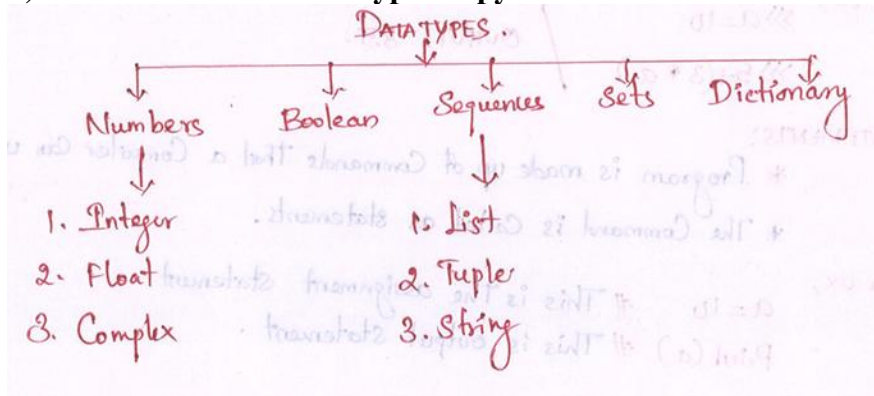


Q. No	Question (s)	Marks
1	<p>a) Write a short note on history of python? Python was created by Guido van Rossum and first released in 1991. It was designed as a successor to the ABC programming language. Python 1.0, released in January 1994, introduced features like functions, exception handling, and core data types such as strings and lists. Python 2.0 arriving in 2000 and Python 3.0 in 2008, each bringing important enhancements and improvements. latest version is 3.13 released in 07th Oct 2024 Next version (3.14) expected, 01st Oct 2025</p>	1M
	<p>b) What is the use of print () function? The <code>print()</code> function in Python is used to output text or other data to the console or standard output. It allows programmers to display messages, results, and information for debugging or interaction with the user.</p>	1M
	<p>c) What are the modes used in python execution?</p> <p>Python can be executed in two primary modes:</p> <ol style="list-style-type: none"> 1. Interactive Mode: Allows the execution of Python commands one at a time in a live interpreter session. 2. Script Mode: Executes Python code from a script file (<code>.py</code>), running all commands sequentially. 	1M
	<p>d) List the standard data types in python.</p>  <p>The diagram shows 'DATATYPES' at the top, branching into 'Numbers', 'Boolean', 'Sequences', 'Sets', and 'Dictionary'. Under 'Numbers', it lists '1. Integer', '2. Float', and '3. Complex'. Under 'Sequences', it lists '1. List', '2. Tuple', and '3. String'.</p> <p>OR</p>	1M

	<table> <tr> <th>Type</th><th>Description</th><th>Example</th></tr> <tr> <td>int</td><td>Integer values.</td><td><code>x = 5</code></td></tr> <tr> <td>float</td><td>Floating-point numbers.</td><td><code>y = 5.5</code></td></tr> <tr> <td>complex</td><td>Complex numbers.</td><td><code>z = 1 + 2j</code></td></tr> <tr> <td>str</td><td>String, sequence of characters.</td><td><code>s = "Hello, World!"</code></td></tr> <tr> <td>list</td><td>Ordered, mutable sequence.</td><td><code>lst = [1, 2, 3, 4]</code></td></tr> <tr> <td>tuple</td><td>Ordered, immutable sequence.</td><td><code>tpl = (1, 2, 3, 4)</code></td></tr> <tr> <td>dict</td><td>Unordered, mutable key-value pairs.</td><td><code>d = {"name": "Alice", "age": 30}</code></td></tr> <tr> <td>set</td><td>Unordered collection of unique elements.</td><td><code>st = {1, 2, 3, 4}</code></td></tr> <tr> <td>frozenset</td><td>Immutable set.</td><td><code>fs = frozenset([1, 2, 3, 4])</code></td></tr> <tr> <td>bool</td><td>Boolean values, <code>True</code> or <code>False</code>.</td><td><code>is_active = True</code></td></tr> <tr> <td>NoneType</td><td>Represents the absence of a value.</td><td><code>n = None</code></td></tr> </table>	Type	Description	Example	int	Integer values.	<code>x = 5</code>	float	Floating-point numbers.	<code>y = 5.5</code>	complex	Complex numbers.	<code>z = 1 + 2j</code>	str	String, sequence of characters.	<code>s = "Hello, World!"</code>	list	Ordered, mutable sequence.	<code>lst = [1, 2, 3, 4]</code>	tuple	Ordered, immutable sequence.	<code>tpl = (1, 2, 3, 4)</code>	dict	Unordered, mutable key-value pairs.	<code>d = {"name": "Alice", "age": 30}</code>	set	Unordered collection of unique elements.	<code>st = {1, 2, 3, 4}</code>	frozenset	Immutable set.	<code>fs = frozenset([1, 2, 3, 4])</code>	bool	Boolean values, <code>True</code> or <code>False</code> .	<code>is_active = True</code>	NoneType	Represents the absence of a value.	<code>n = None</code>	
Type	Description	Example																																				
int	Integer values.	<code>x = 5</code>																																				
float	Floating-point numbers.	<code>y = 5.5</code>																																				
complex	Complex numbers.	<code>z = 1 + 2j</code>																																				
str	String, sequence of characters.	<code>s = "Hello, World!"</code>																																				
list	Ordered, mutable sequence.	<code>lst = [1, 2, 3, 4]</code>																																				
tuple	Ordered, immutable sequence.	<code>tpl = (1, 2, 3, 4)</code>																																				
dict	Unordered, mutable key-value pairs.	<code>d = {"name": "Alice", "age": 30}</code>																																				
set	Unordered collection of unique elements.	<code>st = {1, 2, 3, 4}</code>																																				
frozenset	Immutable set.	<code>fs = frozenset([1, 2, 3, 4])</code>																																				
bool	Boolean values, <code>True</code> or <code>False</code> .	<code>is_active = True</code>																																				
NoneType	Represents the absence of a value.	<code>n = None</code>																																				
	<p>e) Which method is used to convert number to binary number? The <code>bin()</code> function is used to convert a number to its binary representation in Python. Ex: <code>binary_number = bin(10)</code> # Output will be '0b1010'</p>	1M																																				
	<p>f) What is the purpose of break statement in python? The break statement in Python is used to exit a loop prematurely. It allows the program to terminate the loop's iteration and proceed with the next line of code outside the loop. This is useful when a specific condition is met, and further iterations are unnecessary.</p> <table> <tr> <th>Feature</th><th><code>continue</code></th><th><code>break</code></th></tr> <tr> <td>Functionality</td><td>Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.</td><td>Exits the loop entirely, regardless of the iteration count or condition.</td></tr> <tr> <td>Usage</td><td>Used when you want to skip certain iterations based on a condition but continue looping.</td><td>Used when you want to terminate the loop completely based on a condition.</td></tr> <tr> <td>Loop Type</td><td>Can be used in <code>for</code> and <code>while</code> loops.</td><td>Can be used in <code>for</code> and <code>while</code> loops.</td></tr> <tr> <td>Example</td><td><code>python for i in range(5): if i == 3: continue print(i) #</code> Output: 0, 1, 2, 4, 5</td><td><code>python for i in range(5): if i == 3: break print(i) #</code> Output: 0, 1, 2</td></tr> </table>	Feature	<code>continue</code>	<code>break</code>	Functionality	Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.	Exits the loop entirely, regardless of the iteration count or condition.	Usage	Used when you want to skip certain iterations based on a condition but continue looping.	Used when you want to terminate the loop completely based on a condition.	Loop Type	Can be used in <code>for</code> and <code>while</code> loops.	Can be used in <code>for</code> and <code>while</code> loops.	Example	<code>python for i in range(5): if i == 3: continue print(i) #</code> Output: 0, 1, 2, 4, 5	<code>python for i in range(5): if i == 3: break print(i) #</code> Output: 0, 1, 2	1M																					
Feature	<code>continue</code>	<code>break</code>																																				
Functionality	Skips the rest of the code inside the loop for the current iteration and moves to the next iteration.	Exits the loop entirely, regardless of the iteration count or condition.																																				
Usage	Used when you want to skip certain iterations based on a condition but continue looping.	Used when you want to terminate the loop completely based on a condition.																																				
Loop Type	Can be used in <code>for</code> and <code>while</code> loops.	Can be used in <code>for</code> and <code>while</code> loops.																																				
Example	<code>python for i in range(5): if i == 3: continue print(i) #</code> Output: 0, 1, 2, 4, 5	<code>python for i in range(5): if i == 3: break print(i) #</code> Output: 0, 1, 2																																				
	<p>g) What are the identity operators in python? The identity operators in Python are <code>is</code> and <code>is not</code>. They are used to compare the memory locations of two objects.</p> <ul style="list-style-type: none"> <code>is</code>: Evaluates to <code>True</code> if both variables point to the same object. <code>is not</code>: Evaluates to <code>True</code> if the variables point to different objects. 	1M																																				

