UNIT-5
Python packages

Today's Target

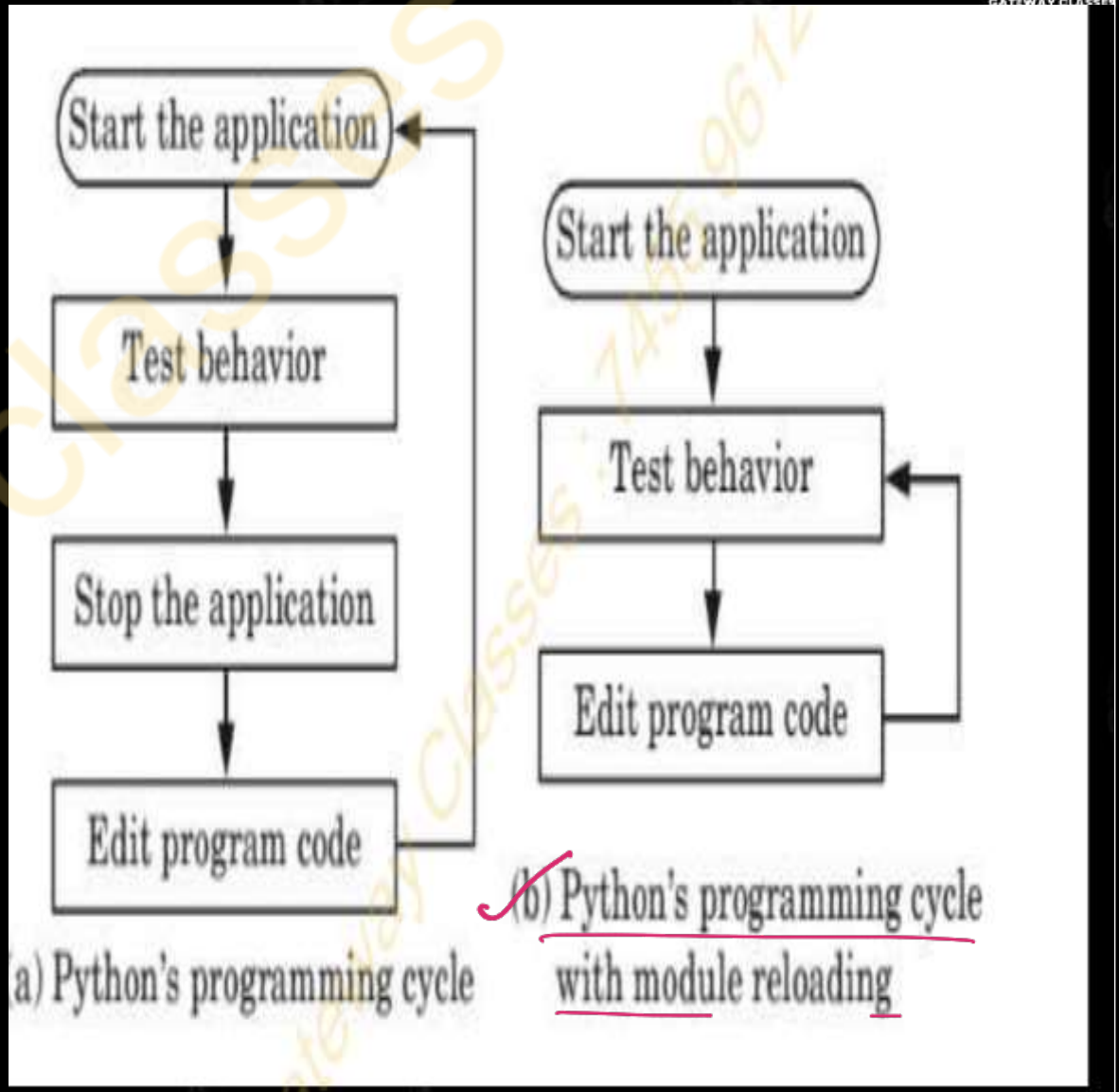
- Most important question

By PRAGYA RAJVANSHI

B.Tech, M.Tech(C.S.E)

Python's programming cycle (AKTU 2019-20) (2021-22)(2022-23 odd+even)

- It is dramatically shorter than that of traditional programming cycle.
- there are no compile or link steps.
- Python programs simply import modules at runtime and use the objects they contain. Because of this, Python programs run immediately after changes are made.
- In cases where dynamic module reloading can be used, it is even possible to change and reload parts of a running program without stopping it at all.



what is IDE (AKTU 2021-22)/(2022-23odd+even)

An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application.

WHY IDE IS IMPORTANT

- It contains many intelligent features for automatically writing or editing the source code
- It can format the written text by automatically making some words bold or italic, or by using different font colors
- an IDE can make suggestions to complete a code statement when the developer begins typing.
- Debugging is the process of fixing any errors or bugs that testing reveals
- **Some python IDE**
- PyCharm
- Spyder
- Pydev
- IDLE
- VISUAL STUDIO

PyCharm

- PyCharm is a hybrid platform developed by JetBrains as an IDE for Python.
- Some of the unicorn organizations such as Twitter, Facebook, Amazon, and Pinterest use PyCharm their Python IDE
- It helps us write high-quality codes!
- It consists of color schemes for keywords, classes, and functions. This helps increase the readability and understanding of the code.
- It helps identify errors easily
- It helps developers create web applications in Python.
- A programmer can locate an element, a symbol, or a variable in the source code within no time.
- Using the lens mode, further, a developer can thoroughly inspect and debug the entire source code.
- SPYDER

Spyder is widely used for data science works. It is mostly used to create a secure and scientific environment for Python. Spyder Python uses PyQt (Python plug-in) which a developer can add as an extension.

