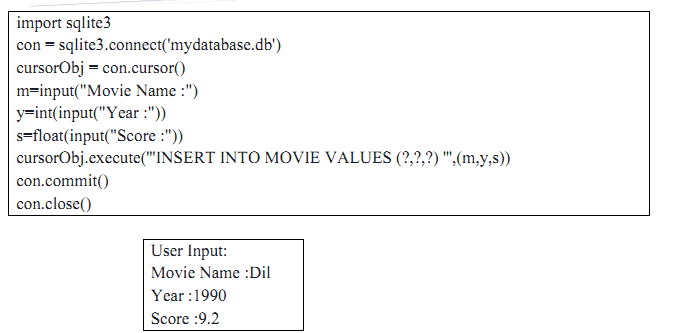


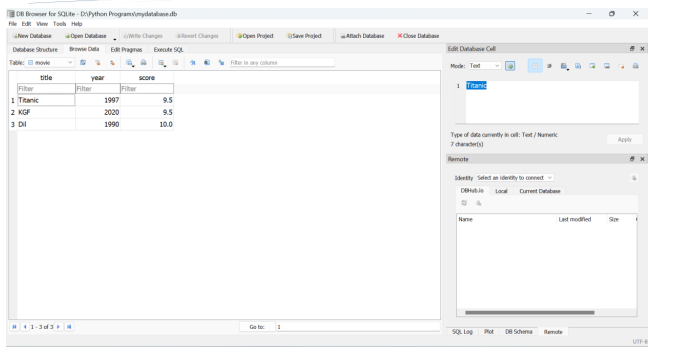


1. **Explain any four SQLite database operations with example.**









1. **Write a Python Program to demonstrate various SQLite Database operations.**

**#importing the module**

import sqlite3

**#create connection object**

con = sqlite3.connect('mydatabase.db')

**#crete a cursor**

cursorObj = con.cursor()

**#creating the table**

cursorObj.execute('''CREATE TABLE movie(title text, year int, score real)''')

**#inserting the data**

cursorObj.execute('''INSERT INTO movie VALUES ("Titanic",1997, 9.5)''')

cursorObj.execute('''INSERT INTO movie VALUES ("KGF",2020, 9.1)''')

cursorObj.execute('''INSERT INTO movie VALUES ("Dil",1990, 8.5)''')

**#Print the Initial Data**

print("Initial Data...")

cursorObj.execute("SELECT \* FROM movie ")

[print(row) for row in cursorObj.fetchall()]

**#Updating**

cursorObj.execute("UPDATE MOVIE SET SCORE=10 WHERE TITLE='Dil' ")

**#Print the after updating**

print("After updating...")

cursorObj.execute("SELECT \* FROM movie ")

[print(row) for row in cursorObj.fetchall()] #Deleting

cursorObj.execute("Delete from movie where title='Titanic' ")

**#Print the after Deleting**

print("After deleting...")

cursorObj.execute("SELECT \* FROM movie ")

[print(row) for row in cursorObj.fetchall()]

**#Drop the table**

cursorObj.execute(" DROP TABLE IF EXISTS MOVIE ")

**#commit changes in the database**

con.commit()

**#close the connection**

con.close()

1. **Explain any five NumPy array attributes with syntax.**
2. **ndarray.ndim** : Gives the number of axes or dimensions in the array

Syntax: **ndarray.ndim**

Example:

import numpy as np

array\_attributes = np.array([[10, 20, 30], [14, 12, 16]])

array\_attributes.ndim

**O/P: 2**

**2**. **ndarray.shape** : Gives the dimensions of the array. For an array with n rows and m columns, shape will be a tuple of integers (n, m).

Syntax: **ndarray.shape**

Example:

import numpy as np

array\_attributes = np.array([[10, 20, 30], [14, 12, 16]])

print a.shape

**O/P: (2, 3)**

**3. ndarray.size** : Gives the total number of elements of the array.

Syntax : **ndarray.size**

Example:

import numpy as np

array\_attributes = np.array([[10, 20, 30], [14, 12, 16]])

array\_attributes.size

**O/P: 6**

**4. ndarray.dtype** : Gives an object describing the type of the elements in the array. One can create or specify dtype’s using standard Python types. Additionally, NumPy provides its own types like np.int32, np.int16, np.float64, and others.

