

PYTHON LAB JOURNAL

AYUSH JAIN

COMPUTER ENGINEERING | TE – B2 | 60004200132

EXPERIMENT – 1

Aim: To study and implement different data types and operators in Python.

Theory:

1. Different Data Types:

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

Text Type: str

Numeric Types: int, float, complex

Sequence Types: list, tuple, range

Mapping Type: dict

Set Types: set

Boolean Type: bool

Binary Types: bytes, bytearray, memoryview

None Type: NoneType

Numeric

In Python, numeric data type represents the data which has numeric value. Numeric value can be integer, floating number or even complex numbers. These values are defined as int, float and complex class in Python.

- **Integers** – This value is represented by int class. It contains positive or negative whole numbers (without fraction or decimal). In Python there is no limit to how long an integer value can be.
- **Float** – This value is represented by float class. It is a real number with floating point representation. It is specified by a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation.
- **Complex Numbers** – Complex number is represented by complex class. It is specified as *(real part) + (imaginary part) j*.

Sequence Type

In Python, sequence is the ordered collection of similar or different data types.

Sequences allows to store multiple values in an organized and efficient fashion. There are several sequence types in Python –

- String

- List
- Tuple

1) **String**

In Python, Strings are arrays of bytes representing Unicode characters. A string is a collection of one or more characters put in a single quote, double-quote or triple quote. In python there is no character data type, a character is a string of length one. It is represented by str class.

Accessing elements of String

In Python, individual characters of a String can be accessed by using the method of Indexing. Indexing allows negative address references to access characters from the back of the String, e.g. -1 refers to the last character, -2 refers to the second last character and so on.

2) **List**

Lists are just like the arrays, declared in other languages which is a ordered collection of data. It is very flexible as the items in a list do not need to be of the same type.

Accessing elements of List

In order to access the list items, refer to the index number. Use the index operator [] to access an item in a list. In Python, negative sequence indexes represent positions from the end of the array. Instead of having to compute the offset as in List[len(List)-3], it is enough to just write List[-3]. Negative indexing means beginning from the end, -1 refers to the last item, -2 refers to the second-last item, etc.

3) **Tuple**

Just like list, tuple is also an ordered collection of Python objects. The only difference between tuple and list is that tuples are immutable i.e. tuples cannot be modified after it is created. It is represented by tuple class.

Accessing elements of Tuple

In order to access the tuple items, refer to the index number. Use the index operator [] to access an item in a tuple. The index must be an integer. Nested tuples are accessed using nested indexing.

Boolean

Data type with one of the two built-in values, True or False. Boolean objects that are equal to True are truthy (true), and those equal to False are false (false). But nonBoolean objects can be evaluated in Boolean context as well and determined to be true or false. It is denoted by the class bool.

Note – True and False with capital ‘T’ and ‘F’ are valid Booleans otherwise python will throw an error.

Set

In Python, [Set](#) is an unordered collection of data type that is iterable, mutable and has no duplicate elements. The order of elements in a set is undefined though it may consist of various elements.

Creating Sets

Sets can be created by using the built-in `set()` function with an iterable object or a sequence by placing the sequence inside curly braces, separated by 'comma'. Type of elements in a set need not be the same, various mixed-up data type values can also be passed to the set.

Accessing elements of Sets

Set items cannot be accessed by referring to an index, since sets are unordered the items has no index. But you can loop through the set items using a for loop, or ask if a specified value is present in a set, by using the `in` keyword.

Dictionary

Dictionary in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds key: value pair. Key-value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon (:), whereas each key is separated by a 'comma'.

Creating Dictionary

In Python, a Dictionary can be created by placing a sequence of elements within curly `{}` braces, separated by 'comma'. Values in a dictionary can be of any datatype and can be duplicated, whereas keys can't be repeated and must be immutable. Dictionary can also be created by the built-in function `dict()`. An empty dictionary can be created by just placing it to curly braces `{}`.

Accessing elements of Dictionary

In order to access the items of a dictionary refer to its key name. Key can be used inside square brackets. There is also a method called `get()` that will also help in accessing the element from a dictionary.

Binary Types

(bytes, bytearray, memoryview)

bytes and **bytearray** are used for manipulating binary data. The **memoryview** uses the buffer protocol to access the memory of other binary objects without needing to make a copy.

Bytes objects are **immutable** sequences of single bytes. We should use them only when working with ASCII compatible data.

The syntax for bytes literals is same as string literals, except that a 'b' prefix is added. bytearray objects are always created by calling the constructor `bytearray()`. These are **mutable** objects.

None

The None keyword is used to define a null value, or no value at all.

None is not the same as 0, False, or an empty string. None is a data type of its own (NoneType) and only None can be None.

2. Types of Operator:

Python language supports the following types of operators.

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Python Arithmetic Operators

Assume variable a hold 10 and variable b holds 20, then –

Operator	Description	Example
+ Addition	Adds values on either side of the operator.	$a + b = 30$
- Subtraction	Subtracts right hand operand from left hand operand.	$a - b = 10$
* Multiplication	Multiplies values on either side of the operator	$a * b = 200$
/ Division	Divides left hand operand by right hand operand	$b / a = 2$
% Modulus	Divides left hand operand by right hand operand and returns remainder	$b \% a = 0$

** Exponent	Performs exponential (power) calculation on operators	$a^{**}b$ = 10 to the power 20
//	Floor Division - The division of operands where the result is the quotient in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero (towards negative infinity) -	$9//2 = 4$ and $9.0//2.0 = 4.0$, $11//3 = -4$, $11.0//3 = -4.0$

Python Comparison Operators

These operators compare the values on either side of them and decide the relation among them. They are also called Relational operators.

Assume variable a hold 10 and variable b holds 20, then -

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	$(a == b)$ is not true.
!=	If values of two operands are not equal, then condition becomes true.	$(a != b)$ is true.

<>	If values of two operands are not equal, then condition becomes true.	(a <> b) is true. This is similar to != operator.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Python Assignment Operators

Assume variable a hold 10 and variable b holds 20, then

Operator	Description	Example

=	Assigns values from right side operands to left side operand	$c = a + b$ assigns value of $a + b$ into c
+= Add AND	It adds right operand to the left operand and assign the result to left operand	$c += a$ is equivalent to $c = c + a$
-= Subtract AND	It subtracts right operand from the left operand and assign the result to left operand	$c -= a$ is equivalent to $c = c - a$
*= Multiply AND	It multiplies right operand with the left operand and assign the result to left operand	$c *= a$ is equivalent
		to $c = c * a$
/= Divide AND	It divides left operand with the right operand and assign the result to left operand	$c /= a$ is equivalent to $c = c / a$ $c /= a$ is equivalent to $c = c / a$

<code>%=</code> Modulus AND	It takes modulus using two operands and assign the result to left operand	<code>c %= a</code> is equivalent to <code>c = c % a</code>
<code>**=</code> Exponent AND	Performs exponential (power) calculation on operators and assign value to the left operand	<code>c **= a</code> is equivalent to <code>c = c ** a</code>
<code>// =</code> Floor Division	It performs floor division on operators and assign value to the left operand	<code>c // = a</code> is equivalent to <code>c = c // a</code>

Python Bitwise Operators

Bitwise operator works on bits and performs bit by bit operation. Assume if `a = 60`; and `b = 13`;

Now in binary format they will be as follows – `a = 0011 1100` `b = 0000 1101`

`a&b = 0000 1100` `a|b = 0011 1101` `a^b= 0011 0001`

`~a = 1100 0011`

There are following Bitwise operators supported by Python language

Operator	Description	Example
<code>&</code> Binary AND	Operator copies a bit to the result if it exists in both operands	<code>(a & b)</code> (means 0000 1100)

Binary OR	It copies a bit if it exists in either operand.	(a b) = 61 (means 0011 1101)
^ Binary XOR	It copies the bit if it is set in one operand but not both.	(a ^ b) = 49 (means 0011 0001)
~ Binary Ones Complement	It is unary and has the effect of 'flipping' bits.	(~a) = -61 (means 1100 0011 in 2's complement form due to a signed binary number.
<< Binary Left Shift	The left operands value is moved left by the number of bits specified by the right operand.	a << 2 = 240 (means 1111 0000)
>> Binary Right Shift	The left operands value is moved right by the number of bits specified by the right operand.	a >> 2 = 15 (means 0000 1111)

Python Logical Operators

There are following logical operators supported by Python language. Assume variable a hold 10 and variable b holds 20 then

Operator	Description	Example
and Logical AND	If both the operands are true then condition becomes true.	(a and b) is true.
or Logical OR	If any of the two operands are non-zero then condition becomes true.	(a or b) is true.
not Logical NOT	Used to reverse the logical state of its operand.	Not(a and b) is false.

Python Membership Operators

Python's membership operators test for membership in a sequence, such as strings, lists, or tuples. There are two membership operators as explained below –

Operator	Description	Example
In	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	<code>x in y,</code> here in results in a 1 if x is a member of sequence y.

not in	Evaluates to true if it does not find a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.
--------	---	--

Python Identity Operators

Identity operators compare the memory locations of two objects. There are two Identity operators explained below –

Operator	Description	Example
Is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).
Is	Evaluates to true if the variables on either side of the operator point to the same object and false otherwise.	x is y, here is results in 1 if id(x) equals id(y).
is not	Evaluates to false if the variables on either side of the operator point to the same object and true otherwise.	x is not y, here is not results in 1 if id(x) is not equal to id(y).

Python Operators Precedence

The following table lists all operators from highest precedence to lowest.

Sr.No.	Operator & Description
1	** Exponentiation (raise to the power)
2	~ + - Complement, unary plus and minus (method names for the last two are +@ and - @)

3	<code>* / % //</code> Multiply, divide, modulo and floor division
4	<code>+ -</code> Addition and subtraction
5	<code>>> <<</code> Right and left bitwise shift
6	<code>&</code> Bitwise 'AND'
7	<code>^ </code> Bitwise exclusive 'OR' and regular 'OR'
8	<code><= < > >=</code> Comparison operators
9	<code><> == !=</code> Equality operators
10	<code>= %= /= //= -= += *= **=</code> Assignment operators
11	Is, is not Identity operators
12	In, not in Membership operators
13	not, or, and Logical operators

Implementation and Output:

• Data Types:

```
x = "Hello World" print(x)
print(type(x)) x = 1
print(x) print(type(x)) x =
10.5 print(x)
print(type(x)) x = 10j
print(x) print(type(x)) x =
["hello", "world"] print(x)
print(type(x)) x = range(0,
5) print(x) print(type(x))
x = (1, 2, 3) print(x)
print(type(x)) x = {"id":
5, "name": "ron"} print(x)
print(type(x)) x = {"DSA",
"DBMS"} print(x)
print(type(x)) x = True
print(x) print(type(x)) x =
b"python" print(x)
print(type(x)) x = None
print(x) print(type(x))
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  JUPYTER
[Running] python -u "c:\Users\djsce.student\Desktop\44_py\data_types.py"
Hello World
<class 'str'>
1
<class 'int'>
10.5
<class 'float'>
10j
<class 'complex'>
['hello', 'world']
<class 'list'>
range(0, 5)
<class 'range'>
(1, 2, 3)
<class 'tuple'>
{'id': 5, 'name': 'ron'}
<class 'dict'>
{'DBMS', 'DSA'}
<class 'set'>
True
<class 'bool'>
b'python'
<class 'bytes'>
None
<class 'NoneType'>

[Done] exited with code=0 in 0.613 seconds
```

