CMR INSTITUTE OF TECHNOLOGY

Affiliated to VTU, Approved by AICTE, and Accredited by NBA, by NAAC with A++

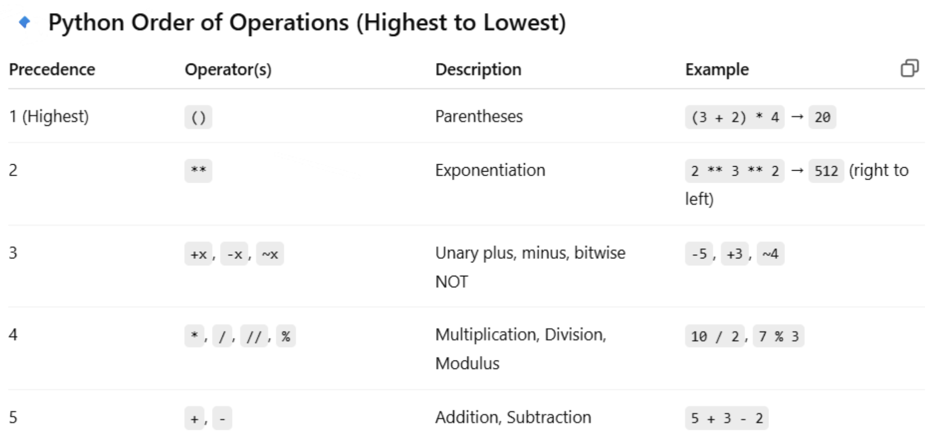
<u>VTU- SoE</u>

VTU 3rd Semester MBA Degree Examination Dec'24/ Jan '25

Introduction to Python, Data and Control Systems

22MBABA303

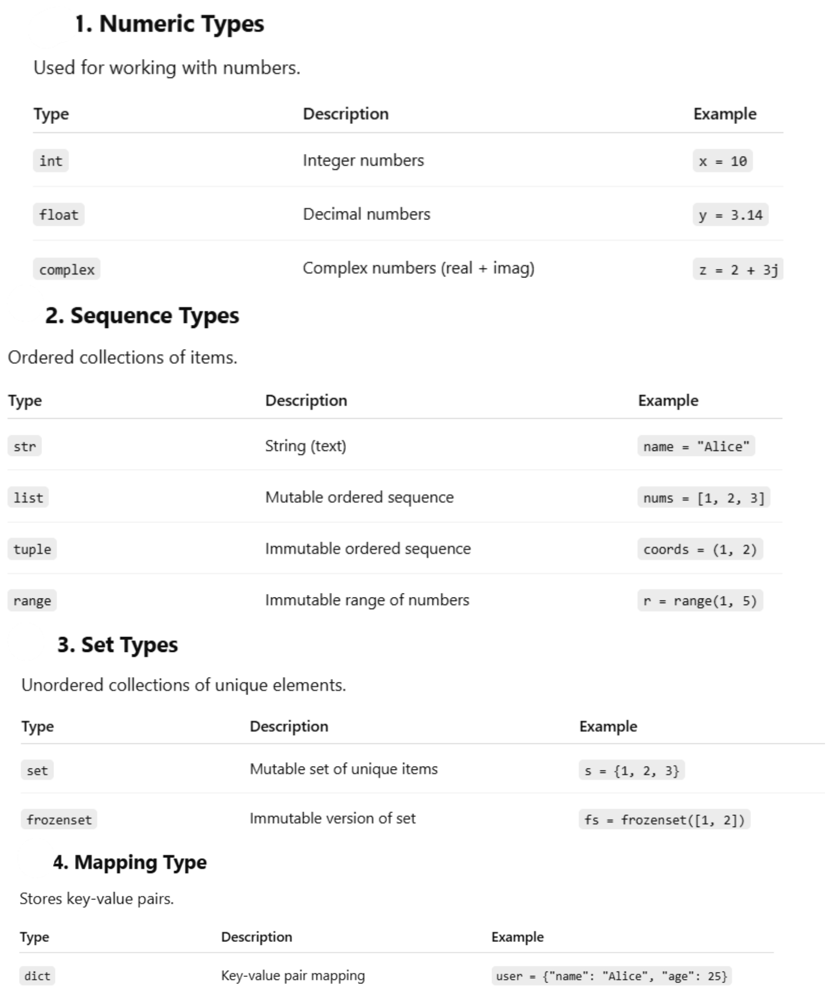
Questions	Solution
1. a.	**Explain about "Creativity" and Innovation" in Python** Creativity in Python Creativity refers to the ability to think of new and original ideas. In Python, this can be expressed through: Examples: 1. Creative Problem Solving: ✓ Writing a concise one-liner instead of a verbose loop. 2. Creative Use of Python Libraries: ✓ Combining libraries like Pandas, Matplotlib, and Scikit-learn in unique ways. 3. Creative Code Design: ✓ Writing elegant, readable, and efficient code. ✓ Using design patterns innovatively in Python projects. Innovation in Python Innovation is about turning creative ideas into practical solutions, tools, or products. In Python, this involves building something new or improving existing solutions. Examples: 1. Developing New Tools or Libraries: ✓ Creating a new Python package that simplifies a complex task. ✓ Contributing to open-source projects with new features. 2. Automating Manual Processes: ✓ Writing Python scripts to automate repetitive business or data processing tasks. 3. Innovative Applications: ✓ Using AI/ML with Python to build chatbots, recommendation systems, or fraud detection systems. Developing new web apps with frameworks like Django or FastAPI.
b	**Explain about the order of operations in Python**