- It has good syntax highlighting and auto code completion
- Spyder Python explores and edits variables directly from GUI.
- It performs very well in multi-language editor.

PYDEV

- It is designed to provide a complete development environment for Python programmers.
- Also, it is built on top of the Eclipse platform and supports various features like debugging, code analysis, code completion, and much more.
- PyDev is compatible with all the major operating systems like Windows, Linux, and Mac OS X, making it a popular choice among developers.

VISUAL STUDI0

- It supports Python coding in visual studio, debugging, and other activities. ii. It has both paid and free versions in the market with great features.
- A visual studio code is a lightweight software application with a powerful source code editor that runs on the desktop

- **Discuss why Python is called as dynamic and strongly typed language (AKTU 2021-22 even)**
- Dynamic typing means that the type of a variable is determined at runtime, not in advance. You don't have to declare the type of a variable when you create it;
- x = 10 # x is an integer
- x = "Hello" # Now x is a string
- Strong typing means that once a variable is assigned a value of a certain type, you cannot use it as if it were another type without an explicit conversion

- x = "Hello"
- y = 10
- print(x + y) # This will raise a TypeError because you can't add a string to an integer

Show the way to import the module in python(AKTU 2022-23 odd)

You can import the entire module using import statement

Import math

Element of python(AKTU 2019-20 ODD)

These are the basic element of python

Datatype ,Operator, Variable, Function, keywords

- **Python Keywords** are some predefined and reserved words in Python that have special meanings. Keywords are used to define the syntax of the coding
- **a variable** is a name that refers to a value stored in memory, which can be changed and used throughout a program

BOOLEAN EXPRESSION(AKTU 2019-20 ODD)

Boolean expressions are the expressions that evaluate a condition and result in a Boolean value i.e true or false.

(a>b and a> c) is a Boolean expression

Decorator in python(AKTU 2919-20)

A decorator is like a wrapper around a function. It allows you to add extra functionality to an existing function without changing its code.

you want to add some extra behavior to this function. For example, you want to print a message before and after saying hello.

```python
def say_hello():
    print("Hello!")

```

```python
@my_decorator
def say_hello():
    print("Hello!")

```

```python
def my_decorator(func):
    def wrapper():
        print("Something is happening before the function is called.")
        func()
        print("Something is happening after the function is called.")
    return wrapper

```

```
Something is happening before the function is called.
Hello!
Something is happening after the function is called
```

Tools to perform bug to perform static analysis (AKTU 2019-20)

- Manually inspecting every line of Python code can be daunting and time-consuming. That is where static code analysis comes into play.
- Static code analysis is the procedure of inspecting application source code without executing it
- Pychecker and Pylint are the static analysis tools that help to find bugs in python.
- Pychecker is an opensource tool for static analysis that detects the bugs from source code and warns about the style and complexity of the bug.
- Pylint is highly configurable and it acts like special programs to control warnings and errors
- pylint is also an open source tool for static code analysis it looks for programming errors and is used for coding standard.

What is PEP(2019-20)

PEP (Python Enhancement Proposal) is like a suggestion box for the Python programming language. It's a way for people who work on Python to suggest new ideas or changes to Python, and it helps everyone discuss, plan, and decide if those ideas are good for Python.

PEP stands for "Python Enhancement Proposal." PEPs are design documents that provide information to the Python community or describe a new feature or improvement in the Python programming language. They are the primary mechanism for proposing and discussing changes to Python's core language and standard library

PEP8

PEP 8 is a Python Enhancement Proposal that outlines the style guide for writing clean and readable Python code. It provides a set of conventions and guidelines to make Python code consistent and easy to understand.

Some key points from PEP 8 in plain language

- Use 4 spaces for each level of indentation. Don't use tabs. Consistent indentation makes your code look neat and organized
- Limit each line of code to a maximum of 79 characters for code, and 72 character
- Write comments to explain complex or non-obvious parts of your code.
- Use blank lines to separate functions, classes, and blocks of code inside functions for better readability.

How memory is managed in python

n In Python, memory management, including the allocation and deallocation of memory on the private heap, is handled by the Python memory manager. Python uses a private heap to manage memory, and it employs a system of reference counting along with a cyclic garbage collector to keep track of and clean up memory when it's no longer in use

a=100

b=100

Every python object holds three things

object type

object value

object reference counter

Type int , Value 100 , reference count 2 , reference a, b Example

```
a = 100
print(id(a)) # 10914336
b = 100 print(id(b)) # 10914336
print(id(a) == id(b)) # True
```

- When you create a new object in Python (e.g., a variable, list, dictionary, etc.), the memory for that object is allocated from the private heap
- When the reference count of an object drops to zero (i.e., there are no more references to the object), Python's memory manager deallocates the memory associated with that object.

Cyclic Garbage Collector: Handles reference cycles where objects reference each other, preventing reference count from reaching zero

What is python module? How to create and import a module in python?(AKTU 2021-22 ODD+EVEN)

Python Module is a file that contains built-in functions, classes, and variables. There are many Python modules, each with its specific work,

A Python module is a file containing Python definitions and statements. A module can define functions, classes, and variables. A module can also include runnable code.

