G H Raisoni College of Engineering & Management, Pune – 412 207

Department of Computer Engineering

D-19

Lab Manual (2024-2025)

Class: SY B. Tech. Computer

Term: III

Python Programming Lab

G H Raisoni College of Engineering and Management, Pune 412207

Department: Computer Engineering

Course Details

Course: Python Programming Lab-(23UCOP2307)

Class: S Y B.Tech. **Division: A, B, C**

Internal Marks: 25 **External marks: 25**

Credits:2 **Pattern: 2023**

COURSE OUTCOME	
CO1:	Learn the basic concepts keywords, identifiers, data types to implement simple Python program
CO2:	Apply control structure, functions to demonstrate Python program
CO3:	Perform the different operations on data structure like list, set, tuple and dictionary
CO4:	Demonstrate Python program for string and file handling

List of Experiments

Sr. No.	Experiment List	CO Mapping	Software Required
1.	Write a program by using basic concepts like input, output, variable, keywords, and identifiers.	CO1	Python or Programiz or Online Python IDE
2.	Write a program to demonstrate the of arithmetic operators.	CO1	Python or Programiz or Online Python IDE
3.	Write a program to demonstrate working of control structures.	CO2	Python or Programiz or Online Python IDE
4.	Write a program to demonstrate working of different types of functions.	CO2	Python or Programiz or Online Python IDE
5.	Write a program to demonstrate working of recursion.	CO2	Python or Programiz or Online Python IDE
6.	Write a program to perform different operations on list: i. Append ii. Extend iii. Insert iv. Remove v. Pop vi. Slice	CO3	Python or Programiz or Online Python IDE
7.	Write a program to perform different operations on set: i. Update ii. Remove the element iii. Clear iv. pop	CO3	Python or Programiz or Online Python IDE
8.	Write a program to perform different operations on tuple: i. Accessing ii. Concatenation iii. Slicing iv. Deleting	CO3	Python or Programiz or Online Python IDE

9.	Write a program to demonstrate the working of dictionary	CO3	Python or Programiz or Online Python IDE
10.	Write a program demonstrates various operations on strings: i. Slicing ii. Concatenating iii. Finding the length of the string iv. Converting the string to uppercase and lowercase v. Replacing a substring with another substring vi. Splitting the string into a list of substrings	CO4	Python or Programiz or Online Python IDE
11.	Write a program to demonstrate different operations on file: i. Create ii. Open iii. Read iv. Write v. Update vi. Delete	CO4	Python or Programiz or Online Python IDE
12.	Mini Project	CO1, CO2, CO3, CO4	Python or Programiz or Online Python IDE

Assignment No – 1

Aim: Write a program by using basic concepts like input, output, variable, keywords, and identifiers.

Theory:

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

Python print() Function

The print() function displays the given object to the standard output device (screen) or to the text stream file.

Unlike the other programming languages, Python print() function is most unique and versatile function.

Taking Input to the User

Python provides the input() function which is used to take input from the user.

```
name = input("Enter a name of student:")

print("The student name is: ", name)
```

Output: The Student name is: ABC

If we want to take input as an integer number, we need to typecast the input() function into an integer.

```
a  = int(input("Enter first number: "))

b = int(input("Enter second number: "))

print(a+b)
```

Output: Enter first number: 50

Enter second number: 100

150

Python Variable

A Python variable is a symbolic name that is a reference or pointer to an object. Once an object is assigned to a variable, you can refer to the object by that name.

Python Data Types

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

```
A = 2
```

The variable **A** holds integer value five and we did not define its type. Python interpreter will automatically interpret variables **A** as an integer type.
Python enables us to check the type of the variable used in the program. Python provides us the **type()** function, which returns the type of the variable passed.

Python supports three types of numeric data.

1. **Int** - Integer value can be any length such as integers 10, 2, 29, -20, -150 etc. Python has no restriction on the length of an integer. Its value belongs to int
2. **Float** - Float is used to store floating-point numbers like 1.9, 9.902, 15.2, etc. It is accurate upto 15 decimal points.
3. **Complex** - A complex number contains an ordered pair, i.e., x + iy where x and y denote the real and imaginary parts, respectively. The complex numbers like 2.14j, 2.0 + 2.3j, etc.

Keyword:

A Python keyword is a word that serves a specific function in Python. It is limited to one single function and it cannot be set as a variable name, a function name, or the value of any other identifier. The purpose of the keyword is to define the syntax of the code.

Identifier in Python:

Identifiers in Python is a name used to identify a variable, function, class, module, or other objects. An identifier can only contain letters, digits, and underscores, and cannot start with a digit. In Python, identifiers are case-sensitive, meaning that foo and Foo are considered to be two different identifiers.

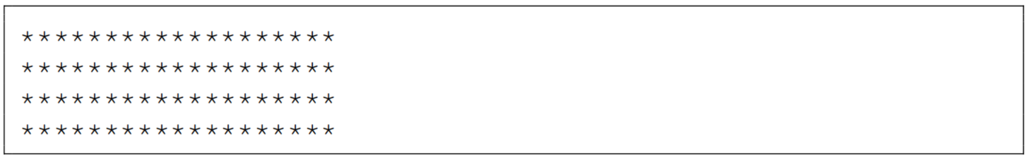
Source code: -

Q.1 Write python program to input user name and greet the user?

Python Code:

```
name = input("Hello, What's your name?")
print("Hello " + name+ " it's nice to meet you"+ "!")
```

Q.2 Print a box like the one below.

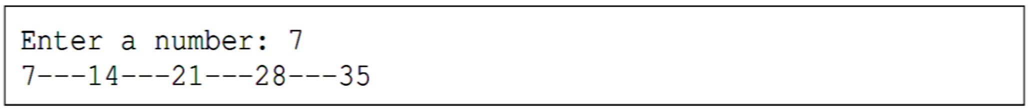


```
*******************
*******************
*******************
*******************
```

Python Code:

```
print ('*' * 19)
print ('*' * 19)
print ('*' * 19)
print ('*' * 19)
```

Q.3 Ask the user to enter a number x. Use the sep optional argument to print out x, 2x, 3x, 4x, and 5x, each separated by three dashes, like below.



```
Enter a number: 7
7---14---21---28---35
```

Python Code:

```
a=int(input())
print(a,"---",a*2,"---",a*3,"---",a*4,"---",a*5,sep='')
```

Q. 4 Write a program that reads marks of three quizzes and outputs the total out of 100.

Python Code:

```
a = int(input("Enter the marks of first subject: "))
b = int(input("Enter the marks of second subject: "))
c = int(input("Enter the marks of third subject: "))
total = a+b+c
avg = total/3
print("Total marks: ",total)
print("Average marks: ",avg)
```

Q.5 What will be the output of the following Python program?

Python Code:

```
print("Hello", "World!", end='####')
print("Bye", "World!", sep='$$')
print("With sep", "and end", sep=' ', end='.\n')
```

Answer:

```
Hello World!####Bye$$World!
With sep and end.
```

Output:-

Conclusion:-

Assignment No: - 2

Aim: Write a program to demonstrate the of arithmetic operators.

Theory:

Arithmetic Operator

An operator is a symbol that tells the compiler that either a mathematical or a logical manipulation has to be done. In this lab you will be studying about the Arithmetic Operations.

They are of the following types:

Addition Operator (+)
The addition operator is used to add two numbers. It is placed between two numbers that are to be added.
Syntax : number1 + number2
Program:

```
x = 3
y = 5
print(‘x + y = ‘, x+y)
```

Output:

```
x + y = 8
```

Subtraction Operator (-)
The subtraction operator is used to subtract two numbers. It is placed between two numbers that are to be subtracted. The right placed number is subtracted from the one that is placed at left.

Syntax : number1 - number2
Program:

```
x = 10
y = 5
print(‘x - y = ‘, x-y)
```

Output:

```
x - y = 5
```

Multiplication Operator (*)
The multiplication operator is used to multiply two numbers. It is also placed between the two numbers that are to be operated.
Syntax : number1 * number2
Program:

```
x = 10
y = 2
```

```
    print(‘x * y = ‘, x8y)
```
Output:
```
        x * y = 20
```

Division Operator (/)
The division operator is used to divide two numbers. It is used between the numbers that are to be operated.

Syntax : number1 / number2

It has some different rules that have to be kept in mind before operating the numbers. Python2 operates the division operator by taking the integral value.

Example : 6 / 4
Answer :
This operation will be solved in Python2 by taking the integral value i.e 1.
Therefore, the answer of 6 / 4 = 1
This problem can be taken care by Type Casting. Type Casting is used to convert the output in a desired form.
To get the correct answer of the above example, we will type cast it using float data type.