	<p>Ex:</p> <pre>a = [1, 2, 3] b = a c = [1, 2, 3]</pre> <pre>print(a is b) # Output: True print(a is c) # Output: False print(a is not c) # Output: True</pre>	
	<p>h) List out any four string methods.</p> <p><code>upper()</code>: Converts all characters in a string to uppercase</p> <p>Ex: <code>text = "hello"</code></p> <pre>print(text.upper()) # Output: "HELLO"</pre> <p><code>lower()</code>: Converts all characters in a string to lowercase</p> <p><code>text = "HELLO"</code></p> <pre>print(text.lower()) # Output: "hello"</pre> <p><code>strip()</code>: Removes leading and trailing whitespace from a string.</p> <p><code>text = " hello "</code></p> <pre>print(text.strip()) # Output: "hello"</pre> <p><code>replace(old, new)</code>: Replaces occurrences of a substring with another substring.</p> <p><code>text = "hello world"</code></p> <pre>print(text.replace("world", "Python")) # Output: "hello Python"</pre>	1M
	<p>i) Define recursion in Python.</p> <p>Recursion in Python is a programming technique where a function calls itself to solve smaller instances of the same problem until a base condition is met.</p> <p>Ex:</p> <pre>def factorial(n): if n == 0: return 1 else: return n * factorial(n - 1)</pre>	1M
	<p>j) State the purpose of math module in Python</p> <p>The math module in Python provides a wide range of mathematical functions and constants, such as trigonometric, logarithmic, and exponential functions, which are essential for performing advanced mathematical operations.</p>	1M

PART – B (20M)

Q. No	Question (s)	Marks
2	<p>a) What are the basic list operations that can be performed in Python? Explain each operation with its syntax and example.</p> <p>Python offers a rich set of list operations that allow for versatile manipulation of list data structures. Here is a detailed explanation of some commonly used list operations:</p> <p>1. Appending Elements</p>	4M

	<p><code>append(element)</code>: Adds an element to the end of the list. Ex: <pre>my_list = [1, 2, 3] my_list.append(4) print(my_list) # Output: [1, 2, 3, 4]</pre></p> <p>2. Extending Lists <code>extend(iterable)</code>: Extends the list by appending elements from the iterable. Ex: <pre>my_list = [1, 2, 3] my_list.extend([4, 5]) print(my_list) # Output: [1, 2, 3, 4, 5]</pre></p> <p>3. Inserting Elements <code>insert(index, element)</code>: Inserts an element at the specified position. Ex: <pre>my_list = [1, 2, 3] my_list.insert(1, 1.5) print(my_list) # Output: [1, 1.5, 2, 3]</pre></p> <p>4. Removing Elements <code>remove(element)</code>: Removes the first occurrence of the element. Ex: <pre>my_list = [1, 2, 3, 2] my_list.remove(2) print(my_list) # Output: [1, 3, 2]</pre></p> <p>5. Popping Elements <code>pop(index)</code>: Removes and returns the element at the specified position. Ex: <pre>my_list = [1, 2, 3] element = my_list.pop(1) print(element) # Output: 2 print(my_list) # Output: [1, 3]</pre></p> <p>6. Clearing Lists <code>clear()</code>: Removes all elements from the list. Ex: <pre>my_list = [1, 2, 3] my_list.clear() print(my_list) # Output: []</pre></p>	
--	---	--

7. Indexing Elements

`index(element)`: Returns the index of the first occurrence of the element.

Ex:

```
my_list = [1, 2, 3]
index = my_list.index(2)
print(index) # Output: 1
```

8. Counting Elements

`count(element)`: Returns the number of occurrences of the element.

Ex:

```
my_list = [1, 2, 3, 2]
count = my_list.count(2)
print(count) # Output: 2
```

9. Sorting Lists

`sort()`: Sorts the list in ascending order.

```
my_list = [3, 1, 2]
my_list.sort()
print(my_list) # Output: [1, 2, 3]
```

10. Reversing Lists

`reverse()`: Reverses the order of the list.

```
my_list = [1, 2, 3]
my_list.reverse()
print(my_list) # Output: [3, 2, 1]
```

11. Copying Lists

`copy()`: Returns a shallow copy of the list.

```
my_list = [1, 2, 3]
new_list = my_list.copy()
print(new_list) # Output: [1, 2, 3]
```

These operations cover a wide range of functionalities for working with lists in Python, making it easier to manage and manipulate data.

b) What are the basic Tuple operations (at least 5) that can be performed in Python? Explain each operation with its syntax and example.

Tuples are immutable sequences in Python, meaning their elements cannot be changed after creation. Here are five basic operations that can be performed on tuples:

1. Indexing:

- Access elements in a tuple using their index.
- **Syntax:**

```
my_tuple = (10, 20, 30, 40)
first_element = my_tuple[0] # Output: 10
```

- **Example:**

```
my_tuple = (10, 20, 30, 40)
print(my_tuple[1]) # Output: 20
```

2. Slicing:

- Access a range of elements in a tuple using slicing.
- **Syntax:**

```
my_tuple = (10, 20, 30, 40, 50)
sub_tuple = my_tuple[1:4] # Output: (20, 30, 40)
```

- **Example:**

```
my_tuple = (10, 20, 30, 40, 50)
print(my_tuple[2:]) # Output: (30, 40, 50)
```

3. Concatenation:

- Combine two or more tuples to form a new tuple.
- **Syntax:**

```
tuple1 = (10, 20)
tuple2 = (30, 40)
combined_tuple = tuple1 + tuple2 #
Output: (10, 20, 30, 40)
```

- **Example:**

```
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
print(tuple1 + tuple2) # Output: (1, 2, 3, 4, 5, 6)
```

