

Department of Data Science & Technology

Certificate

Masters in Computer Applications
Semester I (2023 – 24)

This is to certify that Mr. Karan Dinesh Panchal Roll No. 17030923033 of MCA, has satisfactorily completed the practical course "Python Programming" prescribed by the College for the Partial fulfilment of the Degree by the Somaiya Vidyavihar University, during the academic year 2023-24.

Signature of the Faculty-Incharge
Prof. Mayura Nagar

Signature of the Programme Coordinator
Dr. Bharati Wukkadada

Date of Examination
9th January 2023

Signature of the External Examiner/s

INDEX

Sr. no	Topic	Date	Page No.	Sign
1	Basic Of Python: 1. Simple Commands: <ul style="list-style-type: none"> - Write a program that prints your name. - Create a program that displays "Hello, World!" on the screen. - Write a program that asks the user for their age and then prints it - Write a Python program to add odd numbers from 1 - 10. - Write a Python program to get the Fibonacci series between 0 to 50. 	28-08-23	12	
	2. Mathematical Operators: <ul style="list-style-type: none"> - Create a program to calculate the area of a rectangle given its length and width. - Write a program that calculates the volume of a cube using its side length. - Create a simple calculator program that can perform addition, subtraction, multiplication, and division. 			
	3. Range: <ul style="list-style-type: none"> - Write a program that prints all even numbers between 1 and 50. - Create a program that generates a list of squares of numbers from 1 to 10 using a loop. - Write a program to calculate the sum of all numbers between 1 and 100. 			
2	List: 1. Create a Python program that performs basic list operations. <ul style="list-style-type: none"> - Create an empty list. - Add three different types of elements to the list (e.g., integers, strings, and floats). - Print the list. - Remove one element from the list. - Print the modified list. 	04-09-23	17	
	2. Write a python program for list indexing and slicing in Python.			

	<ul style="list-style-type: none"> - Create a list of numbers from 1 to 10. - Print the first and last elements of the list. - Print a slice of the list containing elements from index 3 to 7. - Modify the list to replace the elements from index 5 to 9 with even numbers (e.g., 10, 12, 14, ...). - Print the modified list. 			
	3. Write a python program to count occurrences of all the elements present in the list. <ul style="list-style-type: none"> - Provide a list. - Print all the elements of the list with their counts. - Print the list. 			
	4. Write a python program for List Sorting and Reversing list. <ul style="list-style-type: none"> - Create a list of unsorted numbers. - Sort the list in ascending order. - Reverse the sorted list. - Print the reversed list. 			
	5. Write a python program to demonstrate list concatenation in Python. <ul style="list-style-type: none"> - Provide two lists of names (e.g., boy's names and girl's names). - Concatenate the two lists to create a combined list of names. - Print the combined list. 			
3	Tuple <ol style="list-style-type: none"> 1. Write a python program to create a tuple of your favourite fruits. <ul style="list-style-type: none"> - Print the tuple. - Access and print individual elements of the tuple. - Attempt to modify an element in the tuple and observe the error. - Calculate and print the length of the tuple. 2. Write a python program to create a tuple containing a student's name and them scores in: <ul style="list-style-type: none"> - Use tuple unpacking to extract and print the student's name and each subject score separately. - Calculate and print the average score for the student. 3. Write a python program to create a list of tuples, each 	08-09-23	21	

	<p>containing a student's name and their exam scores.</p> <ul style="list-style-type: none"> - Sort the list of tuples in ascending order based on the average score (consider using a custom sorting key). - Print the sorted list of tuples. 			
	<p>4. Write a python program to create two tuples containing days of the week (e.g., weekdays and weekends).</p> <ul style="list-style-type: none"> - Concatenate the two tuples to create a single tuple representing all days of the week. - Create a new tuple by repeating one of the original tuples multiple times. 			
	<p>5. Create Python functions that demonstrate tuple packing and unpacking.</p> <ul style="list-style-type: none"> - Implement a function that takes multiple arguments and returns them as a tuple (packing). - Implement a function that receives a tuple as an argument and unpacks it, printing the elements. 			
4	<p>Set</p> <p>1. Write a Python program to find and print the unique elements in a given list.</p> <ul style="list-style-type: none"> - Create a list that contains duplicate elements. - Use a set to identify and print the unique elements from the list. <p>2. Write a python program for Set Operations with Two Sets: Consider two sets of numbers.</p> <ul style="list-style-type: none"> - Calculate and print the union of the two sets. - Calculate and print the intersection of the two sets. - Calculate and print the symmetric difference between the two sets. <p>3. Write a python program to Create two lists of items for books in two libraries and identify common books present in both the libraries.</p> <ul style="list-style-type: none"> - Use sets to identify common books and create a new set. - Print the set of common items. <p>4. Write a python program to create a set of unique words in the text, removing punctuation and converting all words to lowercase</p>	14-09-23	25	
5	<p>Conditional statement:</p>	15-09-23	28	

	1. Write a python program to determine the grade based on a user's input score. Give the grades starts from First class with Distinction to Fail as per the score percentage.			
	2. Write a python program for number comparison. Get two numbers from the users and compare the numbers and display the result. Result could be Like greater than, less than or equal numbers.			
	3. Write a python program to check for leap years. Take a year as an input from the user and depending on the input display appropriate messages.			
	4. Write a python program for age classifier. - If the age is less than zero, then display the message as Invalid age. - If the age in between zero to 18 then Minor. - If the age is in between 18 to 65 then Adult. - If the age is above 65 then Senior Citizen.			
6	Loops 1. Write a python program to Print the multiplication table for a given number. 2. Write a python program to create a countdown timer using a while loop. You can use sleep method for delay in displaying numbers. 3. Write a python program to print a simple pattern using nested loops. <pre> * ** *** **** ***** </pre> 4. Write a python program to print a * pyramid pattern using nested loops. <pre> * ** *** **** ***** </pre> 5. Write a python program to calculate the sum of numbers from 1 to N (e.g., N = 5)	15-09-23	32	
7	Break and Continue: 1. Write a python program to find the first even number and print.	18-09-23	36	

	<p>2. Write a python program to skip printing numbers divisible by 3 within a for loop.</p> <p>3. Write a python program to terminate a while loop when the user enters a specific input (e.g., "exit").</p> <p>4. Write a python program to skip specific values (e.g., negative numbers) when iterating through a list.</p> <p>5. Write a python program to find and print all prime numbers from 1 to 50. Use a for loop and the break statement to optimize the search for prime numbers.</p> <p>6. Write a python program that takes a list of integers and prints all positive numbers, skipping negative ones using the continue statement.</p> <p>7. Write a python program that asks the user to enter a password. If the password contains at least one uppercase letter, one lowercase letter, one digit, and is at least 8 characters long, print "Password accepted." Otherwise, print an appropriate message and continue asking for a password until a valid one is entered.</p> <p>8. Write a python program that takes a list of numbers and a sum limit. Print the numbers in the list one by one until their sum exceeds the limit, then terminate the loop using the break statement.</p> <p>9. Write a python program that takes a list of numbers and prints unique values (skipping duplicates) using the continue statement.</p>			
8	<p>String Functions:</p> <p>1. Write a python program to check if an entered string is a palindrome or not?</p> <p>2. Write a python program to split a given string into lines.</p> <p>3. Write a python program to remove specific characters from the string. (e.g. removing '!' from the string "Hello World!")</p> <p>4. Write a python program to demonstrate more on string function: len(), strip(),rstrip(), lstrip(), find(), rfind(), index(), rindex(), count(), replace(), split(), join(), upper(), lower(), swapcase(), title(), capitalize(), startswith(), endswith()</p>	21-09-23	40	
9	<p>String Operations:</p>	22-09-23	44	

1.	Write a Python program that takes two strings as input and concatenates them. Ensure that the final string is in uppercase. Calculate and print the length of a user-entered string. Remove all leading and trailing spaces from a given string.			
2.	<p>a. Write a program to find and print the index of the first occurrence of a substring in each string.</p> <p>b. Create a Python function that takes a sentence as input and replaces all occurrences of a specified word with a user-defined word.</p>			
3.	<p>a. Write a Python function that extracts the domain name from an email address using string slicing.</p> <p>b. Given a list of words separated by spaces, write a program to split the input into individual words and count the number of words.</p>			
4.	Create a Python program that accepts a user's name and age, and then prints a formatted message like "My name is [name] and I am [age] years old". Format a given string to title case, and then capitalize the first character of the result.			
5.	Write a Python program to count the number of times a specified character appears in a given string. Implement a function that searches for all occurrences of a specified word in a paragraph and counts them.			
6.	<p>a) Develop a program that reverses a given string without using string slicing.</p> <p>b) Create a Python function that accepts a sentence and removes all punctuation marks from it.</p>			
7.	<p>a) Write a Python program that takes a list of words and joins them into a single string with a comma and space as the delimiter.</p> <p>b) Develop a program that reads a paragraph of text and splits it into sentences based on period (.) as the delimiter.</p>			
8.	<p>a) Create a Python function that accepts a string and checks whether it starts with an uppercase letter.</p> <p>b) Implement a program that checks if a given string ends with a specific suffix.</p>			
9.	<p>a) Write a program that removes all leading and trailing zeros from a user-entered string of numbers.</p> <p>b) Develop a function that strips a given string of a specific character from both ends.</p>			

	10. Compare two strings entered by the user and determine if they are equal. Find the index of the first occurrence of a user-specified character in a given string.			
10	Switch Case: <ol style="list-style-type: none"> 1. Write a python program to implement switch cases using a dictionary that maps cases to function and take input from the user for case selection. 2. Write a python program that allows user to manage a to do list the program should display a menu with the following action <ul style="list-style-type: none"> - Add task to the list - Remove task from the list - Display the list 	06-10-2023	51	
11	File handling read operation <ol style="list-style-type: none"> 1. WAPP to read content of the file and count how many times letter 'a' comes in a file. 2. WAPP to read content of a line and display 'T' in place of 'E' while displaying the content of the file, all the other characters should appear as it is. 	09-10-23	55	
12	File handling write operation Create a Python program that merges the contents of multiple text files into a single file, ensuring the lines are sorted in alphabetical order.	13-10-23	57	
13	Exception Handling <ol style="list-style-type: none"> 1. Write a Python program that prompts the user to enter two numbers and then performs division. Handle the "ZeroDivisionError" exception if the second number is zero. 2. Create a Python program that reads a file specified by the user. Handle both "FileNotFoundError" and "PermissionError" exceptions. If the file exists but cannot be opened, inform the user of the issue. 3. Create a program that takes an input number from the user and checks if it's a prime number. Implement the logic inside a function. Use a try-except block to handle any exceptions, and use the else block to print whether the number is prime or not. 4. Write a Python program to simulate a simple ATM machine. Create a function that accepts the user's account balance and withdrawal amount. If the 	23-10-23	60	

	<p>withdrawal amount is greater than the account balance, raise a "InsufficientFundsError" exception.</p> <p>5. Write a program that takes a list of integers from the user and calculates the reciprocal of each number. Handle any exceptions that may occur during the division and continue the loop.</p> <p>6. Develop a program that reads a file and prints its contents. Handle the "FileNotFoundError" exception. Regardless of whether the file is found or not, use the "finally" block to display a message indicating the end of the program.</p> <p>7. Create a Python program that performs a series of mathematical operations on two user-input numbers. Handle exceptions such as "ZeroDivisionError" and "ValueError." If any exception occurs, provide a detailed error message.</p> <p>8. Write a program that defines a function that opens a specified file and reads its contents. If the file does not exist, the function should raise a "FileNotFoundError." In the main program, handle this exception and print an error message.</p> <p>9. Provide a Python code snippet with a deliberate error (e.g., a syntax error or a logical error) and ask the student to identify and fix the issue using the information provided in the exception message.</p>			
14	<p>Class and Constructor:</p> <p>1. Write a program to design a class to represent a car with attributes like make model year and speed each car object should contain 3 methods which can start the engine accelerate and slow down the speed.</p> <p>2. Write a program to design a class to represent a rectangle with attributes like length and width each rectangle object should contain 2 methods which can compare the rectangle and compute the area.</p> <p>3. Design a class to represent a student with attributes like name, roll number, and marks. Create methods to calculate the grade and display student information.</p> <p>4. Implement a class for a library system with functions to add, delete, and search for books.</p> <p>5. Develop a Python class for a simple bank account. Create a constructor that initializes the account</p>	03-11-2023	66	

	holder's name and balance. Include methods for depositing and withdrawing money, and ensure that the balance cannot go negative. Test the class by creating accounts and performing transactions.			
	6. Create a class for representing employees in an organization. The constructor should take parameters for employee name, employee ID, and salary. Implement a method that calculates an annual bonus based on the salary and a given bonus percentage. Use this class to calculate the annual bonus for a set of employees.			
15	Inheritance: 1. Write a python program to demonstrate Inheritance in animals, Call subclasses with parent class member function. 2. Write a python program to demonstrate inheritance in employees. call the	20-11-23	74	
16	Access Modifier: Write a python program to demonstrate access modifiers in: 1. Public 2. Protected 3. Private	21-11-23	77	
17	Module Import: 1. Choose two different modules from the Python standard library. Write a Python script that integrates both modules to perform a specific task or solve a problem. 2. Create a Python module that acts as a simple calculator. Include functions for addition, subtraction, multiplication, and division. Write a script to import and use this module to perform arithmetic operations. 3. Create a Python script that uses the math module to calculate the area and circumference of a circle, the volume of a sphere, and the hypotenuse of a right-angled triangle. Allow the user to input relevant parameters. 4. Write a Python script that generates a random password of a specified length. Utilize the random module to include a mix of uppercase letters, lowercase letters, numbers, and special characters.	08-12-23	80	