To create a Python module, write the desired code and save that in a file with .py extension

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

def subtract(x, y):
    return (x-y)
```

Different methods of importing a module

1.Importing modules

```
1  # importing sqrt() and factorial from the
2  # module math
3  import math
4  print(math.sqrt(16))
5  print(math.factorial(6))
6
7
```

2. Import Specific Attributes from a Python module

```
# importing sqrt() and factorial from the
# module math
from math import sqrt, factorial
print(sqrt(16))
print(factorial(6))
```

Different methods of importing a module

Import the whole module and rename it, usually using a shorter variable name:

```
1  # importing sqrt() and factorial from the
2  # module math
3  import math as mt
4
5  # if we simply do "import math", then
6  # math.sqrt(16) and math.factorial()
7  # are required.
8  print(mt.sqrt(16))
9  print(mt.factorial(6))
10
11
```

```
4.0
720
```

Different methods of importing a module

import specific things from the module and rename them as you're importing them

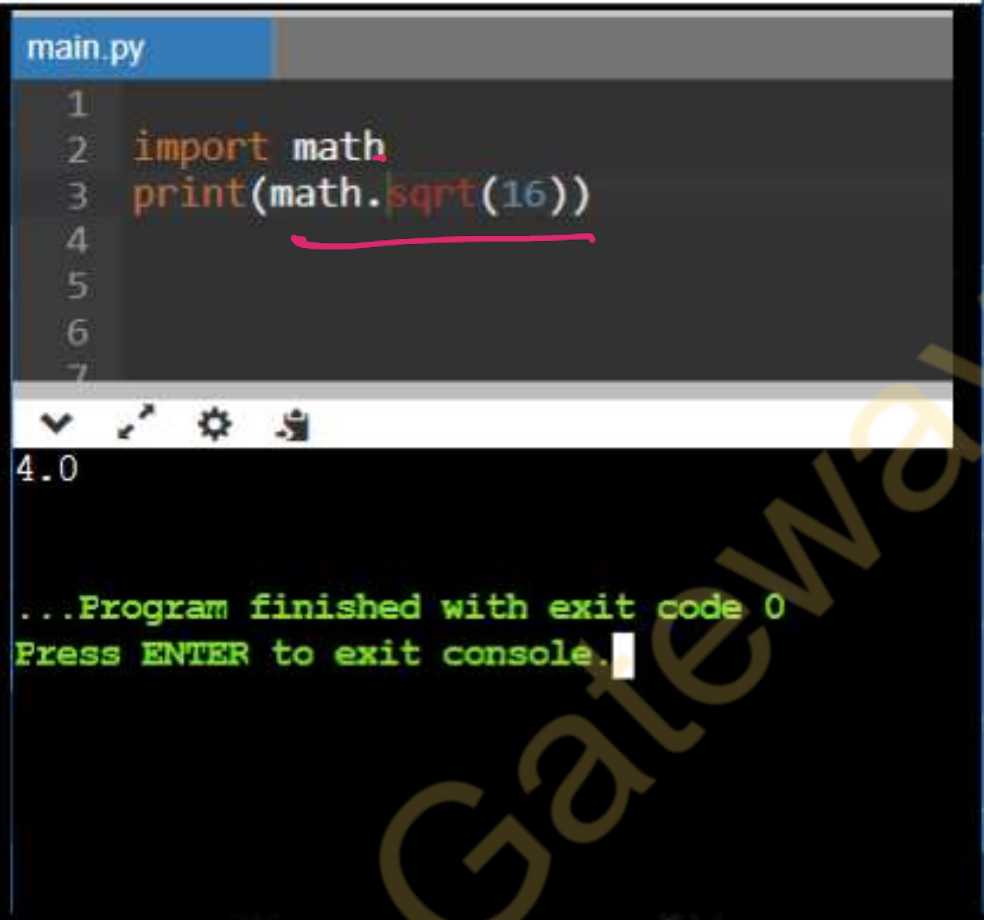
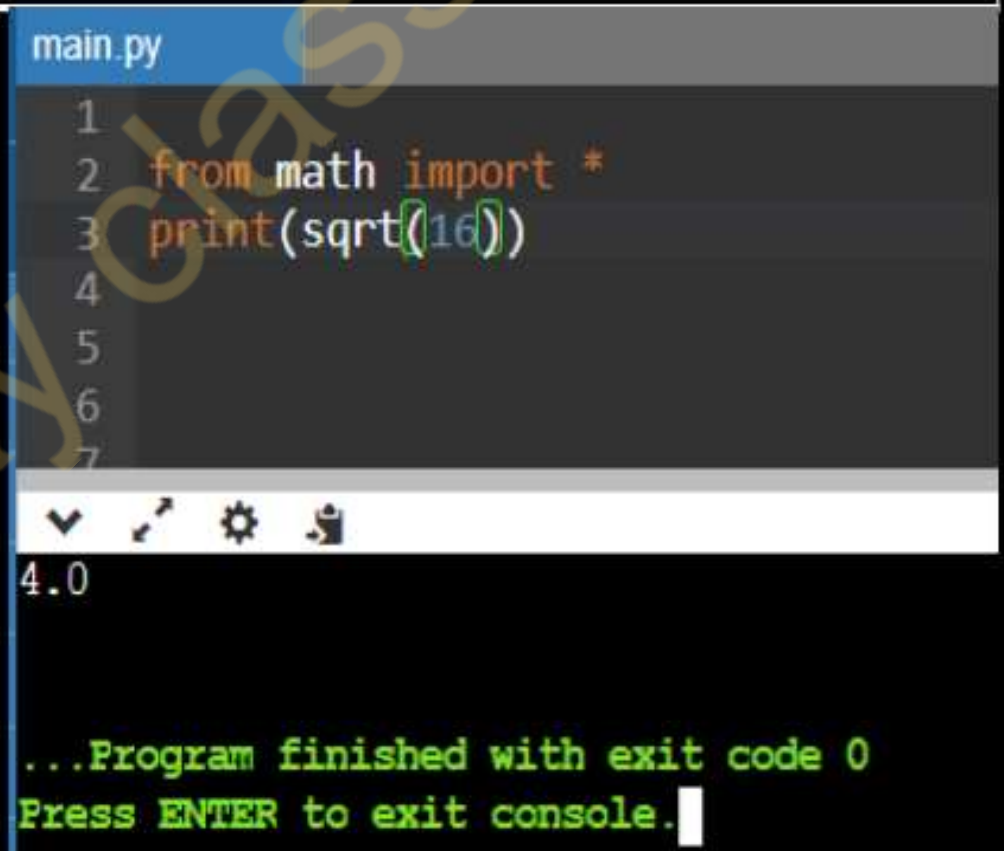
```
1 from math import sqrt as sq
2 print(sq(3))
3
```