Example : float(6 / 4)
Answer : Now, the output will be changed into float type and the answer will be 1.5.

```
float(6 / 4) = 1.5
```

There's another way of solving such problem. By using one float type input, we can get the desired answer.

Example : 6.0 / 4
Answer : 1.5

Modulus Operation (%)
It is used to give out the remainder of a division operation. It is also placed between numbers. The right placed number divides the one on the left and the remainder is given as output.

Syntax : number1 % number2
Program:
```
x = 10
y = 4
print(‘x % y = ‘, x%y)
```
Output:
```
x % y = 2
```

Exponent Operation (**)
It is used to perform exponential calculations. The right placed number acts as the power.
Syntax : number1* *number2
Program:

```
x = 10
y = 2
print('x ** y = ', x**y)
```

Output:

```
x ** y = 100
```

Floor Division Operator (//)
It is used to perform floor division. This gives the result in int format.
Syntax : number1 // number2

```
x=5
y=3
print(x//y)
```

Source Code:

Q.1 Write a program that asks the user for a weight in kilograms and converts it to pounds. There are 2.2 pounds in a kilogram.

Python Code:

```
kg = float(input("What is you're weight in kilograms? "))
lbs = kg * 2.20462
print (lbs)
```

Q. 2 Write a program that asks the user to enter three numbers (use three separate input statements). Create variables called total and average that hold the sum and average of the three numbers and print out the values of total and average.

Python Code:

```
a = int (input(" Please Enter the First Number: "))
b = int (input(" Please Enter the second number: "))
c = int (input(" Please Enter the second number: "))
total=a+b+c
average=total/3
print("The average of three numbers is ", average)
```

Q. 3 Write a program that computes and prints the result of $\frac{512-282}{47\cdot 48+5}$. It is roughly 0.1017.

Python Code:

```
a= ((512-282)/((47*48)+5))
a= round(a,4)
print(a)
```

Q. 4 What will be the output of following Python program?

Python Code:

```
integer_value = 5
float_value = 3.14
complex_value = 2 + 3j
result_float = integer_value * float_value
result_complex = float_value + complex_value
print(result_float)
print(result_complex )
```

Q. 5 Write a program which asks for the number of students on a course and the desired group size. The program will then print out the number of groups formed from the students on the course. If the division is not even, one of the groups may have fewer members than specified.

Python Code:

```
students = int(input("How many students on the course?"))
group = int(input("Desired group size?"))
import math
print(math.ceil(students/group))
```

Output:

Conclusion:

Assignment No – 3

Aim: Write a program to demonstrate working of control structures.

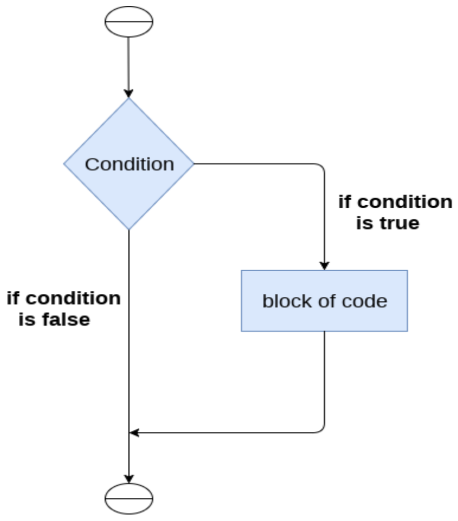
Theory:

Python If-else statements

Decision making is the most important aspect of almost all the programming languages. As the name implies, decision making allows us to run a particular block of code for a particular decision. Here, the decisions are made on the validity of the conditions. Condition checking is the backbone of decision making.

The if statement

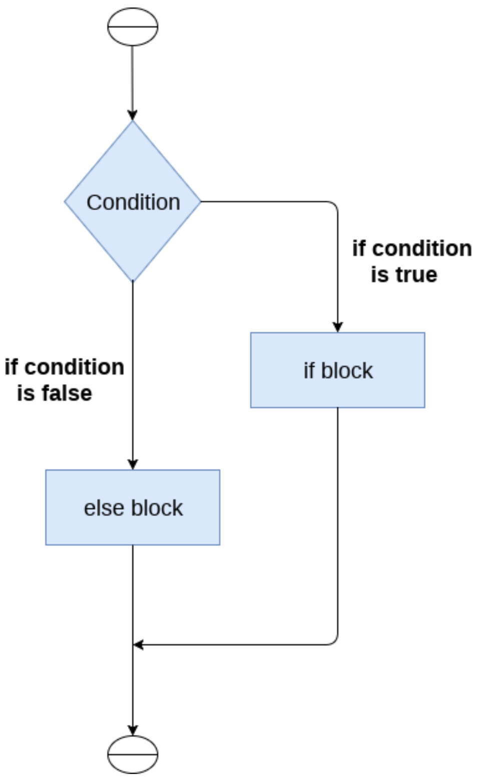
The if statement is used to test a particular condition and if the condition is true, it executes a block of code known as if-block. The condition of if statement can be any valid logical expression which can be either evaluated to true or false.



The if-else statement

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

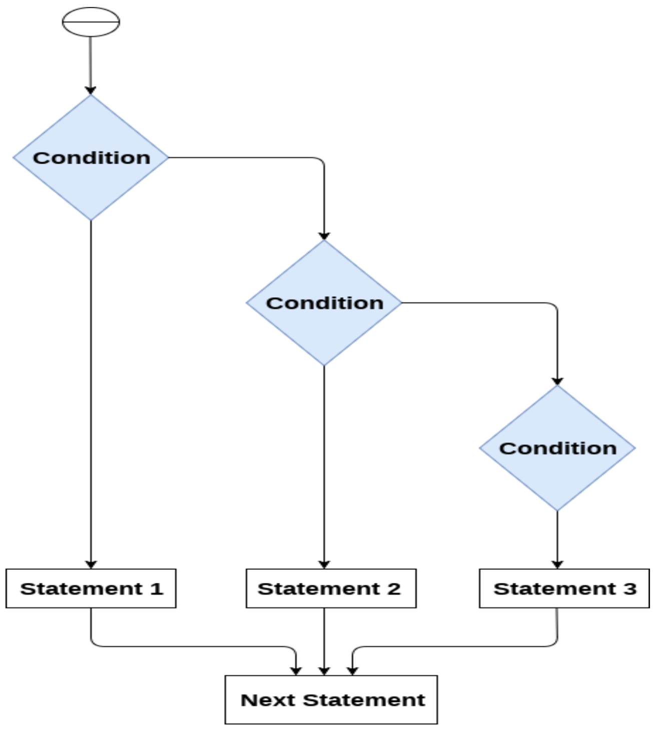
If the condition is true, then the if-block is executed. Otherwise, the else-block is executed.



The elif statement

The elif statement enables us to check multiple conditions and execute the specific block of statements depending upon the true condition among them. We can have any number of elif statements in our program depending upon our need. However, using elif is optional.

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



Loops:

Loop is a sequential set of instructions which gets executed multiple times to reduce minimize the repetition of code.

In Python, we have two types of loops :

i. for loop
ii. while loop

To understand the functioning and flow of a loop, you must get familiar with the term 'block'. A block is the smallest unit in a loop which performs one particular task.

'For' loop :
Syntax :

```
for object in range(initialization, limit, update ):
        Statements
```

The above given syntax is of for loop where we put the object name after 'for' and the limit inside 'range( )'.

Program:

```
n = 3
for i in range (0, n)
print (i)
```

Output:

```
0
1
2
3
```

'While' loop :

Syntax:

```
while expression:
statements
```

The above statement is for while loop, where the testing condition is placed after while and it is followed by the statements placed in the loop body.
Program:

```
count=0
while (count < 3):
        count = count + 1
        print (“Hello!”)
```

Output:

```
Hello!
Hello!
Hello!
```

Nested loops :

A nested loop is an inner loop in the loop body of the outer loop. The inner or outer loop can be any type, such as a while loop or for loop. For example, the outer for loop can contain a while loop and vice versa.

```
for i in range(1, 5):
    for j in range(i):
        print (i, end= ‘  ’)
    print()
```

Output:

```
1
2  2
3  3  3
4  4  4  4
```

Source Code:

Q.1 Write a program that prints your name 5 times.

Python Code:

```
for i in range(5):
    print("Hello world")
```

Q. 2 Write a program that asks the user to enter a length in centimeters. If the user enters a negative length, the program should tell the user that the entry is invalid. Otherwise, the program should convert the length to inches and print out the result. There are 2.54 centimeters in an inch.