Syntax: **ndarray.dtype**

Example:

import numpy as np

array\_attributes = np.array([[10, 20, 30], [14, 12, 16]])

array\_attributes.dtype

**O/P: dtype('int32')**

**5**.**ndarray.itemsize**: Gives the size of each element of the array in bytes.

Syntax: **ndarray.itemsize**

Example:

import numpy as np

array\_attributes = np.array([[10, 20, 30], [14, 12, 16]])

array\_attributes.itemsize

**O/P: 4**

**6.ndarray.data** : Gives the buffer containing the actual elements of the array. Normally, we will not use this attribute because we will access the elements in an array using indexing facilities.

**Syntax** : **ndarray.data**

**Example:**

import numpy as np

array\_attributes = np.array([[10, 20, 30], [14, 12, 16]])

array\_attributes.data

**O/P: <memory at 0x000001E61DB963A8>**

1. **Explain any four NumPy array creation functions with example.**

**np.zeros()** : Creates an array of zeros

**Syntax: np.zeros()**

**Example:**

import numpy as np

np.zeros((2,3))

**O/P:** array([[0., 0., 0.],

[0., 0., 0.]])

**np.linspace() :** Returns evenly spaced numbers over a specified interval where start is the starting value of the sequence, stop is the end value of the sequence, and num (an integer and optional) is the number of samples to generate. Default is 50. Must be non-negative. The optional dtype is the type of the output array.

**The syntax for linspace is,**

**numpy.linspace(start, stop, num= 50, dtype= None)**

1. **Write a note on Indexing, slicing, and iterating operations on NumPy array.**

Contents of ndarray object can be accessed and modified by indexing or slicing**.** NumPy **indexing** is used for accessing an element from an array by giving it an index value that starts from 0**.Slicing** in python means taking elements from one given index to another given index. Use the Plus (+) operator to refer to an index from the beginning. Use the minus (-) operator to refer to an index from the end.**Iterating** means going through elements one by one. As we deal with multi-dimensional arrays in numpy, we can do this using basic for loop of python.

**Example: Integer Indexing, Array Indexing, Boolean Array Indexing, Slicing**

**and Iterating in Arrays**

1. >>> import numpy as np

2. >>> a = np.arange(5)

3. >>> a

array([0, 1, 2, 3, 4])

4. >>> a[2]

2

5. >>> a[2:4]

array([2, 3])

6. >>> a[:4:2] = -999

7. >>> a

array([-999, 1, -999, 3, 4])

8. >>> a[::-1]

array([ 4, 3, -999, 1, -999])

9. >>> for each\_element in a:

10. ... print(each\_element)

-999

1

-999

3

4

**Indexing, slicing, and iterating operations on one-dimensional NumPy arrays ➀–➉.**

**7. Explain basic arithmetic operations on NumPy array with examples.**

**Basic mathematical functions perform element-wise operation on arrays and are available both as operator overloads and as functions in the NumPy module. For example,**

1. >>> import numpy as np

2. >>> a = np.array( [20, 30, 40, 50] )

3. >>> b = np.arange(4)

4. >>> b

array([0, 1, 2, 3])

5. >>> a + b

array([20, 31, 42, 53])

6. >>> np.add(a, b)

array([20, 31, 42, 53])

7. >>> a – b

array([20, 29, 38, 47])

8. >>> np.subtract(a, b)

array([20, 29, 38, 47])

9. >>> A = np.array( [[1, 1], [6, 1]] )

10. >>> B = np.array( [[2, 8], [3, 4]] )

11. >>> A \* B

array([[2, 8],

[18, 4]])

12. >>> np.multiply(A, B)

array([[ 2, 8],

[18, 4]])

13. >>> A / B

array([[0.5 , 0.125],

[2. , 0.25 ]])

14. >>> np.divide(A, B)

array([[0.5 , 0.125],

[2. , 0.25 ]])

15. >>> np.dot(A, B)

array([[ 5, 12],

[15, 52]])

16. >>> B\*\*2

array([[ 4, 64],

[ 9, 16]], dtype=int32)

Element-wise sum, subtract, multiply, and divide operations are performed resulting in an array 5 –14. Matrix product is carried out in 15. Every element is squared in array B as shown in 16.