	5. Develop a simple task scheduler using the time module. Allow the user to input tasks along with their scheduled execution times. The program should execute the tasks at the specified times.			
18	GUI: Imagine you are tasked with developing a registration form for an upcoming event. Create a Python script using Tkinter to design a GUI registration form. The form should include the following fields: A text entry for the participant to enter their full name. An entry for the participant's age. An entry for the participant's email address. Include a dropdown menu or radio buttons for the participant to select their event category (e.g., Workshop, Seminar, Networking). Submit button- to print filled form details	05-12-2023	84	
19	GUI with Database: Develop a database-driven GUI program to manage student information. Set up database connection and Implement features for adding, updating, and deleting student records.	08-12-23	88	
20	Project: Topic: <i>Hospital Management System</i> Description: <ol style="list-style-type: none"> 1. <i>Admin Panel for Hospital with Login</i> 2. <i>Patient Registration</i> 3. <i>Employee Registration</i> 4. <i>Appointment Panel</i> 5. <i>Room Allocation to Patient</i> 6. <i>Billing Panel</i> 	18-12-23	93	



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 01

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No: 33

DATE: 28-08-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To understand basic of python, built in functions, how to create function, run loop, different types of operator and operation using them.
Topics Covered:	python installation basic python commands range() type() built in functions simple python functions assignment operator arithmetic operation
Problem Statement:	<u>Assignments-01</u> 1. Simple Commands: a. Write a program that prints your name. b. Create a program that displays "Hello, World!" on the screen. c. Write a program that asks the user for their age and then prints it. d. Write a Python program to add odd numbers from 1-10 e. Write a Python program to get the Fibonacci series between 0 to 50. 2. Mathematical Operators: a. Create a program to calculate the area of a rectangle given its length and

	<p>width.</p> <p>b. Write a program that calculates the volume of a cube using its side length.</p> <p>c. Create a simple calculator program that can perform addition, subtraction, multiplication, and division.</p> <p>3. Range:</p> <p>a. Write a program that prints all even numbers between 1 and 50.</p> <p>b. Create a program that generates a list of squares of numbers from 1 to 10 using a loop.</p> <p>c. Write a program to calculate the sum of all numbers between 1 and 100.</p>
Theory:	<ol style="list-style-type: none"> 1. First learn how to install latest version and python in our system. 2. Print() function – how to display result with print() , with single print statement how to write multiple line output using \n also tried with \t Take input from user and display it using print() where string and age are concat using (,)separator 3. Create variable and store different datatype(int, float, string) Python treats char and string as same. Also use built in function type() to check datatype of variables Learn typecasting of datatype from one to another 4. Create for loop with range() to run a block of code for specific iteration. Made change in range(start, end, step) parameter to check different scenarios Step – argument is option if not mentioned by default value is 1 End value is exclude because iteration is perform for end -1 times. 5. Then we learn about different operator and arithmetic operations Addition(+), subtraction (-), multiplication(*) , modulus (%), Exponential(**) Complex arithmetic operations based on BODMAS rule 6. Learn to create a function to perform simple calculator operations 7. Print even number using for loop Use end = " " for print() parameter to continue with same line for printing output 8. Save python file with extension .py and run module to see output
Code:	<p>1.a #Write a program that prints your name. print("Karan Panchal - FYMCA - Python Programming")</p> <p>1.b #Create a program that displays "Hello, World!" on the screen. print("Hello, World!")</p> <p>1.c #Write a program that asks the user for their age and then prints it. age = input("Enter your age : ") print(age)</p>

1.d

Write a Python program to add odd numbers from 1-10

```
sum = 0
```

```
for x in range(1,10,2):
```

```
    sum = sum + x
```

```
print(sum)
```

1.e

#Write a Python program to get the Fibonacci series between 0 to 50.

```
num1 = 0
```

```
num2 = 1
```

```
start,end = 2,50
```

```
print(num1 , num2 , end = " ")
```

```
for x in range(start,end+1):
```

```
    num1,num2 = num2,num1+num2
```

```
    print(num2,end = " ")
```

2.a

Create a program to calculate the area of a rectangle given its length and width.

```
length = float(input("Enter Length of rectangle : "))
```

```
width = float(input("Enter Width of rectangle : "))
```

```
area = length * width
```

```
print("Area of rectangle with length ",length , " and " , "width ",width , " is ",area)
```

2.b

#Write a program that calculates the volume of a cube using its side length.

```
length = float(input("Enter side length of cube : "))
```

```
volume = length ** 3
```

```
print("volume of cube is ",volume)
```

2.c

#Create a simple calculator program that can perform addition, subtraction, multiplication, and division.

```
def calculator(x,y,type):
```

```
    if(type == "add"):
```

```
        return x + y
```

```
    elif(type == "sub"):
```

```
        return x - y
```

```
    elif(type == "mul"):
```

```
        return x * y
```

```
    elif(type == "div"):
```

```
        return x / y
```

```
print(calculator(10,2,"add"))
```

```
print(calculator(10,2,"sub"))
```

```
print(calculator(10,2,"mul"))
```

```
print(calculator(10,2,"div"))
```

	<p>3.a #Write a program that prints all even numbers between 1 and 50. start,end = 1,50 #by default step in range is 1 for x in range(start,end+1): if(x%2 == 0): print(x , end = " ")</p> <p>3.b #Create a program that generates a list of squares of numbers from 1 to 10 using a loop. start,end = 1,10 #by default step in range is 1 for x in range(start,end+1): print("Square of " , x , " is " , x ** 2)</p> <p>3.c #Write a program to calculate the sum of all numbers between 1 and 100. start,end = 1,100 sum = 0 for x in range(start,end+1): sum += x</p> <p>print("Sum of all numbers from",start, "to", end , "is",sum)</p>
Screen Shot of Output:	<p>1.a [Running] python -u "c:\Users\kpanchal\Desktop\MCA Assignments\Python Programming\Assignments-01\Assignment-1a.py" Karan Panchal - FYMCA - Python Programming</p> <p>1.b [Running] python -u "c:\Users\kpanchal\Desktop\MCA Assignments\Python Programming\Assignments-01\tempCodeRunnerFile.py" Hello, World!</p> <p>1.c PS C:\Users\kpanchal\Desktop\MCA Assignments\Python Programming\Assignments-01> & C:/Users/kpanchal/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/kpanchal/Desktop/MCA Assignments/Python Programming/Assignments-01/Assignment-1c.py" Enter your age : 22 22</p> <p>1.d [Running] python -u "c:\Users\kpanchal\Desktop\MCA Assignments\Python Programming\Assignments-01\Assignment-1d.py" 25</p> <p>1.e 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987 1597 2584 4181 6765 10946 17711 28657 46368 75025 121393 196418 317811 514229 832040 1346269 2178309 3524578 5702887 9227465 14930352 24157817 39088169 63245986 102334155 1655 80141 267914296 433494437 701408733 1134903170 1836311903 2971215073 4807526976 7778742049 12586269025</p> <p>2.a Enter Length of rectangle : 4 Enter Width of rectangle : 3.5 Area of rectangle with length 4.0 and width 3.5 is 14.0</p> <p>2.b Enter side length of cube : 4.5 volume of cube is 91.125</p> <p>2.c 12 8 20 5.0</p> <p>3.a ktop\MCA Assignments\Python Programming\Assignments-01\Assignment-3a.py" 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50</p> <p>3.b</p>

	<pre> ktop\MCA Assignments\Python Programming\Assignments-01\Assignment-3b.py" Square of 1 is 1 Square of 2 is 4 Square of 3 is 9 Square of 4 is 16 Square of 5 is 25 Square of 6 is 36 Square of 7 is 49 Square of 8 is 64 Square of 9 is 81 Square of 10 is 100 </pre> <p>3.c</p> <pre> ktop\MCA Assignments\Python Programming\Assignments-01\Assignment-3c.py" o Sum of all numbers from 1 to 100 is 5050 </pre>
Observations:	In python, there is no need of main function/additional in libraries to add to make program work. The for is different then traditional for loop. No need to define datatype for defining variable. how to typecast input from user.
Conclusion:	At the conclusion, python is user friendly language for beginner with no exposure to programming language before. Syntax is easier to understand which make code syntax error free. Writing code which is efficient and take less memory space and time and making code reusable make ourself a better programmer.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 02

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 01-09-2023

FULL NAME: Karan Dinesh
Panchal

Aim:	To learn about list creation, manipulation, deleteion in python
Topics Covered:	Create List, Append Value, remove using different methods,clear list, Sorting, count duplicate elements in the list.
Problem Statement:	Assignments-02 1. Create a Python program that performs basic list operations. - Create an empty list. - Add three different types of elements to the list (e.g., integers, strings, and floats). - Print the list. - Remove one element from the list. - Print the modified list. 2. WAPP for list indexing and slicing in Python. - Create a list of numbers from 1 to 10. - Print the first and last elements of the list. - Print a slice of the list containing elements from index 3 to 7. - Modify the list to replace the elements from index 5 to 9 with even

	<p>numbers (e.g., 10, 12, 14, ...).</p> <ul style="list-style-type: none"> - Print the modified list. <p>3. WAPP to count occurrences of all the elements present in the list.</p> <ul style="list-style-type: none"> - Provide a list. - Print all the elements of the list with their counts. - Print the list. <p>4. WAPP for List Sorting and Reversing list</p> <ul style="list-style-type: none"> - Create a list of unsorted numbers. - Sort the list in ascending order. - Reverse the sorted list. - Print the reversed list. <p>5. WAPP to demonstrate list concatenation in Python.</p> <ul style="list-style-type: none"> - Provide two lists of names (e.g., boys names and girls names). - concatenate the two lists to create a combined list of names. - Print the combined list.
Theory:	<ol style="list-style-type: none"> 1. Lists are a fundamental data structure in many programming languages, including Python. We use them to store collections of data, which can be of different data types (integers, strings, floats, etc.), and they allow duplicate values 2. Using append method to add a new value or element at the end of a list. 3. built-in sort method for lists, which by default sorts the elements in ascending order. We can use the reverse=True argument with the sort method to sort the list in descending order. 4. Lists can be concatenated using the + operator in Python. When we use the + operator between two lists, a new list is created containing all the elements from both lists.
Code:	<ol style="list-style-type: none"> 1. <pre> myList = [] print("Empty List :",myList) myList.append(5) myList.append("Karan") myList.append(29.99) print("List after adding elements :",myList) myList.remove("Karan") print("Modified List :",myList) Empty List : [] List after adding elements : [5, 'Karan', 29.99] Modified List : [5, 29.99] </pre> 2. <pre> myList = [1,2,3,4,5,6,7,8,9,10] print("First and Last element of list :", myList[::-1]) </pre>

	<pre> print(myList[3:7]) myList[5] = 10 myList[6] = 12 myList[7] = 14 myList[8] = 16 myList[9] = 18 print("modified list :", myList) First and Last element of list : [1, 10] [4, 5, 6, 7] modified list : [1, 2, 3, 4, 5, 10, 12, 14, 16, 18] 3. myList = [1,2,13,4,4,5,6,7,17,5,6,7,7,1] li = [] for x in myList: if x in li: continue li.append(x) print("Count of",x,"is :",myList.count(x)) print(myList) Count of 1 is : 1 Count of 2 is : 1 Count of 13 is : 1 Count of 4 is : 2 Count of 5 is : 2 Count of 6 is : 1 Count of 7 is : 1 Count of 17 is : 1 [1, 2, 13, 4, 4, 5, 6, 7, 17, 5] 4. myList = [3,2,5,7,12,8,4,0] myList.sort() print("Sorted List :",myList) myList.sort(reverse=True) print("Reverse List :",myList) Sorted List : [0, 2, 3, 4, 5, 7, 8, 12] Reverse List : [12, 8, 7, 5, 4, 3, 2, 0] 5. list1 = ["b1", "b2", "b3", "b4", "b5"] list2 = ["g1", "g2", "g3", "g4", "g5"] list3 = list1 + list2 print(list3) ['b1', 'b2', 'b3', 'b4', 'b5', 'g1', 'g2', 'g3', 'g4', 'g5'] </pre>
Screen Shot of Output:	
Observations :	<p>Python lists store diverse data, allowing duplicates and preserving order. append adds values for dynamic growth. sort orders lists by default, and reverse=True for descending order. + combines lists, creating new ones. Lists are vital, versatile data structures for programming tasks due to their flexibility.</p>

Conclusion:	Python lists offer a flexible and fundamental data structure capable of storing diverse data types, handling duplicates, and preserving insertion order. The append method simplifies dynamic list expansion, while the sort function provides straightforward sorting options. Additionally, list concatenation via the + operator streamlines data manipulation.
--------------------	--

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 03

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No: 33

DATE: 08-09-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To Explore and learn the tuple data type and its various functions.
Topics Covered:	Tuple creation, length, multiple list inside tuple, packing and unpacking of the tuple
Problem Statement:	<ol style="list-style-type: none">1. WAPP to create a tuple of your favorite fruits.<ul style="list-style-type: none">- Print the tuple.- Access and print individual elements of the tuple.- Attempt to modify an element in the tuple and observe the error.- Calculate and print the length of the tuple.2. WAPP to create a tuple containing a student's name and their scores in three subjects (e.g., ("Alice", 90, 85, 92)).<ul style="list-style-type: none">- Use tuple unpacking to extract and print the student's name and each subject score separately.- Calculate and print the average score for the student.3. WAPP to create a list of tuples, each containing a student's name and their exam scores.<ul style="list-style-type: none">- Sort the list of tuples in ascending order based on the average score (consider using a custom sorting key).- Print the sorted list of tuples.4. WAPP to create two tuples containing days of the week (e.g., weekdays and weekends).<ul style="list-style-type: none">- Concatenate the two tuples to create a single tuple representing all days