Operators:

```
# Arithmetic Operators
print("Arithmetic Operators")
x=3 y=2 print(x+y)
print(x-y) print(x*y)
print(x/y) print(x%y)
print(x//y) print(x**y)
print("\n")

# Assignment Operators
print("Assignment Operators")
x=3 x+=2 print(x) x-=2
print(x)
```

```
x*=2 print(x)
x/=2 print(x)
x%=2 print(x)
x//=2
print(x)
x**=2
print(x)
print("\n")

# Comparison Operators
print("Comparison Operators")
x=4 y=2 print(x==y)
print(x!=y) print(x<y)
print(x>y) print(x<=y)
print(x>=y) print("\n")

# Logical Operators
print("Logical Operators")
x=4 y=2 print(x>2 and
y<2) print(x>2 or y<2)
print(not(y<2))
print("\n")

# Identity Operators
print("Identity Operators")
x=5 y=2 print(x is y)
print(x is not y)
print("\n")

# Membership Operators
print("Membership Operators")
x=["DSA", "DBMS", "OS"]
print("DSA" in x) print("DSA"
not in x) print("\n")

# Bitwise Operators
print("Bitwise Operators") x=4
```



```
y=6 print(x&y)
print(x|y)
print(x^y)
print(~y)
print(y<<2)
print(y>>1)
print("\n")
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp1\operators.py"
Arithmetic Operators
5
1
6
1.5
1
1
9

Assignment Operators
5
3
6
3.0
1.0
0.0
0.0

Comparison Operators
False
True
False
True
False
True

Logical Operators
False
True
True
```

```
Identity Operators
False
True
```

```
Membership Operators
True
False
```

```
Bitwise Operators
4
6
2
-7
24
3
```

```
PS C:\Users\USER> 
```

Conclusion:

By studying various data types in python, we came to know that in python data types are classified in 8 types, comprising of Text Type, Numeric Types, Sequence Types, Mapping Type, Set Types, Boolean Type, Binary Types and None Type.

By studying various operators in python, we found that python lacks increment operators but in addition has other operators.

EXPERIMENT 2

Aim: To study and implement different types of input-output and control statements in python.

Theory:

➤ input-output

1. input ()

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

2. print ()

The print () function prints the specified message to the screen, or other standard output device. The message can be a string, or any other object, the object will be converted into a string before written to the screen.

3. append ()

Python's append() function inserts a single element into an existing list. The element will be added to the end of the old list rather than being returned to a new list. Adds its argument as a single element to the end of a list. The length of the list increases by one.

4. add ()

The Python set add () method adds a given element to a set if the element is not present in the set in Python.

• Implementation:

1. Find square root of number

```
num = float(input("Enter Number: "))  
print(f"Square root: {num**0.5}")
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"  
Enter Number: 6.25  
Square root: 2.5  
PS C:\Users\USER> 
```

2. Find area and perimeter of rectangle

```
length = float(input("Enter Length: ")) breadth
= float(input("Enter Breadth: ")) print(f"Area
of reactangle: {length*breadth}")
print(f"Perimenter of reactangle:
{(length+breadth)*2}")
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter Length: 5.5
Enter Breadth: 3.5
Area of reactangle: 19.25
Perimenter of reactangle: 18.0
PS C:\Users\USER> █
```

3. Swapping of two numbers with and without using third variable

```
num1 = int(input("Enter number 1: "))
num2 = int(input("Enter number 2: "))
num1+=num2 num2=num1-num2 num1-
=num2 print("Num1:",num1)
print("Num2:",num2)
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number 1: 5
Enter number 2: 9
Number 1: 9
Number 2: 5
PS C:\Users\USER> █
```

4. Adding elements in List, Set and Tuple

```
List = list() l = int(input("\nEnter
Size Of List: ")) print("Enter
Elements") for i in range(l):
    List.append(int(input()))
print(List)

Set = set()
l = int(input("\nEnter Size Of Set: "))
print("Enter Elements:") for i in
range(l): Set.add(int(input()))
print(Set)

Tuple = (1,2,3)
```

```
print("\nTuple before adding new elements:",Tuple) L=list(Tuple)
L.append(int(input("Enter new Element:")))
T = tuple(L) print("Tuple after adding new
elements:",T)
```

Output:

```
Enter Size Of List: 4
Enter Elements
8
6
4
2
[8, 6, 4, 2]

Enter Size Of Set: 3
Enter Elements:
3
5
7
{3, 5, 7}

Tuple before adding new elements: (1, 2, 3)
Enter new Element:4
Tuple after adding new elements: (1, 2, 3, 4)
PS C:\Users\USER>
```

➤ Control statements and Loops

1. for Loop

A for loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

This is less like the for keyword in other programming languages, and works more like an iterator method as found in other object-orientated programming languages.

With the for loop we can execute a set of statements, once for each item in a list, tuple, set etc.

2. while Loop

Python While Loop is used to execute a block of statements repeatedly until a given condition is satisfied. And when the condition becomes false, the line immediately after the loop in the program is executed.

While loop falls under the category of **indefinite iteration**. Indefinite iteration means that the number of times the loop is executed isn't specified explicitly in advance.

3. if, if-else, nested if, if-elif statement

• if Statement

If the simple code of block is to be performed if the condition holds true then if statement is used. Here the condition mentioned holds true then the code of block runs otherwise not.

- **if. else Statement**

In conditional if Statement the additional block of code is merged as else statement which is performed when if condition is false. Nested if Statement if statement can also be checked inside other if statement. This conditional statement is called a nested if statement. This means that inner if condition will be checked only if outer if condition is true and by this, we can see multiple conditions to be satisfied.

- **if-elif Statement**

The if-elif statement is shortcut of if..else chain. While using if-elif statement at the end else block is added which is performed if none of the above if-elif statement is true.

- **Implementation:**

1. Print the given triangle pattern using for loop

```
for i in range(1,5):  
    for _ in range(i):  
        print(i,end=" ")  
    print()
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"  
1  
2 2  
3 3 3  
4 4 4 4  
PS C:\Users\USER> []
```

2. Find the factorial of a number using for loop

```
num = int(input("Enter number: "))  
s=1 for i in range(1,num+1):  
    s*=i print(s)
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number: 5
120
PS C:\Users\USER> █
```

3. Print the Fibonacci sequence up to given 'n' value using for loop

```
num = int(input("Enter number: "))
sum = 0 sum2 = 1 temp = 0 if (num
== 0): print("0") else:
print("0 1", end=" ") for
i in range(num-1):
print(sum2+sum, end=" ")
temp = sum sum = sum2
sum2 = sum2+temp
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number: 6
0 1 1 2 3 5 8
PS C:\Users\USER> █
```

4. Demonstrate 'while' loop with one example

```
count=0 while
(count<10):
print(count,end=" ")
count+=1
```

Output:

```
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
0 1 2 3 4 5 6 7 8 9
PS C:\Users\USER> █
```

5. Check whether the input number is even or odd using if...else loop

```
num = int(input("Enter number: ")) if
num%2==1:
print("Number is Odd") else:
print('Number is Even')
```

Output:

```

PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number: 8
Number is Even
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number: 21
Number is Odd
PS C:\Users\USER>

```

6. Check whether the input year is leap year or not using nested if

```

year = int(input("Enter number: "))
if year%4==0:
    if year%100==0:
        if year%400==0:
            print("Leap Year")
        else:
            print("Not a Leap Year")
    else:
        print("Leap Year")
else:
    print("Not a Leap Year")

```

Output:

```

PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number: 2020
Leap Year
PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Enter number: 1800
Not a Leap Year
PS C:\Users\USER>

```

7. Demonstrate 'if...elif...else' loop with one example

```

num=4+5j if(type(num)==int):
    print("num is int")
elif(type(num)==float):
    print("num is float")
else:
    print("num is complex")

```

Output:

```

PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\tempCodeRunnerFile.py"
num is complex
PS C:\Users\USER>

```

8. Demonstrate 'continue', 'break' and 'pass' with one example each


```

print("Using continue at 'o' or 'd': ")
for letter in "knowledge":    if
letter == 'o' or letter == 'd':
    continue    print(letter, end="
")    print("\nUsing break at 'o' or 'd':
") for letter in "knowledge":    if
letter == 'o' or letter == 'd':
    break    print(letter, end=" ")
print("\nUsing pass at 'o' or 'd': ") for
letter in "knowledge":    if letter ==
'o' or letter == 'd':
    pass
print(letter, end=" ")

```

Output:

```

PS C:\Users\USER> python -u "d:\college files\Sem 5\Python_programming\Exp2\control.py"
Using continue at 'o' or 'd':
k n w l e g e
Using break at 'o' or 'd':
k n
Using pass at 'o' or 'd':
k n o w l e d g e
PS C:\Users\USER>

```

Conclusion:

Firstly, we learned about different methods to take input and output in python.

There are different types of control statements in Python - break, continue and pass. Different control statements have different functions and can be used according to the need in the program.

We learned about Conditional Statements in Python. These are the statements that alter the control flow of execution in the program. We have different types of conditional statements like if, if-else, elif, nested if, and nested if-else statements which control the execution of our program.

EXPERIMENT 3

Aim: To study and implement different types of functions in Python.

Theory:

Python Functions

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

A function can return data as a result.

Creating a Function

In Python a function is defined using the **def** keyword.

Calling a Function

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

Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

Python Lambda

A lambda function is a small anonymous function.

A lambda function can take any number of arguments, but can only have one expression.

Syntax: *lambda arguments: expression*

Implementation:

A) Mention and explain all the functions with example for following

data structures:

- **LISTS**
- `append ()`: - used for appending and adding items to the end of the list
- `copy ()`: -it returns a shallow copy of the list
- `clear ()`: -this method is used to remove all items from the list
- `count()`:this returns the number of element in the list
- `insert()`:-inserts an element in the given index in a list
- `pop()`:-removes and returns last value from list
- `sort()`:-sorts in ascending or descending order

```
l=['ADBMS',2002, True, {"Python":"Exp3"}] print(l)
l.append('I am a list')
print(l) m=l.copy()
print(m)
m.clear() print(m)
l.insert(5, 'Inserted') print(l)
l.pop() print(l)
```

Output:

```
['ADBMS', 2002, True, {'Python': 'Exp3'}]
['ADBMS', 2002, True, {'Python': 'Exp3'}, 'I am a list']
['ADBMS', 2002, True, {'Python': 'Exp3'}, 'I am a list']
[]
['ADBMS', 2002, True, {'Python': 'Exp3'}, 'I am a list', 'Inserted']
['ADBMS', 2002, True, {'Python': 'Exp3'}, 'I am a list']
```

- **TUPLES** count (): Returns the number of times a value occurs in a tuple
- index (): Searches the tuple for a specified value and returns its index

```
l=(2001,2002,2003,2002)
print(l)
print(l.count(2002))
print(l.index(2001))
```

Output:

```
(2001, 2002, 2003, 2002)
2
0
```

- **SETS** add (): Adds a given element to a set
- clear (): Removes all elements from the set
- discard (): Removes the element from the set
- pop (): Returns and removes a random element from the set
- remove (): Removes the element from the set

```
s={'p','y','t','h','o'} print(s)
s.add('n')
print(s)
s.discard('o')
print(s) s.pop()
```

```
s.pop() print(s)
s.remove('h')
print(s)
s.clear() print(s)
```

Output:

```
{'o', 't', 'h', 'y', 'p'}
{'o', 'n', 't', 'h', 'y', 'p'}
{'n', 't', 'h', 'y', 'p'}
{'t', 'h', 'y', 'p'}
{'t', 'y', 'p'}
set()
```