4. Repetition:

4M

	<ul style="list-style-type: none">○ Repeat elements of a tuple a specified number of times.○ Syntax: <pre>my_tuple = (10, 20) repeated_tuple = my_tuple * 3 # Output: (10, 20, 10, 20, 10, 20)</pre>○ Example: <pre>my_tuple = ('a', 'b') print(my_tuple * 2) # Output: ('a', 'b', 'a', 'b')</pre> <p>5. Length:</p> <ul style="list-style-type: none">○ Determine the number of elements in a tuple using the <code>len()</code> function.○ Syntax: <pre>my_tuple = (10, 20, 30) length = len(my_tuple) # Output: 3</pre>○ Example: <pre>my_tuple = (1, 2, 3, 4, 5) print(len(my_tuple)) # Output: 5</pre> <p>These basic operations provide fundamental ways to interact with tuples, enabling efficient data access and manipulation.</p>																					
3	<p>a) Write are the features of python?</p> <table><tr><th>Feature</th><th>Description</th></tr><tr><td>Simple and Readable Syntax</td><td>Python's syntax is clean and easy to understand, making it suitable for beginners.</td></tr><tr><td>Interpreted Language</td><td>Python code is executed line by line, simplifying debugging and testing.</td></tr><tr><td>Dynamically Typed</td><td>Variables do not require explicit declaration of data types.</td></tr><tr><td>Extensive Standard Library</td><td>A rich standard library with modules for various tasks.</td></tr><tr><td>Object-Oriented</td><td>Supports object-oriented programming with classes and objects.</td></tr><tr><td>Cross-Platform</td><td>Runs on various platforms such as Windows, macOS, and Linux.</td></tr><tr><td>Extensible and Embeddable</td><td>Can be extended with modules written in C or C++, and embedded into applications.</td></tr><tr><td>Large Community and Ecosystem</td><td>A vast and active community, providing extensive support and a wealth of third-party libraries and frameworks.</td></tr><tr><td>Versatile</td><td>Used in web development, data analysis, AI, scientific computing, automation, and more.</td></tr></table>	Feature	Description	Simple and Readable Syntax	Python's syntax is clean and easy to understand, making it suitable for beginners.	Interpreted Language	Python code is executed line by line, simplifying debugging and testing.	Dynamically Typed	Variables do not require explicit declaration of data types.	Extensive Standard Library	A rich standard library with modules for various tasks.	Object-Oriented	Supports object-oriented programming with classes and objects.	Cross-Platform	Runs on various platforms such as Windows, macOS, and Linux.	Extensible and Embeddable	Can be extended with modules written in C or C++, and embedded into applications.	Large Community and Ecosystem	A vast and active community, providing extensive support and a wealth of third-party libraries and frameworks.	Versatile	Used in web development, data analysis, AI, scientific computing, automation, and more.	4M
Feature	Description																					
Simple and Readable Syntax	Python's syntax is clean and easy to understand, making it suitable for beginners.																					
Interpreted Language	Python code is executed line by line, simplifying debugging and testing.																					
Dynamically Typed	Variables do not require explicit declaration of data types.																					
Extensive Standard Library	A rich standard library with modules for various tasks.																					
Object-Oriented	Supports object-oriented programming with classes and objects.																					
Cross-Platform	Runs on various platforms such as Windows, macOS, and Linux.																					
Extensible and Embeddable	Can be extended with modules written in C or C++, and embedded into applications.																					
Large Community and Ecosystem	A vast and active community, providing extensive support and a wealth of third-party libraries and frameworks.																					
Versatile	Used in web development, data analysis, AI, scientific computing, automation, and more.																					
	<p>b) What are all the applications of Python?</p>	4M																				

	<p>Python is a versatile programming language that is used in a wide range of applications. Here are some key areas where Python is commonly applied:</p> <ol style="list-style-type: none"> 1. Web Development: Frameworks like Django and Flask make it easy to build robust and scalable web applications. 2. Data Analysis and Visualization: Libraries such as Pandas, NumPy, and Matplotlib are widely used for data manipulation, analysis, and visualization. 3. Artificial Intelligence and Machine Learning: Python is the preferred language for AI and ML due to libraries like TensorFlow, Keras, and scikit-learn. 4. Scientific Computing: Python is used in scientific research and engineering with tools like SciPy and SymPy. 5. Automation and Scripting: Python is commonly used to automate repetitive tasks and write scripts for various purposes. 6. Game Development: Libraries such as Pygame allow for the development of simple 2D games. 7. Desktop GUI Applications: Frameworks like Tkinter, PyQt, and Kivy enable the creation of desktop applications. 8. Network Programming: Python provides modules such as socket and Twisted for network-related tasks. 9. Embedded Systems: MicroPython and CircuitPython are used to run Python on microcontrollers for embedded systems. 10. Web Scraping: Libraries like BeautifulSoup and Scrapy are used to extract data from websites. 11. Education: Python's simple syntax and readability make it an ideal language for teaching programming. <p>These applications highlight Python's versatility and its wide range of use cases.</p>	
4	<p>a) Write a python program to accept three numbers, find the greatest and print the result.</p> <pre> # Accept three numbers from the user num1 = float(input("Enter the first number: ")) num2 = float(input("Enter the second number: ")) num3 = float(input("Enter the third number: ")) # Find the greatest number if num1 >= num2 and num1 >= num3: greatest = num1 elif num2 >= num1 and num2 >= num3: greatest = num2 else: greatest = num3 # Print the result print(f"The greatest number is: {greatest}") </pre>	4M

	<p>b) Write nested loop in detail with example?</p> <pre># Outer loop for rows for i in range(3): # Inner loop for columns for j in range(3): print(f"({i}, {j})", end=" ") # Newline after each row print()</pre> <p>Output: (0, 0) (0, 1) (0, 2) (1, 0) (1, 1) (1, 2) (2, 0) (2, 1) (2, 2)</p>	4M
5	<p>a) Explain any 2 string manipulation methods with suitable example.</p> <p>1. split() Method</p> <p>The <code>split()</code> method is used to divide a string into a list of substrings based on a specified delimiter. By default, it splits the string at spaces.</p> <p>Ex: <code>text = "Hello World from Python"</code> <code>words = text.split()</code></p> <p>2. join() Method</p> <p>The <code>join()</code> method is used to concatenate the elements of a list or any iterable into a single string, with a specified delimiter separating each element.</p> <p><code>words = ['Hello', 'World', 'from', 'Python']</code> <code>text = " ".join(words)</code> <code>print(text)</code> # Output: "Hello World from Python" <code>print(words)</code> # Output: ['Hello', 'World', 'from', 'Python']</p>	4M
	<p>b) Write logical and relational operators in detail with an example program?</p> <p>Logical and Relational Operators in Python</p> <p>Logical Operators</p> <ul style="list-style-type: none"> • <code>and</code>: Returns <code>True</code> if both statements are true. • <code>or</code>: Returns <code>True</code> if at least one statement is true. • <code>not</code>: Reverses the result, returning <code>False</code> if the result is true. <p>Relational Operators</p> <ul style="list-style-type: none"> • <code>==</code>: Checks if two values are equal. • <code>!=</code>: Checks if two values are not equal. 	4M

	<ul style="list-style-type: none"> • >: Checks if one value is greater than another. • <: Checks if one value is less than another. • >=: Checks if one value is greater than or equal to another. • <=: Checks if one value is less than or equal to another. <p>Example Program:</p> <pre> a = 10 b = 5 c = 15 # Relational Operators print(a == b) # Output: False print(a != b) # Output: True print(a > b) # Output: True print(a < b) # Output: False print(a >= b) # Output: True print(a <= b) # Output: False # Logical Operators print((a > b) and (a < c)) # Output: True print((a > b) or (a < c)) # Output: True print(not(a == b)) # Output: True </pre>	
6	<p>Write a python to check whether the given number is PAN format or not?</p> <p>The Permanent Account Number (PAN) in India is a unique 10-character alphanumeric code. The format of a PAN number is as follows:</p> <ol style="list-style-type: none"> 1. The first five characters are letters. 2. The next four characters are digits. 3. The last character is a letter. <p>Here's a Python program to check if a given number follows the PAN format:</p> <pre> def is_pan_format(pan): if len(pan) != 10: return False if not (pan[:5].isalpha() and pan[:5].isupper()): return False if not (pan[5:9].isdigit()): return False if not (pan[9].isalpha() and pan[9].isupper()): return False return True # Input: PAN number pan_number = input("Enter PAN number: ").upper() # Convert to uppercase for consistency </pre>	4M

	<pre># Check PAN format if is_pan_format(pan_number): print("The given number is in PAN format.") else: print("The given number is not in PAN format.")</pre>	
7	<p>Define function in Python, mention the types of function and state its uses.</p> <p>Functions in Python</p> <p>A function in Python is a reusable block of code designed to perform a specific task. Functions help organize code into modular sections, making it easier to read, maintain, and debug.</p> <p>Types of Functions</p> <ol style="list-style-type: none"> 1. Built-in Functions: These are predefined functions available in Python, such as <code>print()</code>, <code>len()</code>, <code>sum()</code>, etc. 2. User-Defined Functions: These functions are created by the user to perform specific tasks and are defined using the <code>def</code> keyword. 3. Lambda Functions: These are small anonymous functions defined using the <code>lambda</code> keyword. They can have any number of arguments but only one expression. Ex: <code>add = lambda x, y: x + y</code> <code>print(add(3, 5))</code> 4. Recursive Functions: These functions call themselves to solve a smaller instance of the same problem. They must have a base condition to terminate the recursion. 	4M