	<p>of the week.</p> <ul style="list-style-type: none"> - Create a new tuple by repeating one of the original tuples multiple times. <p>5. Create Python functions that demonstrate tuple packing and unpacking.</p> <ul style="list-style-type: none"> - Implement a function that takes multiple arguments and returns them as a tuple (packing). - Implement a function that receives a tuple as an argument and unpacks it, printing the elements.
Theory:	<ol style="list-style-type: none"> 1. Tuples does not contain duplicate values, order is preserved 2. Using len method we can get length of tuple 3. Tuple can stored any data/object. 4. We can concat two tuple using + operator and * operator to multiple tuple value. 5. Unpacking of tuple to get individual values and packing multiple values to create tuple.
Code:	<pre> 1. fruits = ("apple","banana","melons","grapes","mango") print(fruits) for i in fruits: print(i, end= " ") print() #fruits[0] = "orange" #print(fruits) print("Length of the tuple is:",len(fruits)) ('apple', 'banana', 'melons', 'grapes', 'mango') apple banana melons grapes mango Traceback (most recent call last): File "C:\Users\kpanchal\Desktop\MCA Assignments\Python Programmin g\Assignments-03\Q1.py", line 8, in <module> fruits[0] = "orange" TypeError: 'tuple' object does not support item assignment = RESTART: C:\Users\kpanchal\Desktop\MCA Assignments\Python Program ming\Assignments-03\Q1.py ('apple', 'banana', 'melons', 'grapes', 'mango') apple banana melons grapes mango ('apple', 'banana', 'melons', 'grapes', 'mango') Length of the tuple is: 5 2. stud=("Karan", 60, 70, 80) print("student details : ", stud) name, maths, science, history = stud print("Student Name:",name) print("Score in Maths:",maths) print("Score in Science:",science) print("Score in History:",history) </pre>

```

avg=(maths+science+history)/3
print("Average score=",avg)

student details : ('Karan', 60, 70, 80)
Student Name: Karan
Score in Maths: 60
Score in Science: 70
Score in History: 80
Average score= 70.0

3.
student_list = (["Karan", 70], ["Ajay", 80], ["Parth", 75])
def cal_avg(score):
    return score[1]
sorted_scores = sorted(student_list, key=cal_avg)
for student in sorted_scores:
    name, score = student
    print(f"{name}: {score}")
Karan: 70
Parth: 75
Ajay: 80

4.
weekdays = ("Mon", "Tue", "Wed", "Thu", "Fri")
weekend = ("Sat", "Sun")

whole_week = weekdays + weekend
print("Days in week are :", whole_week)

month = whole_week * 4

print("Days in months are :", month)
Days in week are : ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun')
Days in months are : ('Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun', 'Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun')

5.
def pack(roll_number, student_name):
    tup = (roll_number, student_name)
    return tup

def unpack(student_tuple):
    roll_number, student_name = student_tuple
    return student_tuple

student_tuple = pack(34, "Karan")
print(student_tuple)

roll_number, student_name = unpack(student_tuple)
print("Roll Number: ", roll_number, "Name : ", student_name)
(34, 'Karan')
Roll Number: 34 Name : Karan

```

Screen Shot of Output:	
Observations :	Tuples are a valuable data structure in Python due to their immutability, order, and flexibility. We commonly use them for scenarios where we want to represent a collection of related values that should not be modified after creation.
Conclusion:	Tuples are a powerful tool in Python for organizing and working with data, especially when we want to ensure that the data remains unchanged and ordered.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 04

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 11-09-2023

FULL NAME: Karan Dinesh
Panchal

Aim:	To learn about set and list, different operation on set like union, intersection, difference
Topics Covered:	Set, List to Set, union, intersection, difference
Problem Statement:	<ol style="list-style-type: none">1. Write a Python program to find and print the unique elements in a given list.<ul style="list-style-type: none">- Create a list that contains duplicate elements.- Use a set to identify and print the unique elements from the list.2. WAPP for Set Operations with Two Sets: Consider two sets of numbers.<ul style="list-style-type: none">- Calculate and print the union of the two sets.- Calculate and print the intersection of the two sets.- Calculate and print the symmetric difference between the two sets.3. WAPP to Create two lists of items for books in two libraries and identify common books present in both the libraries.<ul style="list-style-type: none">- Use sets to identify common books and create a new set.- Print the set of common items.4. WAPP to create a set of unique words in the text, removing punctuation and converting all words to lowercase

Theory:	<ol style="list-style-type: none"> 1. List with duplicate value can be converted to set using set()method which result into unique elements of list. 2. Two set union result into all unique elements of both sets. 3. Intersection is common elements in both sets 4. Difference is opposite of intersection. 5. String is split with space and convert to lowercase using lower() method and store into list then list is converted to set which contains all unique word of string.
Code:	<pre> 1. myList = [1,2,2,3,4,5,5,6,7,7,8,8] unique_elements = [] for i in myList: if i not in unique_elements: unique_elements.append(i) print("Original List :",myList) print("List with Unique elements :",unique_elements) unique_set = set(myList) print("Set with unique Elements :",unique_set) Original List : [1, 2, 2, 3, 4, 5, 5, 6, 7, 7, 8, 8] List with Unique elements : [1, 2, 3, 4, 5, 6, 7, 8] Set with unique Elements : {1, 2, 3, 4, 5, 6, 7, 8} 2. mySet1 = {1,2,3,4,5,6} mySet2 = {3,4,5,6,7,8} # union of two sets union = mySet1.union(mySet2) print("Union of mySet1 and mySet2:", union) #common elements intersection = mySet1.intersection(mySet2) print(intersection) #opposite of intersection symmetric = mySet1.symmetric_difference(mySet2) print(symmetric) Union of mySet1 and mySet2: {1, 2, 3, 4, 5, 6, 7, 8} {3, 4, 5, 6} {1, 2, 7, 8} 3. library1_list = ["Book1", "Book2", "Book3", "Book4", "Book5", "Book6"] library2_list = ["Book3", "Book4", "Book5", "Book6", "Book7", "Book1"] library1_set = set(library1_list) </pre>

	<pre> library2_set = set(library2_list) common_set = library1_set.intersection(library2_set) print("Common books in both libraries:", common_set) Common books in both libraries: {'Book5', 'Book1', 'Book4', 'Book6', 'Book3'} 4. str = "Peter Piper picked a peck of pickled peppers A peck of pickled peppers Peter Piper picked." myList = str.lower().split(" ") print(myList) unique_set = set(myList) print(unique_set) ['peter', 'piper', 'picked', 'a', 'peck', 'of', 'pickled', 'peppers', 'a', 'pec ', 'of', 'pickled', 'peppers', 'peter', 'piper', 'picked.'] {'pickled', 'of', 'peppers', 'picked', 'peck', 'a', 'peter', 'piper', 'picked.' </pre>
Screen Shot of Output:	
Observations :	<p>set() removes duplicates from lists.</p> <p>Union combines distinct elements.</p> <p>Intersection finds shared values.</p> <p>Difference reveals unique elements.</p> <p>Split, lowercase, and set give unique lowercase words from strings.</p>
Conclusion:	<p>We can remove duplicates, combine data without repeats, find common elements, spot differences, and extract unique lowercase words from text.</p>

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 05

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 17-09-2023

FULL NAME: Karan Dinesh
Panchal

Aim:	To understand different type of conditional statements and operator to control work flow of program
Topics Covered:	If , elif , else conditional statements with and/or operator
Problem Statement:	<ol style="list-style-type: none">1. WAPP to determine the grade based on a user's input score. Give the grades starts from First class with Distinction to Fail as per the score percentage.2. WAPP for number comparison. Get two numbers from the users and compare the numbers and display the result. Result could be Like greater than, less than or equal numbers.3. WAPP to check for leap year. Take year as an input from the user and depending on the input display appropriate message.4. WAPP for age classifier.<ul style="list-style-type: none">• If the age is less than zero, then display the message as Invalid age.• If the age in between zero to 18 then Minor

	<ul style="list-style-type: none"> · If the age is in between 18 to 65 then Adult · If the age is above 65 then Senior Citizen. <p>5. WAPP to create a basic grading system for multiple subjects. It calculates the average grade and provides a final grade based on the average score. If students got less the 35 marks in any of the subjects, then the overall result will be FAIL.</p>
Theory:	<ol style="list-style-type: none"> 1. In Python, when we use an if statement, it first checks the condition associated with it. If the condition evaluates to True, the code block inside the if statement is executed. 2. We can use logical operators like and and or to combine multiple conditions within an if or elif statement. 3. When we use and, all conditions connected by and must evaluate to True for the entire expression to be True. In other words, it's a way to require that multiple conditions are met before executing a code block. 4. The else statement in Python does not have a condition associated with it. It serves as a catch-all block that is executed only when none of the preceding if and elif conditions evaluate to True.
Code:	<ol style="list-style-type: none"> 1. WAPP to determine the grade based on a user's input score. # Give the grades starts from First class with Distinction to Fail as per the score percentage. score = float(input("enter your exam score :")) if score >= 75: print("First Class with Distinction ",score) elif score >= 65 and score < 75: print("Your grade is First Class",score) elif score >= 50 and score < 65: print("Your grade is Second Class",score) elif score >= 40 and score < 50: print("Your grade is Third Class",score) else: print("You are failed",score) 2. WAPP for number comparison. Get two numbers from the users and compare the numbers and display the result. # Result could be Like greater than, less than or equal numbers. num1 = float(input("Enter first number : ")) num2 = float(input("Enter second number : ")) if num1 > num2: print("First Number is greater") elif num1 < num2: print("Second Number is greater") else: print("Both Numbers are equal") 3. WAPP to check for leap year. # Take year as an input from the user and depending on the input display appropriate

	<pre> message. year = int(input("Enter any year :")) if year%4 == 0: print(year ,"is a leap year") else: print(year ,"is not a leap year") 4. WAPP for age classifier. age = int(input("Enter your age : ")) if age < 0: print("invalid age entered") elif age < 18 : print("Minor") elif age < 65: print("Adult") else: print("Senior Citizen") 5. create a marksheet subject_list = ["Maths","Science","History","Social Studies","English"] subject_dict = {} for x in subject_list: str = "Enter " + x + " Score :" sub_score = int(input(str)) subject_dict[x] = sub_score print("Score :",subject_dict) final_result = "You are Pass" print("*****") print("\tSOMAIYA UNIVERSITY") for subject, marks in subject_dict.items(): # print(subject , marks) status = "fail" if marks >= 35: status = "pass" else: final_result = "You are Fail" print(subject,":", marks , status) print("*****",final_result,"*****") print("*****") </pre>
Screen Shot of Output:	<pre> 1. Enter your exam score :61 Your grade is Second Class 61.0 2. Enter first number : 34 Enter second number : 76 Second Number is greater 3. </pre>

	<pre> Enter any year :2000 2000 is a leap year 4. Enter your age : 34 Adult 5. Enter Maths Score :56 Enter Science Score :76 Enter History Score :45 Enter Social Studies Score :23 Enter English Score :67 Score : {'Maths': 56, 'Science': 76, 'History': 45, 'Social Studies': 23, 'English': 67} ***** SOMAIYA UNIVERSITY Maths : 56 pass Science : 76 pass History : 45 pass Social Studies : 23 fail English : 67 pass ***** You are Fail ***** ***** </pre>
Observations :	The if-elif-else structure in Python allows us to create branching logic, making it possible to execute different code blocks based on various conditions. Logical operators like and and or can be used to create complex conditions by combining multiple simpler conditions. The else statement provides a fallback or default action to be taken when none of the preceding conditions (if and elif) are met.
Conclusion:	In conclusion, the if, elif, and else statements in Python are essential components of conditional branching, enabling you to control the flow of your programs based on specific conditions.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 06

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No: 33

DATE: 17-09-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To understand different type of loop/nested loop in python
Topics Covered:	Loops – for , while , nested for , time module
Problem Statement:	<ol style="list-style-type: none">1. WAPP to Print the multiplication table for a given number.2. WAPP to create a countdown timer using a while loop. You can use sleep method for delay in displaying numbers.3. WAPP to print a simple pattern using nested loops. * ** *** **** *****4. WAPP to print a * pyramid pattern using nested loops. * ** *** **** *****

	5. WAPP to calculate the sum of numbers from 1 to N (e.g., N = 5)
Theory:	<ol style="list-style-type: none"> 1. A "for loop" is a control flow statement in programming used to iterate through various data structures like ranges, lists, tuples, sets, and dictionaries. When using a "for loop," you specify a range or a data structure, and the loop iterates through its elements one by one. In Python, the "range" function is often used with "for loops," and it takes three arguments: start, end, and step. By default, the step is set to 1, meaning it will increment by 1 with each iteration 2. while loop is another type of loop in programming that continues iterating until a certain condition becomes false. It's essential to be cautious when using a "while loop" because if the condition never becomes false, the loop will run infinitely, causing your program to hang. 3. The "time" module is a standard library module in Python used for time-related operations. One of its common uses is to introduce delays or pauses in a program. 4. Nested "for loops" are used when you need to create more complex patterns or iterate through multi-dimensional data structures. For example, you can use nested "for loops" to print star patterns.
Code:	<pre> 1. #WAPP to Print the multiplication table for a given number. table = int(input("Enter number for table : ")) print("using while loop") count = 1 while count <= 10: print(table,"X",count," = ",table*count) count +=1 print("-----") print("using for loop") for x in range(1,11): print(table,"X",x," = ",table*x) 2. #WAPP to create a countdown timer using a while loop. import time timer = int(input("Enter no of second for countdown :")) for x in range(timer,-1,-1): if x != timer: time.sleep(1) print(x) print("Timeeeee upsssss.....!!!!!!") 3. #WAPP to print a simple pattern using nested loops. for i in range(0,5): for j in range(0,i+1): print("* ", end = " ") </pre>

	<pre> print() 4. # WAPP to print a * pyramid pattern using nested loops. n_rows = 5 for i in range(0,n_rows): for space in range(1, (n_rows-i)+1): print(" ",end="") for j in range(0,i+1): print("* ", end = "") print() 5. # WAPP to calculate the sum of numbers from 1 to N (e.g., N = 5) user_input = int(input("Enter any natural number : ")) sum = 0 for i in range(1,user_input+1): sum += i print("Sum of nth number is :",sum) </pre>
Screen Shot of Output:	<pre> 1. Enter number for table : 5 using while loop 5 X 1 = 5 5 X 2 = 10 5 X 3 = 15 5 X 4 = 20 5 X 5 = 25 5 X 6 = 30 5 X 7 = 35 5 X 8 = 40 5 X 9 = 45 5 X 10 = 50 ----- using for loop 5 X 1 = 5 5 X 2 = 10 5 X 3 = 15 5 X 4 = 20 5 X 5 = 25 5 X 6 = 30 5 X 7 = 35 5 X 8 = 40 5 X 9 = 45 5 X 10 = 50 </pre> <p>2.</p>

	<pre> Enter no of second for countdown :5 5 4 3 2 1 0 Timeeeee upsssss....!!!!!!! 3. * * * * * * * * * * * * * * * 4. * * * * * * * * * * * * * * * 5. Enter any natural number : 10 Sum of nth number is : 55 </pre>
Observations :	These concepts are fundamental in programming and are commonly used to control the flow of your code, manipulate data structures and create patterns or perform repetitive tasks efficiently.
Conclusion:	In summary, loops are vital for repeating tasks and controlling program flow. These fundamental programming concepts are essential for building effective and efficient software.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 07

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 18-09-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To understand how to control flow in loop using break and continue
Topics Covered:	break, continue
Problem Statement:	<ol style="list-style-type: none">1. WAPP to find the first even number and print.2. WAPP to skip printing numbers divisible by 3 within a for loop.3. WAPP to terminate a while loop when the user enters a specific input (e.g, "exit").4. WAPP to skip specific values (e.g., negative numbers) when iterating through a list.5. WAPP to find and print all prime numbers from 1 to 50. Use a for loop and the break statement to optimize the search for prime numbers.6. WAPP that takes a list of integers and prints all positive numbers, skipping negative ones using the continue statement.7. WAPP that asks the user to enter a password. If the password contains at least one uppercase letter, one lowercase letter, one digit, and is at least 8 characters long, print "Password accepted." Otherwise, print an appropriate message and continue asking for a password until a valid one is entered.