· DICTIONARIES

clear (): removes all items copy (): creates a copy get (): returns the value for the given key items (): return the list with all dictionary keys with values keys (): returns a view object that displays all keys pop (): returns and removes the element with given key

```
d={1:'ADBMS', 2:'DMW'}
e=d.copy() print(e)
e.clear() print(e)
print(d.get(1))
print(d.items())
print(d.keys())
d.pop(2) print(d)
```

Output:

```
{1: 'ADBMS', 2: 'DMW'}
{}
ADBMS
dict_items([(1, 'ADBMS'), (2, 'DMW')])
dict_keys([1, 2])
{1: 'ADBMS'}
```

B) Write a Python function histogram(l) that takes as input a list of integers with repetitions and returns a list of pairs:

```
def histogram(L):
    k = {}
    m = []
    L.sort()
    for j in L:
        x = L.count(j)
        k[j] = x
    for p in k:
        m.append((p, k[p]))
    m = sorted(m, key=lambda a: a[1])
    return m
l = list(int(item) for item in input("Enter Elements in list: ").split())
print(histogram(l))
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\
Enter Elements in list: 1 5 5 4 8 9 5 8
[(1, 1), (4, 1), (9, 1), (8, 2), (5, 3)]
PS D:\college files\Sem 5\Python_programming\My_code> █
```

C) Write a Python function perfect(n) that takes a positive integer argument and returns True if the integer is perfect, and False otherwise.

```
def num_perfect(num):
    sum_n = 0
    for i in range(1, num):
        if num % i == 0:
            sum_n = sum_n + i
    return sum_n == num
print(num_perfect(int(input("Enter a Number: "))))
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\
Enter a Number: 28
True
PS D:\college files\Sem 5\Python_programming\My_code> █
```

D) Implement a recursive function to solve tower of Hanoi Problem:

```
def hanoi(disks,source, auxiliary, target):
if disks==1:
    print(f"Move disk 1 from peg {source} to peg {target}.")
return hanoi(disks-1,source,target,auxiliary) print(f"Move
disk {disks} from peg {source} to peg {target}.")
hanoi(disks-1,auxiliary,source,target)
    disks = int(input("Enter number of disks:
")) hanoi(disks,'A','B','C')
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college fi
Enter number of disks: 4
Move disk 1 from peg A to peg B.
Move disk 2 from peg A to peg C.
Move disk 1 from peg B to peg C.
Move disk 3 from peg A to peg B.
Move disk 1 from peg C to peg A.
Move disk 2 from peg C to peg B.
Move disk 1 from peg A to peg B.
Move disk 4 from peg A to peg C.
Move disk 1 from peg B to peg C.
Move disk 2 from peg B to peg A.
Move disk 1 from peg C to peg A.
Move disk 3 from peg B to peg C.
Move disk 1 from peg A to peg B.
Move disk 2 from peg A to peg C.
Move disk 1 from peg B to peg C.
PS D:\college files\Sem 5\Python_programming\My_code> █
```

E) Implement lambda function to find greater of the 2 input numbers

```
a=int(input("Enter a number: "))
b=int(input("Enter a number: ")) maxi =
lambda a,b: a if a>b else b
print(f"{maxi(a,b)} is maximum number.")
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\co
Enter a number: 8
Enter a number: 2
8 is maximum number.
PS D:\college files\Sem 5\Python_programming\My_code> █
```

F) Using map function perform element wise addition of elements of two lists.

```
def addition(n,m):
```

```

    return n + m
n=list(int(item) for item in
input("Enter Elements in list 1: ").split())
m=list(int(item) for item in input("Enter
Elements in list 2: ").split())
result =
map(addition, n,m)
print(list(result))

```

Output:

```

PS D:\college files\Sem 5\Python_programming\My_code> python -u '
Enter Elements in list 1: 1 2 3 4
Enter Elements in list 2: 5 6 7 8
[6, 8, 10, 12]
PS D:\college files\Sem 5\Python_programming\My_code> 

```

G) Using map and filter find the cube of all the odd numbers from the given input list

```

arr=[]
n=int(input("Enter Number of
elements: "))
print("Enter List
Elements:")
for i in range(n):
    arr.append(int(input()))
print(arr)
print("The cube of all odd numbers in list:")
arr2=list(map(lambda x: x**3, filter(lambda y:y%2==1,arr)))
print(arr2)

```

Output:

```

PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5
Enter Number of elements: 4
Enter List Elements:
1
2
3
4
[1, 2, 3, 4]
The cube of all odd numbers in list:
[1, 27]
PS D:\college files\Sem 5\Python_programming\My_code> 

```

Conclusion:

We studied how a function is declared and called in python. We studied different in-built python functions as well as we created different UserDefined functions in this experiment. Lambda function is a Special type of function used as small anonymous function.

EXPERIMENT 4

Aim: To study and implement concepts of classes and Inheritance in Python.

Theory:

A) Explain Classes and Objects with suitable examples:

A class is a user-defined blueprint or prototype from which objects are created. Classes provide a means of bundling data and functionality together. Creating a new class creates a new type of object, allowing new instances of that type to be made. Each class instance can have attributes attached to it for maintaining its state. Class instances can also have methods (defined by their class) for modifying their state.

An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with *actual values*. It's not an idea anymore, it's an actual dog, like a dog of breed pug who's seven years old. You can have many dogs to create many different instances, but without the class as a guide, you would be lost, not knowing what information is required. An object consists of:

- **State:** It is represented by the attributes of an object. It also reflects the properties of an object.
- **Behaviour:** It is represented by the methods of an object. It also reflects the response of an object to other objects.
- **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

```
class Dog:
    attr1 = "mammal"
attr2 = "dog"
def fun(self):
    print("I'm a", self.attr1)
print("I'm a", self.attr2)

Rodger = Dog()
print(Rodger.attr1)
Rodger.fun()
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_p
mammal
I'm a mammal
I'm a dog
PS D:\college files\Sem 5\Python_programming\My_code>
```


B) Explain use of __init__() function in class with suitable example.

The __init__ method is similar to constructors in C++ and Java. Constructors are used to initialize the object's state. Like methods, a constructor also contains a collection of statements (i.e. instructions) that are executed at the time of Object creation. It runs as soon as an object of a class is instantiated. The method is useful to do any initialization you want to do with your object.

```
class person:
    def __init__(self,age):
        self.age=age
c = person(28)
print("Person has attribut age which is equal to "+str(c.age))
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code\person.py"
Person has attribut age which is equal to 28
PS D:\college files\Sem 5\Python_programming\My_code>
```

C) Code for Object Methods, Modifying Object Properties and Deleting Objects:

```
class person:
    def __init__(self,name,age):
        self.name=name
        self.age=age
    def my_fun(self):
        print("Hello "+self.name)
print("Your age is ",self.age)

p1=person('Bhavik',21)
p1.my_fun()
p1.age=22
print(p1.age)
del p1
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code\person.py"
Hello Bhavik
Your age is 21
22
```

D) Python Inheritance with suitable examples:

Inheritance is an important aspect of the object-oriented paradigm. Inheritance provides code reusability to the program because we can use an existing class to create a new class instead of creating it from scratch.

In inheritance, the child class acquires the properties and can access all the data members and functions defined in the parent class. A child class can also provide its specific implementation to the functions of

the parent class. In this section of the tutorial, we will discuss inheritance in detail.

In python, a derived class can inherit base class by just mentioning the base in the bracket after the derived class name. Consider the following syntax to inherit a base class into the derived class.

```
class Person:
    def __init__(self, fname, lname):
        self.fname = fname
        self.lname = lname
    def printname(self):
        print(f"My name is {self.fname} {self.lname}")
class student(Person):
    def __init__(self, fname, lname):
        Person.__init__(self, fname, lname)
x = student("Bhavik", "Shah")
x.printname()
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python
My name is Bhavik Shah
```

E) Write python code to implement the following inheritance example:

Classes: Employee, Developer, Tester, Manager

Developer, tester, Manager inherit Employee

Manager handles Developer, tester

Manager class: implement functions to add Developer/Tester and Remove Developer/ Tester

Display to see the list of employees he manages

```
class Employee:
    def __init__(self, name, ids):
        self.name = name
self.id = ids
class
Manager(Employee):
    def __init__(self, name, ids, my_id):
        super().__init__(name, ids)
self.manager_id = my_id
self.developer = []
self.testers = []
```

```

    def addDev(self, id):        if
self.developer.count(id) == 0:
    self.developer.append(id)
else:
    print("Already an Developer")
def removeDev(self, id):        if
self.developer.count(id) > 1:
    self.developer.remove(id)
else:
    print("No such Developer")
def addTest(self, id):        if
self.testers.count(id) == 0:
    self.testers.append(id)
else:
    print("Already an Tester")
def removeTesv(self, id):        if
self.testers.count(id) == 0:
    self.testers.append(id)
else:
    print("No such Tester")
class Developer(Employee):    def
__init__(self, name, ids, my_id):
super().__init__(name, ids)
self.developer_id = my_id
class Tester(Employee):        def
__init__(self, name, ids, my_id):
super().__init__(name, ids)
self.tester_id = my_id
m1 = Manager("Bhavik", 101,
1) m2 = Manager("Ram", 102,
2) d1 = Developer("Raj", 103,
1) d2 = Developer("Yash", 104,
2) t1 = Tester("Rohan", 105,
1) m1.addDev(103)
m1.addTest(105) m2.addDev(104)
print(f"Manager with Employee_id {m1.id} manages:")
print(f"Developers: {m1.developer}")
print(f"Tester: {m1.testers}") print(f"Manager with
Employee_id {m2.id} manages:") print(f"Developers:
{m2.developer}") print(f"Tester: {m2.testers}")

```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\
Manager with Employee_id 101 manages:
Developers: [103]
Tester: [105]
Manager with Employee_id 102 manages:
Developers: [104]
Tester: []
PS D:\college files\Sem 5\Python_programming\My_code>
```

Conclusion: In this experiment we learnt about how the concept of classes and objects work in python. How we can create object of a given class and change its attributes, also we learnt about how constructor works in python. And lastly how by inheritance we can transfer properties of one class to another.

EXPERIMENT 5

Aim: To study and implement concepts of exception handling in python.

Theory:

Exceptions are raised when the program is syntactically correct, but the code resulted in an error. This error does not stop the execution of the program. However, it changes the normal flow of the program. Try and except statements are used to catch and handle exceptions in Python. Statements that can raise exceptions are kept inside the try clause and the statements that handle the exception are written inside except clause. You can also use the else clause on the try-except block. The code enters the else block only if the try clause does not raise an exception. Python provides a keyword finally, which is always executed after the try and except blocks. The final block always executes after normal termination of try block or after try block terminates due to some exception.

A) Display and handle at least 5 in-build exceptions:

1) Arithmetic errors:

- a. **Zero division error:** ZeroDivisionError is a built-in Python exception thrown when a number is divided by 0. This means that the exception raised when the second argument of a division or modulo operation is zero.
- b. **Overflow error:** An OverflowError exception is raised when an arithmetic operation exceeds the limits to be represented.

```
try:
    n1,n2=eval(input("Enter 2 numbers separeted by commas: "))
    result=n1/n2      print(f"Result: {result}") except
ZeroDivisionError:
    print("Division by 0 not possible") else:
    print("No exception") finally:
    print("Finally everything Executed")
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_
Enter 2 numbers separeted by commas: 5,0
Division by 0 not possible
Finally everything Executed
PS D:\college files\Sem 5\Python_programming\My_code> |
```

2) Assertion error:

In Python, assert is a simple statement with the following syntax:

assert condition, error_message(optional)

Here, expression can be any valid Python expression or object, which is then tested for truthiness. If expression is false, then the statement throws an AssertionError.