Python Code:

```
y=eval(input('enter a length in centimeters'))
if (y<0):
   print('Invalid Entry')
else:
   print('Inches = ',y/2.54)
```

Q. 3 Ask the user to enter a temperature in Celsius. The program should print a message based

on the temperature:
• If the temperature is less than -273.15, print that the temperature is invalid because it is below absolute zero.
• If it is exactly -273.15, print that the temperature is absolute 0.
• If the temperature is between -273.15 and 0, print that the temperature is below freezing.
• If it is 0, print that the temperature is at the freezing point.
• If it is between 0 and 100, print that the temperature is in the normal range.
• If it is 100, print that the temperature is at the boiling point.
• If it is above 100, print that the temperature is above the boiling point.

Python Code:

```
t=eval(input('Enter the Temperature in Celsius'))
if t<-273.15:
    print('Invalid Temperature')
elif t==-273.15:
    print('The temperature is absolute 0')
elif t>-273.15 and t<0:
    print('The temperature is below freezing')
elif t==0:
    print('The temperature is at the freezing point.')
elif t>0 and t<100:
    print('The temperature is in the normal range.')
elif t==100:
    print('The temperature is at the boiling point.')
elif t>100:
    print('The temperature is above the boiling point.')
```

Q. 4 Generate a random number between 1 and 10. Ask the user to guess the number and print a message based on whether they get it right or not.

Python Code:

```
from random import randint
x = randint(1,10)
y=eval(input('Guess the number '))
if y==x:
    print('Your guess is correct!')
else:
    print('Your guess is wrong! The number is: ',x)
```

Q. 5 Write a program to find the second maximum in a list.

Python Code:

```
list1 = [10, 20, 4, 45, 99]

mx = max(list1[0], list1[1])
secondmax = min(list1[0], list1[1])
n = len(list1)
```

```
for i in range(2,n):
    if list1[i] > mx:
        secondmax = mx
        mx = list1[i]
    elif list1[i] > secondmax and mx != list1[i]:
        secondmax = list1[i]
    elif mx == secondmax and secondmax != list1[i]:
          secondmax = list1[i]
print (list1)
print("Second highest number is : ",str(secondmax))
```

Output:

Conclusion:

Assignment No.: - 4

Aim: Write a program to demonstrate working of different types of functions.

Theory:

In Python, a function is a group of related statements that work together to complete a particular job. The partitioning of our software into smaller, modular chunks is made more accessible by using functions. As the size of our program increases, functions assist in making it more structured and manageable. Moreover, it makes the code reusable and gets rid of duplication.

1. **User-Defined Function:**

User-defined functions are custom procedures that user's program and implement in their software. These functions allow users to define reusable blocks of code that perform specific actions or tasks. Users can name functions, determine input parameters, and write code for the function's functionality.

```
def add_two_numbers(x,y):
        return x+y
add_two_numbers(x=5,y=10)
```

This code defines a user-defined function named "add_two_numbers" which takes two arguments (x and y) and returns their sum.

2. **Build In Function:**

Build-in functions in Python are pre-defined functions that can be used without additional code. They are always available for use in any Python program and do not need to be imported or defined by the programmer. Examples of built-in functions include print(), input(), len(), str(), int(), float(), list(), tuple(), dict(), and range().

Python's wide range of built-in functions is one of the reasons why it is a popular and versatile programming language. These functions perform everyday tasks such as getting user input, printing output, determining the length of objects, converting between data types, and generating sequences. The availability of built-in functions simplifies basic tasks and makes Python easy for beginners yet powerful enough for complex projects.

```
Print(abs(-5))
```

Source Code:

Q.1 Write a program that prints Hello word using function.

```
*******************
*                 *
*                 *
*******************
```

Python Code:

```
def print_hello():
   print('Hello!')
print_hello()
```

Q.2 Print a box like the one below using function.

```
Hello Hello Hello
Hello Hello Hello Hello Hello
Hello Hello
```

Python Code:

```
def draw_square():
      print('*' * 19)
      print('*', ' '*15, '*')
      print('*', ' '*15, '*')
      print('*' * 19)
```

Q.3 Print a box like the one below using function.

Python Code:

```
def print_hello(n):
   print('Hello ' * n)
   print()
print_hello(3)
print_hello(5)
times = 2
print_hello(times)
```

Q. 4 Write Python program to converts temperatures from Celsius to Fahrenheit using function and return the converted value to the calling function.

Python Code:

```
def convert(t):
     return t*9/5+32
print(convert(20))
```

Q. 5 Write a function called first_diff that is given two strings and returns the first location in which the strings differ. If the strings are identical, it should return -1.

Python Code:

```
def first_difference(str1, str2):
    if str1 == str2:
        return -1
    else:
        for str1, str2 in zip(str1, str2):
            if str1 != str2:
                return str1

string1 = input("Enter first string:")
string2 = input("Enter second string:")
print(first_difference(string1, string2))
```

Output:

Conclusion:

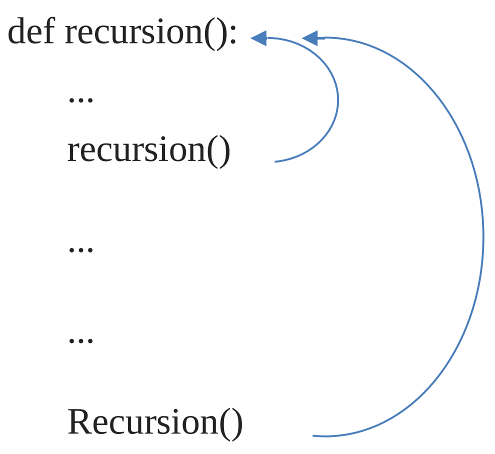
Assignment No. 5

Aim: Write a program to demonstrate working of recursion.

Theory:

Recursion is the process a procedure goes through when one of the steps of the procedure involves invoking the procedure itself. A procedure that goes through recursion is said to be 'recursive'.

This has the benefit of meaning that you can loop through data to reach a result. The developer should be very careful with recursion as it can be quite easy to slip into writing a function which never terminates, or one that uses excess amounts of memory or processor power. However, when written correctly recursion can be a very efficient and mathematically-elegant approach to programming.



```
def recursion():
    ...
    recursion()

    ...

    ...

    Recursion()
```

```
#function
def fact(num):
   if num == 1:
          return 1
  else:
        return (num * fact(num-1))
#Body of the program
num = int(input("Enter a number: "))
result = fact(num)
print("The factorial of the given number", num, "is", result)
```

Source Code:

Q. 1 Write Python program to print the factorial value of the given number.

Python Code:

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fact(n-1)

n = int(input("Enter any number:"))
print(fact(n))
```

Q. 2 Write Python program to print the nth Fibonacci Number.

Python Code:

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

n = int(input("Enter nth position number: "))
print(fibonacci(n))
```

Q. 3 Write a Python program to get the sum of a non-negative integer using recursion technique.

Python Code:

```
def sumDigits(n):
    if n == 0:
        return 0
    else:
        return n % 10 + sumDigits(int(n / 10))
n = int(input("Enter any number: "))
print(sumDigits(n))
```

Q.4 Write a Python program to calculate the sum of the positive integers of n + (n-2) + (n-4) ... (until n-x =< 0) using recursion.

Python Code:

```
def sum_series(n):
   if n < 1:
      return 0
   else:
      return n + sum_series(n - 2)

n = int(input("Enter any number: "))
print(sum_series(n))
```

Q. 5 Write a Python program to calculate the value of 'a' to the power of 'b' using recursion.

Python Code:

```
def power(a, b):
   if b == 0:
      return 1
   elif a == 0:
      return 0
   elif b == 1:
      return a
   else:
      return a * power(a, b - 1)
print(power(3, 4))
```

Output:

Conclusion:

Assignment No 6

AIM: Write a program to perform different operations on list:

i. Append
ii. Extend
iii. Insert
iv. Remove
v. Pop
vi. Slice.

Theory:

- A Python list is defined as an ordered, mutable, and heterogeneous collection of objects.
- Order here implies that the gathering of objects follows a particular order.
- Mutable means the list can be mutated or changed.
- Heterogeneous implies that programmer will be able to mix any kind of object, or data type, within a List like an integer, string, or even another list.
- Lists are contained within a collection of square brackets [ ] and each element is separated by a comma.