◆ Python Order of Operations (Highest to Lowest)

Precedence	Operator(s)	Description	Example
1 (Highest)	`()`	Parentheses	`(3 + 2) * 4` → `20`
2	`**`	Exponentiation	`2 ** 3 ** 2` → `512` (right to left)
3	`+x`, `-x`, `~x`	Unary plus, minus, bitwise NOT	`-5`, `+3`, `~4`
4	`*`, `/`, `//`, `%`	Multiplication, Division, Modulus	`10 / 2`, `7 % 3`
5	`+`, `-`	Addition, Subtraction	`5 + 3 - 2`

Questions	Solution
c	**Explain Python datatypes**



1. Numeric Types

Used for working with numbers.

Type	Description	Example
`int`	Integer numbers	`x = 10`
`float`	Decimal numbers	`y = 3.14`
`complex`	Complex numbers (real + imag)	`z = 2 + 3j`

2. Sequence Types

Ordered collections of items.

Type	Description	Example
`str`	String (text)	`name = "Alice"`
`list`	Mutable ordered sequence	`nums = [1, 2, 3]`
`tuple`	Immutable ordered sequence	`coords = (1, 2)`
`range`	Immutable range of numbers	`r = range(1, 5)`

3. Set Types

Unordered collections of unique elements.

Type	Description	Example
`set`	Mutable set of unique items	`s = {1, 2, 3}`
`frozenset`	Immutable version of set	`fs = frozenset([1, 2])`

4. Mapping Type

Stores key-value pairs.

Type	Description	Example
`dict`	Key-value pair mapping	`user = {"name": "Alice", "age": 25}`

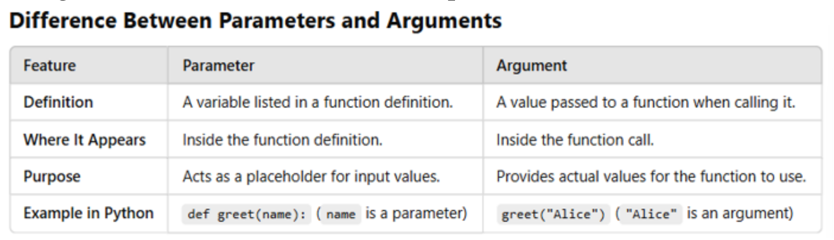
2a

State parameters and arguments with a suitable example

A parameter is the variable name listed in a function definition. It acts as a placeholder for the values (arguments) the function will receive.

Argument

An argument is the actual value that is passed to the function when it is called.



Difference Between Parameters and Arguments

Feature	Parameter	Argument
Definition	A variable listed in a function definition.	A value passed to a function when calling it.
Where It Appears	Inside the function definition.	Inside the function call.
Purpose	Acts as a placeholder for input values.	Provides actual values for the function to use.
Example in Python	`def greet(name):` (`name` is a parameter)	`greet("Alice")` (`"Alice"` is an argument)

Ex:

```
# Function definition
def greet(name):   # 'name' is a parameter
    print("Hello,", name)

# Function call
greet("CMRIT")    # "CMRIT" is an argument
```

b

Explain "Python" debugging

Debugging in Python means identifying and fixing errors (bugs) in your code. It's an essential skill for writing clean, working programs.