```
try:
    import sys    assert('linux'
in sys.platform) except:
    print("Assertion Error
Occured")
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code.py"
Assertion Error Occured
PS D:\college files\Sem 5\Python_programming\My_code> []
```

3) Value error:

The Python ValueError is an exception that occurs when a function receives an argument of the correct data type but an inappropriate value. This error usually occurs in mathematical operations that require a certain kind of value.

```
try:
    a =int(input("Enter value: ")) except:
    print("Value Error: Enter integer
only")
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code.py"
Enter value: bhavik
Value Error: Enter integer only
PS D:\college files\Sem 5\Python_programming\My_code> []
```

4) Key error:

KeyError is raised when a mapping key is accessed and isn't found in the mapping. A mapping is a data structure that maps one set of values to another. The most common mapping in Python is the dictionary.

The Python KeyError is a type of ~~Lookup~~ Lookup Error exception and denotes that there was an issue retrieving the key you were looking for. When you see a KeyError, the semantic meaning is that the key being looked for could not be found.

```
a = {5:"abcc",7:"sdf"} print(a[3])
```

Output:

```
Traceback (most recent call last):
  File "d:\college files\Sem 5\Python_programming\My_code\exception.py", line 10, in <module>
    print(a[3])
KeyError: 3
PS D:\college files\Sem 5\Python_programming\My_code> []
```

5) Index error:

An IndexError means that your code is trying to access an index that is invalid. This is usually because the index goes out of bounds by being too large.

For example, if you have a list with three items and you try to access the fourth item, you will get an IndexError.

This can happen with strings, tuples, lists, and generally any object that is indexable.

```
a = [1, 2, 3]
try:
    print (f"Second element = {a[1]}")
print (f"Fourth element = {a[3]}") except:
    print (f"An error occurred")
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code\exception.py"
Second element = 2
An error occurred
PS D:\college files\Sem 5\Python_programming\My_code>
```

B) Create a user defined exception handling mechanism:

we created a new exception class i.e. User Error. Exceptions need to be derived from the built-in Exception class, either directly or indirectly. Let's look at the given example which contains a constructor and display method within the given class.

```
class LowerNumber(Exception):
    def __str__(self):
        return "Error: Number is Lower"
```

```

class GreaterNumber(Exception):
def __str__(self):
    return "Error: Number is Greater"
def
NumGame(n):
    x = int(input("Enter num: "))
try:
    if x < n:
        raise LowerNumber
elif x > n:
    raise GreaterNumber
except Exception as e:
    print(e)
NumGame(n)
else:
    print("You win")

NumGame(20)

```

Output:

```

PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code\except
Enter num: 15
Error: Number is Lower
Enter num: 25
Error: Number is Greater
Enter num: 20
You win
PS D:\college files\Sem 5\Python_programming\My_code>

```

Conclusion:

In this experiment we learnt about how the concept of exception handling, there are 2 types of exceptions one is predefined exception and other are user defined exception. We use try-except block to handle this exception.

EXPERIMENT 6

Aim: To study and implement concepts of file handling in Python.

Theory:

Python too supports file handling and allows users to handle files i.e., to read and write files, along with many other file handling options, to operate on files. The concept of file handling has stretched over various other languages, but the implementation is either complicated or lengthy, but like other concepts of Python, this concept here is also easy and short. Python treats files differently as text or binary and this is important. Each line of code includes a sequence of characters and they form a text file. Each line of a file is terminated with a special character, called the EOL or End of Line characters like comma {,} or newline character. It ends the current line and tells the interpreter a new one has begun.

Working of open () function

Before performing any operation on the file like reading or writing, first, we have to open that file. For this, we should use Python's inbuilt function open () but at the time of opening, we have to specify the mode, which represents the purpose of the opening file.

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

Where the following mode is supported:

1. **r:** open an existing file for a read operation.
2. **w:** open an existing file for a write operation. If the file already contains some data then it will be overridden but if the file is not present then it creates the file as well.
3. **a:** open an existing file for append operation. It won't override existing data.
4. **r+:** To read and write data into the file. The previous data in the file will be overridden.
5. **w+:** To write and read data. It will override existing data.
6. **a+:** To append and read data from the file. It won't override existing data.

Working of read () mode

To read a file in Python, we must open the file in reading r mode.

There are various methods available for this purpose. We can use the read(size) method to read in the size number of data. If the size

parameter is not specified, it reads and returns up to the end of the file.

Creating a file using write () mode

In order to write into a file in Python, we need to open it in write w, append a or exclusive creation x mode.

We need to be careful with the w mode, as it will overwrite into the file if it already exists. Due to this, all the previous data are erased.

Writing a string or sequence of bytes (for binary files) is done using the write () method. This method returns the number of characters written to the file.

A) Take 10 numbers from the user. Add it to a file (let's say T1.txt). Read the contents of the file and sort the data. Put the sorted data in a different file (T2.txt):

```
f=open("File1_py.txt","w") text = input("Enter
integers (seperated by spaces): ") f.write(text)
f.close() f =
open("File1_py.txt") nums
= f.read() all_nums = []
nums=nums.split(' ') for i
in nums:
all_nums.append(int(i))
all_nums.sort()
f.close()
f=open("File2_py.txt","w") for
i in all_nums:
    f.write(str(i)+" ")
f.close()
```

Output:

1) Terminal

```
PS D:\college files\Sem 5\Python_programming\My_code> python -u "d:\college files\Sem 5\Python_programming\My_code\
s.py"
Enter integers (seperated by spaces): 12 1 15 2 30 45 87 4 63 99
PS D:\college files\Sem 5\Python_programming\My_code> []
```

2) File 1

```
File1_py.txt X File2_py.txt
File1_py.txt
1 12 1 15 2 30 45 87 4 63 99
```

3) File 2

```
File1_py.txt File2_py.txt X
File2_py.txt
1 1 2 4 12 15 30 45 63 87 99 |
```

B) Take a T1.txt file with several words in it. Sort the words lexicographically and put the sorted words in a different file T2.txt.

```
fileRead = open("File1_py.txt", "r")
List = list() f=fileRead.read()
f=f.split(" ") for line in f:
    word = str(line)
    List.append(word)
List.sort()
List.sort(key=len)
fileWrite = open("File2_py.txt",
"w") for i in range(len(List)):
    fileWrite.write(str(List[i]))
fileWrite.write(" ")
```

Output:

1) File 1

```
File1_py.txt X File2_py.txt files.py
File1_py.txt
1 how is the rat not caught yet
```

2) File 2

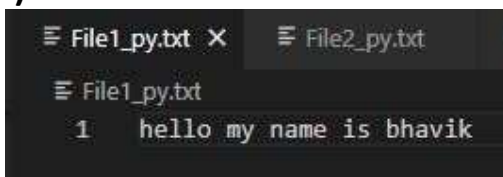
```
File1_py.txt File2_py.txt X files.py
File2_py.txt
1 is how not rat the yet caught |
```

C) Take a T1.txt file with several words in it. Reverse each word and put the reversed words in order in a different file T2.txt.

```
fileRead = open("File1_py.txt", "r")
List = list() f = fileRead.read() f
= f.split(" ") for line in f:
    List.append(str(line[::-1]))
fileWrite = open("File2_py.txt", "w")
for i in range(len(List)):
    fileWrite.write(str(List[i]))
fileWrite.write(" ")
```

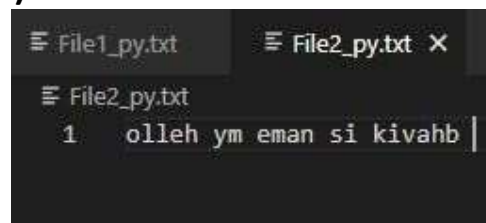
Output:

1) File 1



The screenshot shows a code editor with two tabs: 'File1_py.txt' and 'File2_py.txt'. The 'File1_py.txt' tab is active, displaying the text '1 hello my name is bhavik'.

2) File 2



The screenshot shows a code editor with two tabs: 'File1_py.txt' and 'File2_py.txt'. The 'File2_py.txt' tab is active, displaying the text '1 olleh ym eman si kivahb'.

Conclusion:

In this experiment we learnt about how the concept of file handling in python. How we can read data from a required file and also write or append the data, we want to add in a file.

EXPERIMENT 7

Aim: To study and implement concepts of Regular Expression in Python.

Theory:

A **Regular Expressions (RegEx)** is a special sequence of characters that uses a search pattern to find a string or set of strings. It can detect the presence or absence of a text by matching it with a particular pattern, and also can split a pattern into one or more subpatterns. Python provides a **re** module that supports the use of regex in Python. Its primary function is to offer a search, where it takes a regular expression and a string. Here, it either returns the first match or else none. **MetaCharacters**

To understand the RE analogy, MetaCharacters are useful, important, and will be used in functions of module re. Below is the list of metacharacters.

MetaCharacters Description

\	Used to drop the special meaning of character following it
[]	Represent a character class
^	Matches the beginning
\$	Matches the end
.	Matches any character except newline
	Means OR (Matches with any of the characters separated by it.
?	Matches zero or one occurrence

MetaCharacters Description

*	Any number of occurrences (including 0 occurrences)
+	One or more occurrences

{ } Indicate the number of occurrences of a preceding regex to match.

()	Enclose a group of Regex
----	--------------------------

Q) Use regular expression for the given text to find:

- Names of the User.
- Website name excluding http/s
- Identify email ids
- Identify Phone numbers

Considering a Text File consisting of following data:

Mr. Anderson

Ms. Thareja

Mrs. Morris

Mr. Roy

Ms. Gandhi Mrs. Modi https://www.google.com

http://www.udemy.com www.udacity.com

https://www.stackoverflow.com http://www.djsce.ac.in

https://plus.google.com rishit.grover@gmail.com

kapeesh.grover@yahoo.co.in abhishek.shah@gmail.com

shahp98@gmail.com demo_user@gmail.com rolflmoa@yahoo.co.in

27777647

233*333*88

455-78-888

022-240-93836

02642*221*381

A) Names:

```
import re
file = open('sample.txt', 'r')
text = file.read()
pattern=r'M\w*\.\s*\w*'
names=re.findall(pattern,text)
for i in names:
    print(i)
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> & E:/PythonCOMI
/regex.py"
Mr. Anderson
Ms. Thareja
Mrs. Morris
Mr. Roy
Ms. Gandhi
Mrs. Modi
PS D:\college files\Sem 5\Python_programming\My_code> []
```

B) Emails:

```
import re
file = open('sample.txt', 'r')
text = file.read()
pattern=r'\b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Z|a-z]{2,}\b'
emails=re.findall(pattern,text)
for i in emails:
    print(i)
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> & E:/PythonCOMP/python.exe "/regex.py"
rishit.grover@gmail.com
kapeesh.grover@yahoo.co.in
abhishek.shah@gmail.com
shahp98@gmail.com
demo_user@gmail.com
rolf1moa@yahoo.co.in
PS D:\college files\Sem 5\Python_programming\My_code> █
```

C) Websites:

```
import re
file = open('sample.txt', 'r')
text = file.readlines()
pattern=r'\b(www|plus)[.][a-zA-Z0-9]+([.])com\b|([.])ac[.])in\b|([.])co[.])in\b)'
urls=re.finditer(pattern,str(text))
for i in urls:
    print(i.group())
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> & E:/PythonCOMP/python.exe "/regex.py"
www.google.com
www.udemy.com
www.udacity.com
www.stackoverflow.com
www.djsce.ac.in
plus.google.com
PS D:\college files\Sem 5\Python_programming\My_code> █
```

D) Phone Numbers:

```
import re
file = open('sample.txt', 'r')
text = file.read()
pattern=r'\d{8}|\d{3,5}.\d{2,4}.\d{2,6}'
nums=re.findall(pattern,text)
for i in nums:
    print(i)
```

Output:

```
PS D:\college files\Sem 5\Python_programming\My_code> & E:/PythonCOMP/python.exe "d:/col/regex.py"
27777647
233*333*88
455-78-888
022-240-93836
02642*221*381
PS D:\college files\Sem 5\Python_programming\My_code> █
```

Conclusion:

In this experiment we learnt about how the concept of regular expression and how it works in python. How we can write raw string and use it as regular expression. We also learnt how to make a regular expression with the keywords defined for it.

EXPERIMENT 8

Aim: To implement connection with MySQL database using python.

Theory:

How to Connect to MySQL Database in Python?

1. Install MySQL connector module

Use the pip command to install MySQL connector Python.

pip install mysql-connector-python

2. Import MySQL connector module

Import using a import mysql.connector statement so you can use this module's methods to communicate with the MySQL database.

3. Use the connect() method

Use the connect() method of the MySQL Connector class with the required arguments to connect MySQL. It would return a MySQLConnection object if the connection established successfully

4. Use the cursor() method

Use the cursor() method of a MySQLConnection object to create a cursor object to perform various SQL operations.

5. Use the execute() method

The execute() methods run the SQL query and return the result.

6. Extract result using fetchall()

Use cursor.fetchall() or fetchone() or fetchmany() to read query result.

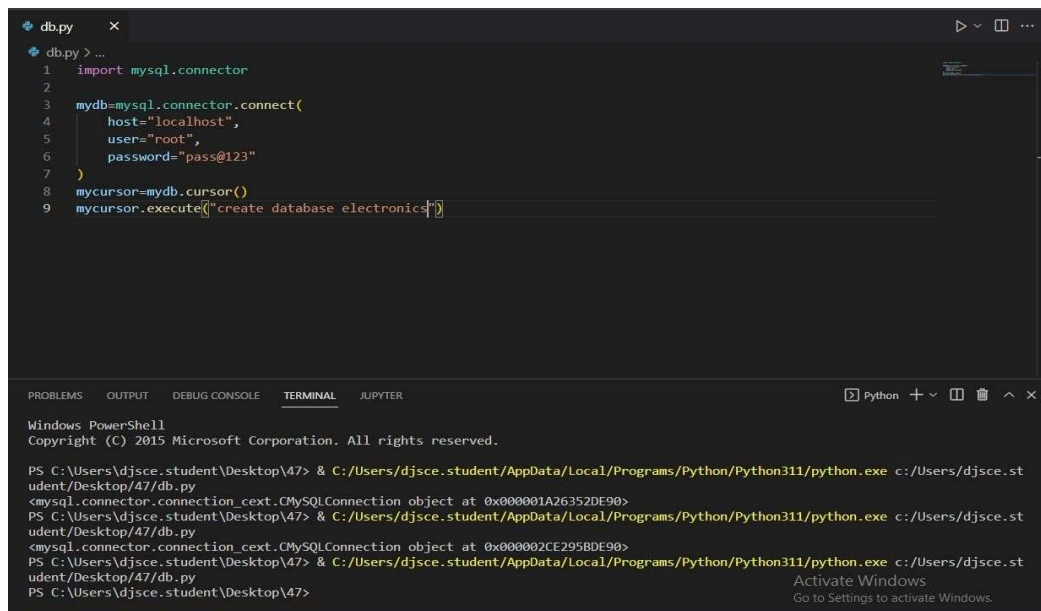
7. Close cursor and connection objects use cursor.close() and connection.close() method to close open connections after your work completes

Implement the following queries using MySQL in Python:

1. Create Database:

```
import mysql.connector
mydb = mysql.connector.connect(host="localhost",
user="root", password="pass@123",) mycursor =
mydb.cursor()
mycursor.execute("CREATE DATABASE electronics")
```

Output:



```
db.py
1 import mysql.connector
2
3 mydb=mysql.connector.connect(
4     host="localhost",
5     user="root",
6     password="pass@123"
7 )
8 mycursor=mydb.cursor()
9 mycursor.execute("create database electronics")
```

Windows PowerShell
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
<mysql.connector.connection_cext.CMySQLConnection object at 0x000001A26352DE00>
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
<mysql.connector.connection_cext.CMySQLConnection object at 0x000002CE2958DE90>
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
PS C:\Users\djsce.student\Desktop\47>

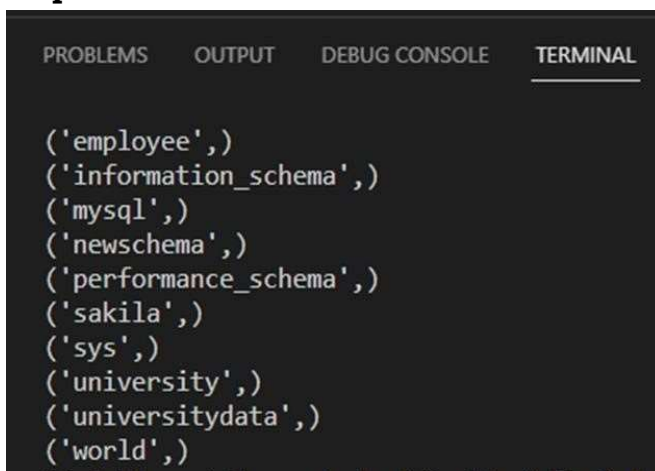
Activate Windows
Go to Settings to activate Windows.

2. Create Table:

```
import mysql.connector

mydb = mysql.connector.connect(    host="localhost",    user="root",
password="pass@123",    database="electronics"
) mycursor =
mydb.cursor()
mycursor.execute("""CREATE TABLE Laptop (
                                Id int(11) NOT NULL,
                                Name varchar(250) NOT NULL,
                                Price float NOT NULL,
                                Purchase_date Date NOT NULL,
                                PRIMARY KEY (Id))""")
```

Output:



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

('employee',)
('information_schema',)
('mysql',)
('newschema',)
('performance_schema',)
('sakila',)
('sys',)
('university',)
('universitydata',)
('world',)
```

3. Insert values (Show examples for Single Row and Multiple Rows):

```
import mysql.connector

mydb = mysql.connector.connect(host="localhost", user="root",
password="pass@123", database="electronics") mycursor = mydb.cursor()
mycursor.execute("""INSERT INTO Laptop (Id, Name, Price, Purchase_date)
VALUES
(15, 'Lenovo ThinkPad P71', 6459, '2019-08-14') """)
mydb.commit() print(mycursor.rowcount, "record inserted.")
```

Output:

1. Single Row

```
db.py
1 password= 'pass@123' ,
2 database="electronics"
3 )
4
5
6 mycursor = mydb.cursor()
7 mycursor.execute("""INSERT INTO Laptop (Id, Name, Price, Purchase_date)
8 VALUES
9 (15, 'Lenovo ThinkPad P71', 6459, '2019-08-14') """)
10
11 mydb.commit()
12
13 print(mycursor.rowcount, "record inserted.")
14
15
16
17
18
19
20
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** JUPYTER

```
in execute
result = self.cnx.cmd_query(
    .....
File "C:\Users\djsce.student\AppData\Local\Programs\Python\Python311\Lib\site-packages\mysql\connector\connection_cext.py", line
573, in cmd_query
    raise get_mysql_exception(
mysql.connector.errors.ProgrammingError: 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to y
our MySQL server version for the right syntax to use near '
                                Name' at line 4
PS C:\Users\djsce.student\Desktop\47> & C:\Users\djsce.student\AppData\Local\Programs\Python\Python311\python.exe c:/Users/djsce.st
udent/Desktop/47/db.py
PS C:\Users\djsce.student\Desktop\47> & C:\Users\djsce.student\AppData\Local\Programs\Python\Python311\python.exe c:/Users/djsce.st
udent/Desktop/47/db.py
1 record inserted.
PS C:\Users\djsce.student\Desktop\47>
```

2. Multiple Rows

```
db.py x
db.py > ...
3 mydb = mysql.connector.connect(
4
5     host="localhost",
6     user="root",
7     password="pass@123",
8     database="electronics"
9 )
10
11 mycursor = mydb.cursor()
12 sql = """INSERT INTO Laptop (Id, Name, Price, Purchase_date)
13 VALUES
14 (%s,%s,%s,%s) """
15
16 val = [(2, 'Area 51M', 6999, '2019-04-14'),(3, 'MacBook Pro', 2499, '2019-06-20'),
17        (4, 'Asus', 8000, '2020-04-14'),(5, 'HP', 15000, '2019-05-10')]
18
19 mycursor.executemany(sql, val)
20
21 mydb.commit() |
22
23 print(mycursor.rowcount, "record inserted.")
24
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

File "C:\Users\djsce.student\AppData\Local\Programs\Python\Python311\Lib\site-packages\mysql\connector\connection_cext.py", line 573, in cmd_query
raise get_mysql_exception(
mysql.connector.errors.ProgrammingError: 1064 (42000): You have an error in your SQL syntax; check the manual that corresponds to your MySQL server vers
ht syntax to use near ' Name' at line 4
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
1 record inserted.
PS C:\Users\djsce.student\Desktop\47>

PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
4 record inserted.
PS C:\Users\djsce.student\Desktop\47>

4. Delete a row based on values:

```
import mysql.connector mydb = mysql.connector.connect(     host="localhost",  
user="root",     password="pass@123",     database="electronics  
" ) mycursor = mydb.cursor() mycursor.execute("DELETE FROM  
Laptop WHERE NAME = 'Asus' ") mydb.commit()  
print(mycursor.rowcount, "record(s) deleted")
```

Output:

```
db.py ×
db.py > ...
1 import mysql.connector
2
3 mydb = mysql.connector.connect(
4
5     host="localhost",
6     user="root",
7     password="pass@123",
8     database="electronics"
9 )
10
11 mycursor = mydb.cursor()
12 mycursor.execute("DELETE FROM Laptop WHERE NAME = 'Asus' ")
13
14 mydb.commit()
15
16 print(mycursor.rowcount, "record(s) deleted") |

(5, 'HP', 15000.0, datetime.date(2019, 5, 10))
PS C:/Users/djsce.student/Desktop/47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.st
udent/Desktop/47/db.py
(2, 'Area 51M', 6999.0, datetime.date(2019, 4, 14))
(4, 'Asus', 8000.0, datetime.date(2020, 4, 14))
PS C:/Users/djsce.student/Desktop/47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.st
udent/Desktop/47/db.py
1 record(s) deleted
PS C:/Users/djsce.student/Desktop/47>
```

5. Display the rows of the table:

```
import mysql.connector mydb =  
mysql.connector.connect(host="localhost",  
user="root", password="pass@123", d  
atabase="electronics") mycursor = mydb.cursor()  
mycursor.execute("SHOW TABLES") for x in mycursor:  
    print(x)
```

6. Update the values of a specific row:

```
import mysql.connector mydb = mysql.connector.connect(host="localhost",
user="root", password="pass@123", database="electronics") mycursor =
mydb.cursor()
mycursor.execute("UPDATE Laptop SET Price = 25000 WHERE NAME = 'HP' ")
mydb.commit() print(mycursor.rowcount, "record(s) deleted")
```

Output:

The screenshot shows a Jupyter Notebook interface with a dark theme. The top bar indicates the file is named 'db.py'. The code cell contains a Python script that connects to a MySQL database, updates the price of a laptop, and prints the number of records affected. The terminal output shows the execution of the script, displaying the number of records deleted and the current date and time.

```
db.py > ...
1 import mysql.connector
2
3 mydb = mysql.connector.connect(
4
5     host="localhost",
6     user="root",
7     password="pass@123",
8     database="electronics"
9 )
10
11 mycursor = mydb.cursor()
12 mycursor.execute("UPDATE Laptop SET Price = 25000 WHERE NAME = 'HP' ")
13
14 mydb.commit()
15
16 print(mycursor.rowcount, "record(s) deleted")
```

Terminal Output:

```
(2, 'Area 51M', 6999.0, datetime.date(2019, 4, 14))
(4, 'Asus', 8000.0, datetime.date(2020, 4, 14))
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.st
udent/Desktop/47/db.py
1 record(s) deleted
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.st
udent/Desktop/47/db.py
1 record(s) deleted
PS C:\Users\djsce.student\Desktop\47>
```

At the bottom of the terminal, there is a watermark that says "Activate Windows Go to Settings to activate Windows."