	<p>8. WAPP that takes a list of numbers and a sum limit. Print the numbers in the list one by one until their sum exceeds the limit, then terminate the loop using the break statement.</p> <p>9. WAPP that takes a list of numbers and prints unique values (skipping duplicates) using the continue statement.</p>
Theory:	<ol style="list-style-type: none"> 1. The break statement is used to forcefully stop a loop when it is encountered inside the loop's code. It not only exits the loop but also continues with the next statement in the program that follows the loop. 2. The continue statement is used to skip the remaining code within the current iteration of a loop and move on to the next iteration of that loop. It helps you ignore specific parts of the loop's code for a particular iteration.
Code:	<ol style="list-style-type: none"> 1. #finding the first even number myList = [1,23,5,3,4,7] for i in myList: if i%2 == 0: print("First even number for list is :",i) break First even number for list is : 4 2. for i in range(1,10): if i%3 == 0: continue print(i, end=" ") 1 2 4 5 7 8 3. #question loop till user enter "exit" keyword while True: txt = input("enter any word") if txt.lower() == "exit": break print("program closed") enter any word :karan enter any word :panchal enter any word :exit' enter any word :exit' enter any word :exit Program closed..... 4. num = [1,-3,-3,4,8,0,-1,-4,6,2,10] for i in num: if i <= 0: continue print(i, end = " ") 1 4 8 6 2 10 5.

```

for num in range(1, 51):
    if num > 1:
        for i in range(2, num):
            if (num % i) == 0:
                break
            else:
                print(num, end= ", ")
2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,
6.
num = [1,4,-7,-5,6,-4,3,7,8,0,-7]
for i in num:
    if i <= 0:
        continue
    print(i, end = " ")
1 4 6 3 7 8
7.
while True:
    password = input("Enter a password: ")
    if len(password) < 8:
        print("Weak: Password is too short.")
    elif not any(char.isdigit() for char in password):
        print("Moderate: Password contains no digits.")
    elif len(password) >= 12 and any(char.isdigit() or not char.isalnum() for char in
password):
        print("Very Strong: Password is strong with special characters.")
    else:
        print("Strong: Password is strong.")
    break
===== RESTART: C:/Users/kpancha
Enter a password: karan29
Weak: Password is too short.

===== RESTART: C:/Users/kpancha
Enter a password: Karan2910%
Strong: Password is strong.

8.
number = list(map(int,input("Enter number for list :").split(" ")))
limit = 100
total = 0
for x in number:
    if total > limit:
        break
    newTotal = total + x
    if newTotal < 100:
        total = newTotal
print(total)
Enter number for list :20 30 15 30 35
95

9.
li = list(map(int,input("Enter number for list :").split(" ")))
l2 = []

```

	<pre> for x in li: if x in l2: #print("exist") continue else: l2.append(x) print(x, end= " ") Enter number for list :1 2 3 1 4 5 3 8 -8 7 7 1 2 3 4 5 8 -8 7 </pre>
Screen Shot of Output:	
Observations :	These observations capture the essential functions of break and continue in loops, helping control the flow of your program based on certain conditions.
Conclusion:	In conclusion, These statements are valuable tools for managing the flow of loops in programming and can be used to control when and how a loop should execute its code.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 08

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No: 33

DATE: 21-09-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about different string methods in python.
Topics Covered:	len(),strip(),rstrip(),lstrip(),find(),rfind(),index(),rindex(),count(),replace(),split(),join(),upper(),lower(),swapcase(),title(),capitalize(),startswith(),endswith()
Problem Statement:	1. WAPP to check if an entered string is a palindrome or not? 2.WAPP to split a given string into lines. 3.WAPP to remove specific character from the string. (e.g. removing ! from the string "Hello World!") 4.WAPP to demonstrate more on string function - i) len() ii) strip() iii) rstrip() iv) lstrip() v) find() vi) rfind() vii) index() viii) rindex() ix) count() x) replace() xi) split() xii) join() xiii) upper() xiv) lower() xv) swapcase() xvi) title() xvii) capitalize() xviii) startswith() xix) endswith()
Theory:	Loop is traverse in reverse to get string and compare with original. Using regex remove special character from string len(str): Gets the length (number of characters) of the string. str.strip(): Removes leading and trailing whitespace from the string. str.rstrip(): Removes trailing whitespace from the string.

	<p>str.lstrip(): Removes leading whitespace from the string.</p> <p>str.find("p"): Searches for the character "p" in the string and returns the index of its first occurrence or -1 if not found.</p> <p>str.rfind("a"): Searches for the character "a" in reverse order in the string and returns the index of its last occurrence or -1 if not found.</p> <p>str.index("a"): Searches for the character "a" in the string and returns the index of its first occurrence, raising an error if not found.</p> <p>str.rindex("a"): Searches for the character "a" in reverse order in the string and returns the index of its last occurrence, raising an error if not found.</p> <p>str.count("a"): Counts the number of occurrences of the character "a" in the string.</p> <p>str.replace("fy", "sy"): Replaces all occurrences of "fy" with "sy" in the string.</p> <p>str.split(" "): Splits the string into a list of substrings using space as the delimiter.</p> <p>"".join(str.split(" ")) : Joins the split substrings back together into a single string without spaces.</p> <p>str.upper(): Converts the string to uppercase.</p> <p>str.lower(): Converts the string to lowercase.</p> <p>str.swapcase(): Swaps the case (uppercase to lowercase and vice versa) of characters in the string.</p> <p>str.title(): Converts the string to title case where the first character of each word is capitalized.</p> <p>str.capitalize(): Capitalizes the first character of the string while making the rest of the characters lowercase.</p> <p>str.startswith("A"): Checks if the string starts with the letter "A" and returns a boolean value (case-sensitive).</p>
Code:	<pre> 1. og_string = "madam" reverse_string = "" og_len = len(og_string) for i in range(og_len-1,-1,-1): reverse_string = reverse_string + og_string[i] if reverse_string == og_string: print("its a palindrome") else: print("not a palindrome") Original : hello Reverse : olleh not a palindrome = RESTART: C:/Users/kpanchal/Desktop/Assignments-08/assignment.py Original : madam Reverse : madam its a palindrome 2. inp = input("Enter your name :") str = inp.split() </pre>

```

for i in str:
    print(i)
Enter your name :hello welcome to MCA
hello
welcome
to
MCA
3.
string = "Hello World!%#@#"
regex = "!%#@#"
output = string.replace(regex, "")
print(output)
Origianl : Hello World!%#@#
After replace Hello World
4.
str=" karan panchal - fy mca  "
print("Length: ",len(str))
print("Strip: ",str.strip())
print("rstrip: ",str.rstrip())
print("lstrip: ",str.lstrip())
print("find: ",str.find("dfg"))
print("rfind: ",str.rfind("dfg"))
print("index: ",str.index("dfg"))
print("rindex: ",str.rindex("dfg"))
print("count: ",str.count("d"))
print("replace: ",str.replace("dfg","afg"))
print("Split: ",str.split("dfg"))
print("Join: ","".join(str.split("dfg")))
print("Upper: ",str.upper())
print("Lower: ",str.lower())
print("Swapcase: ",str.swapcase())
print("Title: ",str.title())
print("Capitalize: ",str.capitalize())
print("Startswith: ",str.startswith("A"))
print("Endswith: ",str.endswith("."))

```

	String is : karan panchal - fy mca Length: 23 Strip: karan panchal - fy mca rstrip: karan panchal - fy mca lstrip: karan panchal - fy mca find: 7 rfind: 22 index: 2 rindex: 22 count: 5 replace: karan panchal - sy mca Split: ['', 'karan', 'panchal', '-', 'fy', 'mca'] Join: karanpanchal-fymca Upper: KARAN PANCHAL - FY MCA Lower: karan panchal - fy mca Swapcase: KARAN PANCHAL - FY MCA Title: Karan Panchal - Fy Mca Capitalize: karan panchal - fy mca Startswith: False Endswith: False
Screen Shot of Output:	
Observations :	These methods provide a wide range of string manipulation and analysis in Python, making it easier to work with and manipulate text data. Depending on your specific task, you can choose the appropriate method to perform the desired operation on a string.
Conclusion:	Python string method can be used to perform different operation on string and get the output as required.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 09

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 22-09-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn different string methods in python.
Topics Covered:	len(),strip(),rstrip(),lstrip(),find(),rfind(),index(),rindex(),count(),replace(),split(),join(),upper(),lower(),swapcase(),title(),capitalize(),startswith(),endswith()
Problem Statement:	<ol style="list-style-type: none">1. Write a Python program that takes two strings as input and concatenates them. Ensure that the final string is in uppercase. Calculate and print the length of a user-entered string. Remove all leading and trailing spaces from a given string.2. a)Write a program to find and print the index of the first occurrence of a substring in each string. b)Create a Python function that takes a sentence as input and replaces all occurrences of a specified word with a user-defined word.3. a)Write a Python function that extracts the domain name from an email address using string slicing. b)Given a list of words separated by spaces, write a program to split the input into individual words and count the number of words.4. Create a Python program that accepts a user's name and age, and then prints a formatted message like "My name is [name] and I am [age] years old".

	<p>Format a given string to title case, and then capitalize the first character of the result.</p> <p>5. Write a Python program to count the number of times a specified character appears in a given string. Implement a function that searches for all occurrences of a specified word in a paragraph and counts them.</p> <p>6. a) Develop a program that reverses a given string without using string slicing.</p> <p>b) Create a Python function that accepts a sentence and removes all punctuation marks from it.</p> <p>7. a) Write a Python program that takes a list of words and joins them into a single string with a comma and space as the delimiter.</p> <p>b) Develop a program that reads a paragraph of text and splits it into sentences based on period (.) as the delimiter.</p> <p>8. a) Create a Python function that accepts a string and checks whether it starts with an uppercase letter.</p> <p>b) Implement a program that checks if a given string ends with a specific suffix.</p> <p>9. a) Write a program that removes all leading and trailing zeros from a user-entered string of numbers.</p> <p>b) Develop a function that strips a given string of a specific character from both ends.</p> <p>10. Compare two strings entered by the user and determine if they are equal. Find the index of the first occurrence of a user-specified character in a given string.</p>
Theory:	<p>str.upper(): Converts a string to uppercase.</p> <p>len(): Returns the length of a string.</p> <p>strip(): Removes leading and trailing whitespace.</p> <p>string.find(): Searches for the first occurrence of a substring in a string.</p> <p>string.find(): Searches for the first occurrence of a substring in a string.</p> <p>String slicing using [:] to extract parts of the email address.</p> <p>split(): Splits a string into a list of substrings based on whitespace.</p> <p>input(): Takes user input.</p> <p>String concatenation using +.</p> <p>string.title(): Converts the string to title case.</p> <p>string.capitalize(): Capitalizes the first letter of the string.</p> <p>string.count(): Counts the occurrences of a substring in a string.</p> <p>lower(): Converts the string to lowercase.</p> <p>Looping and string manipulation to check for palindromes.</p>

	<p>replace(): Replaces substrings in a string.</p> <p>String cleaning by removing punctuation characters.</p> <p>List creation and manipulation.</p> <p>join(): Joins a list of strings into a single string.</p> <p>split('.'): Splits a string into a list of substrings based on periods.</p> <p>string[0].isupper(): Checks if the first character of a string is uppercase.</p> <p>endswith(): Checks if a string ends with a specified suffix.</p> <p>lstrip(): Removes leading characters from the left side of a string.</p> <p>rstrip(): Removes trailing characters from the right side of a string.</p> <p>input(): Takes user input.</p> <p>user_input1.find(): Searches for the first occurrence of a substring in a string.</p>
Code:	<pre> 1. str1 = "hello" str2 = "world" str = str1+ " "+str2 print(str.upper()) user_input = input("Enter any word :") user_len = len(user_input) print("Original length of string is :",user_len) print("After removing white spaces :",user_input.strip() ,"New length is :",len(user_input.strip())) HELLO WORLD Enter any word : karan panchal Original length of string is : 23 After removing white spaces : karan panchal New length is : 13 2. string = "hello world!!!" print("first occurence of string is :",string.find("l")) str = "welcome to mca - python programming - kj somaiya management" print(string) def fnc(): replace_inp = input("Enter the word u want to replace :") new_word = input("Enter the new word :") new_string = str.replace(replace_inp,new_word) print("new string is :\n",new_string) fnc() </pre>

```

first occurrence of string is : 2
hello world!!!
Enter the word u want to replace :python
Enter the new word :web
new string is :
welcome to mca - web programming - kj somaiya management

3.
email = "karan.dp@somaiya.edu"
a=email.find("@")
print(a)
slicing=email[a:]
print("Username:",email[:a])
print("Domain name:",email[a:])

string = "welcome to mca karan panchal"
string = string.split()
print("Original String :",string)
print("No of words are :",len(string))

8
Username: karan.dp
Domain name: @somaiya.edu
Original String : ['welcome', 'to', 'mca', 'karan', 'panchal']
No of words are : 5

4.
user_name = input("Enter name :")
user_age = input("Enter age :")
string = "my name is "+ user_name+" and I am "+user_age+" years old"
print("Original string :",string)
print("TitleCase :", string.title())
print("Capitalize :",string.capitalize())
Enter name :karan
Enter age :55
Original string : my name is karan and I am 55 years old
TitleCase : My Name Is Karan And I Am 55 Years Old
Capitalize : My name is karan and i am 55 years old

5.

string = "la la laa la lala lalal lllaaaaaa"
print("Original String",string)
print("count of l letter :",string.count('l'))

para = "Peter Piper picked a peck of pickled peppers.A peck of pickled peppers Peter Piper
picked.If Peter Piper picked a peck of pickled peppers,Where's the peck of pickled peppers
Peter Piper picked?"
print("Original para",para)
print(para.lower().count("peter"))