Types of Bugs

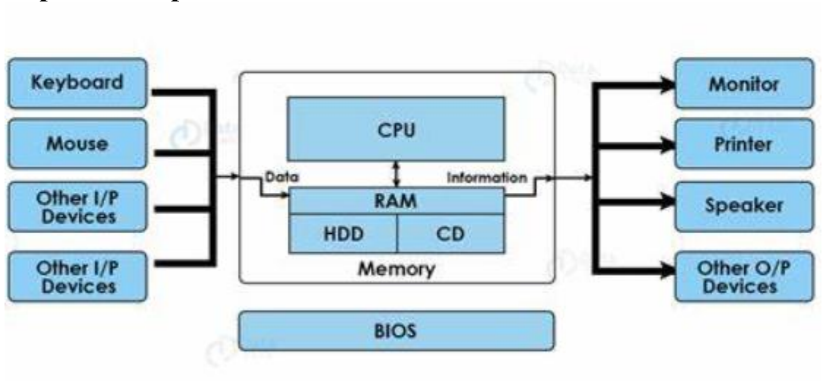
1. Syntax Errors – Mistakes in code structure
   Example: if x == 5 print(x)
2. Runtime Errors – Errors that occur during execution
   Example: dividing by zero
3. Logical Errors – Code runs but gives wrong output
   Example: wrong formula or loop condition

Best Practices for Debugging

- Reproduce the bug consistently.
- Add print/log statements to isolate where it occurs.
- Use a debugger to inspect variables.
- Keep your code modular to test parts independently.
- Remove or disable debug code when deploying.

c

Explain Computer Hardware architecture



A Computer Hardware Architecture Block Diagram typically consists of the main components of a computer system and how they interact. Here's a simple breakdown of the key components:

Main Components of Computer Architecture

1. Central Processing Unit (CPU)
   - Control Unit (CU): Directs operations within the computer.
   - Arithmetic Logic Unit (ALU): Performs mathematical and logical operations.
   - Registers: Small storage areas inside the CPU for temporary data.
2. Memory (Primary Storage/RAM & ROM)
   - RAM (Random Access Memory): Temporary storage for running programs.
   - ROM (Read-Only Memory): Stores firmware and system boot instructions.
3. Input Devices
   - Examples: Keyboard, Mouse, Scanner, Microphone.
4. Output Devices
   - Examples: Monitor, Printer, Speakers.
5. Storage (Secondary Memory)
   - HDD (Hard Disk Drive) / SSD (Solid-State Drive): Permanent data storage.
6. System Bus (Data Pathways)
   - Data Bus: Transfers actual data between components.
   - Address Bus: Transfers memory addresses.
   - Control Bus: Sends control signals between CPU and other components.

	7. I/O Interfaces & Peripherals o Includes USB ports, network cards, and external devices. 8. Power Supply Unit (PSU) o Converts electrical power for the system.

3 a	**State syntax of if—else statements**
	Basic Syntax:
	if condition:
	   # block of code if condition is True
	else:
	   # block of code if condition is False
	
	Ex:
	age = 18
	if age >= 18:
	   print("You are eligible to vote.")
	else:
	   print("You are not eligible to vote.")
	
	elif (else if)
	Used to check multiple conditions.
	marks = 85
	if marks >= 90:
	   print("Grade: A")
	elif marks >= 75:
	   print("Grade: B")
	else:
	   print("Grade: C")
	
	• Use colons (:) at the end of if, elif, and else.
	• Indentation (usually 4 spaces) is required for the code blocks.

b	**Explain various types of string methods and what it does?** 1. len() Returns the length of the string. len("hello") # Output: 5 2. lower() and upper() Convert the string to lowercase or uppercase. "Hello".lower() # Output: 'hello' "hello".upper() # Output: 'HELLO' 3. strip() Removes leading and trailing whitespace (or specified characters). " hello ".strip() # Output: 'hello' 4. replace(old, new) Replaces all occurrences of a substring with another. "apple".replace("p", "b") # Output: 'abble' 5. split(separator) Splits a string into a list using the specified separator. "one,two,three".split(",") # Output: ['one', 'two', 'three'] 6. join(iterable)

Joins elements of a list into a single string.

```
",".join(["a", "b", "c"])  # Output: 'a,b,c'
```

7. find(sub) and index(sub)