SET-II

Q. No	Question (s)	Marks
1	<p>a) What is list with syntax?</p> <p>A list in Python is a collection of items that can be of different types, ordered, and changeable. Lists are created by placing items inside square brackets <code>[]</code>, separated by commas.</p>	1M
	<p>b) What operators does python support?</p> <p>Python supports several types of operators, including:</p> <ol style="list-style-type: none"> Arithmetic Operators: <code>+</code>, <code>-</code>, <code>*</code>, <code>/</code>, <code>%</code>, <code>**</code>, <code>//</code> Comparison Operators: <code>==</code>, <code>!=</code>, <code>></code>, <code><</code>, <code>>=</code>, <code><=</code> Logical Operators: <code>and</code>, <code>or</code>, <code>not</code> Bitwise Operators: <code>&</code>, <code> </code>, <code>^</code>, <code>~</code>, <code><<</code>, <code>>></code> Assignment Operators: <code>=</code>, <code>+=</code>, <code>-=</code>, <code>*=</code>, <code>/=</code>, <code>%=</code>, <code>**=</code>, <code>//=</code> 	1M

	<p>6. Identity Operators: <code>is</code>, <code>is not</code></p> <p>7. Membership Operators: <code>in</code>, <code>not in</code></p> <p>These operators allow you to perform various operations on data within your Python programs.</p>																			
	<p>c) What is meant by dictionary with syntax?</p> <p>A dictionary in Python is a collection of key-value pairs, where each key is unique and associated with a value. Dictionaries are unordered and mutable, allowing for efficient data retrieval and manipulation.</p> <p>Ex:</p> <pre>student = {"name": "John", "age": 21, "major": "Computer Science"}</pre>	1M																		
	<p>d) Differentiate List and Tuple with an example.</p> <table border="1"> <thead> <tr> <th>Feature</th><th>List</th><th>Tuple</th></tr> </thead> <tbody> <tr> <td>Definition</td><td>Ordered, mutable collection of items.</td><td>Ordered, immutable collection of items.</td></tr> <tr> <td>Syntax</td><td><code>list_name = [item1, item2, item3, ...]</code></td><td><code>tuple_name = (item1, item2, item3, ...)</code></td></tr> <tr> <td>Mutability</td><td>Mutable (can be changed).</td><td>Immutable (cannot be changed).</td></tr> <tr> <td>Performance</td><td>Slower due to extra memory operations.</td><td>Faster due to immutability.</td></tr> <tr> <td>Use Case</td><td>Suitable for collections that need to change.</td><td>Suitable for collections that should remain constant.</td></tr> </tbody> </table>	Feature	List	Tuple	Definition	Ordered, mutable collection of items.	Ordered, immutable collection of items.	Syntax	<code>list_name = [item1, item2, item3, ...]</code>	<code>tuple_name = (item1, item2, item3, ...)</code>	Mutability	Mutable (can be changed).	Immutable (cannot be changed).	Performance	Slower due to extra memory operations.	Faster due to immutability.	Use Case	Suitable for collections that need to change.	Suitable for collections that should remain constant.	1M
Feature	List	Tuple																		
Definition	Ordered, mutable collection of items.	Ordered, immutable collection of items.																		
Syntax	<code>list_name = [item1, item2, item3, ...]</code>	<code>tuple_name = (item1, item2, item3, ...)</code>																		
Mutability	Mutable (can be changed).	Immutable (cannot be changed).																		
Performance	Slower due to extra memory operations.	Faster due to immutability.																		
Use Case	Suitable for collections that need to change.	Suitable for collections that should remain constant.																		
	<p>e) Write the syntax of while loop?</p> <pre>count = 0 while count < 5: print(count) count += 1</pre>	1M																		
	<p>f) What is the purpose of continue statement in python?</p> <p>The <code>continue</code> statement in Python is used to skip the rest of the current loop iteration and proceed directly to the next iteration of the loop.</p> <pre>for num in range(1, 11): # Loop through numbers from 1 to 10 if num % 2 == 0: # Check if the number is even continue # Skip the current iteration if the number is even print(num) # Print the number if it is odd</pre>	1M																		
	<p>g) What is short-circuit (lazy) evaluation?</p> <p>Short-circuit (lazy) evaluation is a technique where the evaluation of logical expressions is stopped as soon as the result is determined.</p> <pre>x = 0 y = 1 result = (x != 0) and (y / x > 1) # This won't raise a ZeroDivisionError</pre>	1M																		
	<p>h) Write the syntax of nested if statement in python?</p>	1M																		

	<pre> age = 18 if age >= 18: print("Adult") if age >= 65: print("Senior Citizen") else: print("Not a Senior Citizen") else: print("Minor") </pre>	
	i) Write down the syntax of function declaration in python. <pre> def greet(name): return f"Hello, {name}!" </pre>	1M
	j) Write any two benefits of using functions in Python. <ol style="list-style-type: none"> 1. Code Reusability: Functions allow you to write a piece of code once and reuse it multiple times, reducing redundancy and making your code more efficient. 2. Improved Readability: Functions help break down complex problems into smaller, manageable pieces, making your code easier to read, understand, and maintain. 	1M

JPART – B (20M)

Q. No	Question (s)	Marks
2	<p>a) Explain in detail about tuple in python with different methods and example.</p> <p>A tuple is an ordered and immutable collection of elements. Tuples are similar to lists, but unlike lists, they cannot be modified after creation. Tuples are useful for grouping related data and ensuring that the data remains unchanged.</p> <p>Ex: <pre> my_tuple = (1, 2, 3, "apple", "banana") print(my_tuple) # Output: (1, 2, 3, 'apple', 'banana') </pre> </p> <p>Tuple Methods:</p> <ol style="list-style-type: none"> 1. count(): <ul style="list-style-type: none"> ○ Returns the number of occurrences of a specified value. ○ Example: <code>my_tuple.count(2)</code> returns 1. 2. index(): <ul style="list-style-type: none"> ○ Returns the index of the first occurrence of a specified value. ○ Example: <code>my_tuple.index('banana')</code> returns 4. <p>Common Methods and Operations with Tuples:</p>	4M

	<ol style="list-style-type: none"> 1. Accessing Elements: <ul style="list-style-type: none"> ○ Use indexing to access elements. ○ Example: <code>my_tuple[1]</code> returns 2. 2. Slicing: <ul style="list-style-type: none"> ○ Extract a part of the tuple. ○ Example: <code>my_tuple[1:4]</code> returns (2, 3, 'apple'). 3. Concatenation: <ul style="list-style-type: none"> ○ Combine two tuples. ○ Example: <code>(1, 2) + (3, 4)</code> returns (1, 2, 3, 4). 4. Repetition: <ul style="list-style-type: none"> ○ Repeat the elements of a tuple. ○ Example: <code>("apple", "banana") * 2</code> returns ('apple', 'banana', 'apple', 'banana'). 5. Length: <ul style="list-style-type: none"> ○ Use <code>len()</code> to get the number of elements. ○ Example: <code>len(my_tuple)</code> returns 5. 6. Membership: <ul style="list-style-type: none"> ○ Check if an element exists within a tuple. ○ Example: <code>'apple' in my_tuple</code> returns True. 	
	<p>b) State the type conversion in python with an example.</p> <p>Type conversion refers to converting one data type to another. This can be done implicitly by the Python interpreter or explicitly by the programmer using built-in functions. There are two types of type conversion in Python:</p> <ol style="list-style-type: none"> 1. Implicit Type Conversion (Coercion): <ul style="list-style-type: none"> ○ Python automatically converts one data type to another without any user intervention. ○ Typically occurs when mixing data types in expressions. <p>Example:</p> <pre>num_int = 10 num_float = 10.5 result = num_int + num_float print(type(result)) # Output: <class 'float'></pre> <p>Explicit Type Conversion (Type Casting):</p> <ul style="list-style-type: none"> • Programmers manually convert one data type to another using built-in functions such as <code>int()</code>, <code>float()</code>, <code>str()</code>, 	4M