```
1.7320508075688772
```

Difference between-

S.NO	Import library	From library import*
1	Python import loads a python module/library into its own namespace	From loads a python module into the current name space
2	By using import , we include all the codes inside the module or library	We includes all the code or only the required function , classes present inside the module or library
3	Dot operator is used	Dot operator is not required
4	It is required to mention the module name followed by dot to access any names in the module	It is not needed to mention the module name to access any names in the module

Difference between-

S.NO	Import library	From library import*
		

what is pandas and matplotlib (AKTU 2023-24 ODD0

- Matplotlib is an amazing visualization library in Python for 2D plots of arrays. Matplotlib is a multiplatform data visualization library built on NumPy arrays and designed to work with the broader SciPy stack.
- It was introduced by John Hunter in the year 2002. One of the greatest benefits of visualization is that it allows us visual access to huge amounts of data in easily digestible visuals.
- Matplotlib consists of several plots like line, bar, scatter, histogram, etc.

PANDAS

- Pandas is a powerful and versatile library that simplifies tasks of data manipulation in Python. Pandas is built on top of the NumPy library and is particularly well-suited for working with tabular data, such as spreadsheets or SQL tables.
- Its versatility and ease of use make it an essential tool for data analysts, scientists, and engineers working with structured data in Python

How can you create a python file that can be imported as library as well as run as a standalone script

- Module/ library are units that store code and data.
- It provided code reuse to python projects, and also useful in partitioning the system's namespace in self-contained packages
- They are self-contained because we can only access attributes of a module/library after importing it.
- Steps for importing python file as a library:

Step 1: create a file and name it test.py

Step 2: inside test.py create a function called display message().

For example

```
Def display_message(name):
return("welcome"+name +"hello")
```

How can you create a python file that can be imported as library as well as run as a standalone script

Step 3: now create another file display.py

Step 4: inside display .py import the module test.py file as shown below :

```
Import test
```

While importing we do not have to mention the test.py but just the name of the file

Step 5: then we can call the function display _message () from test.py inside displau.py, we need to make use of module _name, function.name

For example

```
Import test
Print(test.display_message("Aditya))
```

Step6

When we execute display.py we will get the following output