Finds the position of a substring. find() returns -1 if not found; index() raises an error.

```
"hello".find("e")   # Output: 1
"hello".index("e")  # Output: 1
```

8. startswith() and endswith()

Check if a string starts or ends with a specific substring.

```
"hello".startswith("he")  # Output: True
"hello".endswith("lo")    # Output: True
```

9. isalnum(), isalpha(), isdigit()

Check character properties.

```
"abc123".isalnum()  # True
"abc".isalpha()     # True
"123".isdigit()     # True
```

10. capitalize(), title()

Format capitalization.

```
"hello world".capitalize()  # Output: 'Hello world'
"hello world".title()       # Output: 'Hello World'
```

c. **Explain chain conditional statements in Python.**

Chained conditionals let you check multiple conditions in a structured way using if, elif, and else.

Syntax:

```
if condition1:
   # Code block if condition1 is True
elif condition2:
   # Code block if condition2 is True
elif condition3:
   # Code block if condition3 is True
else:
   # Code block if none of the above conditions are True
```

Example:

```
marks = 78
if marks >= 90:
   print("Grade: A")
elif marks >= 75:
   print("Grade: B")
elif marks >= 60:
   print("Grade: C")
else:
   print("Grade: D")
```

Output:

```
Grade: B
```

Note:

- Python checks the conditions from top to bottom.
- As soon as one condition is True, it executes that block and skips the rest.
- The else block is optional, but useful for a default case.

Nested conditionals

You can nest one if...else block inside another:

```
age = 20
citizen = True
if age >= 18:
    if citizen:
        print("You are eligible to vote.")
    else:
        print("You must be a citizen to vote.")
else:
    print("You are not old enough to vote.")
Output:
You are eligible to vote.
```

4 a

State the working of a Python ‘break’ statement

The break statement is used to exit a loop early — before the loop condition becomes False or the loop finishes all iterations.

<u>Inside for loops:</u>

```
for item in iterable:
    if condition:
        break  # Exit the loop
```

<u>while condition:</u>

```
    if condition_to_stop:
        break
```

Example 1: Using break in a for loop

```
for num in range(1, 10):
    if num == 5:
        break
    print(num)
```

Output:

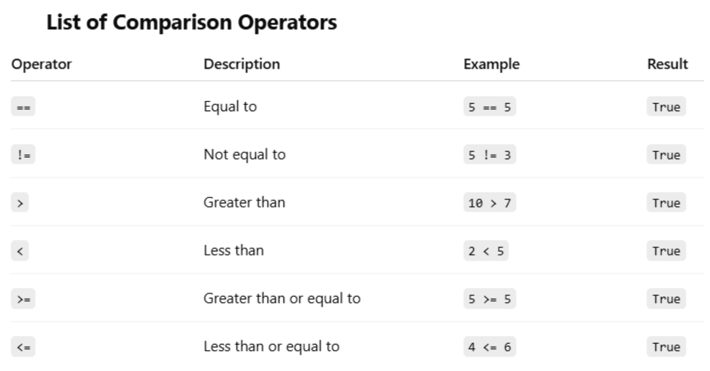
```
1
2
3
4
```

➞ The loop stops when num == 5.

b

Explain comparison operators in Python

Comparison operators are used to compare two values. They return a Boolean value:
- True if the comparison is correct
- False otherwise