**8.With code examples explain creating pandas series using Scalar data and**

**Dictionary. (Refer Q.No:10)**

**9. Explain any four string processing methods supported by Pandas Library with example.**

The Pandas Series supports a set of string processing methods that make it easy to operate on each element of the array. These methods are accessible via the str attribute and they generally have the same name as that of the built-in Python string methods.

**1. Str.lower(): convert to lower case**

1. >>> import numpy as np

2. >>> import pandas as pd

3. >>> empires\_ds = pd.Series(["Vijayanagara", "Roman", "Chola", "Mongol", "A k k a d i a n" ] )

4. >>> empires\_ds.str.lower()

0 vijayanagara

1 roman

2 chola

3 mongol

4 akkadian

dtype: object

**2. str.upper(): convert to upper case**

1. >>> import numpy as np

2. >>> import pandas as pd

3. >>> empires\_ds = pd.Series(["Vijayanagara", "Roman", "Chola", "Mongol", "A k k a d i a n" ] )

4. >>> empires\_ds.str.upper()

0 VIJAYANAGARA

1 ROMAN

2 CHOLA

3 MONGOL

4 AKKADIAN

dtype: object

**3. str.len(): displays the length**

1. >>> import numpy as np

2. >>> import pandas as pd

3. >>> empires\_ds = pd.Series(["Vijayanagara", "Roman", "Chola", "Mongol", "A k k a d i a n" ] )

4>>> empires\_ds.str.len()

0 11

1 5

2 5

3 6

4 8

dtype: int64

**4. str.count: no of times the character occurrence**

>>> names\_ds = pd.Series(['Jahnavi', 'Adelmo', 'Pietro', 'Alejandro'])

>>> names\_ds.str.count('e')

0 0

1 1

2 1

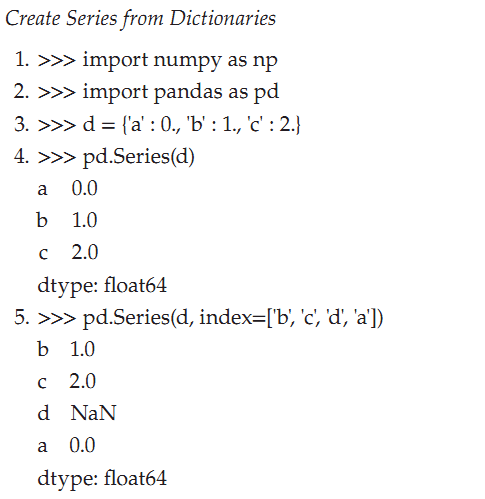
3 1

dtype: int64

**10. Explain with example any two methods of creating DataFrame.**



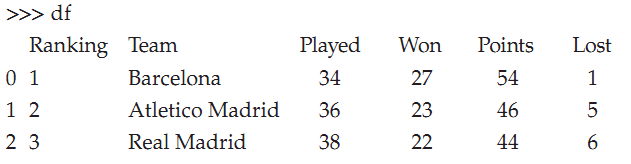
You can create a Pandas Series from scalar value. Here scalar value is five ➂. If data is a scalar value, an index must be provided. The value will be repeated to match the length of the index.

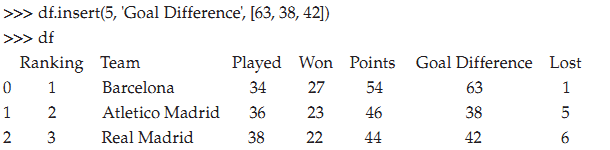


Series can be created from the dictionary. Create a dictionary ➂ and pass it to Series() method ➃. When a series is created using dictionaries, by default the keys will be index labels. While creating series using a dictionary, if labels are passed for the index, the values corresponding to the labels in the index will be pulled out ➄.