```

	<pre> Original String la la laa la lala lalal lllaaaaaa count of l letter : 12 Original para Peter Piper picked a peck of pickled peppers.A peck of pickled peppers Peter Piper picked.If Peter Piper pick d a peck of pickled peppers,Where's the peck of pickled peppers Peter Piper picked? 4 6. og_string = "madam" reverse_string = "" og_len = len(og_string) print("Original String",og_string) for i in range(og_len-1,-1,-1): reverse_string = reverse_string + og_string[i] if reverse_string == og_string: print("its a palindrome") else: print("not a palindrome") string = "string. with, punctuation!!" print("Original String with punctuation :",string) punc = "'!()-[]{};:'\"<>./?@#\$\$%^&*~_" for i in string: if i in punc: string = string.replace(i, "") print("The string after punctuation filter :",string) Original String madam its a palindrome Original String with punctuation : string. with, punctuation!! The string after punctuation filter : string with punctuation 7. word_list = ["hello","world","welcome","to","mca"] print("Words of list :",word_list) new_string = "" for i in word_list: new_string += i + ", " print(new_string) para = "this is para1. this is para2. this is para3. this is para4" print("Original Paragraph :",para) para = para.split(".") print("New sentence :",para) Words of list : ['hello', 'world', 'welcome', 'to', 'mca'] hello, world, welcome, to, mca, Original Paragraph : this is para1. this is para2. this is para 3. this is para4 New sentence : ['this is para1', ' this is para2', ' this is pa ra3', ' this is para4'] 8. </pre>
--	--


```

string = "This string start with uppercase"
print("Original String:",string)
if string[0].isupper():
    print("string starts with uppercase")
else:
    print("does not starts with uppercase")

user_input = input("Enter any word :")
end_word_check = input("Enter end word to check")
if user_input.endswith(end_word_check):
    print("ends with this word")
else:
    print("does not ends with this word")
Original String: This string start with uppercase
string starts with uppercase
Enter any word :karan panchal
Enter end word to checkpanchal
ends with this word
|

9.
user_input = input("Please with 0 in start and end :")
user_input = user_input.lstrip("0")
user_input = user_input.rstrip("0")
print(user_input)

user_input = input("Please enter any word :")
remove_input = input("enter letter to remove from start and end :")
change = user_input.lstrip(remove_input)
change = change.rstrip(remove_input)
print("original string :", user_input)
print("after strip :",change)
Please with 0 in start and end :0000karan000panchal0000
karan000panchal
Please enter any word :1111111karan 111 panchal 111111
enter letter to remove from start and end :1
original string : 1111111karan 111 panchal 111111
after strip : karan 111 panchal

10.
user_input1 = input("Please enter first word")
user_input2 = input("Please enter second word")

if user_input1 == user_input2:
    print("Both strings are identical")
else:
    print("Strings are not identical")

print("first occurence of letter in user_input :",user_input1.find("k"))
Please enter first word :karan
Please enter second word :karan
Both strings are identical
first occurence of a letter in user_input : 1

```

Screen Shot of Output:	
Observations :	The Python code showcase essential string manipulation techniques, user input handling, and conditional checks. They utilize functions like upper(), find(), split(), count(), and input(). The code also demonstrates string cleaning, looping for palindrome checks, and list manipulation. These operations are fundamental for text processing and interaction with users, enabling a wide range of text-based programming tasks.
Conclusion:	In conclusion, code snippets demonstrate how to work with text in Python. They show how to change text to uppercase, find words, clean up text, and more. These are basic skills for working with words and sentences in programming. The code also includes user input, so you can interact with the program. It's a useful starting point for learning how to handle and manipulate text in Python.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 10

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 06-10-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about switch case in python language.
Topics Covered:	Switch case using dictionary and function in python
Problem Statement:	<p>Q1. Implement a "switch" using dictionaries and functions define function for each case</p> <p>Q2. Write a python program that allow user to manage a to do list. The program should display a menu from following options</p> <ol style="list-style-type: none">1.add task to the list2.remove task from the list3.display the list4.sort the list5.quit/exit
Theory:	<p>A dictionary called switch maps user input (1, 2, 3, 4, 5) to corresponding functions for each operation. A while loop repeatedly prompts the user for input until they opt to quit. Depending on their choice, the code calls specific functions, like add, remove, printList, sort, or quit, which carry out the desired task and display the updated list or a message when quitting.</p> <p>The code handles potential ValueError to ensure users input valid integers for their choices. This program provides a straightforward way to manage a list of tasks</p>

	through a text-based menu.
Code:	<pre> 1. #implementing a "Switch" using dictio #define functions for each case def case1(): print("Case 1 is selected") def case2(): print("Case 2 is selected") def case3(): print("Case 3 is selected") switch = { 1:case1, 2:case2, 3:case3, } try: selected_case = int(input("Enter a case(1,2 or 3):")) selected_function = switch.get(selected_case) if selected_function: selected_function() else: print("Invalid Case") except ValueError: print("Invalid input. Please enter a number (1,2 or 3).") Enter a case(1,2 or 3):1 Case 1 is selected = RESTART: C:/Users/kpanchal/Desktop/ ming/Assignments-10/Assignment.py Enter a case(1,2 or 3):4 Invalid Case 2. l = ["study","office work"] def add(data): l.append(data) print(l) def remove(data): l.remove(data) print(l) </pre>

```

def printList():
    print(l)
def sort():
    l.sort()
    print(l)
def quit1():
    print("Thank you!!!")

switch = {
    1:add,
    2:remove,
    3 : printList,
    4:sort,
    5:quit1
}

print("*****Menu*****")
print("1.Add Task\n2.Remove Task\n3.Print Task")
print("4.Sort Task\n5.Quit Menu")
print("*****")

try:
    while True:
        user_input = int(input("choose operation to perform form list : "))
        selected_function = switch.get(user_input)
        if user_input == 1:
            user_add_task = input("enter task : ")
            selected_function(user_add_task)
        elif user_input == 2:
            user_remove_task = input("enter task to remove : ")
            selected_function(user_remove_task)
        elif user_input == 3:
            selected_function()
        elif user_input == 4:
            selected_function()
        elif user_input == 5:
            selected_function()
            break
        else:
            print("Invalid operation")
except ValueError:
    print("Invalid input. Please enter a number (1,2 or 3).")

```

	<pre> *****Menu***** 1.Add Task 2.Remove Task 3.Print Task 4.Sort Task 5.Quit Menu ***** choose operation to perform form list : 1 enter task : lunch ['study', 'office work', 'lunch'] choose operation to perform form list : 2 enter task to remove : study ['office work', 'lunch'] choose operation to perform form list : 1 enter task : dinner ['office work', 'lunch', 'dinner'] choose operation to perform form list : 4 ['dinner', 'lunch', 'office work'] choose operation to perform form list : 5 Thank you!!! </pre>
Screen Shot of Output:	
Observations :	The code creates a basic task list manager in Python where Users can add, remove, print, sort tasks, and quit from menu. The code is structured well, but lacks advanced features like data persistence or extensive error handling. Enhancements for user experience, data management, and extensibility could be made for a more comprehensive task management solution.
Conclusion:	Python doesn't have a built-in switch-case statement. Instead, conditional if-elif-else statements are used for similar logic. Dictionaries can be used to map keys to functions, providing a form of switch-case behavior, as shown in the provided code. This allows for flexible branching and method invocation based on user choices.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 11

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 09-10-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about file handling in python and perform different operation on it.
Topics Covered:	Read text file, write file, append data, read by line, read by no. of characters
Problem Statement:	<ol style="list-style-type: none">1. WAP to read content of the file and count how many times letter 'a' comes in a file.2. WAP to read content of a line and display 'I' in place of 'E' while displaying the content of the file, all the other character should appear as it is.
Theory:	This code reads the contents of a text file named sample in read-only mode, storing the data in the file_data variable. It then prints the file's content and counts the occurrences of the letter 'a' within it. After that, it replaces all instances of the letter 'i' with 'e' in the file_data and prints the modified content. Finally, it closes the file using myfile.close(). This code demonstrates basic file handling in Python, including reading, processing, and manipulating text data from a file.
Code:	<pre>1. myfile = open("sample.txt","r") file_data = myfile.read() print(file_data) print("No. of count of 'a' in file is :",file_data.count('a')) myfile.close()</pre>

	<p>Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.</p> <p>Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented and functional programming.</p> <p>No. of count of 'a' in file is : 21</p> <p>2.</p> <pre>myfile = open("sample.txt","r") file_data = myfile.read() print(file_data) print("After replacing i with e data following is the data: \n",file_data.replace('i','e')) myfile.close()</pre> <p>-----</p> <p>Python is a high-level, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation. Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured, object-oriented and functional programming.</p> <p>After replacing i with e data following is the data:</p> <p>Python es a hegh-level, general-purpose programmeng language. Its desegn phelosophy emphasezes code readabelety weth the use of segnefecant endentateon. Python es dymamecally typed and garbage-collected. It supports mutleple programmeng paradegms, encludeng structured, object-orented and functeonal programmeng.</p>
Screen Shot of Output:	
Observations :	This code opens a file read the contents and checks how many times 'a' appears and changes 'i' to 'e' in the text, and displays the updated result before closing the file. This show basic file handling operations in Python for reading and manipulating text data.
Conclusion:	In conclusion, this code teaches how to Open and read text files. Count occurrences of specific characters or words in the file. Modify the file content by replacing specific characters. Close the file properly. It's a basic introduction to text file manipulation in Python.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 12

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 13-10-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn handle multiple file handling and operation in python
Topics Covered:	Read from multiple file, write read data to new file and sort the line alphabetic order.
Problem Statement:	Create a Python program that merges the contents of multiple text files into a single file, ensuring the lines are sorted in alphabetical order.
Theory:	The code processes a list of file names and saves their content to a file named sample.txt. It iterates through the list, reads the content of each file, and appends it to sample.txt. Then, it reads the contents of sample into a list, sorts that list alphabetically, and updates sample with the sorted content. However, it should be noted that the code may need modifications to sort the list while considering newline characters as part of each list element to ensure correct sorting.
Code:	<pre>1. file_list = ["file1.txt","file2.txt","file3.txt"] str1 = " " book_file = open("sample.txt",'r+') #truncate file data book_file.truncate(0) for x in file_list:</pre>

	<pre> #reading data of files myfile = open(x,"r+") data = myfile.read() str1 += data + "\n" myfile.close() #writing data of files book_file.write(data + "\n") #save read file data to new file book_file.close() #get readfile data in list book_file = open("sample.txt",'r+') new_list = book_file.readlines(); print("Original list is :\n",new_list) new_list.sort() print("Sorted List is :\n",new_list) book_file.write("\nAfter sorting: \n\n") #append sorted list to file for x in new_list: book_file.write(x) book_file.close() Original list is : ['Python is a high-level, general-purpose programming language.\n', 'Its design philosophy emphasizes code readability with the use o f significant indentation.\n', 'Python is dynamically typed and gar bage-collected. It supports multiple programming paradigms, includi ng structured, object-oriented and functional programming.\n'] Sorted List is : ['Its design philosophy emphasizes code readability with the use o f significant indentation.\n', 'Python is a high-level, general-pur pose programming language.\n', 'Python is dynamically typed and gar bage-collected. It supports multiple programming paradigms, includi ng structured, object-oriented and functional programming.\n'] </pre>
Screen Shot of Output:	
Observations:	The code combines data from multiple files into one file, sorts it, and updates the original file. However, it lacks newline character handling during sorting, potentially leading to incorrect results.
Conclusion:	In conclusion, the provided code successfully combines and sorts data from multiple files into one file. However, it requires modification to address newline character handling during sorting, accuracy in the final sorted result.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 13

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 20-10-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about file handling, exception handling in python
Topics Covered:	File handling read exception, Index range error, ZeroDivision error
Problem Statement:	<ol style="list-style-type: none">1. Write a Python program that prompts the user to enter two numbers and then performs division. Handle the "ZeroDivisionError" exception if the second number is zero.2. Create a Python program that reads a file specified by the user. Handle both "FileNotFoundError" and "PermissionError" exceptions. If the file exists but cannot be opened, inform the user of the issue.3. Create a program that takes an input number from the user and checks if it's a prime number. Implement the logic inside a function. Use a try-except block to handle any exceptions, and use the else block to print whether the number is prime or not.4. Write a Python program to simulate a simple ATM machine. Create a function that accepts the user's account balance and withdrawal amount. If the withdrawal amount is greater than the account balance, raise a "InsufficientFundsError" exception.