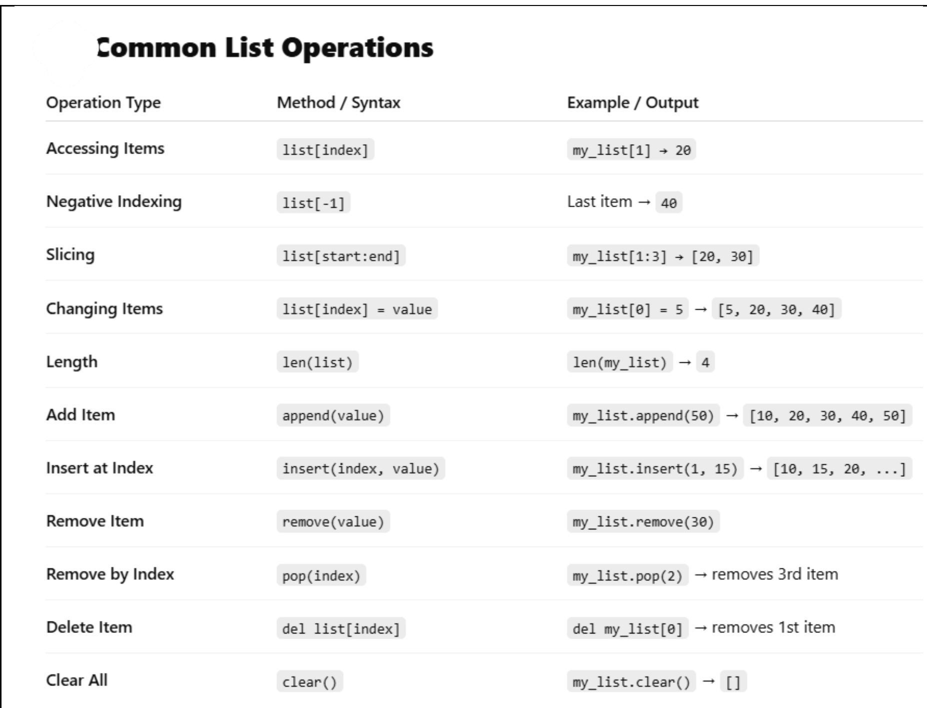
List of Comparison Operators

Operator	Description	Example	Result
`==`	Equal to	`5 == 5`	`True`
`!=`	Not equal to	`5 != 3`	`True`
`>`	Greater than	`10 > 7`	`True`
`<`	Less than	`2 < 5`	`True`
`>=`	Greater than or equal to	`5 >= 5`	`True`
`<=`	Less than or equal to	`4 <= 6`	`True`

Example:

```
a = 10
b = 20
print(a == b)   # False
print(a != b)   # True
print(a < b)    # True
print(a > b)    # False
```

	print(a <= b) # True print(a >= b) # False
c	**Explain “String slices” and “Strings are mutable”** <u>String Slicing</u> Slicing allows you to extract a part (substring) of a string using index ranges. Syntax: string[start : end : step] • start: index to begin (inclusive) • end: index to stop (exclusive) • step: interval (default is 1) Ex: text = "Python" print(text[0:3]) # 'Pyt' print(text[2:]) # 'thon' print(text[:4]) # 'Pyth' print(text[-1]) # 'n' (last character) print(text[::-1]) # 'nohtyP' (reverse string) <u>Strings Are Immutable</u> In Python, strings are immutable, which means you cannot change them after creation. word = "hello" word[0] = "H" # This will raise an error Output: TypeError: 'str' object does not support item assignment So, a new string can be created using slicing and concatenation: word = "hello" new_word = "H" + word[1:] print(new_word) # Output: 'Hello'
5 a	**State Python Tuple with an example** Python Tuple A tuple is a built-in Python data type used to store multiple items in a single, ordered, and immutable collection. Key Characteristics of Tuples: • Ordered (items have a fixed position) • Immutable (cannot be changed after creation) • Allow duplicates • Can contain elements of different data types
b	**Explain various List operations in Python** A list is a built-in Python data type used to store multiple items in a mutable, ordered collection. Basic List Syntax: my_list = [10, 20, 30, 40]



Common List Operations

Operation Type	Method / Syntax	Example / Output
Accessing Items	`list[index]`	`my_list[1] → 20`
Negative Indexing	`list[-1]`	Last item → `40`
Slicing	`list[start:end]`	`my_list[1:3] → [20, 30]`
Changing Items	`list[index] = value`	`my_list[0] = 5` → `[5, 20, 30, 40]`
Length	`len(list)`	`len(my_list)` → `4`
Add Item	`append(value)`	`my_list.append(50)` → `[10, 20, 30, 40, 50]`
Insert at Index	`insert(index, value)`	`my_list.insert(1, 15)` → `[10, 15, 20, ...]`
Remove Item	`remove(value)`	`my_list.remove(30)`
Remove by Index	`pop(index)`	`my_list.pop(2)` → removes 3rd item
Delete Item	`del list[index]`	`del my_list[0]` → removes 1st item
Clear All	`clear()`	`my_list.clear()` → `[]`

c

Explain the various types of errors

1. Syntax Errors – Mistakes in code structure
   Example: if x == 5 print(x)
2. Runtime Errors – Errors that occur during execution
   Example: dividing by zero
3. Logical Errors – Code runs but gives wrong output
   Example: wrong formula or loop condition

Python programs can produce different types of errors (also called exceptions). These errors must be handled or fixed for the program to run correctly.