	<p>etc.</p> <p>Example:</p> <pre># Converting string to integer num_str = "100" num_int = int(num_str) print(type(num_int)) # Output: <class 'int'> # Converting float to integer float_val = 10.7 int_val = int(float_val) print(int_val) # Output: 10 # Converting integer to string num = 123 num_str = str(num) print(type(num_str)) # Output: <class 'str'> # Converting string to integer num_str = "100" num_int = int(num_str) print(type(num_int)) # Output: <class 'int'> # Converting float to integer float_val = 10.7 int_val = int(float_val) print(int_val) # Output: 10 # Converting integer to string num = 123 num_str = str(num) print(type(num_str)) # Output: <class 'str'></pre>	
3	<p>What is Dictionary? Explain Python dictionaries in detail discussing its operations and methods.</p> <p>A dictionary in Python is an unordered collection of key-value pairs. Each key is unique and maps to a value. Dictionaries are mutable, meaning they can be modified after their creation.</p> <p>Ex:</p> <pre>person = { "name": "Alice", "age": 30, "city": "New York" } print(person) # Output: {'name': 'Alice', 'age': 30, 'city': 'New York'}</pre>	<p>8M</p> <p>4M</p>

Dictionary Operations:

1. **Accessing Values:**
 - Use the key to access its corresponding value.
 - Example: `person["name"]` returns "Alice".
2. **Adding/Updating Key-Value Pairs:**
 - Add a new key-value pair or update an existing one.
 - Example: `person["email"] = "alice@example.com".`
3. **Removing Key-Value Pairs:**
 - Use `del` statement or `pop()` method.
 - Example: `del person["age"]` or `person.pop("age").`
4. **Checking for Key Existence:**
 - Use the `in` keyword.
 - Example: `"name" in person` returns `True`.
5. **Getting All Keys, Values, and Items:**
 - Use `keys()`, `values()`, and `items()` methods.
 - Example: `person.keys()`, `person.values()`, `person.items()`.

Dictionary Methods:

1. `get(key, default):`
 - Returns the value for the specified key. If the key is not found, it returns the default value.
 - Example: `person.get("name", "Unknown")` returns "Alice".
2. `update(other_dict):`
 - Updates the dictionary with key-value pairs from another dictionary.
 - Example: `person.update({"age": 31, "city": "San Francisco"})`.
3. `pop(key, default):`
 - Removes the specified key and returns the corresponding value. If the key is not found, it returns the default value.
 - Example: `person.pop("age", "Not Found")` returns 31.
4. `clear():`
 - Removes all key-value pairs from the dictionary.
 - Example: `person.clear()`.
5. `copy():`
 - Returns a shallow copy of the dictionary.
 - Example: `person_copy = person.copy()`.

b) Differentiate List and Tuple with an example

Feature	List	Tuple
Definition	An ordered collection of items.	An ordered collection of items.
Syntax	Defined using square brackets <code>[]</code> .	Defined using parentheses <code>()</code> .
Mutability	Mutable: Can be modified (items can be added, removed, or changed).	Immutable: Cannot be modified after creation.
Methods	Supports various methods like <code>append()</code> , <code>remove()</code> , <code>sort()</code> , etc.	Supports limited methods like <code>count()</code> , <code>index()</code> .
Usage	Used when you need a collection that can change over time.	Used when you need a collection of items that should remain constant.

List (Ex):

```
fruits = ['apple', 'banana', 'cherry']
fruits.append('date')
print(fruits) # Output: ['apple', 'banana', 'cherry', 'date']
```

Tuple (Ex):

```
vegetables = ('carrot', 'potato', 'spinach')
# Trying to modify will raise an error
# vegetables.append('onion') # This will raise an
# AttributeError
print(vegetables) # Output: ('carrot', 'potato', 'spinach')
```

a) What are the different loop control statements available in Python? Explain with suitable example program.

Loop control statements in Python allow you to control the execution flow of loops. The primary loop control statements are `break`, `continue`, and `pass`.

1. break Statement:

- The `break` statement is used to terminate the loop immediately.
- When encountered, the loop stops executing, and the control exits the loop.

Example:

```
for i in range(1, 6):
    if i == 3:
        break
    print(i)
# Output: 1 2
```

continue Statement:

4

8M
4M

	<ul style="list-style-type: none"> The <code>continue</code> statement is used to skip the rest of the code inside the loop for the current iteration and move to the next iteration. <p>Example:</p> <pre>for i in range(1, 6): if i == 3: continue print(i) # Output: 1 2 4 5</pre> <p>pass Statement:</p> <ul style="list-style-type: none"> The <code>pass</code> statement is a null operation; it does nothing. It is used as a placeholder in loops or functions where code is syntactically required but you have not yet written it. <p>Example:</p> <pre>for i in range(1, 6): if i == 3: pass # Placeholder for future code print(i) # Output: 1 2 3 4 5</pre>	
	<p>b) Explain in detail about decision structures with a suitable example.</p> <p>Decision Structures in Python</p> <p>Decision structures, also known as control structures, allow a program to execute certain blocks of code based on specific conditions. They are fundamental to any programming language and enable the creation of dynamic, responsive programs that can handle different scenarios.</p> <p>Types of Decision Structures:</p> <ol style="list-style-type: none"> if Statement: <ul style="list-style-type: none"> Executes a block of code if a specified condition is true. if-else Statement: <ul style="list-style-type: none"> Executes one block of code if the condition is true, and another block of code if the condition is false. if-elif-else Statement: <ul style="list-style-type: none"> Executes a block of code based on multiple conditions, where only the first true condition's 	

	<p>block is executed.</p> <p>4. Nested if Statements:</p> <ul style="list-style-type: none"> o An <code>if</code> statement inside another <code>if</code> statement, allowing for more complex decision-making. <p>Example: if-elif-else Statement</p> <p># Input: Student's score <code>score = 85</code></p> <p># Determine grade based on score using if-elif-else structure <code>if score >= 90:</code> <code> grade = 'A'</code> <code>elif score >= 80:</code> <code> grade = 'B'</code> <code>elif score >= 70:</code> <code> grade = 'C'</code> <code>elif score >= 60:</code> <code> grade = 'D'</code> <code>else:</code> <code> grade = 'F'</code></p> <p># Output: Display the grade <code>print("The student's grade is:", grade)</code></p>	
5	<p>a) What is the use of break statement? Illustrate with an example program.</p> <p>The <code>break</code> statement is used to terminate a loop immediately when a specific condition is met. When the <code>break</code> statement is encountered within a loop (whether it's a <code>for</code> or <code>while</code> loop), the loop stops executing, and the control exits the loop.</p> <p>Example Program</p> <p>Let's consider a scenario where we want to find the first even number in a list of numbers and stop processing further once we find it:</p> <pre>numbers = [1, 3, 5, 6, 7, 9, 10] for num in numbers: if num % 2 == 0: # Check if the number is even print(f"First even number found: {num}") break # Terminate the loop print(f"Checked number: {num}")</pre> <p>Output: Checked number: 1</p>	4M

	Checked number: 3 Checked number: 5 First even number found: 6	
	<p>b) Write short note on bitwise operator with an example.</p> <ul style="list-style-type: none"> • Bitwise AND (&): <ul style="list-style-type: none"> • Compares each bit of two integers. The result is a new integer where a bit is set to 1 if the corresponding bits of both operands are 1; otherwise, it is set to 0. • Example: $a \ \& \ b$ • Bitwise OR (): <ul style="list-style-type: none"> • Compares each bit of two integers. The result is a new integer where a bit is set to 1 if at least one of the corresponding bits of either operand is 1; otherwise, it is set to 0. • Example: $a \ \ b$ • Bitwise XOR (^): <ul style="list-style-type: none"> • Compares each bit of two integers. The result is a new integer where a bit is set to 1 if the corresponding bits of the operands are different; otherwise, it is set to 0. • Example: $a \ ^ \ b$ • Bitwise NOT (~): <ul style="list-style-type: none"> • Inverts each bit of an integer. The result is a new integer where each bit of the operand is flipped (1 becomes 0 and 0 becomes 1). • Example: $\sim a$ • Bitwise Left Shift (<<): <ul style="list-style-type: none"> • Shifts the bits of an integer to the left by a specified number of positions. The empty positions on the right are filled with 0s. • Example: $a \ << \ n$ • Bitwise Right Shift (>>): <ul style="list-style-type: none"> • Shifts the bits of an integer to the right by a specified number of positions. The empty positions on the left are filled with the sign bit for signed numbers. 	4M