	<p>5. Write a program that takes a list of integers from the user and calculates the reciprocal of each number. Handle any exceptions that may occur during the division and continue the loop.</p> <p>6. Develop a program that reads a file and prints its contents. Handle the "FileNotFoundError" exception. Regardless of whether the file is found or not, use the "finally" block to display a message indicating the end of the program.</p> <p>7. Create a Python program that performs a series of mathematical operations on two user-input numbers. Handle exceptions such as "ZeroDivisionError" and "ValueError." If any exception occurs, provide a detailed error message.</p> <p>8. Write a program that defines a function that opens a specified file and reads its contents. If the file does not exist, the function should raise a "FileNotFoundError." In the main program, handle this exception and print an error message.</p> <p>9. Provide a Python code snippet with a deliberate error (e.g., a syntax error or a logical error) and ask the student to identify and fix the issue using the information provided in the exception message</p>
Theory:	<ol style="list-style-type: none"> 1. If zero is divided then it throw ZeroDivision error then appropriate message should be shown 2. file handling techniques are demonstrated, with a focus on catching both FileNotFoundError and PermissionError. 3. use of the finally block to actions are performed regardless of exceptions. 4. Create custom exception and raise exception when withdrawal amount is greater than balance_amount
Code:	<pre> 1. def divide(): try: a = int(input("Enter Number 1 : ")) b = int(input("Enter Number 2 : ")) c = a/b print("Answer is : ", c) except ZeroDivisionError: print("Cannot divide by zero... \nenter again... \n") divide() divide() Enter Number 1 : 9 Enter Number 2 : 0 Cannot divide by zero... enter again... Enter Number 1 : 5 Enter Number 2 : 2 Answer is : 2.5 </pre> <p>2.</p>

```

try:
    myfile = open("sample.txt","r")
    print("file does exist")
    content = myfile.read()
    print(content)
except FileNotFoundError:
    print("file does not exist")
except PermissionError:
    print("Premission denied")
file does exist
Lorem Ipsum is simply dummy text of the printing and typesetting industry.

3.
def isPrime(num):
    flag = False
    try:
        for i in range(2, num):
            if (num % i) == 0:
                flag = True
                break

        if flag:
            print(num, "- prime number")
        else:
            print(num, "- prime number")
    except:
        print("something went wrong")

isPrime(5)
isPrime(4)
isPrime(10)

----- ,
5 - prime number
4 - prime number
10 - prime number

4.
class InsufficientFundsError(Exception):
    "withdrawal amount is greater than the account balance"
    pass
main_balance = 10000
try:
    withdraw_amt = int(input("Enter a withdrawal amount: "))
    if withdraw_amt > main_balance:
        raise InsufficientFundsError
    else:
        print("collect money from machine.....")
        print("update balance : ", main_balance - withdraw_amt)

except InsufficientFundsError:
    print("Balance Low..!!")

```

```
Enter a withdrawal amount: 1266
collect money from machine.....
update balance : 8734
```

```
5.
def reciprocal():
    try:
        number_list = list(map(int,input("Enter number for list :").split(" ")))

        for x in number_list:
            c = 1/x
            print(c)
        except:
            print("something went wrong... \ntry agian...")
            reciprocal()
reciprocal()
```

```
Enter number for list :0 8
something went wrong...
try agian...
Enter number for list :7 3 5
0.14285714285714285
0.3333333333333333
0.2
```

```
6.
try:
    myfile = open("sample.txt","r")
    content = myfile.read()
    print(content)
except FileNotFoundError:
    print("File not found 404....")
finally:
    print("File read process ended....")
Lorem Ipsum is simply dummy text of the printing and typesetting industry.
File read process ended....
```

```
7.
def operations():
    try:
        a = int(input("Enter Number 1 : "))
        b = int(input("Enter Number 2 : "))
        add = a + b
        sub = a - b
        mul = a * b
        div = a / b
        print("all operation perform successfully")
    except ZeroDivisionError:
        print("Contains zero... \nenter again... \n")
        operations()
operations()
```

	<pre> Enter Number 1 : 4 Enter Number 2 : 0 Contains zero... enter again... Enter Number 1 : 4 Enter Number 2 : 5 all operation perform successfully </pre> <p>8.</p> <pre> def fileOperate(): try: file_name = input("Enter file name : ") myfile = open(file_name,"r") print("file does exist") content = myfile.read() print(content) except FileNotFoundError: print("file does not exist... try agian") fileOperate() fileOperate() Enter file name : sample.t file does not exist... try agian Enter file name : sample.txt file does exist Lorem Ipsum is simply dummy text of the printing and typesetting industry. </pre> <p>9.</p> <pre> def calculate_area(length, width): area = length * width return area length = 8 width = "4" try: result = calculate_area(length, width) print(f"Area: {result} square units") except TypeError as e: print(f"An error occurred: {e}") Area: 5555555555555555 square units </pre>
Screen Shot of Output:	
Observations:	Code effectively uses functions for modularity, incorporates error handling with try...except blocks, and engages user input. Showcases file handling practices, custom exceptions, mathematical operations, data type handling, and user-friendly messaging.

Conclusion:	In conclusion, the provided code examples offer a well-rounded introduction to file handling and exception handling. They cover a range of fundamental concepts and best practices, including error handling, user interaction, file operations, and custom exceptions. These examples provide a strong foundation for beginners, helping them develop coding skills and gain a deeper understanding of how to write robust and user-friendly Python applications.
--------------------	--

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 14

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 03-11-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about create class in python with constructor, data members and functions. And creating object of class.
Topics Covered:	Class creation, constructor, property , methods , and using data members with object creation.
Problem Statement:	<p>1.write a program to design a class to represent a car with attributes like make model year and speed each car object should contain 3 methods which can start the engine accelerate and slow down the speed.</p> <p>2.write a program to design a class to represent a rectangle with attributes like length and width each rectangle object should contain 2 methods which can compare the rectangle and compute the area.</p> <p>3.Design a class to represent a student with attributes like name, roll number, and marks. Create methods to calculate the grade and display student information.</p> <p>4.Implement a class for a library system with functions to add, delete, and search for books.</p> <p>5.Develop a Python class for a simple bank account. Create a constructor that initializes the account holder's name and balance. Include methods for depositing and withdrawing money, and ensure that the balance</p>

	<p>cannot go negative. Test the class by creating accounts and performing transactions.</p> <p>6. Create a class for representing employees in an organization. The constructor should take parameters for employee name, employee ID, and salary. Implement a method that calculates an annual bonus based on the salary and a given bonus percentage. Use this class to calculate the annual bonus for a set of employees.</p>
Theory:	<ol style="list-style-type: none"> 1. Object-Oriented Programming (OOP): The code demonstrates the use of classes and objects, encapsulating data and behavior in a structured manner. Class Construction: The code showcases class creation, constructor methods (e.g., <code>__init__</code>), and attributes (e.g., <code>self.name</code>, <code>self.salary</code>). 2. Method Usage: The code uses methods within classes to perform actions or calculations on class attributes. 3. Code Reusability: The principles of DRY (Don't Repeat Yourself) are followed by defining methods once and reusing them for different instances. 4. Pythonic Conventions: The code follows Python conventions for class names, method names, and the use of underscores in variable and function names. 5. Class Design and Modeling: The code exemplifies how to model real-world entities (e.g., employees, library books) in a program using classes and objects.
Code:	<pre> 1. class Car: def __init__(self,model,year,speed): self.model = model self.year = year self.speed = speed def start_engine(self): print(self.model, " - starting the car engine") def accelerate(self, plus_speed = 10): self.speed += plus_speed print(self.model, " - accelerarating the car engine at - ",self.speed , "km/h") def slow(self, minus_speed = 10): self.speed -= minus_speed if self.speed >= minus_speed: print(self.model, " - decresase the car engine at - ",self.speed, "km/h") else: print(self.model, " - car is stopeed") car1 = Car("Ferrai", "Karan", 7) car1.start_engine() car1.accelerate(10) car1.accelerate(10) car1.slow() car1.slow() car1.slow(5) car1.accelerate() car1.slow(10) </pre>

```

Ferrai - starting the car engine
Ferrai - accelarating the car engine at - 17 km/h
Ferrai - accelarating the car engine at - 27 km/h
Ferrai - decresase the car engine at - 17 km/h
Ferrai - car is stopeed
Ferrai - car is stopeed
Ferrai - accelarating the car engine at - 12 km/h
Ferrai - car is stopeed

```

2.

```
class Shape:
```

```
    def __init__(self, length, width):
        self.length = length
        self.width = width
```

```
    def area(self):
```

```
        raise NotImplementedError("Subclasses must implement this method")
```

```
    def compare(self, other_shape):
```

```
        if self.area() > other_shape.area():
            return "This shape is larger."
```

```
        elif self.area() < other_shape.area():
            return "This shape is smaller."
```

```
        else:
```

```
            return "Both shapes have the same area."
```

```
class Rectangle(Shape):
```

```
    def area(self):
```

```
        return self.length * self.width
```

```
rect1 = Rectangle(11, 2)
```

```
rect2 = Rectangle(30, 6)
```

```
print("Area of rectangle 1:", rect1.area(), "cm2")
```

```
print("Area of rectangle 2:", rect2.area(), "cm2")
```

```
print(rect1.compare(rect2))
```

```
Area of rectangle 1: 22 cm2
```

```
Area of rectangle 2: 180 cm2
```

```
This shape is smaller.
```

3.

```
class Student:
```

```
    def __init__(self, name, roll_number, marks):
```

```
        self.name = name
```

```
        self.roll_number = roll_number
```

```
        self.marks = marks
```

```
    def calculate_grade(self):
```

```
        average_marks = sum(self.marks) / len(self.marks)
```

```
        if average_marks >= 90:
```

```

        return 'A+'
    elif average_marks >= 80:
        return 'A'
    elif average_marks >= 70:
        return 'B'
    elif average_marks >= 60:
        return 'C'
    elif average_marks >= 50:
        return 'D'
    else:
        return 'F'

def display_info(self):
    print("Name: ",self.name)
    print("Roll Number: ",self.roll_number)
    print("Marks: ",self.marks)
    print("Average Marks: ",sum(self.marks) / len(self.marks))
    print("Grade: ",self.calculate_grade())
marks_list = list(map(int, input("Enter marks for each subject: ").split()))
s1 = Student("Karan Panchal", "33", marks_list)
s1.display_info()

```

```

-----
Enter marks for each subject: 60 70 80 90 99
Name:  Karan Panchal
Roll Number:  33
Marks:  [60, 70, 80, 90, 99]
Average Marks:  79.8
Grade:  B
-----

```

```

4.
class Library:
    books = []

    def add(cls, book):
        cls.books.append(book)
        print("Added", book)

    def delete(cls, book):
        if book in cls.books:
            cls.books.remove(book)
            print("Deleted", book)
        else:
            print(book, "is not in the library")

```

```

def search(cls, book):
    if book in cls.books:
        print(book, "is available in the library")
    else:
        print(book, "is not in the library")

def main():
    while True:
        option = int(input("Enter an option:
1 to add a book
2 to delete a book
3 to search for a book
4 to exit
"))

        if option == 1:
            book = input("Enter the book title to add: ")
            Library.add(book)
        elif option == 2:
            book = input("Enter the book title to delete: ")
            Library.delete(book)
        elif option == 3:
            book = input("Enter the book title to search: ")
            Library.search(book)
        elif option == 4:
            print("Thank you")
            break
        else:
            print("Invalid option. Please try again.")

main()

```

```

Enter an option:
1 to add a book
2 to delete a book
3 to search for a book
4 to exit
1
Enter the book title to add: cant hurt me
Added cant hurt me
Enter an option:
1 to add a book
2 to delete a book
3 to search for a book
4 to exit
2
Enter the book title to delete: man
man is not in the library
Enter an option:
1 to add a book
2 to delete a book
3 to search for a book
4 to exit
3
Enter the book title to search: cant hurt me
cant hurt me is available in the library
Enter an option:
1 to add a book
2 to delete a book
3 to search for a book
4 to exit
4
Thank you
5.

import random

class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_number = random.randint(1000, 9999)
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            return "Deposited ₹" + str(amount) + ". New balance: ₹" + str(self.balance)
        else:
            return "Invalid deposit amount. Please enter a positive amount."

    def withdraw(self, amount):
        if amount > 0:
            if amount <= self.balance:
                self.balance -= amount
                return "Withdrew ₹" + str(amount) + ". New balance: ₹" + str(self.balance)
            else:
                return "Insufficient funds. Cannot withdraw."
        else:
            return "Invalid withdrawal amount. Please enter a positive amount."

    def get_balance(self):

```

	<pre> return "Account Number: " + str(self.account_number) + ", Holder: " + self.account_holder + ", Balance: ₹" + str(self.balance) account1 = BankAccount("Karan", 10000) account2 = BankAccount("Panchal", 5000) print(account1.get_balance()) print(account2.get_balance()) print(account1.deposit(200)) print(account2.deposit(300)) print(account1.withdraw(400)) print(account2.withdraw(700)) print(account1.withdraw(1500)) Account Number: 4440, Holder: Karan, Balance: ₹10000 Account Number: 1773, Holder: Panchal, Balance: ₹5000 Deposited ₹200. New balance: ₹10200 Deposited ₹300. New balance: ₹5300 Withdrew ₹400. New balance: ₹9800 Withdrew ₹700. New balance: ₹4600 Withdrew ₹1500. New balance: ₹8300 6. class Employee: def __init__(self, name, employee_id, salary): self.name = name self.employee_id = employee_id self.salary = salary def calculate_annual_bonus(self, bonus_percentage): bonus = self.salary * (bonus_percentage / 100) return "Employee ID: " + str(self.employee_id) + ", Name: " + self.name + ", Annual Bonus: ₹" + str(bonus) # Example usage: employee1 = Employee("KP", 1001, 50000) employee2 = Employee("PS", 1002, 60000) employee3 = Employee("P", 1003, 70000) bonus_percentage = 10 # Example bonus percentage print(employee1.calculate_annual_bonus(bonus_percentage)) print(employee2.calculate_annual_bonus(bonus_percentage)) print(employee3.calculate_annual_bonus(bonus_percentage)) Employee ID: 1001, Name: KP, Annual Bonus: ₹5000.0 Employee ID: 1002, Name: PS, Annual Bonus: ₹6000.0 Employee ID: 1003, Name: PP, Annual Bonus: ₹7000.0 </pre>
Screen Shot of Output:	