7. Search whether a particular record is present in the table or not:

```
import mysql.connector mydb =
mysql.connector.connect(
    host="localhost",    user="root",    password="pass@123",    database="electronics
")
mycursor = mydb.cursor()
mycursor.execute("SELECT * FROM Laptop WHERE Name = 'HP' ")
myresult = mycursor.fetchall() for x in myresult:
    print(x)
```

Output:

```
db.py X
db.py > ...
3 mydb = mysql.connector.connect(
4
5     host="localhost",
6     user="root",
7     password="pass@123",
8     database="electronics"
9 )
10
11 mycursor = mydb.cursor()
12 mycursor.execute("SELECT * FROM Laptop WHERE Name = 'HP' ")
13
14 myresult = mycursor.fetchall()
15
16 for x in myresult:
17     print(x)
```

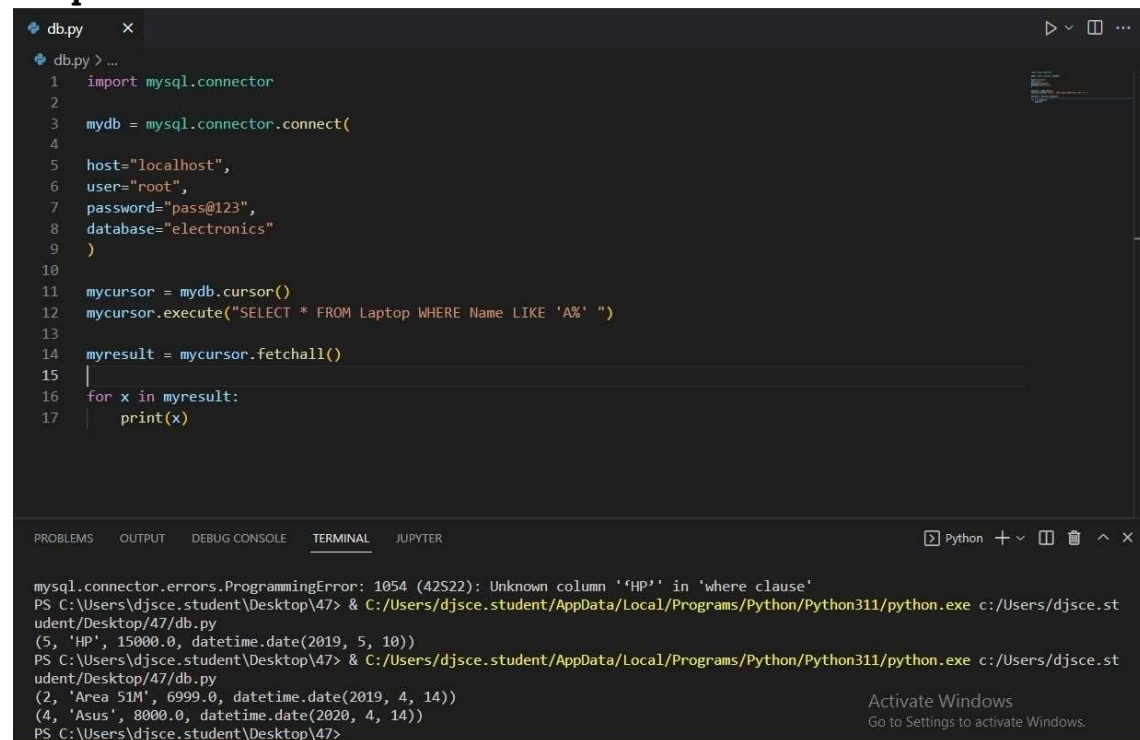
File "C:\Users\djsce.student\AppData\Local\Programs\Python\Python311\Lib\site-packages\mysql\connector\connection_cext.py", line 573, in cmd_query
raise get_mysql_exception(
mysql.connector.errors.ProgrammingError: 1054 (42S22): Unknown column 'HP' in 'where clause'
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
(5, 'HP', 15000.0, datetime.date(2019, 5, 10))
PS C:\Users\djsce.student\Desktop\47>

Python + - [] [] ^ X
Activate Windows
Go to Settings to activate Windows.

8.Wildcards:

```
import mysql.connector mydb =  
mysql.connector.connect(     host="localhost",  
user="root",     password="pass@123",  
database="electronics"  
) mycursor = mydb.cursor() mycursor.execute("SELECT * FROM  
Laptop WHERE Name LIKE 'A%' ") myresult = mycursor.fetchall()  
for x in myresult:  
    print(x)
```

Output:



The screenshot shows a Jupyter Notebook with a file named 'db.py'. The code in the notebook is as follows:

```
1 import mysql.connector
2
3 mydb = mysql.connector.connect(
4
5     host="localhost",
6     user="root",
7     password="pass@123",
8     database="electronics"
9 )
10
11 mycursor = mydb.cursor()
12 mycursor.execute("SELECT * FROM Laptop WHERE Name LIKE 'A%' ")
13
14 myresult = mycursor.fetchall()
15
16 for x in myresult:
17     print(x)
```

The output of the script is displayed in the terminal pane at the bottom. It shows a MySQL error message followed by the execution of the script and the resulting data from the 'Laptop' table:

```
mysql.connector.errors.ProgrammingError: 1054 (42S22): Unknown column 'HP' in 'where clause'
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
(5, 'HP', 15000.0, datetime.date(2019, 5, 10))
PS C:\Users\djsce.student\Desktop\47> & C:/Users/djsce.student/AppData/Local/Programs/Python/Python311/python.exe c:/Users/djsce.student/Desktop/47/db.py
(2, 'Area 51M', 6999.0, datetime.date(2019, 4, 14))
(4, 'Asus', 8000.0, datetime.date(2020, 4, 14))
PS C:\Users\djsce.student\Desktop\47>
```

An 'Activate Windows' watermark is visible in the bottom right corner of the terminal pane.

9. Delete the table:

```
import mysql.connector mydb = mysql.connector.connect(host="localhost",
user="root", password="pass@123", database="electronics" ) mycursor =
mydb.cursor() mycursor.execute("DROP TABLE Laptop") mydb.commit()
```

10. Delete the database:

```
import mysql.connector mydb
=
mysql.connector.connect(host="localhost", user="root", password="pass@123", database="electronics" ) mycursor = mydb.cursor() mycursor.execute("DROP DATABASE
electronics ") mydb.commit()
```

Conclusion:

In this experiment, we saw how to use MySQL Connector/Python to integrate a MySQL database with our Python application. We also saw some unique features of a MySQL database that differentiate it from other SQL databases.

EXPERIMENT 9

Aim: Implement a client server communication application based on socket Programming in python.

Theory:

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server. They are the real backbones behind web browsing. In simpler terms, there is a server and a client.

Server:

A server has a bind() method which binds it to a specific IP and port so that it can listen to incoming requests on that IP and port. A server has a listen() method which puts the server into listening mode. This allows the server to listen to incoming connections. And last a server has an accept() and close() method. The accept method initiates a connection with the client and the close method closes the connection with the client.

Client:

Now we need something with which a server can interact. We could connect to the server like this just to know that our server is working.

Server:

```
import socket
def mpm():
    host = '127.0.0.1'
    port = 6000    s =
    socket.socket()
    s.bind((host, port))
    s.listen(1)    c, addr =
    s.accept()    print("Client
Address : ", addr)    while True:
        data = c.recv(1024)
    d = data.decode('ascii')
    print("Client: ", d)
    x = input(">>> ")    y
    = x.encode('ascii')
        c.send(y)
    mpm()
```

Client:

```
import socket
```

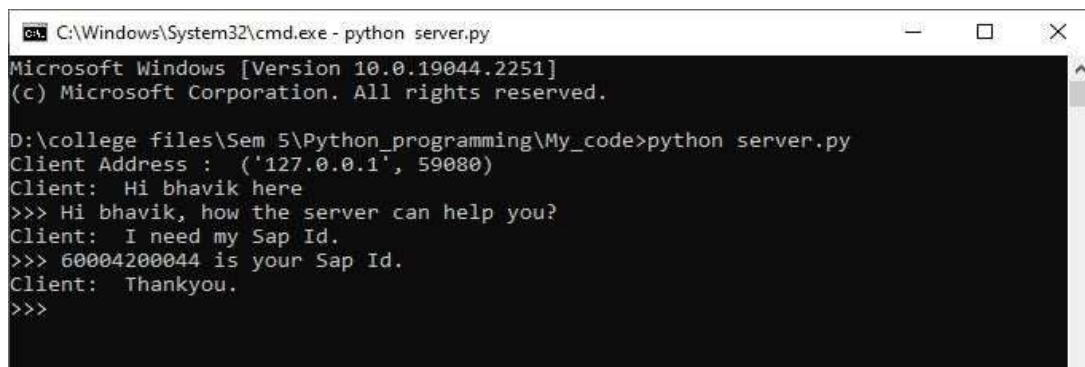
```

def mpm():
    host = '127.0.0.1'
    port = 6000
    s = socket.socket()
    s.connect((host, port))
while True:
    x = input("Enter New Message : ")
    y = x.encode('ascii')
    s.send(y)
    data = s.recv(1024)
    d = data.decode('ascii')
    print("Server: ",d)
    mpm()

```

Output:

Server:



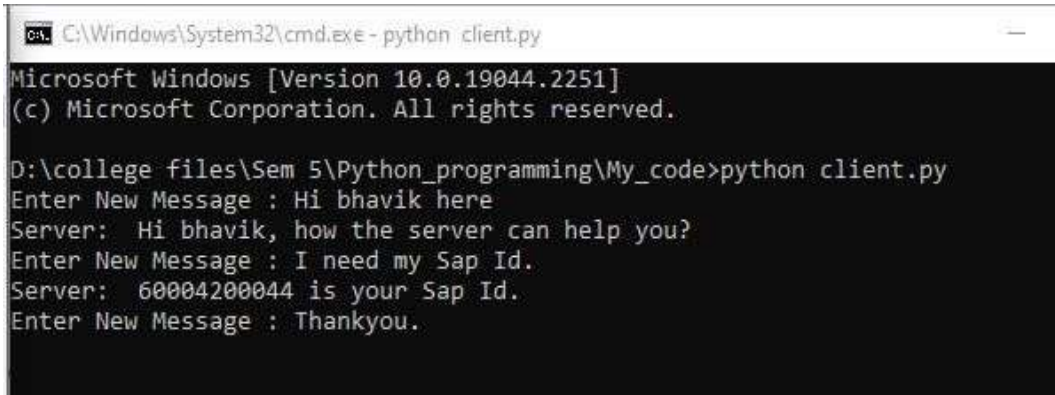
```

C:\Windows\System32\cmd.exe - python server.py
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

D:\college files\Sem 5\Python_programming\My_code>python server.py
Client Address : ('127.0.0.1', 59080)
Client: Hi bhavik here
>>> Hi bhavik, how the server can help you?
Client: I need my Sap Id.
>>> 60004200044 is your Sap Id.
Client: Thankyou.
>>>

```

Client:



```

C:\Windows\System32\cmd.exe - python client.py
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

D:\college files\Sem 5\Python_programming\My_code>python client.py
Enter New Message : Hi bhavik here
Server: Hi bhavik, how the server can help you?
Enter New Message : I need my Sap Id.
Server: 60004200044 is your Sap Id.
Enter New Message : Thankyou.