**11. Explain any five operations on Dataframe with example.**

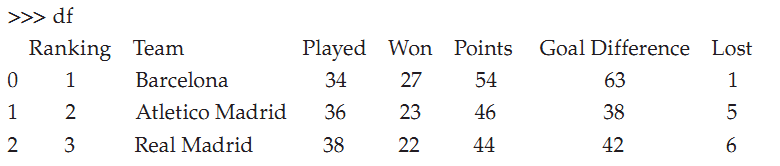
**1. Insert():The insert function is available to insert at a particular location in the columns**

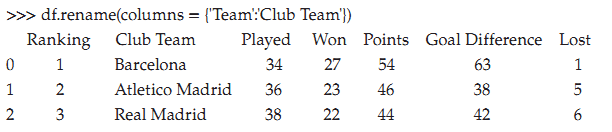
****

****

**Column goal difference inserted**

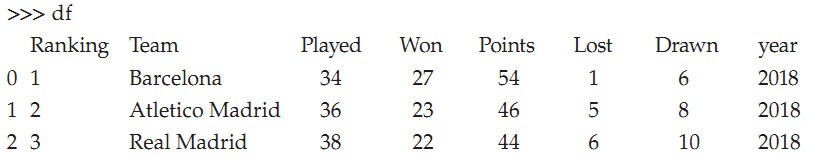
**2. Rename():You can rename the column label using therename() method**

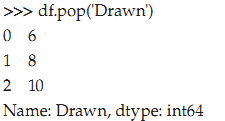
****

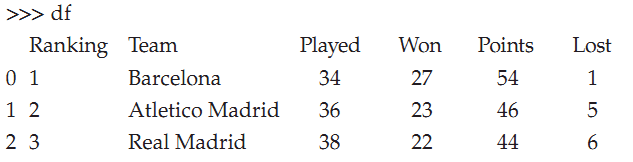
****

**Column team renamed as club team**

**3. Pop(): columns can be poped.**

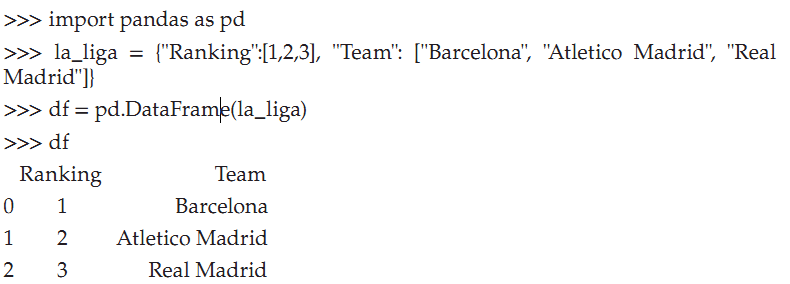
****

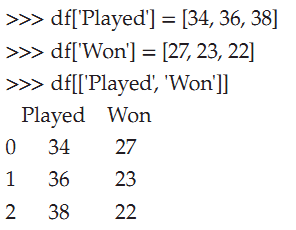
****

****

**Column drawn poped out**

**4. Arithmatic operation: we can do arithmetic operation like addition, subtraction, multiplication, division.**

****

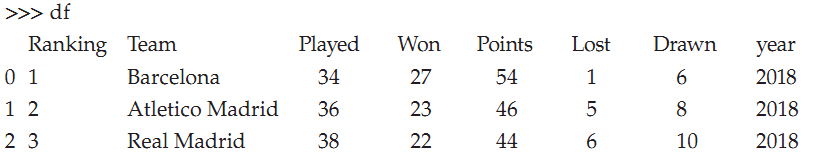
****

****

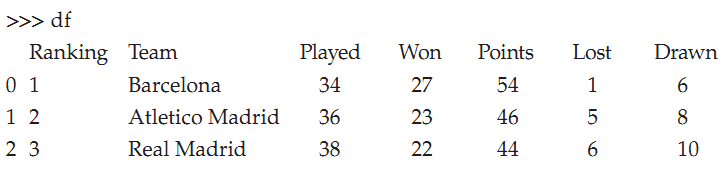
**Arithmetic multiplication function applied in column won.**