```
Welcome Aditya hello
```

Printed Pages: **Sub Code: BCC302**
Paper Id: **Roll No.**

BTech
(SEM III) THEORY EXAMINATION 2023-24
PYTHON PROGRAMMING

Time: 3 Hours ***Total Marks: 70***

Note: 1. Attempt all Sections. If require any missing data; then choose suitably.

SECTION A

1. Attempt *all* questions in brief. **2 x 7 = 14**

Q no.	Question	Marks	CO
a.	**Describe the concept of list comprehension with a suitable example.** List comprehension is a concise and elegant way of creating lists in Python. It allows you to create a new list by applying an expression to each element of an existing iterable (such as a list, tuple, or range) and filtering the elements based on certain conditions. Eg: #Creating a list of squares of numbers from 1 to 5 using list comprehension squares = [x ** 2 for x in range(1, 6)] print(squares) # Output: [1, 4, 9, 16, 25]	2	1
b.	**Differentiate between / and // operator with an example.** In Python, both the / and // operators are used for division, but they behave differently depending on the types of operands and the desired result. 1. Division Operator (/): - The / operator performs regular division, and it always returns a floating-point result. - It performs true division regardless of the types of operands. # Eg: Regular division (/) result = 10 / 3 print(result) # Output: 3.3333333333333335 2. Floor Division Operator (//): - The // operator performs floor division, and it returns the floor of the division result as an integer. - It truncates the decimal part and returns the integer quotient. # Eg: Floor division (//) result = 10 // 3 print(result) # Output: 3	2	2
c.	**Compute the output of the following python code:** **def count(s):**	2	3

	for str in string.split(): **s = "&".join(str)** **return s** **print(count("Python is fun to learn."))** Error will be generated NameError: name 'string' is not defined		
d.	**How to use the functions defined in library.py in main.py** To use functions defined in a Python file (**library.py**) within another Python file (**main.py**), we need to import the module **library** in **main.py** and then access the functions using dot notation. ```# library.py def greet(name): return f"Hello, {name}!" def add(a, b): return a + b # main.py import library # Call the greet function from library.py print(library.greet("Kumar")) # Call the add function from library.py result = library.add(10, 20) print("Result of addition:", result)```	2	4
e.	**Describe the difference between linspace and argspace.** We use **linspace** when we need a specified number of evenly spaced points over a given interval. We use **arange** when we need a sequence of numbers with a specified step size within a given range.	2	5
f.	**Explain why the program generates an error.** **x = ['12', 'hello', 456]** **x[0] *= 3** **x[1][1]='bye'** The program generates an error because of the difference in mutability between strings and lists in Python, as well as an attempt to modify a string in place. **x[0] *= 3**: This line tries to multiply the string **'12'** three times and assign the result back to **x[0]**. In Python, the ***=** operator on strings concatenates the string with itself multiple times. So, **'12' * 3** results in **'121212'**. Therefore, **x[0]** becomes **'121212'**, and there is no error here. **x[1][1] = 'bye'**: This line tries to modify the second character of the string **'hello'** to **'bye'**. However, strings in Python are immutable,	2	4

	meaning you cannot modify them in place. Therefore, this line will raise an error.		
g.	**Describe about different functions of matplotlib and pandas.** Both Matplotlib and Pandas are popular libraries for data visualization and data manipulation, respectively. Let's discuss the different functions provided by each library: **Matplotlib:** Matplotlib is a comprehensive library for creating static, interactive, and animated visualizations. It provides a wide range of functions for creating plots, charts, histograms, scatterplots, and more. **Pandas:** Pandas is a powerful library for data manipulation and analysis. It provides data structures like DataFrame and Series, along with a variety of functions for data cleaning, transformation, aggregation, and visualization.	2	5

SECTION B

2. Attempt any *three* of the following: 7 x 3 = 21

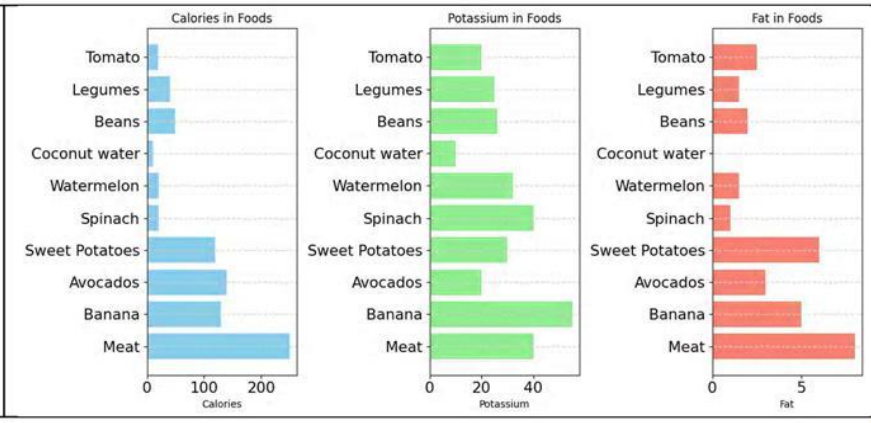
Q no.	Question	Marks	CO
a.	**Illustrate Unpacking tuples, mutable sequences, and string concatenation with examples.** Unpacking Tuples: Tuple unpacking allows you to assign the elements of a tuple to individual variables. It's a convenient way to extract values from a tuple. # Eg: Unpacking a tuple t = (1, 2, 3) a, b, c = t print(a) # Output: 1 print(b) # Output: 2 print(c) # Output: 3 # Eg: Unpacking a tuple with * operator t = (1, 2, 3, 4, 5) a, *b, c = t print(a) # Output: 1 print(b) # Output: [2, 3, 4] print(c) # Output: 5 Mutable Sequences: Mutable sequences in Python are data structures that can be modified after creation. Lists are the most common mutable sequence in Python. # Eg: Modifying a list my_list = [1, 2, 3] my_list[0] = 10 print(my_list) # Output: [10, 2, 3] # Example: Appending to a list	7	1

	my_list.append(4) print(my_list) # Output: [10, 2, 3, 4] String Concatenation: String concatenation is the process of combining two or more strings into a single string. # Eg: Concatenating strings str1 = "Hello" str2 = "World" result = str1 + " " + str2 print(result) # Output: Hello World # Eg: Using join() method for string concatenation words = ["Hello", "World"] result = " ".join(words) print(result) # Output: Hello World		
b.	**Illustrate different list slicing constructs for the following operations on the following list:** **L = [1, 2, 3, 4, 5, 6, 7, 8, 9]** **1. Return a list of numbers starting from the last to second item of the list** **2. Return a list that start from 3rd item to second last item.** **3. Return a list that has only even position elements of list L to list M.** **4. Return a list that starts from the middle of the list L.** **5. Return a list that reverses all the elements starting from element at index 0 to middle index only and return the entire list.** **6. Divide each element of the list by 2 and replace it with the remainder.** The output of the following python code: L = [1, 2, 3, 4, 5, 6, 7, 8, 9] 1. Return a list of numbers starting from the last to second item of the list slice1 = L[-1:-3:-1] print(slice1) # Output: [9, 8] 2. Return a list that starts from 3rd item to second last item. slice2 = L[2:-1] print(slice2) # Output: [3, 4, 5, 6, 7, 8] 3. Return a list that has only even position elements of list L to list M. M = L[1::2] print(M) # Output: [2, 4, 6, 8] 4. Return a list that starts from the middle of the list L. mid_index = len(L) // 2 slice3 = L[mid_index:] print(slice3) # Output: [5, 6, 7, 8, 9]	7	2

	5. Return a list that reverses all the elements starting from element at index 0 to middle index only and return the entire list. mid = len(L)//2 M = L[mid::-1]+ L[mid+1:] print(M) # Output: [5, 4, 3, 2, 1, 6, 7, 8, 9] 6. Divide each element of the list by 2 and replace it with the remainder. remainder_list = [num % 2 for num in L] print(remainder_list) # Output: [1, 0, 1, 0, 1, 0, 1, 0, 1]		
c.	**Construct a function perfect_square(number) that returns a number if it is a perfect square otherwise it returns -1.** **For example:** **perfect_square(1) returns 1** **perfect_square (2) returns -1** def perfect_square(number): sqrt_num = int(number ** 0.5) # Calculate the integer square root if sqrt_num * sqrt_num == number: # Check if the square of the integer square root equals the original number return number else: return -1 # Test cases print(perfect_square(1)) # Output: 1 print(perfect_square(2)) # Output: -1	7	3
d.	**Construct a program to change the contents of the file by separating each character by comma:** **Hello!!** **Output** **H,e,l,l,o,!,!** # Open the input file in read mode and output file in write mode with open("input.txt", "r") as input_file, open("output.txt", "w") as output_file: # Read the contents of the input file character by character for char in input_file.read(): # Write the character followed by a comma to the output file output_file.write(char + ",") # input.txt Contents of the file have been separated by comma and written to output.txt. #output.txt C,o,n,t,e,n,t,s, ,o,f, ,t,h,e, ,f,i,l,e, ,h,a,v,e, ,b,e,e,n, ,s,e,p,a,r,a,t,e,d, ,b,y, ,c,o,m,m,a, ,a,n,d, ,w,r,i,t,t,e,n, ,t,o, ,o,u,t,p,u,t,.,t,x,t,	7	4
e.	**Construct a plot for following dataset using matplotlib :**	7	5

Food	Calories	Potassium	fat
Meat	250	40	8
Banana	130	55	5
Avocados	140	20	3
Sweet Potatoes	120	30	6
Spinach	20	40	1
Watermelon	20	32	1.5
Coconut water	10	10	0
Beans	50	26	2
Legumes	40	25	1.5
Tomato	19	20	2.5