```

Conclusion:

In this experiment we learnt about how python provides two levels of access to network services. At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.

EXPERIMENT 10

Aim: To study concepts GUI using Tkinter in python and design a GUI application to show input and output operations.

Theory:

Tkinter tutorial provides basic and advanced concepts of Python Tkinter. Our Tkinter tutorial is designed for beginners and professionals.

Python provides the standard library Tkinter for creating the graphical user interface for desktop based applications.

Developing desktop based applications with python Tkinter is not a complex task. An empty Tkinter top-level window can be created by using the following steps.

1. import the Tkinter module.
2. Create the main application window.
3. Add the widgets like labels, buttons, frames, etc. to the window.
4. Call the main event loop so that the actions can take place on the user's computer screen.

Tkinter widgets

There are various widgets like button, canvas, checkbox, entry, etc. that are used to build the python GUI applications.

SN	Widget	Description
1	<u>Button</u>	The Button is used to add various kinds of buttons to the python application.
2	<u>Canvas</u>	The canvas widget is used to draw the canvas on the window.

3	<u>Checkbutton</u>	The Checkbutton is used to display the Checkbutton on the window.
4	<u>Entry</u>	The entry widget is used to display the single-line text field

		to the user. It is commonly used to accept user values.
5	<u>Frame</u>	It can be defined as a container to which, another widget can be added and organized.
6	<u>Label</u>	A label is a text used to display some message or information about the other widgets.
7	<u>ListBox</u>	The ListBox widget is used to display a list of options to the user.
8	<u>Menubutton</u>	The Menubutton is used to display the menu items to the user.
9	<u>Menu</u>	It is used to add menu items to the user.
10	<u>Message</u>	The Message widget is used to display the message-box to the user.

11	<u>Radiobutton</u>	The Radiobutton is different from a checkbox. Here, the user is provided with various options and the user can select only one option among them.
12	<u>Scale</u>	It is used to provide the slider to the user.
13	<u>Scrollbar</u>	It provides the scrollbar to the user so that the user can scroll the window up and down.
14	<u>Text</u>	It is different from Entry because it provides a multi-line text field to the user so that the user can write the text and edit the text inside it.
14	<u>Toplevel</u>	It is used to create a separate window container.
15	<u>Spinbox</u>	It is an entry widget used to select from options of values.
16	<u>PanedWindow</u>	It is like a container widget that contains horizontal or vertical panes.
17	<u>LabelFrame</u>	A LabelFrame is a container widget that acts as the container
18	<u>MessageBox</u>	This module is used to display the message-box in the desktop based applications.

Python Tkinter Geometry

The Tkinter geometry specifies the method by using which, the widgets are represented on display. The python Tkinter provides the following geometry methods.

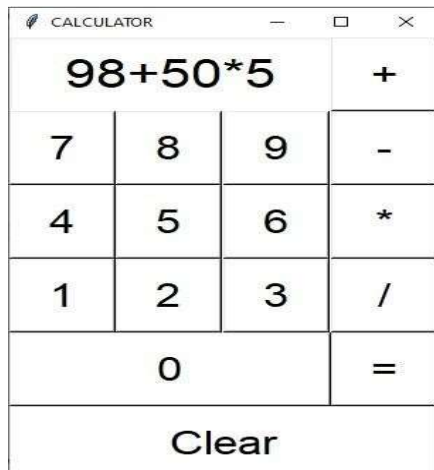
1. The pack() method
2. The grid() method
3. The place() method

```
import tkinter as tk
window = tk.Tk()
window.title("CALCULATOR")
window.configure(bg="white")
w = tk.Canvas(window, width=500, height=500, bg="white")
w.grid(row=0, column=0)
v = tk.StringVar()
txt = tk.Label(w, textvariable=v, font=("Arial", 30), bg="white", padx=5, pady=5)
v.set("")
txt.grid(row=0, column=0, columnspan=3)
btn0 = tk.Button(w, text="0", font=("Arial", 25), padx=5, pady=5, bg="white", width=11, command=lambda: v.set(v.get()+"0"))
btn0.grid(row=4, column=0, columnspan=3)
btn1 = tk.Button(w, text="1", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"1"))
btn1.grid(row=3, column=0)
btn2 = tk.Button(w, text="2", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"2"))
btn2.grid(row=3, column=1)
btn3 = tk.Button(w, text="3", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"3"))
btn3.grid(row=3, column=2)

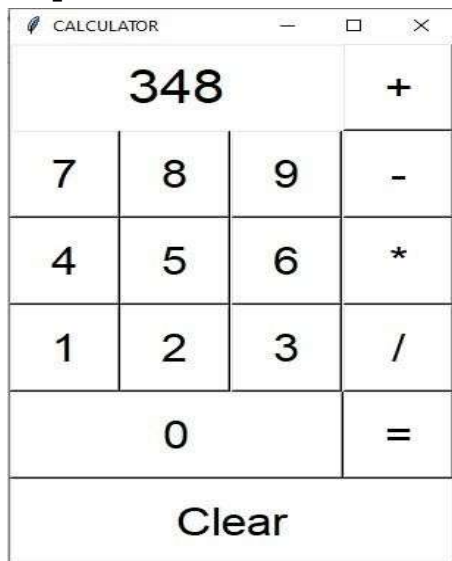
btn4 = tk.Button(w, text="4", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"4"))
btn4.grid(row=2, column=0)
btn5 = tk.Button(w, text="5", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"5"))
btn5.grid(row=2, column=1)
btn6 = tk.Button(w, text="6", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"6"))
btn6.grid(row=2, column=2)
btn7 = tk.Button(w, text="7", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"7"))
btn7.grid(row=1, column=0)
btn8 = tk.Button(w, text="8", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"8"))
btn8.grid(row=1, column=1)
btn9 = tk.Button(w, text="9", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"9"))
btn9.grid(row=1, column=2)
btn_plus = tk.Button(w, text="+", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+""))
btn_plus.grid(row=0, column=3)
btn_minus = tk.Button(w, text="-", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"-"))
btn_minus.grid(row=1, column=3)
btn_mul = tk.Button(w, text="*", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"*"))
btn_mul.grid(row=2, column=3)
btn_div = tk.Button(w, text="/", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(v.get()+"/"))
btn_div.grid(row=3, column=3)
btn_eq = tk.Button(w, text="=", font=("Arial", 25), padx=5, pady=5, width=3, bg="white", command=lambda: v.set(eval(str(v.get()))))
btn_eq.grid(row=4, column=3)
btn_clr = tk.Button(w, text="Clear", font=("Arial", 25), padx=5, pady=5, width=15, bg="white", command=lambda: v.set(""))
btn_clr.grid(row=5, column=0, columnspan=4)
window.mainloop()
```

Output:

Input:



Output:



Conclusion:

In this experiment we learnt about the concept of GUI in python which we can implement using tkinter. Tkinter provide us many different classes to make an interactive application. We had built a calculator application which can perform basics mathematical operations in this experiment.

EXPERIMENT 11

Aim: To study and implement concepts of Pandas and Matplotlib in Python.

Theory:

Pandas:

Pandas is an open-source library that is made mainly for working with relational or labeled data both easily and intuitively. It provides various data structures and operations for manipulating numerical data and time series. This library is built on top of the NumPy library. Pandas is fast and it has high performance & productivity for users.

Advantages

- Fast and efficient for manipulating and analyzing data.
- Data from different file objects can be loaded.
- Easy handling of missing data (represented as NaN) in floating point as well as non-floating point data
- Size mutability: columns can be inserted and deleted from DataFrame and higher dimensional objects
- Data set merging and joining.
- Flexible reshaping and pivoting of data sets
- Provides time-series functionality.
- Powerful group by functionality for performing split-apply-combine operations on data sets.

Matplotlib:

Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multi-platform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.

One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals. Matplotlib consists of several plots like line, bar, scatter, histogram etc.

A. Pandas

1. Read Csv

```
[4] #write code here
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

[5] train_path = "/content/titanic_data.csv"
#Read Data
train_df = pd.read_csv(train_path)
```

2. Show various operations using dataframe to read data , clean data and analyse data.

```
[6] train_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
[ ] train_df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

3. Create series, create own dataframe

```
[9] #regex example
import re
title_list = []
for i in train_df["Name"]:
    title_search = re.search('(\w+)\.', i)
    if title_search:
        title = title_search.group(1)
        title_list.append(title)

train_df["Title"] = title_list
```

```

[10] unique_title = train_df["Title"].unique()

survival_dict = {} #will keep count of survived people per title
total_dict = {} #will keep track of total people per title

for i in zip(train_df["Title"], train_df["Survived"]):
    if i[0] in total_dict.keys():
        total_dict[i[0]] += 1
        survival_dict[i[0]] += i[1]
    else:
        total_dict[i[0]] = 1
        survival_dict[i[0]] = i[1]

print(survival_dict)
print()
print(total_dict)

survival_rate_dict = {}
for i in total_dict.keys():
    survival_rate_dict[i] = survival_dict[i]/total_dict[i]

print()
print(survival_rate_dict)

#creating new dataframe
SR_df = pd.DataFrame()
SR_df["Title"] = list(survival_rate_dict.keys())
SR_df["Survival Rate"] = list(survival_rate_dict.values())

{'Mr': 81, 'Mrs': 99, 'Miss': 127, 'Master': 23, 'Don': 0, 'Rev': 0, 'Dr': 3, 'Mme': 1, 'Ms': 1, 'Major': 1, 'Lady': 1, 'Sir': 1, 'Mlle': 2, 'Col': 1, 'C
'Mr': 517, 'Mrs': 125, 'Miss': 182, 'Master': 40, 'Don': 1, 'Rev': 6, 'Dr': 7, 'Mme': 1, 'Ms': 1, 'Major': 2, 'Lady': 1, 'Sir': 1, 'Mlle': 2, 'Col': 2,
{'Mr': 0.15667311411992263, 'Mrs': 0.792, 'Miss': 0.6978021978021978, 'Master': 0.575, 'Don': 0.0, 'Rev': 0.0, 'Dr': 0.42857142857142855, 'Mme': 1.0, 'Ms

```

4. Delete NA values from the dataframe(all NA and NA values of specific columns)