Some of the list functions are

1. sort(): Sorts the list in ascending order.
2. append(): Adds one element to a list.
3. extend(): Adds multiple elements to a list.
4. index(): Returns the first appearance of a particular value.
5. max(list): It returns an item from the list with a max value.
6. min(list): It returns an item from the list with a min value.
7. len(list): It gives the overall length of the list.
8. clear(): Removes all the elements from the list.
9. insert(): Adds a component at the required position.
10. count(): Returns the number of elements with the required value.
11. pop(): Removes the element at the required position.
12. remove(): Removes the primary item with the desired value.
13. reverse(): Reverses the order of the list.
14. copy(): Returns a duplicate of the list.

Source Code:

Q. 1 Write a Python code to add an item at the end of the list.

Python Code:

```
currencies = ['Dollar', 'Euro', 'Pound']
currencies.append('Yen')
print(currencies)
```

Q. 2 Write a Python code to extend the list with other list items.

Python Code:

```
numbers1 = [3, 4, 5]
numbers2 = [10, 20]
numbers2.extend(numbers1)
print(f"numbers1 = {numbers1}")
print(f"numbers2}")
```

Q. 3 Write a Python code to inserts an element to the list at the specified index.

Python Code:

```
vowel = ['a', 'e', 'i', 'u']
vowel.insert(3, 'o')
print('List:', vowel)
```

Q. 4 Write a Python code to remove the first matching element from the list.

Python Code:

```
prime_numbers = [2, 3, 5, 7, 9, 11]
prime_numbers.remove(5)
print('Updated List: ', prime_numbers)
```

Q. 5 Write a Python code to removes the item at the specified index.

Python Code:

```
prime_numbers = [2, 3, 5, 7]
removed_element = prime_numbers.pop(2)
print('Removed Element:', removed_element)
print('Updated List:', prime_numbers)
```

Q. 6 Write a Python code to removes slices from a list.

Python Code:

```
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9]
del my_list[2]
print(my_list)
```

```
del my_list[1:4]
print(my_list)
del my_list[:]
print(my_list)
```

Output:

Conclusion:

Assignment No 7

Aim: Write a program to perform different operations on set:

i) Update
ii) Remove the element
iii) Clear
iv) pop

Theory:

- A Set in Python is an unordered collection data type that is iterable, mutable and has no duplicate elements.
- Sets are used to store multiple items in a single variable.
- Sets are written with curly brackets.

Characteristics of Sets

The Sets has the following characteristics:

- The sets are unordered
- The element of the sets are unindexed.
- The sets are the mutable type.
- A sets can store multiple items in a single variable.

Operations on sets:

i. Update:

- Once a set is created, you cannot change its items, but you can add new items.
- To add items from another set into the current set, use the update () method.

Python Code:

```
thisset = {"apple", "banana", "cherry"}
tropical = {"pineapple", "mango", "papaya"}
thisset. update(tropical)
print(thisset)
```

ii. Remove the element

- To remove an item in a set, use the remove () method.

Python Code:

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

iii. Clear

- The clear () method empties the set:

Python Code:

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

iv. Pop

- You can also use the pop () method to remove an item, but this method will remove a random item, so you cannot be sure what item that gets removed.
- The return value of the pop () method is the removed item.
  e.g. Remove a random item by using the pop() method:

Python Code:

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

Source Code:

```
#update set
      thisset = {"apple", "banana", "cherry"}
      tropical = {"pineapple", "mango", "papaya"}
      thisset. update(tropical)
      print(thisset)

 #delete set
      thisset = {"apple", "banana", "cherry"}
              thisset.remove("banana")
              print(thisset)
      thisset = {"apple", "banana", "cherry"}
             thisset.clear()
             print(thisset)
             thisset = {"apple", "banana", "cherry"}
             x = thisset.pop()
             print(x)
             print(thisset)
```

Conclusion: Students implemented different operations on sets.

Viva Questions:

Q1. What is a set in Python?

Q2. How do you create an empty set in Python?

Q.3.What is output for following?

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)
```

Q.4. Define characteristics of sets.

Q.5.How to update element in set?

Assignment No 8

Aim: Write a program to perform different operations on tuple:

i. Accessing
ii. Concatenation
iii. Slicing
iv. Deleting

Theory:

- Python Tuple is a collection of Python Programming objects much like a list.
- The sequence of values stored in a tuple can be of any type, and they are indexed by integers.
- Values of a tuple are syntactically separated by ‘commas‘. Although it is not necessary, it is more common to define a tuple by closing the sequence of values in parentheses. This helps in understanding the Python tuples more easily.

1. Accessing

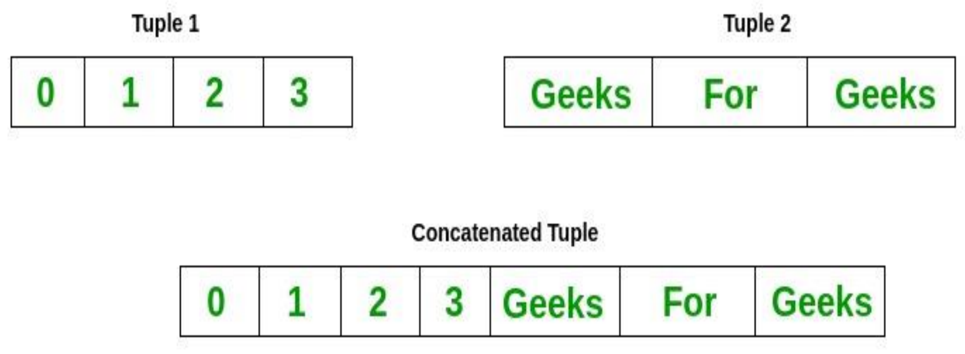
- Tuples are immutable, and usually, they contain a sequence of heterogeneous elements that are accessed via unpacking or indexing (or even by attribute in the case of named tuples)
- In unpacking of tuple number of variables on the left-hand side should be equal to a number of values in given tuple a.

Python Code:

```
e.g. Accessing Tuple
Tuple1 = tuple("Geeks")
print("\nFirst element of Tuple: ")
print(Tuple1[0])
         Tuple1 = ("Geeks", "For", "Geeks")
          a, b, c = Tuple1
print("\nValues after unpacking: ")
print(a)
print(b)
print(c)
```

Concatenation of Tuples

- Concatenation of tuple is the process of joining two or more Tuples. Concatenation is done by the use of '+' operator.
- Concatenation of tuples is done always from the end of the original tuple. Other arithmetic operations do not apply on Tuples.
- Only the same datatypes can be combined with concatenation, an error arises if a list and a tuple are combined.



Python Code:

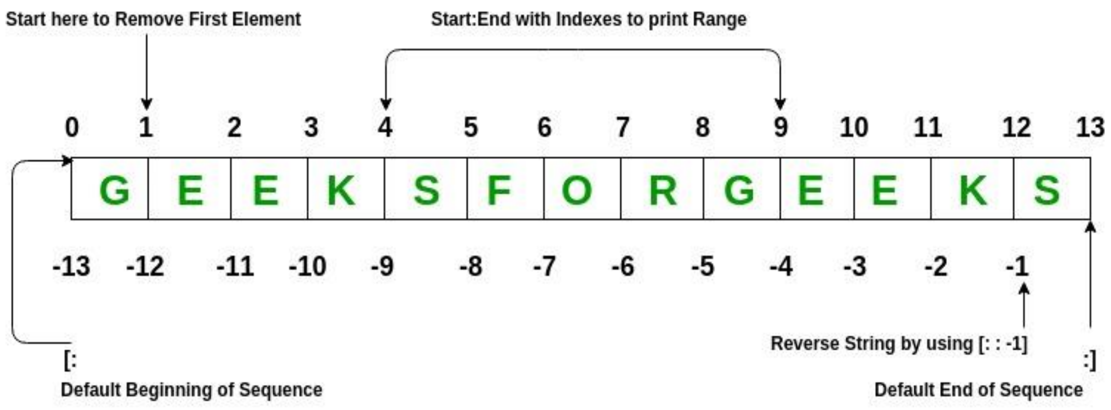
```
# Concatenation of tuples
Tuple1 = (0, 1, 2, 3)
Tuple2 = ('Geeks', 'For', 'Geeks')

Tuple3 = Tuple1 + Tuple2

# Printing first Tuple
print("Tuple 1: ")
print(Tuple1)

# Printing Second Tuple
print("\nTuple2: ")
print(Tuple2)

# Printing Final Tuple
print("\nTuples after Concatenation: ")
print(Tuple3)
```

Slicing of Tuple

- Slicing of a Tuple is done to fetch a specific range or slice of sub-elements from a Tuple.
- Slicing can also be done to lists and arrays.
- Indexing in a list results to fetching a single element whereas Slicing allows

to fetch a set of elements.

- Negative Increment values can also be used to reverse the sequence of Tuples.


Start here to Remove First Element
Start:End with Indexes to print Range
0 1 2 3 4 5 6 7 8 9 10 11 12 13
G E E K S F O R G E E K S
-13 -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1
Reverse String by using [: : -1]
[:
Default Beginning of Sequence
:]
Default End of Sequence

Python Code:

```
# Slicing of a Tuple
# with Numbers
Tuple1 = tuple('GEEKSFORGEEKS')

# Removing First element
print("Removal of First Element: ")
print(Tuple1[1:])

# Reversing the Tuple
print("\nTuple after sequence of Element is reversed: ")
print(Tuple1[::-1])

# Printing elements of a Range
print("\nPrinting elements between Range 4-9: ")
print(Tuple1[4:9])
```

Deleting a Tuple

- Tuples are immutable and hence they do not allow deletion of a part of it. The entire tuple gets deleted by the use of del() method.
- Printing of Tuple after deletion results in an Error.

Python Code:

```
# Deleting a Tuple
Tuple1 = (0, 1, 2, 3, 4)
del Tuple1
print(Tuple1)
```

Source Code:

```
T = ("apple", "banana", "cherry","mango","grape","orange")
print("\n Created tuple is :",T)
print("\n Second fruit is :",T[1])
print("\n From 3-6 fruits are :",T[3:6])
print("\n List of all items in Tuple :")
for x in T:
  print(x)
if "apple" in T:
  print("\n Yes, 'apple' is in the fruits tuple")
print("\n Length of Tuple is :",len(T))
```

Conclusion: Students implemented different operations on tuples.

Viva Questions:

Q.1.What is tuples?

Q.2.Which are different types of operations?

Q.3.Differentiate between list and tuples.

Q.4.How to delete or remove items from tuples?

Q.5.How to perform Concatenation in tuples?

Assignment No 8

Aim: Write a program to demonstrate the working of dictionary.

Theory:

Python Dictionary

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

- Keys must be a single element
- Value can be any type such as list, tuple, integer, etc.
- In other words, we can say that a dictionary is the collection of key-value pairs where the value can be any Python object. In contrast, the keys are the immutable Python object, i.e., Numbers, string, or tuple.

Creating the dictionary

Dictionaries are written with curly brackets, and have keys and values:

Python Code:

```
Thisdict= {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

Accessing Items

You can access the items of a dictionary by referring to its key name, inside square brackets: To get the value of ‘model’ key:

Python Code:

```
  thisdict= {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964}
x = thisdict["model"]
```

Adding Items

Adding an item to the dictionary is done by using a new index key and assigning a value to it:

```
thisdict= {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict["color"]= "red"
print(thisdict)
```

Update Dictionary

The update () method will update the dictionary with the items from a given argument. If the item does not exist, the item will be added.

The argument must be a dictionary, or an iterable object with key:value pairs.

Example Add a color item to the dictionary by using the update() method:

```
 thisdict= {
 "brand": "Ford",
 "model": "Mustang",
 "year": 1964
}
thisdict.update({"color": "red"})
```

Removing Items

There are several methods to remove items from a dictionary: Example Get your own Python Server The pop () method removes the item with the specified key name:

```
thisdict= {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
thisdict.pop("model")
print(thisdict)
```

Source Code:

```
dict1 = {'StdNo':'532','StuName': 'Naveen', 'StuAge': 21,
'StuCity': 'Hyderabad'}
print("\n Dictionary is :",dict1)
#Accessing specific values
print("\n Student Name is :",dict1['StuName'])
print("\n Student City is :",dict1['StuCity'])
#Display all Keys
print("\n All Keys in Dictionary ")
for x in dict1:
   print(x)
#Display all values
print("\n All Values in Dictionary ")
for x in dict1:
   print(dict1[x])
#Adding items
dict1["Phno"]=85457854
#Updated dictoinary
print("\n Uadated Dictionary is :",dict1)
#Change values
dict1["StuName"]="Madhu"
#Updated dictoinary
print("\n Uadated Dictionary is :",dict1)
#Removing Items
dict1.pop("StuAge");
#Updated dictoinary
print("\n Uadated Dictionary is :",dict1)
#Length of Dictionary
print("Length of Dictionary is :",len(dict1))
#Copy a Dictionary
dict2=dict1.copy()
#New dictoinary
print("\n New Dictionary is :",dict2)
#empties the dictionary
dict1.clear()
print("\n Uadated Dictionary is :",dict1)
```

Conclusion: Students implemented working of dictionary.

Viva Questions:
Q.1. What is a dictionary?
Q.2. How do you add an element in Dictionary?
Q.3.What are different ways of creating a Dictionary?
Q.4. Are dictionaries case-sensitive?
Q.5. Discuss different methods used with Dictionary.

Assignment No. 10

Aim: Write a program demonstrates various operations on strings:

I. Slicing
II. Concatenating
III. Finding the length of the string
IV. Converting the string to uppercase and lowercase
V. Replacing a substring with another substring
VI. Splitting the string into a list of substrings

Theory:

Python string is the collection of the characters surrounded by single quotes, double quotes, or triple quotes. The computer does not understand the characters; internally, it stores manipulated character as the combination of the 0's and 1's. Each character is encoded in the ASCII or Unicode character. So we can say that Python strings are also called the collection of Unicode characters. In Python, strings can be created by enclosing the character or the sequence of characters in the quotes. Python allows us to use single quotes, double quotes, or triple quotes to create the string.

Creating String in Python

We can create a string by enclosing the characters in single-quotes or double-quotes. Python also provides triple-quotes to represent the string, but it is generally used for multiline string or docstrings.

- **Indexing :**
  Like other languages, the indexing of the Python strings starts from 0.
- **Slicing :**
  You can return a range of characters by using the slice syntax. The slice starts at index 1 but ends at index 4.
- **Concatenating:**
  To concatenate, or combine, two strings you can use the + operator.
- **Finding the length of the string:**
  To get the length of a string, use the len() function.
- **Converting the string to uppercase and lowercase :**
  upper() method on a string converts all of the characters to uppercase, whereas the lower() method converts all of the characters to lowercase
- **Replacing a substring with another substring :**
  The replace() method replaces a specified phrase with another specified phrase
- **Splitting the string into a list of substrings**
  Splits a string up into its elements into a list.

Source Code:

Q. 1 Write a Python code to apply indexing on string.
For example, The string "mystring" is indexed as given in the below figure.

Python Code:

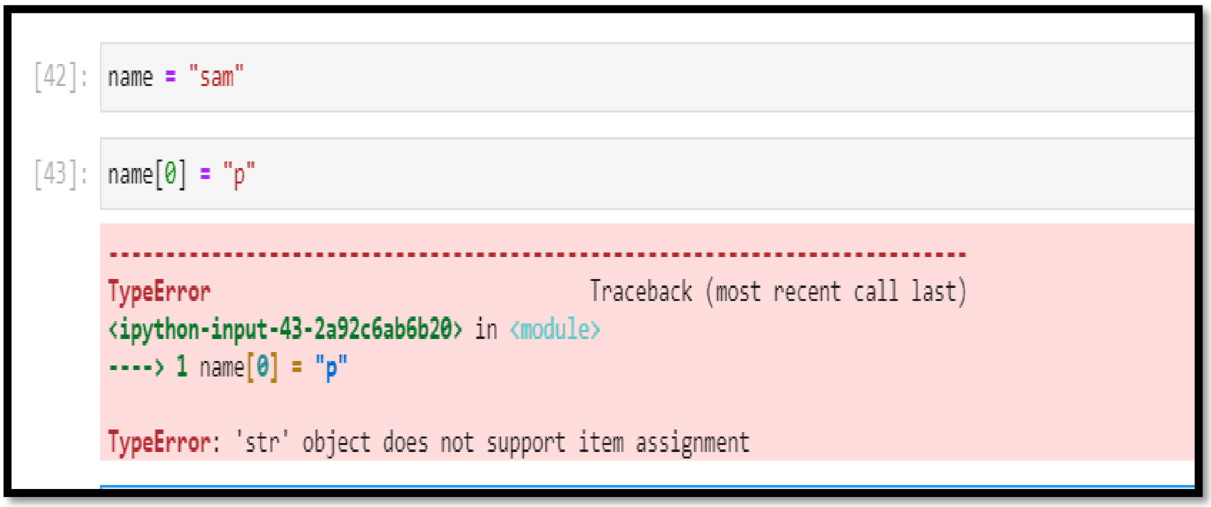
```
mystring = "my world"
mystring[0]
        Output : ‘m’

mystring = "my world"
mystring[-1]
        Output : ‘m’
```