```
import matplotlib.pyplot as plt

# Define the dataset
foods = ['Meat', 'Banana', 'Avocados', 'Sweet Potatoes', 'Spinach', 'Watermelon', 'Coconut water', 'Beans', 'Legumes', 'Tomato']
calories = [250, 130, 140, 120, 20, 20, 10, 50, 40, 19]
potassium = [40, 55, 20, 30, 40, 32, 10, 26, 25, 20]
fat = [8, 5, 3, 6, 1, 1.5, 0, 2, 1.5, 2.5]

# Create subplots for each numerical column
plt.figure(figsize=(12, 6))

# Subplot for Calories
plt.subplot(1, 3, 1)
plt.barh(foods, calories, color='skyblue')
plt.xlabel('Calories')
plt.title('Calories in Foods')

# Subplot for Potassium
plt.subplot(1, 3, 2)
plt.barh(foods, potassium, color='lightgreen')
plt.xlabel('Potassium')
plt.title('Potassium in Foods')

# Subplot for Fat
plt.subplot(1, 3, 3)
plt.barh(foods, fat, color='salmon')
plt.xlabel('Fat')
plt.title('Fat in Foods')

# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```



SECTION C

3. Attempt any *one* part of the following: 7 x 1 = 7

Q no.	Question	Marks	CO
a.	**Determine a python function removenth(s,n) that takes an input a string and an integer n>=0 and removes a character at index n. If n is beyond the length of s, then whole s is returned. For example:** **removenth("MANGO",1) returns MNGO** **removenth("MANGO",3) returns MANO** def removenth(s, n): if n < 0 or n >= len(s): # Check if n is within the valid range return s # Return the original string if n is beyond the length of s else: return s[:n] + s[n+1:] # Return the string with character at index n removed # Test cases print(removenth("MANGO", 1)) # Output: MNGO print(removenth("MANGO", 3)) # Output: MANO	7	1
b.	**Construct a program that accepts a comma separated sequence of words as input and prints the words in a comma-separated sequence after sorting them alphabetically.** **Suppose the following input is supplied to the program:** **without, hello, bag, world** **Then, the output should be:** **bag, hello, without, world** def sort_words(input_str): # Split the input string into a list of words words = input_str.split(", ") # Sort the list of words alphabetically sorted_words = sorted(words) # Join the sorted words into a comma-separated string sorted_str = ", ".join(sorted_words)	7	1

```
    return sorted_str

# Test the function with example input
input_str = "without, hello, bag, world"
output_str = sort_words(input_str)
print(output_str)  # Output: bag, hello, without, world
```

4. Attempt any *one* part of the following: 7 x 1 = 7

Q no.	Question	Marks	CO
a.	**A website requires the users to input username and password to register. Construct a program to check the validity of password input by users.** **Following are the criteria for checking the password:** **1. At least 1 letter between [a-z]** **2. At least 1 number between [0-9]** **3. At least 1 letter between [A-Z]** **4. At least 1 character from [$#@]** **5. Minimum length of transaction password: 6** **6. Maximum length of transaction password: 12** **Your program should accept a sequence of comma separated passwords and will check them according to the above criteria. Passwords that match the criteria are to be printed, each separated by a comma.**	7	2

```
import re

passwords = input("Type in: ")
passwords = passwords.split(",")

accepted_pass = []
for i in passwords:

    if len(i) < 6 or len(i) > 12:
        continue

    elif not re.search("([a-z])+", i):
        continue

    elif not re.search("([A-Z])+", i):
        continue

    elif not re.search("([0-9])+", i):
        continue

    elif not re.search("([!@$%^&])+", i):
        continue

    else:
        accepted_pass.append(i)

print((" ").join(accepted_pass))
```