	<ul style="list-style-type: none"> • Example: a >> n 	
6	<p>Write a python program using function to find the sum of first 'n' numbers</p> <pre>def sum_of_n_numbers(n): # Initialize the sum variable total = 0 # Use a loop to add numbers from 1 to n for i in range(1, n + 1): total += i return total # Input: The value of n n = int(input("Enter a positive integer: ")) # Output: The sum of the first n numbers print(f"The sum of the first {n} numbers is: {sum_of_n_numbers(n)}")</pre>	4M
7	<p>Write short notes on local variables and global variables.</p> <p>Local Variables</p> <p>Definition: Local variables are variables that are declared inside a function and can only be used within that function. They are not accessible outside the function in which they are defined.</p> <p>Scope: The scope of a local variable is limited to the function block in which it is defined.</p> <p>Example:</p> <pre>def my_function(): local_var = 10 # This is a local variable print(local_var) my_function() # print(local_var) # This will raise an error because local_var is not accessible outside the function</pre> <p>Global Variables</p> <p>Definition: Global variables are variables that are declared outside of any function and can be accessed from any function within the same module.</p> <p>Scope: The scope of a global variable extends to the entire module</p>	4M

	<p>in which it is declared.</p> <p>Example:</p> <pre>global_var = 20 # This is a global variable def my_function(): print(global_var) # Accessing the global variable inside a function my_function() print(global_var) # Accessing the global variable outside the function</pre> <p>Using Global Variables Inside Functions:</p> <p>To modify a global variable inside a function, you need to use the <code>global</code> keyword.</p> <p>Example:</p> <pre>global_var = 20 # This is a global variable def my_function(): global global_var global_var = 30 # Modifying the global variable inside the function print(global_var) my_function() print(global_var) # The global variable is modified globally</pre>	
--	---	--

Q. No	Question (s)	Marks
1	<p>a) List the applications of Python?</p> <p>Applications of Python</p> <ol style="list-style-type: none"> Web Development: Used for creating dynamic websites and web applications (e.g., Django, Flask). Data Science and Analysis: Used for data manipulation, analysis, and visualization (e.g., Pandas, NumPy, Matplotlib). Artificial Intelligence and Machine Learning: Used for developing AI and ML models (e.g., TensorFlow, scikit-learn). Automation and Scripting: Used for automating repetitive tasks and writing scripts. Game Development: Used for creating video games (e.g., Pygame). Desktop GUI Applications: Used for building desktop applications (e.g., Tkinter, PyQt). Networking: Used for network programming and developing network applications. <p>Python's versatility and extensive libraries make it suitable for a wide range of applications across various domains.</p>	1M
	<p>b) State the type conversion in python with an example.</p> <p>Type conversion in Python refers to converting one data type to another. It can be done using built-in functions.</p>	1M
	<p>c) What are the modes used in python execution?</p> <p>Modes of Python Execution</p> <ol style="list-style-type: none"> Interactive Mode: Allows you to execute Python commands one at a time in a prompt, useful for testing and experimenting with code snippets. <ul style="list-style-type: none"> Example: Running Python in a terminal or command prompt by typing <code>python</code>. Script Mode: Executes a sequence of Python statements stored in a file, useful for running complete programs. <ul style="list-style-type: none"> Example: Running a Python script file by typing <code>python script_name.py</code> in the terminal or command prompt. <p>These modes provide flexibility for both quick tests and full-scale program execution.</p>	1M
	<p>d) What is the use of input () function?</p> <p>The <code>input ()</code> function in Python is used to take input from the user. It reads a line of text from the standard input (usually the</p>	1M

	keyboard) and returns it as a string. Ex. name = input("Enter your name: ") print(f"Hello, {name}!")													
	e) Define String slice. A string slice in Python is a substring extracted from a given string using a specified range of indices. Ex: text = "Hello, World!" slice = text[7:12] print(slice) # Output: World	1M												
	f) Write the syntax of for loop? for i in range(5): print(i)	1M												
	g) What are the identity operators in python? Identity Operators in Python Python provides two identity operators to compare the memory locations of two objects: 1. is : Returns True if both variables point to the same object. 2. is not : Returns True if both variables point to different objects.	1M												
	h) What is the purpose of break and continue statement in python? <table><tr><th>Statement</th><th>Purpose</th><th>Example</th></tr><tr><td>break</td><td>Terminates the loop immediately.</td><td><pre>python for i in range(5): if i == 3: break print(i) # Output: 0 1 2</pre></td></tr><tr><td>continue</td><td>Skips the rest of the code in the current iteration and moves to the next iteration.</td><td><pre>python for i in range(5): if i == 3: continue print(i) # Output: 0 1 2 4</pre></td></tr></table>	Statement	Purpose	Example	break	Terminates the loop immediately.	<pre>python for i in range(5): if i == 3: break print(i) # Output: 0 1 2</pre>	continue	Skips the rest of the code in the current iteration and moves to the next iteration.	<pre>python for i in range(5): if i == 3: continue print(i) # Output: 0 1 2 4</pre>	1M			
Statement	Purpose	Example												
break	Terminates the loop immediately.	<pre>python for i in range(5): if i == 3: break print(i) # Output: 0 1 2</pre>												
continue	Skips the rest of the code in the current iteration and moves to the next iteration.	<pre>python for i in range(5): if i == 3: continue print(i) # Output: 0 1 2 4</pre>												
	i) Define Local and global variables. <table><tr><th>Variable Type</th><th>Definition</th><th>Scope</th><th>Example</th></tr><tr><td>Local</td><td>Defined inside a function.</td><td>Accessible only within the function.</td><td><pre>python def my_func(): local_var = 10</pre></td></tr><tr><td>Global</td><td>Defined outside any function.</td><td>Accessible throughout the entire program.</td><td><pre>python global_var = 20 def my_func(): print(global_var)</pre></td></tr></table>	Variable Type	Definition	Scope	Example	Local	Defined inside a function.	Accessible only within the function.	<pre>python def my_func(): local_var = 10</pre>	Global	Defined outside any function.	Accessible throughout the entire program.	<pre>python global_var = 20 def my_func(): print(global_var)</pre>	1M
Variable Type	Definition	Scope	Example											
Local	Defined inside a function.	Accessible only within the function.	<pre>python def my_func(): local_var = 10</pre>											
Global	Defined outside any function.	Accessible throughout the entire program.	<pre>python global_var = 20 def my_func(): print(global_var)</pre>											
	j) State the purpose of math module in Python. The math module in Python provides a wide range of mathematical functions and constants, allowing you to perform complex mathematical operations with ease. Ex: import math	1M												

	<pre># Using some functions from the math module print(math.sqrt(16)) # Output: 4.0 print(math.pi) # Output: 3.141592653589793 print(math.sin(math.radians(90))) # Output: 1.0</pre>	
--	---	--

PART – B (20M)