**5. Del()( delete() ):Columns can be deleted**

**Example:**

****

****

****

**Column year deleted**

**12. Explain Bar Graph creation using Matplot Library module.**

To create a bar graph using the Matplotlib library in Python, you can usethe `matplotlib.pyplot.bar()` function. The bar graph is a visual representation ofcategorical data with rectangular bars, where the length of each bar represents aspecific value. Here's an example that demonstrates how to create a bar graphusing Matplotlib:

import matplotlib.pyplot as plt

**# Sample data**

categories = ['A', 'B', 'C', 'D']

values = [10, 20, 15, 25]

**# Create a bar graph**

plt.bar(categories, values)

**# Add labels and title**

plt.xlabel('Categories')

plt.ylabel('Values')

plt.title('Bar Graph')

**# Display the graph**

plt.show()

In this example, we start by importing the `matplotlib.pyplot` module as`plt`. We define the sample data we want to plot, where `categories` representsthe categories on the x-axis and `values` represents the corresponding values foreach category.Next, we create a bar graph using the `plt.bar()` function. We pass the`categories` and `values` as the first two arguments to the function.

To enhance the graph, we add labels and a title using the `plt.xlabel()`,`plt.ylabel()`, and `plt.title()` functions, respectively.Finally, we display the graph using `plt.show()`.

Running this code will generate a bar graph with the categories on the x-axis and the corresponding values on the y-axis. Each category will berepresented by a rectangular bar, and the length of each bar will represent itscorresponding value.You can customize the appearance of the bar graph further by adjusting thecolor, width, alignment, and other properties using additional parameters in the`plt.bar()` function. Matplotlib offers extensive customization options to createvisually appealing and informative bar graphs.

**13. Write a program to display histogram .**

import matplotlib.pyplot as plt

**# Sample data**

data = [10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80]

**# Create a histogram**

plt.hist(data)

**# Add labels and title**

plt.xlabel('Values')

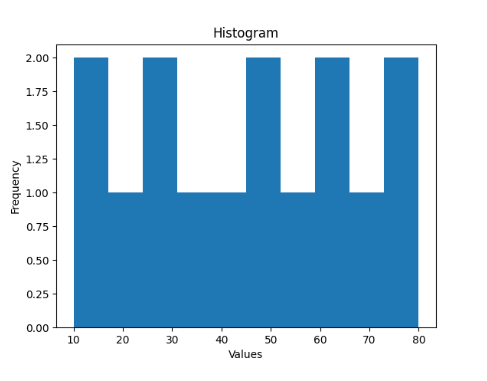
plt.ylabel('Frequency')

plt.title('Histogram')

**# Display the histogram**

plt.show()

**OUTPUT**



In this example, we start by importing the `matplotlib.pyplot` module as`plt`. We define the sample data that we want to visualize using a histogram.The `data` list contains the values for which we want to create a histogram.Next, we create a histogram using the `plt.hist()` function. We pass the`data` as the first argument to the function.

To enhance the histogram, we add labels and a title using the `plt.xlabel()`,`plt.ylabel()`, and `plt.title()` functions, respectively. These labels provideinformation about the x-axis, y-axis, and the title of the histogram.Finally, we display the histogram using `plt.show()`.

Running this code will generate a histogram with bins representing therange of values and the frequency of occurrence for each bin. The x-axisrepresents the values, and the y-axis represents the frequency or count of valuesfalling within each bin.

