CMR INSTITUTE OF TECHNOLOGY, BENGALURU	USN										

Internal Assesment Test - I

Sub:	Introduction to Python, Data and Control Systems							Code:	MBABA313
Date:	05-12-2025	Duration:	90 mins	Max Marks:	50	Sem:	III	Branch:	MBA

SET- III

		Marks	OBE CO	OBE RBT
	Part A - Answer Any Two Full Questions (2* 20 = 40 marks)			

1 (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.
		[03]	CO1	L1

(b)	**Write a Python program to check whether the user-given number is odd or even.**
	# Program to check if a number is odd or even
	
	num = int(input("Enter a number: "))
	if num % 2 == 0:
	print(num, "is Even")
	else:
	print(num, "is Odd")
	
	Enter a number: 2
	2 is an Even number.
	
	Enter a number: 5
	5 is an Odd number.
		[07]	CO1	L3

(c)	**Apply the concepts of conditional, alternate and chained executions for grading student performance based on marks.**
	Below is a Python program that uses **conditional**, **alternate**, and **chained executions** to grade student performance based on marks.
	
	# Program to grade student performance using conditional, alternate, and chained execution
	
	marks = float(input("Enter the student's marks (0–100): "))
	
	# Conditional execution
	if marks < 0 or marks > 100:
	  print("Invalid marks! Please enter a value between 0 and 100.")
	
	# Alternate execution
	elif marks >= 90:
	  print("Grade: A+ (Excellent)")
	
	# Chained execution (elif)
	elif marks >= 80:
	  print("Grade: A (Very Good)")
	elif marks >= 70:
	  print("Grade: B (Good)")
	elif marks >= 60:
	  print("Grade: C (Average)")
	elif marks >= 50:
	  print("Grade: D (Pass)")
	else:
	  print("Grade: F (Fail)")
		[10]	CO1	L3

2 (a)	**State the working of a Python 'break' statement.**
	The **break statement in Python** is used to **immediately stop a loop** (either for or while) and exit from it, even if the loop condition is still true.
	What break Does
	• Stops the loop instantly.
	• Control moves to the first statement **after** the loop.
	Example 1: Using break in a for loop
	for i in range(1, 10):
	  if i == 5:
	  break
	  print(i)
	Output:
	1
	2
	3
	4
		[03]	CO1	L1

(b)	**Illustrate the concept of operator precedence in Python.** Operator precedence in Python determines **which operators are evaluated first** in an expression. Just like in mathematics (where multiplication happens before addition), Python also follows a hierarchy. **Operator Precedence (Highest to Lowest)** Here is a simplified hierarchy: 1. **Parentheses** → ()	[07]	CO1	L3

2. **Exponents** → **
3. **Unary operators** → +x, -x, ~x
4. **Multiplication / Division / Floor division / Modulus** → *, /, //, %
5. **Addition / Subtraction** → +, -
6. **Relational operators** → <, <=, >, >=
7. **Equality operators** → ==, !=
8. **Logical NOT** → not
9. **Logical AND** → and
10. **Logical OR** → or

Example of Operator Precedence

```python
result = 3 + 5 * 2  # Multiplication (*) happens before addition (+)
print(result)  # Output: 13

result = (3 + 5) * 2  # Parentheses override precedence
print(result)  # Output: 16

result = 10 > 5 and 3 < 8  # Comparison happens before 'and'
print(result)  # Output: True
```

(c) **Outline the importance of operators in python with examples.** [10] CO1 L4

Operators are special symbols that perform operations on variables and values. Python has the following types of operators:

1. Arithmetic Operators

Used for mathematical operations.

Operator	Meaning	Example
+	Addition	5 + 3 = 8
-	Subtraction	9 - 4 = 5
*	Multiplication	6 * 2 = 12
/	Division	8 / 2 = 4.0
%	Modulus (remainder)	10 % 3 = 1
**	Exponent (power)	2 ** 3 = 8
//	Floor division	7 // 2 = 3

✔ Example:

```
a = 10
b = 3

print(a + b)  # 13
print(a - b)  # 7
print(a * b)  # 30
print(a / b)  # 3.3333
print(a % b)  # 1
print(a ** b) # 1000
print(a // b) # 3
```

2. Comparison (Relational) Operators

Used to compare values. Result is either **True** or **False**.

Operator	Meaning	Example

	== Equal to 5 == 5 → True != Not equal to 5 != 3 → True > Greater than 6 > 2 → True < Less than 3 < 7 → True >= Greater or equal 6 >= 6 → True <= Less or equal 4 <= 5 →True **3. Logical Operators** Used to combine conditions. **Operator Meaning Example** and True if both conditions are True (5 > 3 and 8 > 6) → True or True if at least one condition is True (5 > 10 or 3 < 8) → True not Reverses the condition not(5 > 3) → False **4. Assignment Operators** Used to assign and update values. **Operator Meaning Example** = Assign x = 10 += Add and assign x += 5 (x = x + 5) -= Subtract and assign x -= 3 *= Multiply and assign x *= 2 /= Divide and assign x /= 2 %= Modulus and assign x %= 3 **= Power and assign x **= 2 //= Floor divide and assign x //= 4 **5. Bitwise Operators** Work on **bits** (0s and 1s). **Operator Meaning** & AND ` ` ^ XOR ~ NOT << Left shift >> Right shift **6. Membership Operators** Used to check membership in sequences (list, string, etc.) **Operator Meaning** in True if value present not in True if value not present				
3 (a)	**Difference between data cleaning and data analysis?** **Feature** | **Data Cleaning** | **Data Analysis** **Meaning** | Process of fixing or removing incorrect, incomplete, or inconsistent data. | Process of examining cleaned data to find patterns, insights, and conclusions. **Purpose** | Improve data quality so analysis becomes accurate. | Make decisions, answer questions, and find meaningful insights.		[03]	CO2	L1