```

[11] df_nan = pd.DataFrame()

nan_dict = {}
for i in train_df.columns:
    nan_dict[i] = (train_df[i].isnull().sum()/len(train_df))*100

df_nan["Column"] = nan_dict.keys()
df_nan["nan_percentage"] = nan_dict.values()
df_nan

df_nan

```

	Column	nan_percentage
0	PassengerId	0.000000
1	Survived	0.000000
2	Pclass	0.000000
3	Name	0.000000
4	Sex	0.000000
5	Age	19.865320
6	SibSp	0.000000
7	Parch	0.000000
8	Ticket	0.000000
9	Fare	0.000000
10	Cabin	77.104377
11	Embarked	0.224467
12	Title	0.000000

```

[12] train_df.drop(["Cabin"], axis=1, inplace=True)

```

5. Fill NA values with random values, mean, median)

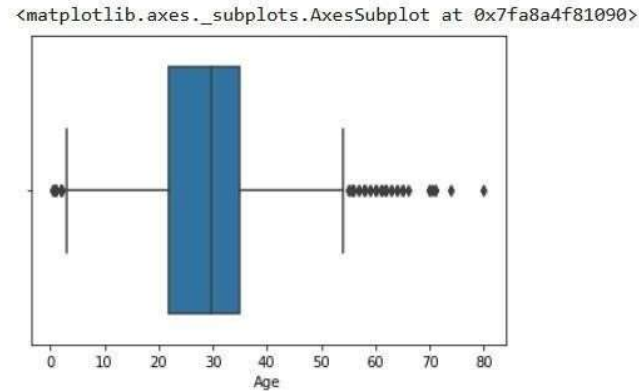
```

[13] train_df["Embarked"].fillna(train_df["Embarked"].mode()[0], inplace=True)
train_df["Age"].fillna(train_df["Age"].mean(), inplace=True)

```

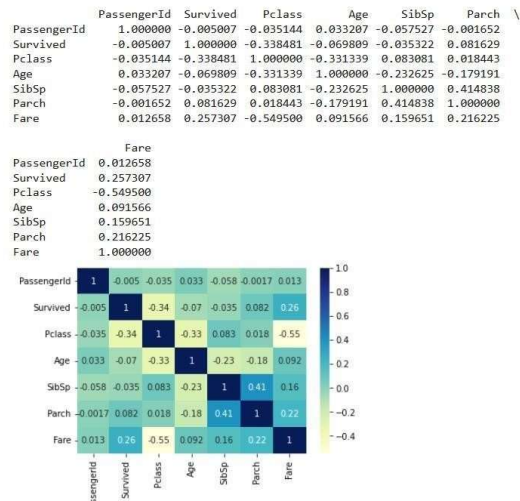
6. Display statistical information of the data frame

```
[19] sb.boxplot(x=train_df["Age"])
```



7. Establish relationship between the columns of the data frame

```
[14] import seaborn as sb
print(train_df.corr())
dataplot = sb.heatmap(train_df.corr(), cmap="YlGnBu", annot=True)
plt.show()
```



B. Matplotlib

Plot following graphs

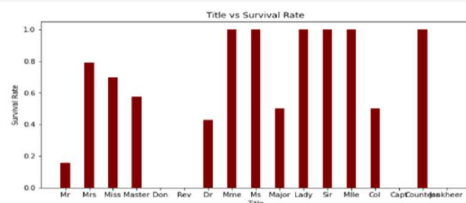
1. Barchart

```
[15] titles = list(SR_df["Title"])
sr = list(SR_df["Survival Rate"])

fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(titles, sr, color = 'maroon', width = 0.4)

plt.xlabel("Title")
plt.ylabel("Survival Rate")
plt.title("Title vs Survival Rate")
plt.show()
```

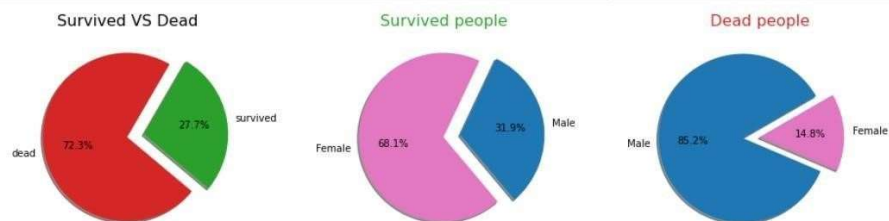


2. Pie Chart

```
[20] plt.subplot(1,3,2)
plt.pie(x = [len(train_df[(train_df.Survived == 1)&(train_df.Sex == "male")]),
             len(train_df[(train_df.Survived == 1)&(train_df.Sex == "female")])], labels = ['Male', 'Female'],
        , shadow = True, explode = [0.2, 0], startangle = -50,
        colors = ['tab:blue', 'tab:pink'], autopct = '%.1f%%')
plt.title('Survived people', fontdict = {'color':'tab:green', 'size':16})

plt.subplot(1,3,3)
plt.pie(x = [len(train_df[(train_df.Survived == 0)&(train_df.Sex == "male")]),
             len(train_df[(train_df.Survived == 0)&(train_df.Sex == "female")])], labels = ['Male', 'Female'],
        , shadow = True, explode = [0.2, 0], startangle = 30,
        colors = ['tab:blue', 'tab:pink'], autopct = '%.1f%%')
plt.title('Dead people', fontdict = {'color':'tab:red', 'size':16})

plt.show()
```



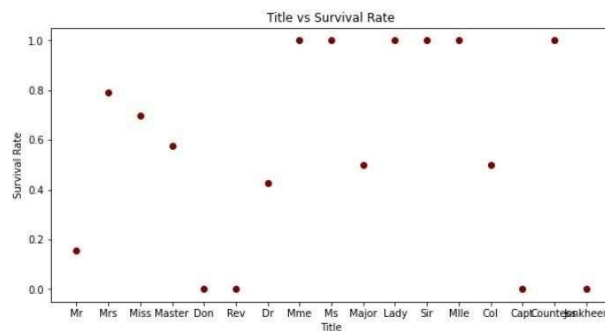
3. Scatter plot

```
[21] titles = list(SR_df["Title"])
sr = list(SR_df["Survival Rate"])

fig = plt.figure(figsize = (10, 5))

# creating the scatter plot
plt.scatter(titles, sr, color = 'maroon')

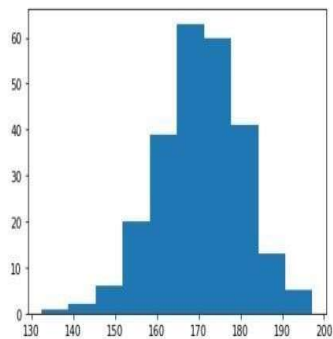
plt.xlabel("Title")
plt.ylabel("Survival Rate")
plt.title("Title vs Survival Rate")
plt.show()
```



4. Histogram

```
[18] import matplotlib.pyplot as plt
import numpy as np

x = np.random.normal(170, 10, 250)
#histogram
plt.hist(x)
plt.show()
```



Conclusion: Thus from the above experiment we have concluded that Pandas DataFrame is a twodimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). The data produced by Pandas are often used as input for plotting functions of Matplotlib, statistical analysis in SciPy, and machine learning algorithms in Scikit-learn.

EXPERIMENT 12

Aim: Implement a web application using Django Framework and understand CRUD functionality.

Theory:

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.



Ridiculously fast.

Django was designed to help developers take applications from concept to completion as quickly as possible.



Reassuringly secure.

Django takes security seriously and helps developers avoid many common security mistakes.



Exceedingly scalable.

Some of the busiest sites on the web leverage Django's ability to quickly and flexibly scale.

Django makes use of a directory structure to arrange different parts of the web application.

It creates a project and an app folder for this.

Creating a proper project and organizing it helps in keeping the project DRY (Don't Repeat Yourself) and clean.

When we create a Django project, the Django itself creates a root directory of the project with the project name you have given on it. It contains the necessary files that would provide basic functionalities to your web applications.

Code:

MyProject/settings.py

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',
```

```
'django.contrib.contenttypes',
'django.contrib.sessions',
'django.contrib.messages',
'django.contrib.staticfiles',
'MyApp'
]
```

MyProject/urls.py

```
urlpatterns = [path('admin/', admin.site.urls),path('', include('MyApp.urls'))]
```

MyApp/admin.py

```
admin.site.register(ToDoModel)
```

MyApp/models.py

```
class ToDoModel(models.Model):
    task_name=models.CharField(max_length=200)    def __str__(self):
    return self.task_name
```

MyApp/urls.py

```
urlpatterns = [path('', views.gettodo, name='todo'), path('add/', views.posttodo,
name='add'), path('delete/<str:i>/', views.deletetodo, name='delete'),
path('update/<str:i>/', views.updatetodo, name='update'),]
```

MyApp/views.py

```
def gettodo(request):    if
request.method=='GET':
    list=ToDoModel.objects.all()    return
    render(request,'index.html',{'todo_list' : list})
def posttodo(request):    if
request.method=='POST':

task=request.POST.get('task_name')
add_task=ToDoModel(task_name=task)
add_task.save()    return
    HttpResponseRedirect('/')
def deletetodo(request,i):
if request.method=='POST':
    w=ToDoModel.objects.get(id=i)
    w.delete()
return
    HttpResponseRedirect('/')
def updatetodo(request,i):
if request.method=='GET':
    x=ToDoModel.objects.get(id = i)
return
    render(request,'update.html',{'item':x})
if request.method=='POST':
    object=ToDoModel.objects.get(id = i)
    object.task_name=request.POST.get('task_name')
    object.save()    return
    HttpResponseRedirect('/')
```

MyApp/templates/index.html

```

<!DOCTYPE html>
<head>
  <title>ToDoList</title>
  <style>    div {
margin: auto;
width: 50%;
border: 3px;
padding: 10px;
    }    h1 {
text-align: center
    }
  </style>
</head>
<body>
  <h1> Todo List </h1>
  <form action="{% url 'add' %}" method="POST">
    {% csrf_token %}
    <div class="mb-3">
      <label for="formGroupExampleInput" class="form-label">ToDo Item: </label>
<input type="text" class="form-control" id="task_name" name="task_name"
placeholder="ToDo Item">
    </div>
    <div>
      <button type="submit" class="btn btn-secondary btn-lg btn-block" name="add">Add
Item</button>
    </div>
  </form>
  <ol>
    {% for i in todo_list%}
    <li>
      {{i}}
      <tr>
        <form action="{% url 'delete' i.id %}" method="POST"> {% csrf_token %}
<button type="delete" class="btn btn-primary btn-lg">Delete</button>      </form>
        <a href="{% url 'update' i.id %}"><button type="update" class="btn btn-primary
btn-lg">Edit</button></a>
      </tr>
    </li>
    <br>
    {% endfor%}
  </ol>
</body>

```

MyApp/templates/update.html

```

<h1>Update ToDo</h1>
<form method="POST">
{% csrf_token %}
  <div class="mb-3">
    <label for="formGroupExampleInput" class="form-label">ToDo Item: </label>
    <input type="text" class="form-control" id="task_name" name="task_name" placeholder="ToDo
Item" value="{{item.task_name}}">
  </div>
  <div>
    <button type="submit" class="btn btn-secondary btn-lg btn-block" name="add">Add
Item</button>

```



```
</div>  
</form>
```

Output:

1. Add a todo item



2. Update a todo item



Conclusion:

Django is a rapid web development framework and if you want to get your application built fast within a few days, there is no better framework than Django Web Framework. Django gives all the features included, also called “Batteries Included Framework”. It has a built-in admin interface which makes it easy to work with it.