**14. Write a Python program to display Pie Chart showing percentage of**

**employees in each department. Assume there are 4 departments namely**

**Sales , Production , HR and Finance.**

import matplotlib.pyplot as plt

# Sample data

departments = ['Sales', 'Production', 'HR', 'Finance']

employee\_counts = [40, 30, 20, 10]

# Create a pie chart

plt.pie(employee\_counts, labels=departments, autopct='%1.1f%%')

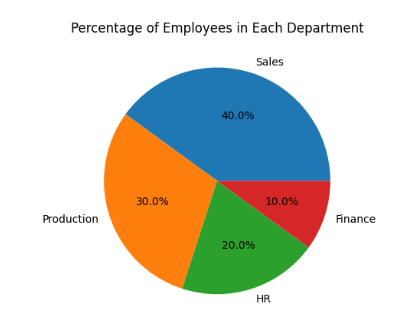
# Add title

plt.title('Percentage of Employees in Each Department')

# Display the pie chart

plt.show()

**OUTPUT:**

****

```

In this example, we start by importing the `matplotlib.pyplot` module as`plt`. We define the sample data that we want to visualize. The `departments` listcontains the department names, and the `employee\_counts` list contains thecorresponding number of employees in each department.Next, we create a pie chart using the `plt.pie()` function. We pass the`employee\_counts` as the first argument and the `departments` as the `labels`parameter to the function. The `autopct='%1.1f%%'` parameter is used to displaythe percentage values on each slice of the pie chart.To enhance the pie chart, we add a title using the`plt.title()` function.Finally, we display the pie chart using `plt.show()`.

Running this code will generate a pie chart where each department isrepresented by a slice. The size of each slice represents the percentage ofemployees in that department. The legend labels indicate the department names,and the percentage values are displayed on each slice.

You can further customize the appearance of the pie chart by adjustingparameters such as colors, explode (to highlight a particular slice), shadow, and more using additional arguments in the `plt.pie()` function. Matplotlib provides various options to create visually appealing and informative pie charts.

**15. Write a Python Program to create Line Graph showing number of students of a college in various Years. Consider 8 years data.**

import matplotlib.pyplot as plt

**# Sample data**

years = [2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022]

student\_counts = [500, 600, 700, 800, 900, 1000, 1100, 1200]

**# Create a line graph**

plt.plot(years, student\_counts, marker='o')

**# Add labels and title**

plt.xlabel('Year')

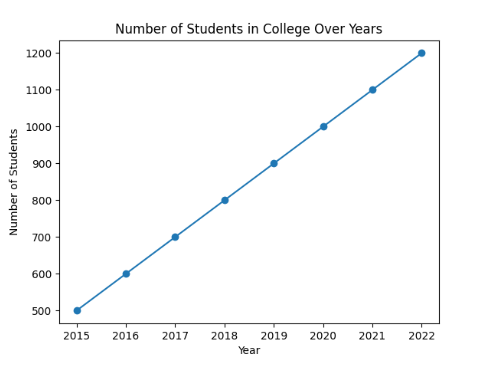
plt.ylabel('Number of Students')

plt.title('Number of Students in College Over Years')

**# Display the line graph**

plt.show()

**OUTPUT:**

****

In this example, we start by importing the `matplotlib.pyplot` module as `plt`. We define the sample data that represents the years and the corresponding number of students in the college. The `years` list contains the years, and the `student\_counts` list contains the respective number of students in each year. Next, we create a line graph using the `plt.plot()` function.

We pass the `years` as the first argument and `student\_counts` as the second argument to the function. The `marker='o'` parameter is used to display circular markers at each data point.To enhance the line graph, we add labels and a title using the `plt.xlabel()`, `plt.ylabel()`, and `plt.title()` functions, respectively.

Finally, we display the line graph using `plt.show()`. Running this code will generate a line graph where the x-axis represents the years, and the y-axis represents the number of students in the college. Each data point is connected by a line, and circular markers are displayed at each data point. You can further customize the appearance of the line graph by adjusting parameters such as line color, line style, marker type, and more using additional arguments in the `plt.plot()` function. Matplotlib offers various options to create visually appealing and informative line graphs.