b.	**Explore the working of while, and for loop with examples.** Both while and for loops are control flow constructs in Python used for iteration. They help execute a block of code repeatedly until a certain condition is met (while loop) or until the elements of a sequence have been exhausted (for loop). while loop: The while loop repeatedly executes a block of code as long as a specified condition evaluates to True. Syntax: while condition: # code block Eg: # Print numbers from 1 to 5 using a while loop num = 1 while num <= 5: print(num) num += 1 for loop: The for loop iterates over the elements of a sequence (such as lists, tuples, strings, etc.) or any iterable object, executing a block of code for each element. Syntax: for element in sequence: # code block Eg: # Print each character of a string using a for loop word = "Hello" for char in word: print(char)	7	2

5. Attempt any *one* part of the following: 7 x 1 = 7

Q no.	Question	Marks	CO
a.	**Construct a function ret_smaller(l) that returns smallest list from a nested list. If two lists have same length then return the first list that is encountered. For example:** **ret_smaller([[-2, -1, 0, 0.12, 1, 2], [3, 4, 5], [6 , 7, 8, 9, 10], [11, 12, 13, 14, 15]]) returns [3,4,5]** **ret_smaller([[-2, -1, 0, 0.12, 1, 2], ['a', 'b', 'c', 'd', 3, 4, 5], [6 , 7, 8, 9, 10], [11, 12, 13, 14, 15]]) returns [6 , 7, 8, 9, 10]** def ret_smaller(l): # Initialize the smallest list variable with the first sublist smallest_list = l[0] # Iterate through each sublist in the nested list	7	3

```
    for sublist in l:
        # Compare the length of the current sublist with the length of the
smallest sublist found so far
        if len(sublist) < len(smallest_list):
            # If the current sublist is smaller, update the smallest_list
variable
            smallest_list = sublist
        # If the lengths are the same, keep the first encountered sublist
        elif len(sublist) == len(smallest_list) and l.index(sublist) <
l.index(smallest_list):
            smallest_list = sublist

    return smallest_list

# Test the function
nested_list1 = [[-2, -1, 0, 0.12, 1, 2], [3, 4, 5], [6, 7, 8, 9, 10], [11, 12,
13, 14, 15]]
result1 = ret_smaller(nested_list1)
print(result1)  # Output: [3, 4, 5]

nested_list2 = [[-2, -1, 0, 0.12, 1, 2], ['a', 'b', 'c', 'd', 3, 4, 5], [6, 7, 8, 9,
10], [11, 12, 13, 14, 15]]
result2 = ret_smaller(nested_list2)
print(result2)  # Output: [6, 7, 8, 9, 10]
```

b. **Construct following filters:**

1. **Filter all the numbers**
2. **Filter all the strings starting with a vowel**
3. **Filter all the strings that contains any of the following noun: Agra, Ramesh, Tomato, Patna.**

Create a program that implements these filters to clean the text.

7 | 3

```
# Function to filter all the numbers
def filter_numbers(text):
    return [word for word in text if word.isdigit()]

# Function to filter all the strings starting with a vowel
def filter_strings_starting_with_vowel(text):
    vowels = {'a', 'e', 'i', 'o', 'u'}
    return [word for word in text if word[0].lower() in vowels]

# Function to filter all the strings that contain any of the specified nouns
def filter_strings_containing_nouns(text):
    nouns = {'Agra', 'Ramesh', 'Tomato', 'Patna'}
    return [word for word in text if any(noun in word for noun in nouns)]

# Test text
text = ["apple", "123", "Agra", "banana", "Ramesh", "tomato", "1234",
"Patna", "orange"]

# Apply filters
filtered_numbers = filter_numbers(text)
filtered_vowel_strings = filter_strings_starting_with_vowel(text)
filtered_noun_strings = filter_strings_containing_nouns(text)
```

```
# Print results
print("Filtered numbers:", filtered_numbers)
print("Strings starting with a vowel:", filtered_vowel_strings)
print("Strings containing specified nouns:", filtered_noun_strings)
```

6. Attempt any *one* part of the following: 7 x 1 = 7

Q no.	Question	Marks	CO
a.	**Change all the numbers in the file to text. Construct a program for the same.** **Example:** **Given 2 integer numbers, return their product only if the product is equal to or lower than 10.** **And the result should be:** **Given two integer numbers, return their product only if the product is equal to or lower than one zero** `def number_to_text(number):` `  # Define a dictionary to map numbers to text` `  num_to_text = {` `  '0': 'zero', '1': 'one', '2': 'two', '3': 'three', '4': 'four',` `  '5': 'five', '6': 'six', '7': 'seven', '8': 'eight', '9': 'nine'` `  }` `  # Convert each digit of the number to text and join them` `  return ''.join(num_to_text[digit] for digit in str(number) if digit in num_to_text)` `def convert_numbers_to_text(file_path):` `  # Read the content of the file` `  with open(file_path, 'r') as file:` `  content = file.read()` `  # Replace all numbers in the content with their text equivalent` `  modified_content = ''.join(` `  number_to_text(int(c)) if c.isdigit() else c for c in content` `  )` `  # Write the modified content back to the file` `  with open(file_path, 'w') as file:` `  file.write(modified_content)` `# Test the function with a file containing numbers` `file_path = 'example.txt'  # Specify the path to your file` `convert_numbers_to_text(file_path)` `print("Numbers converted to text successfully.")`	7	4
b.	**Construct a program which accepts a sequence of words separated by whitespace as file input. Print the words composed of digits only.** `def print_words_with_digits(file_path):` `  # Read the content of the file`	7	4

```
    with open(file_path, 'r') as file:
        content = file.read()

    # Split the content into words
    words = content.split()

    # Filter words containing only digits
    words_with_digits = [word for word in words if word.isdigit()]

    # Print the words composed of digits only
    if words_with_digits:
        print("Words composed of digits only:")
        for word in words_with_digits:
            print(word)
    else:
        print("No words composed of digits found.")

# Test the function with a file containing words separated by whitespace
file_path = 'example.txt'  # Specify the path to your file
print_words_with_digits(file_path)
```