1. Syntax Errors
Occurs when Python code is not written correctly according to the rules of the language.
Ex:
```
if x == 5
   print("Hello")  # Missing colon
```
Error Message:
Syntax Error: expected ':'

2. Indentation Errors
Happens when the code is not properly indented (Python uses indentation to define blocks).
Example:
```
if True:
print("Hi")  # Not indented
```
Error Message:
IndentationError: expected an indented block

3. Name Errors
Occurs when you use a variable or function name that has not been defined.
Example:
```
print(score)  #  score not defined
```
Error Message:
NameError: name 'score' is not defined

	4. Type Errors Raised when an operation is applied to the wrong data type. Example: x = "5" + 2 # can't add string and integer Error Message: TypeError: can only concatenate str (not "int") to str 5. Value Errors Occurs when the data type is correct, but the value is not suitable. Example: int("abc") # cannot convert to integer Error Message: ValueError: invalid literal for int() with base 10 6. Index Errors Happens when you try to access an index that doesn't exist in a list or string. Example: my_list = [1, 2, 3] print(my_list[5]) # index out of range Error Message: IndexError: list index out of range 7. ZeroDivisionError Occurs when dividing by zero. Example: x = 10 / 0 # division by zero Error Message: ZeroDivisionError: division by zero 8. Attribute Errors Raised when an object does not have the attribute you're trying to access. Example: x = 5 x.append(3) # integers don't have 'append' Error Message: AttributeError: 'int' object has no attribute 'append' 9. Import Errors Occurs when Python cannot find the module you're trying to import. Example: import maths # typo: should be 'math' Error Message: ModuleNotFoundError: No module named 'maths'

6 a	**State python string with an example**
	A string in Python is a sequence of characters enclosed in quotes. It is one of the most commonly used data types for handling text.
	String Declaration:
	# String using single quotes
	greeting = 'Hello, World!'
	
	# String using double quotes
	name = "Python Programming"
	
	# Multi-line string using triple quotes
	info = '''This is a

```
multi-line string'''

print(greeting)
print(name)
print(info)
```

b

Tuples are immutable. Explain.
A tuple is a Python data type used to store multiple items in an ordered and unchangeable collection.
Example:

```
my_tuple = (10, 20, 30)
```

Immutability means you cannot change, add, or remove items after the tuple is created. Tuples are immutable – contents can't be changed. So, No append(), remove(), or item assignment. You can access, iterate, and create new tuples, but not modify existing ones
In contrast, lists are mutable, meaning their elements can be changed.

Example: Trying to Modify a Tuple (Will Cause Error)

```
my_tuple = (1, 2, 3)
my_tuple[0] = 100  # Attempt to change value
```

Output:

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

Hence,

- Access elements using indexing.
- Use slicing to read portions of the tuple.
- Reassign the whole tuple (create a new one).

```
my_tuple = (1, 2, 3)
new_tuple = my_tuple + (4, 5)  # This creates a new tuple
print(new_tuple)  # (1, 2, 3, 4, 5)
```

Why Use Immutable Tuples?

Advantage	Description
Safer	Prevents accidental changes
Faster	More efficient in memory and performance
Suitable for Keys	Can be used as keys in dictionaries

c

Explain the Dictionary operation in Python.

A dictionary in Python is a collection of key-value pairs. It is unordered, mutable, and indexed by keys (not positions).

Creating a Dictionary

```
student = {
    "name": "Suma",
    "age": 21,
    "course": "Data Science"
}
```

Common Dictionary Operations

Operation Type	Syntax / Method	Example Output
Access value	dict[key]	student["name"] → 'Alice'
Add item	dict[key] = value	student["grade"] = "A"
Update value	dict[key] = new_value	student["age"] = 22

Delete item	del dict[key]	del student["course"]
Get value safely	dict.get(key)	student.get("name") → 'Alice'
Keys list	dict.keys()	dict_keys(['name', 'age'])
Values list	dict.values()	dict_values(['Alice', 21])
Items (pairs)	dict.items()	dict_items([('name', 'Alice'), ...])
Length	len(dict)	2 (number of key-value pairs)
Clear all items	dict.clear()	{}
Check key	'key' in dict	'age' in student → True
Copy dictionary	dict.copy()	Makes a shallow copy
Remove with return	dict.pop(key)	student.pop("age") → returns value
Set default	dict.setdefault(key, default)	Adds key if not present
Update dictionary	dict.update({key: value})	Adds or modifies multiple keys

Example:

```
employee = {"id": 101, "name": "John", "salary": 50000}

# Access
print(employee["name"])      # John

# Add
employee["department"] = "HR"

# Update
employee["salary"] = 55000

# Delete
del employee["id"]
# Loop through items
for key, value in employee.items():
    print(key, ":", value)
```

Output:

```
name : John
salary : 55000
department : HR
```

Why Use Dictionaries?