	Focus	Removing errors.	Understanding and interpreting data.			
	Activities involved	Handling missing values, removing duplicates, correcting errors, formatting data.	Applying statistical methods, visualizations, trends, predictions, and reports.			
	Tools used	Excel, Python (Pandas), Power BI, SQL.	Excel, Python, Power BI, Tableau, R.			
	Output	Clean, structured, error-free dataset.	Charts, insights, reports, predictions.			
	When performed?	Before data analysis.	After data cleaning.			
	Example	Removing rows with missing age, fixing spelling errors ("Bananaa" → "Banana").	Finding which fruit is sold the most each month.			

(b) **What is a python library? Outline the importance of Pandas, NumPy and Matplotlib.** [07] CO2 L4

A **Python library is a set of ready-made tools that help you do specific tasks easily.**

Example: Instead of writing your own code for math calculations or data analysis, you can use a library.

1. NumPy (Numerical Python)

Importance: Foundation for scientific computing

NumPy is used for:

- Fast mathematical and numerical operations
- Handling large multi-dimensional arrays
- Performing matrix operations
- Serving as a base for many other libraries (Pandas, SciPy, Scikit-learn)

Why it is important?

- Much faster than Python lists
- Supports vectorization (performing operations on entire arrays at once)
- Provides built-in mathematical functions
- Essential for machine learning calculations

2. Pandas

Importance: Data cleaning, manipulation, and analysis

Pandas is used for:

- Loading and handling datasets (CSV, Excel, SQL, etc.)
- Cleaning data (handling missing values, duplicates, etc.)
- Data analysis using DataFrame
- Filtering, grouping, merging datasets

Why it is important?

- Makes raw data usable
- Provides DataFrame, which is like an Excel sheet in Python
- Most important tool for data preprocessing in ML and analytics
- Very easy to use compared to NumPy arrays

3. Matplotlib

Importance: Data Visualization

Matplotlib is used for:

- Creating charts and graphs
- Visualizing patterns and trends
- Understanding data before analysis

	• Creating bar graphs, line charts, scatter plots, histograms, etc. Why it is important? • Helps convert numbers into meaningful visuals • Useful in presentations and reports • Works well with NumPy and Pandas • Basis for advanced visualization libraries like Seaborn			
(c)	**Illustrate the concept of data visualization in python?** In today's world, a lot of data is being generated on a daily basis. And sometimes to analyze this data for certain trends, patterns may become difficult if the data is in its raw format. To overcome this data visualization comes into play. Data visualization provides a good, organized pictorial representation of the data which makes it easier to understand, observe, analyze. In this tutorial, we will discuss how to visualize data using Python. Python provides various libraries that come with different features for visualizing data. All these libraries come with different features and can support various types of graphs. In this tutorial, we will be discussing four such libraries. • Matplotlib • Seaborn • Bokeh • Plotly ***Matplotlib*** Matplotlib is an easy-to-use, low-level data visualization library that is built on NumPy arrays. It consists of various plots like scatter plot, line plot, histogram, etc. Matplotlib provides a lot of flexibility. ***Seaborn*** **Seaborn** is a high-level interface built on top of the Matplotlib. It provides beautiful design styles and color palettes to make more attractive graphs. ***Bokeh*** Let's move on to the third library of our list. Bokeh is mainly famous for its interactive charts visualization. Bokeh renders its plots using HTML and JavaScript that uses modern web browsers for presenting elegant, concise construction of novel graphics with high-level interactivity. ***Plotly*** This is the last library of our list and you might be wondering why plotly. Here's why - • Plotly has hover tool capabilities that allow us to detect any outliers or anomalies in numerous data points. • It allows more customization. • It makes the graph visually more attractive.	[10]	CO2	L3
	Part B - Compulsory (01*10=10 marks) – CASE STUDY			
4	**Develop a Python program for a calculator that provides a menu with options to add, subtract, multiply, and divide. The program should allow the user to choose an operation, enter two numbers, and display the result. Additionally, the user should have the option to continue using the calculator or exit the program.** **# This function adds two numbers** def add(x, y): return x + y **# This function subtracts two numbers**	[10]	CO2	L4

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

# This function multiplies two numbers
def multiply(x, y):
    return x * y

# This function divides two numbers
def divide(x, y):
    return x / y
print("Select operation.")
print("1.Add")
print("2.Subtract")
print("3.Multiply")
print("4.Divide")

while True:
    # take input from the user
    choice = input("Enter choice(1/2/3/4): ")

    # check if choice is one of the four options
    if choice in ('1', '2', '3', '4'):
        try:
            num1 = float(input("Enter first number: "))
            num2 = float(input("Enter second number: "))
        except ValueError:
            print("Invalid input. Please enter a number.")
            continue

        if choice == '1':
            print(num1, "+", num2, "=", add(num1, num2))

        elif choice == '2':
            print(num1, "-", num2, "=", subtract(num1, num2))

        elif choice == '3':
            print(num1, "*", num2, "=", multiply(num1, num2))

        elif choice == '4':
            print(num1, "/", num2, "=", divide(num1, num2))

        # check if user wants another calculation
        # break the while loop if answer is no
        next_calculation = input("Let's do next calculation? (yes/no): ")
        if next_calculation == "no":
          break
    else:
        print("Invalid Input")
```

*****