7. Attempt any *one* part of the following: 7 x 1 = 7

Q no.	Question	Marks	CO
a.	**Construct a program to read cities.csv dataset, remove last column and save it in an array. Save the last column to another array. Plot the first two columns.**	7	5
b.	**Design a calculator with the following buttons and functionalities like addition, subtraction, multiplication, division and clear.**	7	5

```
import pandas as pd
import matplotlib.pyplot as plt

# Step 1: Read the CSV file
df = pd.read_csv('cities.csv')

# Step 2: Remove the last column and save it to a separate array
last_column = df.iloc[:, -1]
df_without_last_column = df.iloc[:, :-1]

# Step 3: Plot the first two columns
plt.scatter(df_without_last_column.iloc[:,                    0],
df_without_last_column.iloc[:, 1])
plt.xlabel(df.columns[0])
plt.ylabel(df.columns[1])
plt.title("Scatter plot of the first two columns")
plt.show()
```


```
#Description: Create Calculator using tkinter
from tkinter import *
from tkmacosx import Button # not required for windows and linux
root = Tk()
root.title("Calculator")
root.geometry("240x340")
root.configure(bg="#333333")
root.resizable(0,0)
entry = None
expression = ""
#Define functions for the buttons
def press_0():
   entry.insert(END,"0")
def press_1():
   entry.insert(END,"1")
def press_2():
   entry.insert(END,"2")
def press_3():
   entry.insert(END,"3")
def press_4():
   entry.insert(END,"4")
def press_5():
   entry.insert(END,"5")
def press_6():
   entry.insert(END,"6")
def press_7():
   entry.insert(END,"7")
def press_8():
   entry.insert(END,"8")
def press_9():
   entry.insert(END,"9")

def press_plus():
   entry.insert(END,"+")

def press_minus():
```

```
    entry.insert(END,"-")

def press_multiply():
    entry.insert(END,"*")

def press_divide():
    entry.insert(END,"/")

def press_equal():
    global expression
    expression = entry.get()
    entry.delete(0,END)
    entry.insert(0,eval(expression))

def press_clear():
    entry.delete(0,END)

#Add widgets
# create a menubar
menubar = Menu(root)
root.config(menu=menubar)
# create a menu
file_menu = Menu(menubar)
# add a menu item to the menu
file_menu.add_command(
    label='Exit',
    command=root.destroy
)
# add the File menu to the menubar
menubar.add_cascade(
    label="File",
    menu=file_menu
)



#Entry box for displaying the values and results
entry = Entry(root,font=("Times New Roman",12))

#Labels from 0 to 2
label_0 = Label(root, text="0",bg="#333333",fg="white",font=("Times
New Roman",12))
label_1 = Label(root, text="1",bg="#333333",fg="white",font=("Times
New Roman",12))
label_2 = Label(root, text="2",bg="#333333",fg="white",font=("Times
New Roman",12))
#Buttons from 0 to 2, +
button_0                                                    =
Button(root,text="0",bg="#A73278",fg="white",borderless=1,font=("T
imes New Roman",12),command=press_0)
button_1                                                    =
Button(root,text="1",bg="#A73278",fg="white",borderless=1,font=("T
imes New Roman",12),command=press_1)
```

```
button_2 = Button(root,text="2",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_2)
button_3 = Button(root,text="3",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_3)
button_4 = Button(root,text="4",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_4)
button_5 = Button(root,text="5",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_5)
button_6 = Button(root,text="6",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_6)
button_7 = Button(root,text="7",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_7)
button_8 = Button(root,text="8",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_8)
button_9 = Button(root,text="9",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_9)

button_plus = Button(root,text="+",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_plus)
button_minus = Button(root,text="-",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_minus)
button_multiply = Button(root,text="*",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_multiply)
button_divide = Button(root,text="/",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_divide)

button_equal = Button(root,text="=",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_equal)
button_clear = Button(root,text="Clear",bg="#A73278",fg="white",borderless=1,font=("Times New Roman",12),command=press_clear)

#Placing widgets on the window
entry.grid(row=0,column=0,columnspan=3,sticky="news",pady=40)

button_1.grid(row=1,column=0)
button_2.grid(row=1,column=1)
button_3.grid(row=1,column=2)
button_4.grid(row=2,column=0)
button_5.grid(row=2,column=1)
button_6.grid(row=2,column=2)
```

	button_7.grid(row=3,column=0) button_8.grid(row=3,column=1) button_9.grid(row=3,column=2) button_0.grid(row=4,column=0,columnspan=3) button_plus.grid(row=5,column=0) button_minus.grid(row=5,column=1) button_multiply.grid(row=5,column=2) button_divide.grid(row=6,column=0) button_equal.grid(row=6,column=1) button_clear.grid(row=6,column=2) root.mainloop()		