Q. No	Question (s)	Marks																		
2	<p>a) Differentiate List and Tuple with an example.</p> <table border="1"> <thead> <tr> <th>Feature</th><th>List</th><th>Tuple</th></tr> </thead> <tbody> <tr> <td>Definition</td><td>Ordered, mutable collection of items.</td><td>Ordered, immutable collection of items.</td></tr> <tr> <td>Syntax</td><td><code>list_name = [item1, item2, item3, ...]</code></td><td><code>tuple_name = (item1, item2, item3, ...)</code></td></tr> <tr> <td>Mutability</td><td>Mutable (can be changed).</td><td>Immutable (cannot be changed).</td></tr> <tr> <td>Performance</td><td>Slower due to extra memory operations.</td><td>Faster due to immutability.</td></tr> <tr> <td>Use Case</td><td>Suitable for collections that need to change.</td><td>Suitable for collections that should remain constant.</td></tr> </tbody> </table> <p>List: <pre>fruits_list = ["apple", "banana", "cherry"] fruits_list.append("orange") # Output: ['apple', 'banana', 'cherry', 'orange']</pre> Tuple: <pre>fruits_tuple = ("apple", "banana", "cherry") # Attempting to modify will result in an error</pre> Lists are versatile and mutable, allowing changes to the elements, while tuples offer performance benefits and immutability, ensuring the elements remain constant once defined.</p>	Feature	List	Tuple	Definition	Ordered, mutable collection of items.	Ordered, immutable collection of items.	Syntax	<code>list_name = [item1, item2, item3, ...]</code>	<code>tuple_name = (item1, item2, item3, ...)</code>	Mutability	Mutable (can be changed).	Immutable (cannot be changed).	Performance	Slower due to extra memory operations.	Faster due to immutability.	Use Case	Suitable for collections that need to change.	Suitable for collections that should remain constant.	4M
Feature	List	Tuple																		
Definition	Ordered, mutable collection of items.	Ordered, immutable collection of items.																		
Syntax	<code>list_name = [item1, item2, item3, ...]</code>	<code>tuple_name = (item1, item2, item3, ...)</code>																		
Mutability	Mutable (can be changed).	Immutable (cannot be changed).																		
Performance	Slower due to extra memory operations.	Faster due to immutability.																		
Use Case	Suitable for collections that need to change.	Suitable for collections that should remain constant.																		
	<p>b) Explain in detail about python type conversion and type casting?</p> <table border="1"> <thead> <tr> <th>Feature</th><th>Type Conversion</th><th>Type Casting</th></tr> </thead> <tbody> <tr> <td>Definition</td><td>Implicit conversion of one data type to another by Python automatically.</td><td>Explicit conversion of one data type to another using built-in functions.</td></tr> <tr> <td>Performed By</td><td>Python interpreter</td><td>Programmer</td></tr> <tr> <td>Ease of Use</td><td>Easier as it is done automatically by Python</td><td>Requires explicit instruction from the programmer</td></tr> <tr> <td>Risk of Error</td><td>Lower risk of errors as Python manages the conversion</td><td>Higher risk of errors if conversion is not handled properly</td></tr> </tbody> </table> <p>Type Conversion: <pre># Implicit conversion num_int = 10 num_float = 10.5 result = num_int + num_float print(type(result)) # Output: <class 'float'></pre> Type Casting:</p>	Feature	Type Conversion	Type Casting	Definition	Implicit conversion of one data type to another by Python automatically.	Explicit conversion of one data type to another using built-in functions.	Performed By	Python interpreter	Programmer	Ease of Use	Easier as it is done automatically by Python	Requires explicit instruction from the programmer	Risk of Error	Lower risk of errors as Python manages the conversion	Higher risk of errors if conversion is not handled properly	4M			
Feature	Type Conversion	Type Casting																		
Definition	Implicit conversion of one data type to another by Python automatically.	Explicit conversion of one data type to another using built-in functions.																		
Performed By	Python interpreter	Programmer																		
Ease of Use	Easier as it is done automatically by Python	Requires explicit instruction from the programmer																		
Risk of Error	Lower risk of errors as Python manages the conversion	Higher risk of errors if conversion is not handled properly																		

	<pre># Explicit conversion num_str = "100" num_int = int(num_str) print(type(num_int)) # Output: <class 'int'></pre> <p>In summary, type conversion is automatic and managed by Python, while type casting is manual and requires the programmer to explicitly convert data types using built-in functions like <code>int()</code>, <code>float()</code>, <code>str()</code>, etc.</p>	
3	<p>a) What is Dictionary? Explain Python dictionaries in detail discussing its operations and methods.</p> <p>dictionary in Python is a collection of key-value pairs. Each key is unique and associated with a value. Dictionaries are mutable, meaning they can be changed after creation, and they are unordered, meaning that the items do not have a defined order.</p> <p>Syntax:</p> <pre>my_dict = { 'key1': 'value1', 'key2': 'value2', 'key3': 'value3' }</pre> <p>Key Features:</p> <ul style="list-style-type: none"> • Key-Value Pairs: Each key is associated with a value. • Mutable: Can be modified (e.g., adding, updating, or deleting items). • Unordered: Items do not have a defined order. • Keys: Must be unique and immutable (e.g., strings, numbers, tuples). <p>Common Operations:</p> <ul style="list-style-type: none"> • Creating a Dictionary: <pre>my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'}</pre> • Accessing Values: <pre>print(my_dict['name']) # Output: Alice</pre> • Adding or Updating Key-Value Pairs: <pre>my_dict['email'] = 'alice@example.com'</pre> 	<p>8M 4M</p>

	<ul style="list-style-type: none"> • Removing Key-Value Pairs: <code>del my_dict['city']</code> • Checking if a Key Exists: <code>if 'name' in my_dict: print("Key 'name' exists.")</code> <p>Common Methods:</p> <ul style="list-style-type: none"> • <code>get(key, default=None)</code>: Returns the value for a specified key. <code>age = my_dict.get('age', 0)</code> • <code>keys()</code>: Returns a view object containing all keys. <code>print(my_dict.keys())</code> • <code>values()</code>: Returns a view object containing all values. <code>print(my_dict.values())</code> • <code>items()</code>: Returns a view object containing all key-value pairs. <code>print(my_dict.items())</code> • <code>pop(key, default=None)</code>: Removes a specified key and returns the corresponding value. <code>email = my_dict.pop('email', 'Not Available')</code> • <code>update([other])</code>: Updates the dictionary with key-value pairs from another dictionary or iterable. <code>my_dict.update({'city': 'Boston', 'country': 'USA'})</code> <p>dictionaries are versatile and powerful, making them essential for various applications.</p>	
	<p>b) What is SET? Explain Python SET in detail discussing its operations and methods</p> <p>A set in Python is an unordered collection of unique elements. Sets are used to store multiple items in a single variable, but unlike lists or tuples, they do not allow duplicate elements. Sets are mutable,</p>	4M

meaning you can add or remove items after creation, but the elements themselves must be immutable.

Syntax:

```
my_set = {1, 2, 3, 4}
```

Alternatively, you can create a set using the `set()` constructor:

```
my_set = set([1, 2, 3, 4])
```

Key Features:

- **Unordered:** The elements do not have a defined order.
- **Unique Elements:** Duplicate elements are not allowed.
- **Mutable:** Sets can be modified after creation.

Common Operations:

1. Creating a Set:

```
my_set = {1, 2, 3, 4}
my_set = set([1, 2, 3, 4]) # Using the set()
                           constructor
```

2. Adding Elements:

```
my_set.add(5)
```

3. Removing Elements:

```
my_set.remove(3) # Raises KeyError if the
                 element does not exist
my_set.discard(3) # Does not raise an error if
                  the element does not exist
```

4. Checking Membership:

```
if 2 in my_set:
    print("2 is in the set")
```

5. Iterating Through a Set:

```
for element in my_set:
    print(element)
```

Common Methods:

	<ol style="list-style-type: none"> 1. <code>add(element)</code>: Adds an element to the set. <code>my_set.add(5)</code> 2. <code>remove(element)</code>: Removes an element from the set. Raises <code>KeyError</code> if the element is not found. <code>my_set.remove(3)</code> 3. <code>discard(element)</code>: Removes an element from the set if it exists. <code>my_set.discard(3)</code> 4. <code>clear()</code>: Removes all elements from the set. <code>my_set.clear()</code> 5. <code>union(set)</code>: Returns a new set with all elements from both sets. <pre>set1 = {1, 2, 3} set2 = {3, 4, 5} set3 = set1.union(set2) # Output: {1, 2, 3, 4, 5}</pre> 6. <code>intersection(set)</code>: Returns a new set with elements common to both sets. <pre>set1 = {1, 2, 3} set2 = {3, 4, 5} set3 = set1.intersection(set2) # Output: {3}</pre> 7. <code>difference(set)</code>: Returns a new set with elements in the first set but not in the second. <pre>set1 = {1, 2, 3} set2 = {3, 4, 5} set3 = set1.difference(set2) # Output: {1, 2}</pre> 8. <code>symmetric_difference(set)</code>: Returns a new set with elements in either set, but not in both. <pre>set1 = {1, 2, 3} set2 = {3, 4, 5} set3 = set1.symmetric_difference(set2)</pre> 	
4	a) Explain in detail about decision structures with a suitable example.	4M