Q. 2 Write a Python code to apply Slicing on string.

Python Code:

```
mystring = "abcdefgh"
mystring[2:]
        Output: 'cdefgh'

mystring[:5]
        Output: 'abcde'

mystring[3:6]
        Output: 'def'

mystring[::]
        Output: 'abcdefgh'
```

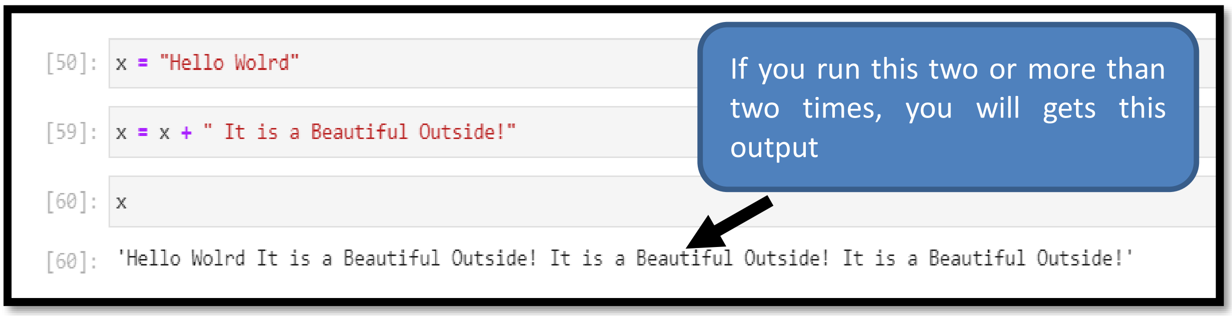
Q. 3 Write a Python code to Concatenating the string.



```
[42]: name = "sam"

[43]: name[0] = "p"

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-43-2a92c6ab6b20> in <module>
----> 1 name[0] = "p"

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

Python Code:

```
name = "sam"
last_letters = name[1:]
'p' + last_letters
        Output: 'pam'

x = "Hello Wolrd"
x = x + " It is a Beautiful Outside!"
x
        Output: 'Hello World It is a Beautiful Outside!'
```



```
[50]: x = "Hello Wolrd"

[59]: x = x + " It is a Beautiful Outside!"

[60]: x

[60]: 'Hello Wolrd It is a Beautiful Outside! It is a Beautiful Outside! It is a Beautiful Outside!'
```

If you run this two or more than two times, you will gets this output

Q. 4 Write a Python code to Find the length of the string.

Python Code:

```
len("hello")
        Output: 5

len("I am")
        Output: 4
```

Q. 5 Write a Python code to Converting the string to uppercase and lowercase.

Python Code:

```
x = 'Hello World'
x.upper()
        Output: 'HELLO WORLD'
x.lower()
        Output: 'hello world'
```

Q. 6 Write a Python code to Replacing a substring with another substring.

Python Code:

```
string = "Hello World"
new_string = string.replace("Hello","Happy")
print(new_string)
        Output: Happy World
```

Q. 7 Write a Python code to Splitting the string into a list of substrings

Python Code:

```
x.split()
        Output: ['Hello', 'World']

x = 'Hi This is a string'
x.split('i')
        Output: ['H', ' Th', 's ', 's a str', 'ng']
```

Conclusion: Strings in Python are an integral part of the programming language and are used to represent text data. Strings are immutable so they cannot be changed once they are created. However, you can perform various operations on strings without modifying the original value. These operations include concatenation, slicing, indexing, and utilizing a wide range of built-in methods for string manipulation.

Viva Questions:

1. **What is the meaning of string immutability in python?**
2. **How do you convert a string to a list of characters?**
3. **Write a Python function to remove duplicate characters from a string.**
4. **How can you concatenate two strings in Python?**
5. **Explain the difference between single, double, and triple quotes for defining strings.**

Assignment No. 11

Aim: Write a program to demonstrate different operations on file:

i. Create
ii. Open
iii. Read
iv. Write
v. Update
vi. Delete

Theory:

File handling in Python is a powerful and versatile tool that can be used to perform a wide range of operations. However, it is important to carefully consider the advantages and disadvantages of file handling when writing Python programs, to ensure that the code is secure, reliable, and performs well.

- **Python File Handling**

Python 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, 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.

Apart from the most commonly used file handling functions: open(), read(), write(), and close(), there are a lot more functions that are used for file handling in Python.

Here is a list of some commonly used file-handling functions in Python.

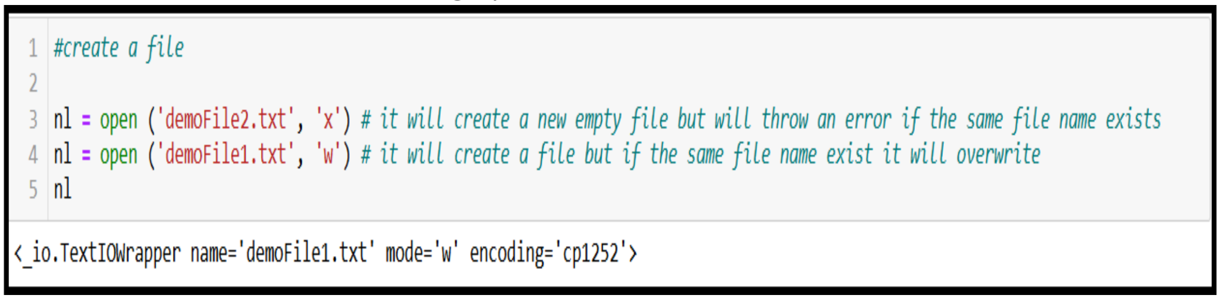
Function	**Description**
open()	Used to open a file.
readable()	Used to check whether the file can be read or not.
readline()	Used to read a single line from the file.
readlines()	Used to read all the content in the form of lines.
read()	Used to read whole content at a time.
writable()	Used to check whether the file is writable or not.
write()	Used to write a string to the file.
writelines()	Used to write multiple strings at a time.
close()	Used to close the file from the program.

Mode	Description
r	Read Mode (default value)
w	Write Mode (file is opened in write-only mode)
a	Append Mode (Opens a file for appending at the end of the file without truncating)
x	Create Mode (Creates a new file but will return an error if the file already exists.
t	Open the file in text mode.
b	Opens the file in binary mode. Binary mode returns bytes. It is mainly used while dealing with the non-text file such as images.
+	Opens the file for updating (reading and writing)

Source Code:

i. Create

To create the file use following syntax:

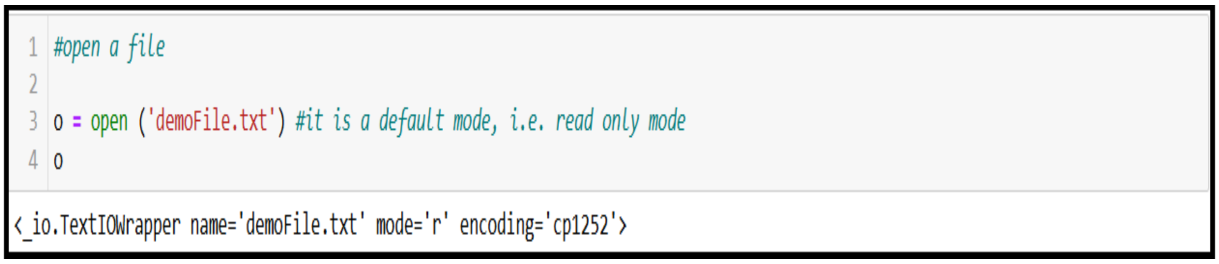


```
#create a file

nl = open ('demoFile2.txt', 'x') # it will create a new empty file but will throw an error if the same file name exists
nl = open ('demoFile1.txt', 'w') # it will create a file but if the same file name exist it will overwrite
nl
```