- Fast lookup by key
- Useful for storing structured data (like JSON)
- Great for mapping relationships

7 a

State Python Dictionary with an example

A dictionary in Python is a collection of key-value pairs. It is unordered, mutable, and indexed by keys (not positions).

Creating a Dictionary

```
student = {
```

```
            "name": "Suma",
            "age": 21,
            "course": "Data Science"
        }
```

b

Explain Python Exception handling using 'try' statement

In Python, exception handling is used to catch and handle runtime errors gracefully, without crashing the program.

Basic Structure of try-except:

```
try:
   # Code that might raise an error
except SomeError:
   # Code to handle the error
```

Full Syntax Example:

```
try:
   num = int(input("Enter a number: "))
   result = 10 / num
   print("Result:", result)
except ZeroDivisionError:
   print("You can't divide by zero!")
except ValueError:
   print("Invalid input! Please enter a number.")
```

Output (example):

```
Enter a number: 0
You can't divide by zero!
```

Optional Blocks:

Block	Purpose
try	Code that may cause an exception
except	Catches and handles the exception
else	Executes if no exception occurs
finally	Executes no matter what (used for clean-up)

Example with else and finally:

```
try:
   n = int(input("Enter a number: "))
   print(100 / n)
except ZeroDivisionError:
   print("Cannot divide by zero.")
except ValueError:
   print("Please enter a valid number.")
else:
   print("Division successful.")
finally:
   print("Execution complete.")
```

Why Use Exception Handling?

- Prevents program crashes
- Provides user-friendly error messages
- Useful for file handling, user input, APIs, etc.

c

List different packages you are familiar with. Write a brief note on the determine (of) package in python.

Common Python Packages (Libraries)
Here are several popular Python packages used across various domains:

Category	Package Name	Purpose
Data Analysis	pandas	Data manipulation and analysis
Numerical Computation	numpy	Handling arrays and numerical operations
Visualization	matplotlib, seaborn	Plotting graphs and charts
Machine Learning	scikit-learn	ML algorithms and tools
Deep Learning	tensorflow, keras, torch	Neural networks
Web Development	flask, django	Web app frameworks
APIs & Web Scraping	requests, beautifulsoup4	HTTP and web scraping
File Handling	os, shutil, pathlib	File system operations
Math & Statistics	math, statistics, random	Math functions and stats

8

Compulsory Case study:
a)Develop a python code to solve quadratic equation by importing “sqrt” from “math”. The formula is given below:

Sol 1= $-\boldsymbol{b} + \sqrt{\frac{\boldsymbol{b^2-4ac}}{\boldsymbol{2a}}}$ **; Sol 2=** $\sqrt{\frac{\boldsymbol{b2-4ac}}{\boldsymbol{2a}}}$

```
from math import sqrt

# Input coefficients
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))

# Calculate the discriminant
d = b**2 - 4*a*c

# Check if real solutions exist
if d >= 0:
    sol1 = (-b + sqrt(d)) / (2 * a)
    sol2 = (-b - sqrt(d)) / (2 * a)
    print("Solution 1:", sol1)
    print("Solution 2:", sol2)
else:
    print("No real solutions (discriminant is negative)")
```

b) **Develop a python code to find the area of a triangle given three sides ‘a’. ‘b’ and ‘c’ as per the following formulae.(Heron’s formula)**

$$S = \frac{a+b+c}{2}$$
$$Area = s(s-a)(s-b)(s-c)$$

```
from math import sqrt

# Input sides of triangle
a = float(input("Enter side a: "))
b = float(input("Enter side b: "))
```

```
c = float(input("Enter side c: "))

# Calculate semi-perimeter
s = (a + b + c) / 2

# Calculate area using Heron's formula
area = sqrt(s * (s - a) * (s - b) * (s - c))
print("Area of the triangle:", area)
```