	<p>Decision Structures in Python</p> <p>Decision structures, also known as control flow statements, allow a program to make decisions and execute specific blocks of code based on certain conditions. They enable the program to respond dynamically to different inputs or states.</p> <p>Types of Decision Structures:</p> <ol style="list-style-type: none"> 1. if Statement: <ul style="list-style-type: none"> ◦ Executes a block of code if a specified condition is true. 2. if-else Statement: <ul style="list-style-type: none"> ◦ Executes one block of code if the condition is true and another block of code if the condition is false. 3. if-elif-else Statement: <ul style="list-style-type: none"> ◦ Executes a block of code based on multiple conditions, where only the first true condition's block is executed. 4. Nested if Statements: <ul style="list-style-type: none"> ◦ An if statement inside another if statement, allowing for more complex decision-making. <p>Example: if-elif-else Statement</p> <pre>def determine_grade(score): # Determine grade based on score using if-elif-else structure if score >= 90: grade = 'A' elif score >= 80: grade = 'B' elif score >= 70: grade = 'C' elif score >= 60: grade = 'D' else: grade = 'F' return grade # Input: Student's score score = int(input("Enter the student's score: ")) # Output: Display the grade print(f"The student's grade is: {determine_grade(score)}")</pre>	
	<p>b) What is meant by string? How to create and access the string with an example?</p> <p>A string in Python is a sequence of characters enclosed within</p>	4M

	<p>single quotes (' '), double quotes (" "), or triple quotes (''' ''')</p> <p>or """ """</p> <p># Using single quotes</p> <pre>string1 = 'Hello, World!'</pre> <p># Using double quotes</p> <pre>string2 = "Hello, World!"</pre> <p># Using triple quotes</p> <pre>string3 = """Hello, World!"""</pre> <p>Accessing Elements in a String:</p> <p>You can access individual characters in a string using indexing. Python uses zero-based indexing.</p> <p># Example string</p> <pre>my_string = "Hello, World!"</pre> <p># Accessing characters by index</p> <pre>first_char = my_string[0] # Output: 'H' fifth_char = my_string[4] # Output: 'o' last_char = my_string[-1] # Output: '!'</pre> <p># Creating a string</p> <pre>greeting = "Hello, Python!"</pre> <p># Accessing characters</p> <pre>print(greeting[0]) # Output: H print(greeting[7]) # Output: P print(greeting[-1]) # Output: !</pre>	
5	<p>a) Write a short note on membership operator with suitable programs.</p> <p>Membership Operator in Python</p> <p>Membership operators in Python are used to test whether a value or variable is found in a sequence (such as a string, list, tuple, set, or dictionary). There are two membership operators: <code>in</code> and <code>not in</code>.</p> <p>Operators:</p> <ol style="list-style-type: none"> <code>in</code> Operator: <ul style="list-style-type: none"> Returns <code>True</code> if the specified value is present in the sequence. Syntax: <code>value in sequence</code> 	4M

	<p>2. not in Operator:</p> <ul style="list-style-type: none"> ○ Returns <code>True</code> if the specified value is not present in the sequence. ○ Syntax: <code>value not in sequence</code> <p>Examples:</p> <p>1. Using in Operator with a List:</p> <pre>fruits = ['apple', 'banana', 'cherry'] print('apple' in fruits) # Output: True print('orange' in fruits) # Output: False</pre> <p>2. Using not in Operator with a String:</p> <pre>greeting = "Hello, World!" print('Hello' in greeting) # Output: True print('Python' not in greeting) # Output: True</pre> <p>3. Using in Operator with a Dictionary:</p> <pre>student = {'name': 'John', 'age': 20, 'major': 'CS'} print('name' in student) # Output: True (checks keys by default) print('John' in student.values()) # Output: True (checks values)</pre> <ul style="list-style-type: none"> • Lists and Strings: The <code>in</code> operator checks if an element exists in a list or a substring exists in a string. • Dictionaries: The <code>in</code> operator checks if a key exists in a dictionary. To check for values, use the <code>.values()</code> method. <p>Membership operators provide a simple and effective way to check the presence of elements in various data structures, ensuring efficient and readable code.</p>	
	<p>b) Write a short note on number conversion (binary to others).</p> <p>Binary Number Conversion</p> <p>Binary numbers (base-2) can be converted to other number systems like decimal (base-10), octal (base-8), and hexadecimal (base-16). Here's a brief overview of the conversions:</p> <p>Binary to Decimal:</p> <p>Convert binary to decimal by multiplying each bit by 2 raised to the power of its position (starting from 0) and summing the results.</p>	4M

	<p>Example: Binary: 1011 Decimal: $(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 8 + 0 + 2 + 1 = 11$ $(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) = 8 + 0 + 2 + 1 = 11$</p> <p>Binary to Octal:</p> <p>Group binary digits into sets of three (starting from the right) and convert each group to its octal equivalent. Example: Binary: 1011 Group: 001 011 Octal: 1 3 => 13</p> <p>Binary to Hexadecimal:</p> <p>Group binary digits into sets of four (starting from the right) and convert each group to its hexadecimal equivalent. Example: Binary: 1011 Group: 0001 0110 Hexadecimal: 1 6 => 16</p> <p>Python Code Examples:</p> <ol style="list-style-type: none"> Binary to Decimal: <pre>binary = '1011' decimal = int(binary, 2) print(decimal) # Output: 11</pre> Binary to Octal: <pre>binary = '1011' decimal = int(binary, 2) octal = oct(decimal) print(octal) # Output: 0o13</pre> Binary to Hexadecimal: <pre>binary = '1011' decimal = int(binary, 2) hexadecimal = hex(decimal) print(hexadecimal) # Output: 0xb</pre> 	
6	<p>Explain the types of arguments with an example program .</p> <p>Types of Arguments in Python</p> <ol style="list-style-type: none"> Positional Arguments: <ul style="list-style-type: none"> Passed to a function in a specific order. Keyword Arguments: <ul style="list-style-type: none"> Passed to a function with a specific keyword. Default Arguments: <ul style="list-style-type: none"> Assume a default value if not provided. Variable-Length Arguments: <ul style="list-style-type: none"> Allow a function to accept an arbitrary number of arguments. Denoted by <code>*args</code> for non-keyword 	4M

	<p>arguments and <code>**kwargs</code> for keyword arguments.</p> <p>Example Program:</p> <pre>def display_info(name, age=25, *skills, **details): # Positional argument print(f"Name: {name}") # Default argument print(f"Age: {age}") # Variable-length non-keyword argument print("Skills:", ", ".join(skills)) # Variable-length keyword argument for key, value in details.items(): print(f"{key}: {value}") # Calling the function with different types of arguments display_info("Alice", 30, "Python", "Machine Learning", city="New York", country="USA")</pre> <p>Explanation:</p> <ol style="list-style-type: none"> Positional Argument: <ul style="list-style-type: none"> name: "Alice" Default Argument: <ul style="list-style-type: none"> age: 30 (overrides default 25) Variable-Length Non-Keyword Argument: <ul style="list-style-type: none"> *skills: "Python", "Machine Learning" Variable-Length Keyword Argument: <ul style="list-style-type: none"> **details: {"city": "New York", "country": "USA"} <p>Output:</p> <p>Name: Alice</p> <p>Age: 30</p> <p>Skills: Python, Machine Learning</p> <p>city: New York</p> <p>country: USA</p>	
7	Explain the Recursion function with an example program.	4M

	<pre>def factorial(n): # Base Case: If n is 0 or 1, return 1 if n == 0 or n == 1: return 1 # Recursive Case: Call the function with (n-1) and multiply by n else: return n * factorial(n - 1) # Input: Number for which factorial is to be found number = 5 # Output: Factorial of the number print(f"The factorial of {number} is {factorial(number)}")</pre>	
--	--	--