Observations :	<p>When you create a class in Python, it's like designing a blueprint for an object. The constructor is like the instruction manual for creating that object. It tells you how to set up the object when you first make it.</p> <p>Think of the object as something you want to describe. It has characteristics (attributes) and things it can do (methods). The constructor helps you define these characteristics and abilities when you create an object based on your class. So, it's like setting up the object with its initial information and what it can do.</p>
Conclusion:	<p>We learned how to create a Python class to represent employees in an organization. The class constructor initializes attributes for employee name, ID, and salary. It includes a method for calculating annual bonuses based on a specified percentage. The code follows essential Object-Oriented Programming (OOP) principles, such as encapsulation and method usage, while ensuring code reusability by defining the bonus calculation logic in one place. Additionally, the code illustrates string manipulation techniques for constructing output . Overall, we exemplify how to design and work with classes to model real-world entities and apply OOP concepts in Python.</p>

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 15

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 20-11-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about inheritance in python.
Topics Covered:	Inheritance , sub class, parent class
Problem Statement:	Q1) WAPP to demonstrate inheritance in animal. Call subclasses with parent class member function Q) WAPP to demonstrate inheritance in employee. Call subclass Trainee and manager with parent class member function
Theory:	1. Base class called Animal with an initializer (<code>__init__</code>) method and an abstract method <code>make_sound</code> . Dog and Cat are derived classes from Animal and they override the <code>make_sound</code> method. dog and cat are instances of Dog and Cat classes. The print statements demonstrate how to access the attributes and call the overridden methods for dog and cat objects. 2. Base class Employee with an initializer and a method <code>display_details</code> . Trainee and Manager are derived classes from Employee and they override the <code>display_details</code> method. trainee and manager are instances of Trainee and Manager classes. The print statements demonstrate how to access the attributes and call the overridden methods for trainee and manager objects.
Code:	1. class Animal: def <code>__init__</code> (self, name, species): self.name = name

```

        self.species = species
    # abstract method
    def make_sound(self):
        pass
class Dog(Animal):
    def make_sound(self):
        return "woofff.....!!!"
class Cat(Animal):
    def make_sound(self):
        return "Meow.....!!!!"
dog = Dog("Leo", "Canine")
cat = Cat(species="Luna",name="Feline")
print(dog.name + " the " + dog.species + " says: " + dog.make_sound())
print(cat.name + " the " + cat.species + " says: " + cat.make_sound())

```

```

Leo the Canine says: woofff.....!!!
Feline the Luna says: Meow.....!!!!

```

```

2.
class Employee:
    def __init__(self, emp_name, emp_salary):
        self.emp_name = emp_name
        self.emp_salary = emp_salary

    def display_details(self):
        print("Employee Name:", self.emp_name)
        print("Employee Salary: ₹", self.emp_salary)

class Trainee(Employee):
    def __init__(self, tr_name, tr_salary, training_duration):
        # constructor of the parent class
        Employee.__init__(self, tr_name, tr_salary)
        self.training_duration = training_duration

    def display_details(self):
        # display_details method of the parent class
        Employee.display_details(self)
        print("Training Duration:", self.training_duration, "months")

class Manager(Employee):
    def __init__(self, mgr_name, mgr_salary, team_size):
        Employee.__init__(self, mgr_name, mgr_salary)
        self.team_size = team_size

    def display_details(self):
        Employee.display_details(self)
        print("Team Size:", self.team_size)

trainee = Trainee("Karan", 150000, 6)
manager = Manager("KP", 90000, 10)

print("Manager Details:")
manager.display_details()

```

	<pre>print("\nTrainee Details:") trainee.display_details() Manager Details: Employee Name: KP Employee Salary: ₹ 90000 Team Size: 10 Trainee Details: Employee Name: Karan Employee Salary: ₹ 150000 Training Duration: 6 months</pre>
Screen Shot of Output:	
Observations :	The use of inheritance allows the derived classes (Dog, Cat, Trainee, and Manager) to reuse and extend the functionality of the base class (Animal and Employee). Both examples illustrate the object-oriented concept of organizing code hierarchically for clarity, efficiency, and extensibility making reusability and maintainability for scaling.
Conclusion:	Polymorphism is showcased through overridden methods, enabling objects of different classes to be used interchangeably. This approach enhances code organization, reusability, and flexibility. Overall, the code illustrates the power of object-oriented programming in creating modular and extensible systems, where commonalities are captured in base classes, and specific functionalities are extended in derived classes.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 16

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 21-11-2023

FULL NAME: Karan Dinesh Panchal

Aim:	Implement a python program based on Access modifier in python Private
Topics Covered:	Encapsulation of data using private, public(default), protected
Problem Statement:	1. WAPP to demonstrate access modifiers in <ul style="list-style-type: none">• private• public• protected
Theory:	<ol style="list-style-type: none">1. class myclass is defined with a private attribute <code>_private_attribute</code>. The <code>get_private_attribute</code> method is defined to retrieve the value of the private attribute. The instance <code>obj</code> of <code>myclass</code> is created. To access the private attribute, the syntax used is <code>obj._myclass__private_attribute</code>. name mangling is applied to make the attribute private by adding a prefix with the class name.2. It single method public <code>print1</code> that prints a message. An instance <code>obj</code> is created, and the <code>print1</code> method is called.No mangling means its public by default.3. Baseclass is defined with a protected variable <code>_protected_variable</code> and a protected method <code>_protected_method</code>. Then, a derivedclass is defined that inherits from baseclass. An instance <code>derived_object</code> is created.The <code>access_protected</code> method of <code>derivedclass</code> is called, which prints the value of the protected variable and calls the protected method.

Code:	<pre> 1. class myclass: def __init__(self): self.__private_attribute=33 def get__private_attribute(self): return self.__private_attribute obj=myclass() print("from private data member ",obj.__myclass__private_attribute) print("from private data member from method",obj.get__private_attribute()) from private data member 33 from private data member from method 33 2. class myclass: def print1(self): print("This is a public method") obj=myclass() obj.print1() This is a public method 3. class baseclass: def __init__(self): self._protected_variable=40 def _protected_method(self): print("This is protected method") class derivedclass(baseclass): def access_protected(self): print(self._protected_variable) self._protected_method() derived_object=derivedclass() derived_object.access_protected() print(derived_object._protected_variable) derived_object._protected_method() derived class - 40 baseClass - This is protected method protected variable - 40 baseClass - This is protected method </pre>
Screen Shot of Output:	
Observations :	<p>Above Code, myclass with a private attribute accessed using name mangling, myclass with a public method, and baseclass with a protected variable and method, inherited by derivedclass. The code demonstrates encapsulation concepts: private attributes with name mangling for restricted access, public methods for open access, and protected members for inheritance. It showcases different levels of member visibility, emphasizing encapsulation principles in object-oriented programming.</p>
Conclusion:	<p>The assignment code provides a concise illustration of encapsulation concepts, from pseudo-privacy to transparent accessibility and controlled inheritance, capturing the essence of object-oriented programming in Python.</p>

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 17

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 01-12-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about import inbuilt and custom module in python
Topics Covered:	Python inbuilt import, custom import from same parent project folder
Problem Statement:	<ol style="list-style-type: none">1. Choose two different modules from the Python standard library. Write a Python script that integrates both modules to perform a specific task or solve a problem.2. Create a Python module that acts as a simple calculator. Include functions for addition, subtraction, multiplication, and division. Write a script to import and use this module to perform arithmetic operations.3. Create a Python script that uses the math module to calculate the area and circumference of a circle, the volume of a sphere, and the hypotenuse of a right-angled triangle. Allow the user to input relevant parameters.4. Write a Python script that generates a random password of a specified length. Utilize the random module to include a mix of uppercase letters, lowercase letters, numbers, and special characters.5. Develop a simple task scheduler using the time module. Allow the user to input tasks along with their scheduled execution times. The program

	should execute the tasks at the specified times.
Theory:	<p>In this code , the 'import' statement facilitates the integration of external modules, enhancing code flexibility and reusability. It allows access to functions, classes, or variables from other modules, with syntax like 'import module' or 'from module import item.' Aliasing with 'as' provides name customization, while '*' imports all items. Python's standard library boasts diverse modules, and packages organize related ones. Ensure modules are in the script's directory or modify 'sys.path.'</p> <p>Importing self-made modules follows similar principles.</p>
Code:	<pre> 1. import time import random def handleRandom(): length = random.randint(1, 10) breadth = random.randint(1, 50) area = length * breadth print("Area of rectange is :", area) def handleAfterTime(): print("called handleAfterTime!!! \nprogram will now wait for some seconds.....") for x in range(3,0,-1): time.sleep(1) print(x , " seconds...") handleRandom() print("called after handleRandom()!!!!") handleAfterTime() called handleAfterTime!!! program will now wait for some seconds..... 3 seconds... 2 seconds... 1 seconds... Area of rectange is : 8 called after handleRandom()!!!! 2. calculator.py def add(a, b): return a + b def subtract(a, b): return a - b def multiply(a, b): return a * b def divide(a, b): if b == 0: return "Cannot divide by zero!" return a / b main.py import calculator </pre>

```

num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))
print("Addition :",calculator.add(num1, num2))
print("Subtracion :",calculator.subtract(num1, num2))
print("Multiplication : ",calculator.multiply(num1, num2))
print("Division : ",calculator.divide(num1, num2))

```

```

Enter first number: 34
Enter second number: 23
Addition : 57
Subtracion : 11
Multiplication : 782
Division : 1.4782608695652173
|

```

```

3.
import math
def circle(radius):
    area = math.pi * radius ** 2
    circumference = 2 * math.pi * radius
    return area, circumference
def sphere(radius):
    volume = (4/3) * math.pi * radius ** 3
    return volume
def triangle(a, b):
    hypotenuse = math.sqrt(a ** 2 + b ** 2)
    return hypotenuse
radius_circle = int(input("Enter the radius of the circle: "))
radius_sphere = int(input("Enter the radius of the sphere: "))
side_a = int(input("Enter the length of side a of the triangle: "))
side_b = int(input("Enter the length of side b of the triangle: "))

```

```

circle_area, circle_circumference = circle(radius_circle)
print("Area of the circle: ", circle_area)
print("Circumference of the circle: ",circle_circumference)

```

```

sphere_vol = sphere(radius_sphere)
print("Volume of the sphere: ",sphere_vol)

```

```

hypotenuse = triangle(side_a, side_b)
print("Hypotenuse of the triangle: ",hypotenuse)

```

```

Enter the radius of the circle: 5
Enter the radius of the sphere: 7
Enter the length of side a of the triangle: 3
Enter the length of side b of the triangle: 6
Area of the circle: 78.53981633974483
Circumference of the circle: 31.41592653589793
Volume of the sphere: 1436.7550402417319
Hypotenuse of the triangle: 6.708203932499369

```

```

4.
import random
import string
def generate_password(length=12):
    characters = string.ascii_letters + string.digits + string.punctuation
    password = ''.join(random.choice(characters) for _ in range(length))
    return password

```

	<pre> random_password = generate_password(int(input("enter a number"))) print("Random Password:", random_password) enter a number10 Random Password: 8^0~f\UM-2 5. import time import math def getInput(): radius = int(input("enter radius of circle : ")) get_time = int(input("enter time to wait in seconds")) print("program execution will wait for",get_time,"seconds.....") time.sleep(get_time) area_of_circle(radius) def area_of_circle(radius): print("area of circle :", math.pi * radius * radius) getInput() enter radius of circle : 10 enter time to wait in seconds3 program execution will wait for 3 seconds..... area of circle : 314.1592653589793 </pre>
Screen Shot of Output:	
Observations :	The import statement in Python serves as a fundamental mechanism for incorporating external modules into scripts, promoting code modularity and reuse. It provides flexibility in choosing specific elements from modules, supports aliasing for readability, and facilitates organization through packages.
Conclusion:	Proficiency in using the import statement is crucial for writing efficient, modular, and maintainable Python code. It enables developers to leverage existing libraries, improve code structure, and enhance collaboration. Overall, a solid understanding of import mechanisms contributes to the development of scalable and robust Python applications.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 18

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No:33

DATE: 04-09-2023

FULL NAME: Karan Dinesh Panchal

Aim:	To learn about developing GUI using tkinter
Topics Covered:	Tkinter GUI window, components, handle submit button, and display information
Problem Statement:	1. Imagine you are tasked with developing a registration form for an upcoming event. Create a Python script using Tkinter to design a GUI registration form. The form should include the following fields: A text entry for the participant to enter their full name. An entry for the participant's age. An entry for the participant's email address. Include a dropdown menu or radio buttons for the participant to select their event category (e.g., Workshop, Seminar, Networking). Submit button- to print filled form details
Theory:	In this code snippet, the Tkinter library is employed by importing it with the statement <code>import tkinter as tk</code> . Tkinter is a popular library for creating graphical user interfaces in Python. The script then defines a function named <code>submit_form</code> , which is called when the user clicks the "Submit" button. The GUI components, known as widgets, are created using Tkinter elements like Label, Entry, <code>tk.Combobox</code> , and Button. Labels are used to display text prompts such as "Full Name," and Entry widgets allow users to input data, such as their name, age, and email. The event category is collected through a dropdown, providing a dropdown