```
<_io.TextIOWrapper name='demoFile1.txt' mode='w' encoding='cp1252'>
```

ii. Open

To open the file, use the built-in open() function.
The open() function returns a file object, which has a read() method for reading the content of the file:

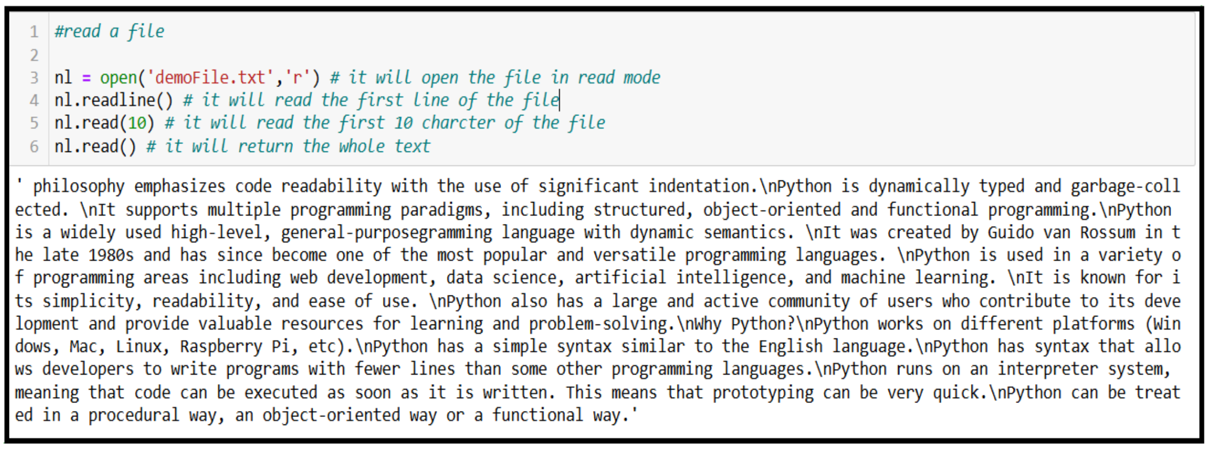


```
#open a file

o = open ('demoFile.txt') #it is a default mode, i.e. read only mode
o
```

```
<_io.TextIOWrapper name='demoFile.txt' mode='r' encoding='cp1252'>
```

iii. Read

By default the read() method returns the whole text, but you can also specify how many lines you want to return:

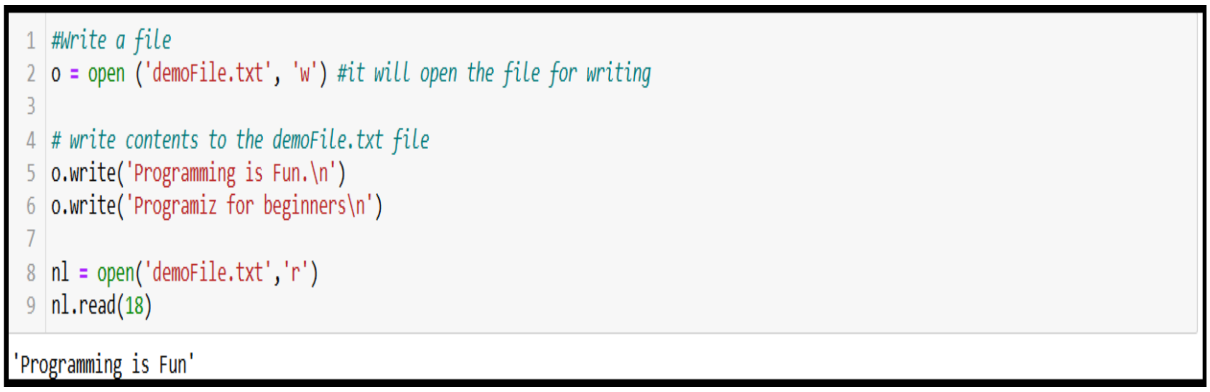


```python
#read a file

nl = open('demoFile.txt','r') # it will open the file in read mode
nl.readline() # it will read the first line of the file
nl.read(10) # it will read the first 10 charcter of the file
nl.read() # it will return the whole text
```

```
' philosophy emphasizes code readability with the use of significant indentation.\nPython is dynamically typed and garbage-coll
ected. \nIt supports multiple programming paradigms, including structured, object-oriented and functional programming.\nPython
is a widely used high-level, general-purposegramming language with dynamic semantics. \nIt was created by Guido van Rossum in t
he late 1980s and has since become one of the most popular and versatile programming languages. \nPython is used in a variety o
f programming areas including web development, data science, artificial intelligence, and machine learning. \nIt is known for i
ts simplicity, readability, and ease of use. \nPython also has a large and active community of users who contribute to its deve
lopment and provide valuable resources for learning and problem-solving.\nWhy Python?\nPython works on different platforms (Win
dows, Mac, Linux, Raspberry Pi, etc).\nPython has a simple syntax similar to the English language.\nPython has syntax that allo
ws developers to write programs with fewer lines than some other programming languages.\nPython runs on an interpreter system,
meaning that code can be executed as soon as it is written. This means that prototyping can be very quick.\nPython can be treat
ed in a procedural way, an object-oriented way or a functional way.'
```

iv. Write and Update

we can write the content of a file using the write() function in Python. For updates in the file, we use same write() function as shown below:

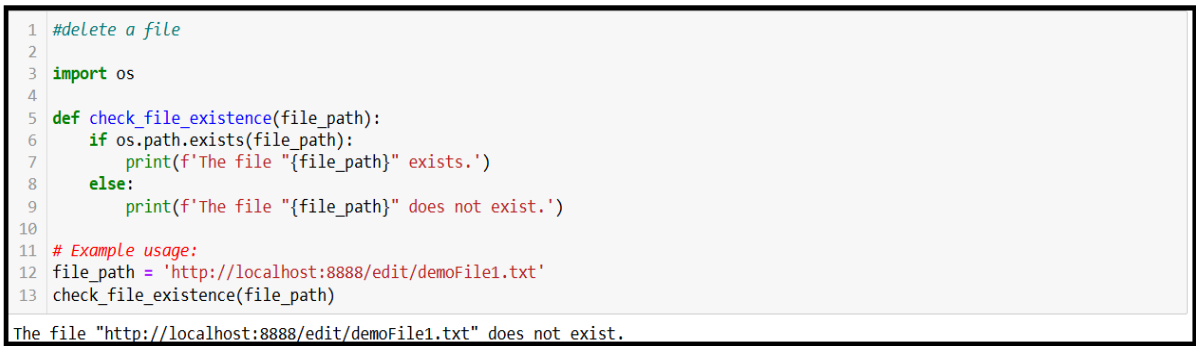


```python
#Write a file
o = open ('demoFile.txt', 'w') #it will open the file for writing

# write contents to the demoFile.txt file
o.write('Programming is Fun.\n')
o.write('Programiz for beginners\n')

nl = open('demoFile.txt','r')
nl.read(18)
```

```
'Programming is Fun'
```

vi. Delete

If the file is located in a different location, you will have to specify the file path, like this:

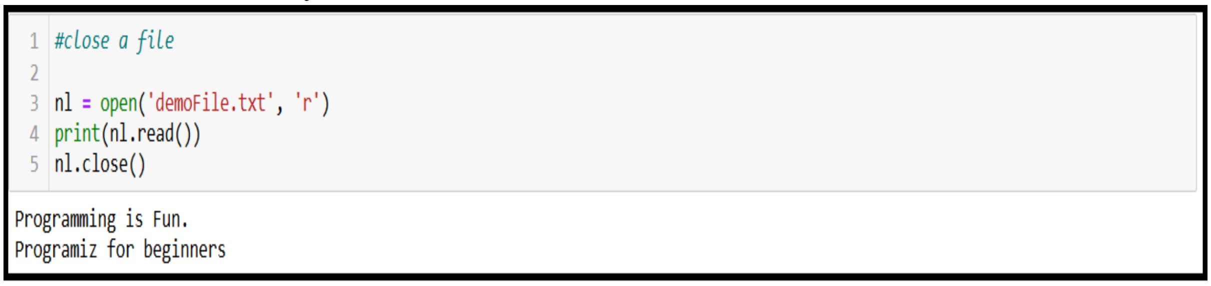


```python
#delete a file

import os

def check_file_existence(file_path):
    if os.path.exists(file_path):
        print(f'The file "{file_path}" exists.')
    else:
        print(f'The file "{file_path}" does not exist.')