	<p>menu with event options. The values entered by the user are retrieved through functions like get() for Entry widgets and get() for the dropdown. Finally, the script employs the print function to display the entered information in the console when the user clicks the "Submit" button. Overall, this code effectively combines Tkinter widgets to create a functional and user-friendly registration form.</p>
Code:	<pre> 1. import tkinter from tkinter import * root=tkinter.Tk() root.title("Registration Form") root.geometry("800x800") def handleSubmit(): name_fetch_value=name_Input.get() age_fetch_value=age_Input.get() email_fetch_value=email_Input.get() drop_fetch_value=drop.get() print("Name: ",name_fetch_value) print("Age: ",age_fetch_value) print("Email: ",email_fetch_value) print("Drop: ",drop_fetch_value) # frame formFrame = Frame(root) formFrame.pack(pady=100) # heading heading = Label(formFrame, text="Register", font=("Bookman Old Style", 20 , "bold")) heading.pack() name = Label(formFrame, text="Full Name", font=("Bookman old style", 16),) name.pack(padx=10,pady=5) name_Input = Entry(formFrame) name_Input.pack(padx=10,pady=5) age = Label(formFrame, text="Age", font=("Bookman old style", 16),) age.pack(padx=10,pady=5) age_Input = Entry(formFrame) age_Input.pack(padx=10,pady=5) </pre>

```

email = Label(
    formFrame,
    text="Email",
    font=("Bookman old style", 16),
)
email.pack(padx=10,pady=5)

email_Input = Entry(formFrame)
email_Input.pack(padx=10,pady=5)

options = [
    "Workshop",
    "Seminar",
    "Networking",
]

clicked = StringVar()
clicked.set( "Seminar" )
drop = OptionMenu( formFrame , clicked , *options )
drop.pack(padx=10,pady=5)
submit_button=Button(root,text="Submit",command=handleSubmit)
submit_button.pack()
root.mainloop()

```

```

Name: karan
Age: 21
Email: karan.dp@somaiya.edu
Drop: Networking

```

Screen Shot of Output:	
Observations :	Tkinter to create a user-friendly GUI registration form. It features text entry fields for the participant's name, age, and email, alongside a dropdown menu for event category selection. The layout is well-organized, and the "Submit" button triggers a function to retrieve and print the entered details. This code serves as a clear example of using Tkinter widgets for basic interactive applications, providing a foundation for more complex GUI development. It showcases an efficient approach to capture user input and demonstrates the potential for expanding functionality based on specific project needs.
Conclusion:	In conclusion learn how to create a user interface for a registration form using Tkinter. It demonstrates how to design text entry fields, a dropdown menu, and a submit button to capture user information. The script's structure and use of Tkinter functions provide insights into building interactive applications. Overall, it serves as a practical introduction to GUI development in Python, offering a foundation for creating more interfaces tailored to project requirements.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar



SOMAIYA
VIDYAVIHAR UNIVERSITY

K J Somaiya Institute of Management

Department of Data Science and Technology

Practical No: 19

Subject: Python Programming Lab

MCA / Sem I / Python Programming [Course Code : 217P09L102]

ROLL No: 33

DATE: 11-12-2023

FULL NAME: Karan Dinesh Panchal

Aim:	Develop a GUI program with Database connection.
Topics Covered:	GUI development using tkinter, database connection using SQLite, CRUD operations.
Problem Statement:	1. Develop a database-driven GUI program to manage student information. Set up database connection and Implement features for adding, updating, and deleting student records.
Theory:	The underlying theory behind GUI development in Python involves utilizing libraries like Tkinter, which provides tools for creating interactive interfaces. GUIs comprise widgets such as buttons, text fields, and labels arranged using layout management techniques. These interfaces operate on an event-driven paradigm, responding to user actions like button clicks or text input changes.
Code:	1. import sqlite3 import tkinter as tk # initialization


```

conn = sqlite3.connect('student_records.db')
cursor = conn.cursor()

# Create student table if it doesn't exist
cursor.execute("""
    CREATE TABLE IF NOT EXISTS students (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        name TEXT NOT NULL,
        age INTEGER,
        grade TEXT
    )
""")
conn.commit()

# new student record
def add_student():
    name = name_entry.get()
    age = age_entry.get()
    grade = grade_entry.get()
    cursor.execute('INSERT INTO students (name, age, grade) VALUES (?, ?, ?)', (name,
age, grade))
    conn.commit()
    success_label.config(text='Student record added successfully')
    display_data_in_listbox()

# existing student record
def update_student():
    selected_id = id_entry.get()
    new_name = name_entry.get()
    new_age = age_entry.get()
    new_grade = grade_entry.get()
    cursor.execute('UPDATE students SET name=?, age=?, grade=? WHERE id=?',
(new_name, new_age, new_grade, selected_id))
    conn.commit()
    success_label.config(text='Student record updated successfully')
    display_data_in_listbox()

# student record
def delete_student():
    selected_id = id_entry.get()
    cursor.execute('DELETE FROM students WHERE id=?', (selected_id,))
    conn.commit()
    success_label.config(text='Student record deleted successfully')
    display_data_in_listbox()

# display table data in Listbox
def display_data_in_listbox():
    data_list.delete(0, tk.END) # Clear existing data in Listbox
    cursor.execute('SELECT * FROM students')
    records = cursor.fetchall()

    for record in records:
        data_list.insert(tk.END, f"ID: {record[0]}, Name: {record[1]}, Age: {record[2]},
Grade: {record[3]}")

```

```

root = tk.Tk()
root.title('Student Records Management')

# Labels
tk.Label(root, text='ID:').grid(row=0, column=0)
tk.Label(root, text='Name:').grid(row=1, column=0)
tk.Label(root, text='Age:').grid(row=2, column=0)
tk.Label(root, text='Grade:').grid(row=3, column=0)
success_label = tk.Label(root, text="")
success_label.grid(row=7, columnspan=2)

# Entry fields
id_entry = tk.Entry(root)
id_entry.grid(row=0, column=1)
name_entry = tk.Entry(root)
name_entry.grid(row=1, column=1)
age_entry = tk.Entry(root)
age_entry.grid(row=2, column=1)
grade_entry = tk.Entry(root)
grade_entry.grid(row=3, column=1)

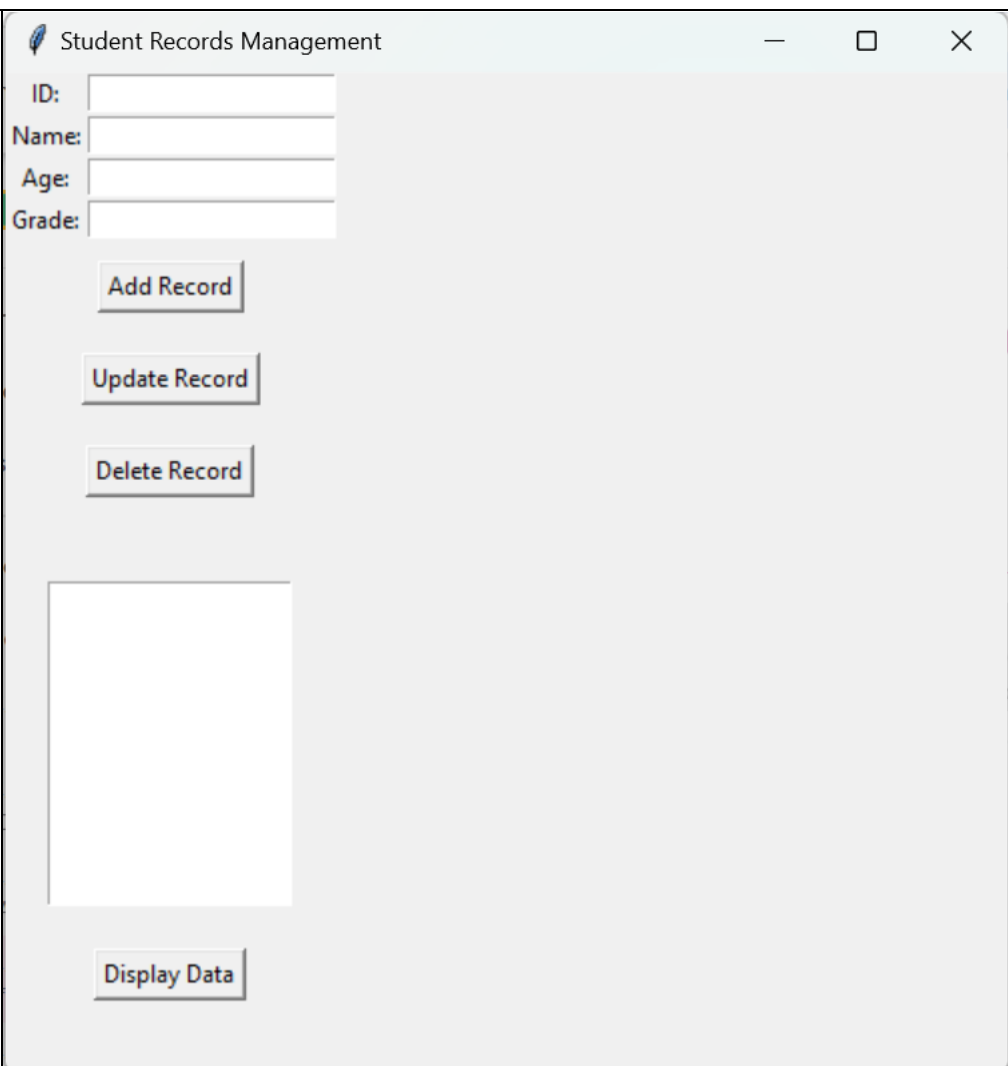
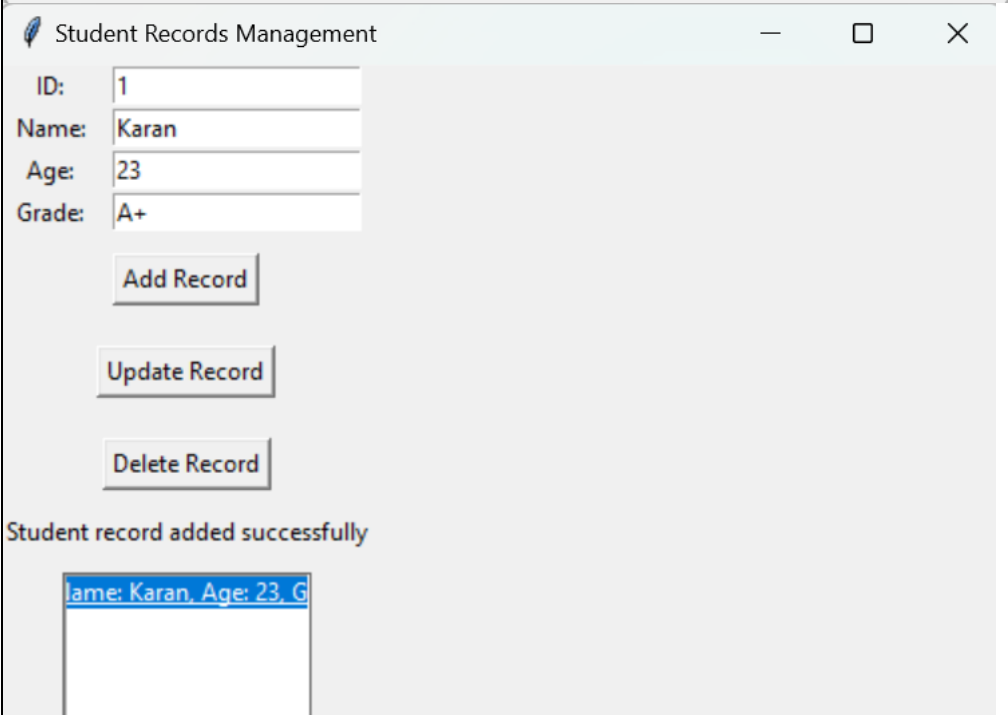
# Listbox for displaying data
data_list = tk.Listbox(root)
data_list.grid(row=8, column=0, columnspan=2, pady=10)

# Buttons
tk.Button(root, text='Add Record', command=add_student).grid(row=4, column=0,
columnspan=2, pady=10)
tk.Button(root, text='Update Record', command=update_student).grid(row=5, column=0,
columnspan=2, pady=10)
tk.Button(root, text='Delete Record', command=delete_student).grid(row=6, column=0,
columnspan=2, pady=10)
tk.Button(root, text='Display Data', command=display_data_in_listbox).grid(row=9,
column=0, columnspan=2, pady=10)

# Initial display of data in Listbox
display_data_in_listbox()

root.mainloop()

```

	<div data-bbox="354 183 1366 1245">  </div> <div data-bbox="354 1245 1366 1957">  </div>
Screen Shot of Output:	

Observations:	Observing the interface in action reveals its user-friendly nature, allowing seamless data entry, updates, and deletions. However, the absence of robust error handling mechanisms could potentially lead to issues with invalid inputs or database connectivity problems, affecting the program's reliability.
Conclusion:	The conclusion drawn from this are the GUI's effectiveness in enhancing user interaction with the application and its ability to integrate with databases for data management. Nevertheless, there's room for improvement, particularly in implementing error handling mechanisms to fortify the program's stability and incorporating additional functionalities like data validation and search capabilities. Expanding the project's scope could involve exploring advanced GUI features and other Python GUI libraries to create more comprehensive and versatile applications.

Subject-In-Charge:

Sign: _____

Prof. Mayura Nagar

Practical No : 20

Project overview

1. Objective:

The Hospital Management System (HMS) is designed to streamline and automate administrative and operational tasks within a healthcare facility. By utilizing Python and Tkinter, this project aims to provide a user-friendly Graphical User Interface (GUI) that enhances the efficiency of hospital management.

2. Key Features:

1. User Authentication:
 - . Secure login system with different access levels for administrators, staff, and receptionists.
 - . Protection of sensitive information through encrypted authentication.
2. Patient Management:
 - . Registration and tracking of patient information, including personal details and medical history.
 - . Search and update functionalities for managing patient records efficiently.
2. Appointment Scheduling:
 - . Intuitive scheduling interface with a calendar widget for selecting dates and times.
 - . Integration with patient management to assign and track appointments.
2. Billing:
 - . Automation of billing processes to generate accurate invoices for services rendered.
 - . Transparent and accountable financial system for efficient financial management.
2. Doctor and Staff Management:
 - . Efficient management of doctors and staff schedules, assignments, and evaluations.
 - . Streamlined allocation of responsibilities to enhance operational efficiency.
2. User-Friendly Interface:
 - . Intuitive GUI design using Tkinter for a seamless and visually appealing user experience.
 - . Accessibility for both novice and experienced users.