# Example usage:
file_path = 'http://localhost:8888/edit/demoFile1.txt'
check_file_existence(file_path)
```

```
The file "http://localhost:8888/edit/demoFile1.txt" does not exist.
```

vii. Close

Close the file when you are finish with it:



```python
#close a file

nl = open('demoFile.txt', 'r')
print(nl.read())
nl.close()
```

```
Programming is Fun.
Programiz for beginners
```

Conclusion: File handling in python is very easy for user. It allows us to store data that can be accessed by our code for various purposes like reading, writing, modifying and deleting data from files.

Viva Questions:

1. **What is file handling in Python?**
2. **Which function is used to open a file in Python?**
3. **Write a function in Python to count and display the total number of words in a text file.**
4. **What is the purpose of the 'rb' mode in file opening?**
5. **Write a function in Python to count uppercase character in a text file.**
6. **Write a function in python to read the content from a text file (take any .txt file) line by line and display the same on screen.**

Assignment No. 12

Aim: Mini Project in Python

Including a mini project in the Second-Year B. Tech Python Programming Lab aims to achieve several educational and practical objectives:

Objectives:

1. **Hands-on Experience**:

Provide students with practical experience in applying Python programming concepts to real-world problems.

2. **Skill Development**:

Enhance coding skills by requiring students to design, develop, test, and debug their own projects.

3. **Problem-Solving Abilities**:

Foster critical thinking and problem-solving skills by challenging students to devise solutions for complex tasks.

4. **Understanding Software Development Life Cycle (SDLC)**:

Introduce students to various phases of SDLC including planning, analysis, design, implementation, and testing.

5. **Project Management**:

Teach students the basics of project management, including time management, resource allocation, and documentation.

6. **Collaboration and Teamwork**:

Encourage teamwork and collaboration, especially if the project is done in groups, promoting communication and cooperation skills.

7. **Application of Theoretical Knowledge**:

Bridge the gap between theoretical concepts and practical applications, reinforcing the understanding of concepts learned in lectures.

8. **Exposure to Tools and Technologies**:

Provide exposure to development environments, version control systems, and other tools commonly used in software development.

9. **Innovation and Creativity**:

Inspire creativity and innovation by allowing students to explore new ideas and technologies in their projects.

10. **Self-Learning and Research**:

Motivate students to engage in self-learning and research to overcome challenges encountered during the project.

11. **Evaluation and Feedback**:

Facilitate continuous evaluation and constructive feedback from instructors, helping students improve their coding practices and methodologies.

12. **Portfolio Development**:

Help students build a portfolio of projects that can be showcased to potential employers or used as a foundation for future academic or professional endeavors.

Specific Learning Outcomes:

1. **Programming Proficiency**:

Demonstrate proficiency in Python programming by developing functional and efficient code.

2. **File Handling and Data Management**:

Implement file handling and data management techniques to store, retrieve, and manipulate data.

3. **User Interface Design**:

Develop basic user interfaces (command-line or GUI) to interact with users.

4. **Error Handling**:

Apply error handling techniques to create robust and user-friendly applications.

5. **Modular Programming**:

Write modular code using functions, classes, and modules to enhance code readability and reusability.

6. **Documentation**:

Prepare comprehensive documentation that explains the project's design, implementation, and usage.

7. **Testing and Debugging**:

Develop skills in testing and debugging to ensure the reliability and correctness of the software.

Some titles for mini projects in Python suitable for second-year B.Tech students are as follow.

These projects will help students apply their Python programming skills to solve practical problems and develop a deeper understanding of software development principles.

1. Library Management System
2. Online Quiz Application
3. Bank Account Management System
4. Inventory Management System
5. Weather Forecast Application
6. Simple Chat Application
7. Personal Expense Tracker
8. Hotel Booking System
9. To-Do List Application
10. Employee Management System
11. Flight Reservation System
12. E-commerce Website Scraper
13. Student Result Management System
14. Automated Email Sender
15. Movie Recommendation System
16. Text-Based Adventure Game
17. Simple File Encryption/Decryption Tool

18. Note-Taking Application
19. Online Polling System
20. Health Management System
21. Social Media Post Scheduler
22. Traffic Management System
23. Vehicle Rental System
24. Voting System
25. Portfolio Website Generator

Sample Short report

Title: "Student Management System"

Objective:

The objective of this mini project is to develop a simple yet functional Student Management System using Python. This project aims to introduce students to basic concepts of Python programming, file handling, and database operations.

Introduction:

The Student Management System (SMS) is a software application designed to manage student data, including adding new students, updating existing student information, deleting student records, and displaying a list of students. This project will help students understand how to interact with files and databases in Python and provide practical experience in building a small-scale application.

Requirements:

- Basic knowledge of Python programming
- Understanding of file handling in Python
- Familiarity with basic data structures such as lists and dictionaries

Tools and Technologies:

- Python 3.x
- A text editor or Integrated Development Environment (IDE) like
- VSCode, PyCharm, or Jupyter Notebook

Project Outline:

1. Setup and Installation:

- Install Python from the official website
- Set up the development environment

2. Project Structure:

- Create a project directory
- Define the necessary files and folders (e.g., main.py, students.txt)

3. Features and Functionalities:

- **Add New Student**: Collect student details (name, roll number, class, etc.) and save them to a file.
- **View All Students**: Read and display student details from the file.
- **Search Student**: Search for a student by roll number and display their details.
- **Update Student Information**: Update the details of an existing student.
- **Delete Student**: Remove a student record from the file.
- **Exit**: Exit the application.

4. Implementation:

- **Main Menu**: Provide a menu-driven interface for user interaction.
- **File Handling**: Use file operations to read from and write to a text file.
- **Functions**: Define functions for each feature (add, view, search, update, delete).
- **Error Handling**: Implement error handling for invalid inputs and file operations.

Sample Code:

```
import os
def add_student():
    with open("students.txt", "a") as file:
        roll_number = input("Enter Roll Number: ")
        name = input("Enter Name: ")
        class_name = input("Enter Class: ")
        file.write(f"{roll_number},{name},{class_name}\n")
        print("Student added successfully.")

def view_students():
    if os.path.exists("students.txt"):
        with open("students.txt", "r") as file:
            for line in file:
                roll_number, name, class_name = line.strip().split(",")
                print(f"Roll Number: {roll_number}, Name: {name}, Class:
{class_name}")
    else:
        print("No student records found.")

def search_student():
    roll_number = input("Enter Roll Number to search: ")
    found = False
    if os.path.exists("students.txt"):
        with open("students.txt", "r") as file:
            for line in file:
                stored_roll_number, name, class_name = line.strip().split(",")
                if stored_roll_number == roll_number:
                    print(f"Roll Number: {stored_roll_number}, Name: {name},
Class: {class_name}")
                    found = True
                    break
    if not found:
        print("Student not found.")

def update_student():
    roll_number = input("Enter Roll Number to update: ")
    updated_data = []
    found = False
    if os.path.exists("students.txt"):
        with open("students.txt", "r") as file:
            for line in file:
                stored_roll_number, name, class_name = line.strip().split(",")
                if stored_roll_number == roll_number:
                    name = input("Enter New Name: ")
                    class_name = input("Enter New Class: ")
                    found = True

updated_data.append(f"{stored_roll_number},{name},{class_name}\n")

        if found:
            with open("students.txt", "w") as file:
                file.writelines(updated_data)
            print("Student updated successfully.")
        else:
            print("Student not found.")
    else:
        print("No student records found.")

def delete_student():
    roll_number = input("Enter Roll Number to delete: ")
    updated_data = []
    found = False
    if os.path.exists("students.txt"):
        with open("students.txt", "r") as file:
            for line in file:
                stored_roll_number, name, class_name = line.strip().split(",")
                if stored_roll_number != roll_number:

updated_data.append(f"{stored_roll_number},{name},{class_name}\n")
                else:
                    found = True

        if found:
            with open("students.txt", "w") as file:
                file.writelines(updated_data)
            print("Student deleted successfully.")
        else:
            print("Student not found.")
    else:
        print("No student records found.")

def main():
    while True:
        print("\nStudent Management System")
        print("1. Add Student")
        print("2. View Students")
        print("3. Search Student")
        print("4. Update Student")
        print("5. Delete Student")
        print("6. Exit")

        choice = input("Enter your choice: ")

        if choice == '1':
            add_student()
        elif choice == '2':
            view_students()
        elif choice == '3':
            search_student()
        elif choice == '4':
            update_student()
        elif choice == '5':
            delete_student()
        elif choice == '6':
            break
        else:
            print("Invalid choice, please try again.")

if __name__ == "__main__":
    main()
```

Evaluation Criteria:

- Correctness and functionality of the implemented features
- Code readability and proper use of comments
- Efficient use of file handling and data structures
- Error handling and user input validation

Conclusion:

This mini project provides a practical approach to learning Python programming by implementing a real-world application. It covers essential programming concepts and offers hands-on experience in managing data using files and basic operations.

References:

- Python Official Documentation: https://docs.python.org/3/
- Online Python Tutorials: https://www.w3schools.com/python/

Conclusion:

Including a mini project in the Python programming lab for Second-Year B.Tech students is an effective way to enhance their technical skills, promote active learning, and prepare them for future challenges in their academic and professional careers.