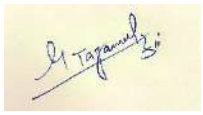
Department of BCA

Details of Lesson Plan

S.No.	Particulars	Details
1.	Course Name	Python Programming
2.	Course Code	BCAMJ-301
3.	Academic Year	2024-2025
4.	Semester	3rd
5.	Number of Lesson plans	45
6.	Faculty Assigned	Tajamul Hassan



Faculty Signature

Lesson Plan No. 1.1	Course Name: Python Programming Topic: Introduction to python Programming	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand what Python is and its applications. b. Recognize the features that make Python popular. c. Write and execute basic Python programs.
Teaching Aids (if any)	a. PowerPoint presentation b. Live demonstrations on a shared screen c. Handouts with key concepts
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you know about Python as a programming language? • Can you name any applications or tools that use Python? b. Overview of Python: • Define Python as a high-level, interpreted programming language. • Discuss its creation by Guido van Rossum in 1991 and its growth in popularity. 2. **Development** (30 minutes) a. Applications of Python (10 minutes): • Web Development: Frameworks like Django and Flask. • Data Science: Libraries such as Pandas, NumPy, and Matplotlib. • Machine Learning: Frameworks like TensorFlow and scikit-learn. • Automation: Scripting tasks and automating workflows. • Game Development: Using libraries like Pygame. b. Key Features of Python (10 minutes): • Easy to Learn and Use: Readable syntax and simplicity. • Interpreted Language: No need for compilation; code runs directly. • Dynamically Typed: Variable types are determined at runtime. • Extensive Libraries: Rich ecosystem of libraries and frameworks. • Community Support: Strong community and plenty of resources available. c. Writing Your First Python Program (10 minutes): • Live Demonstration: 1. Open a Python environment (IDLE, Jupyter Notebook, or a text editor). 2. Write a simple program:

	print("Welcome to Python Programming!") 3. Run the program to show the output. 3. Exercise (5 minutes) – • Ask students to write a similar program that prints their name and a fun fact about themselves. • Encourage them to experiment with different print statements. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize Key Points: • Overview of Python's versatility and ease of use. • Importance of Python in various fields. • Review of the first Python program written. 2. Suggested Readings: • BOOK 1: *Automate the Boring Stuff with Python* by Al Sweigart: Chapter 1 (Python Basics). • BOOK 2: *Python Crash Course* by Eric Matthes: Chapter 2 (Variables and Simple Data Types). Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What excites you the most about learning Python? • How do you see yourself using Python in the future? 2. Homework: • Explore Python's official website and find two interesting projects or libraries you would like to learn more about. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.2	Course Name: Python Programming Topic: Installation of python	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the prerequisites for installing Python. b. Successfully install Python on their operating system. c. Verify the installation and run a simple Python script.
Teaching Aids (if any)	a. PowerPoint presentation b. Live demonstrations on a shared screen c. Installation guides (handouts)
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • Why do you think installation is an important first step in programming? • Have you installed any programming languages before? What was your experience? b. Overview of Python: • Briefly discuss Python's popularity and its applications in various fields. 2. **Development** (30 minutes) a. Prerequisites for Installation (5 minutes): • Discuss system requirements (Windows, macOS, Linux). • Explain the importance of having administrative privileges for installation. b. Installing Python (15 minutes): • For Windows: 1. Go to the official Python website: https://www.python.org/ 2. Download the latest version of Python. 3. Run the installer and check "Add Python to PATH." 4. Complete the installation process. • For macOS: 1. Open the Terminal. 2. Install Homebrew (if not already installed): https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)" 3. Install Python using Homebrew: brew install python • For Linux: 1. Open the terminal. 2. Use the package manager to install Python (e.g., for Ubuntu): sudo apt update sudo apt install python3 c. Verifying the Installation (5 minutes): • Show how to check the Python version in the terminal or command prompt:

	python --version # or python3 --version • Explain the difference between python and python3 on some systems. d. Running a Simple Python Script (5 minutes): • Demonstrate creating a simple Python script: 1. Open a text editor or IDE (e.g., IDLE, VSCode). 2. Write a simple script: print("Hello, World!") 3. Save the file as hello.py. 4. Run the script from the terminal or command prompt: python hello.py # or python3 hello.py 3. Exercise (5 minutes) – • Ask students to install Python on their machines and run the simple script shown in the demonstration. • Encourage them to troubleshoot any issues and ask for help if needed. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize Key Points: • Importance of installing Python and verifying the installation. • Brief overview of writing and executing a simple script. 2. Suggested Readings: • BOOK 1: *Automate the Boring Stuff with Python* by Al Sweigart: Chapter 1 (Introduction to Python). • BOOK 2: *Python Crash Course* by Eric Matthes: Chapter 1 (Getting Started) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What challenges did you face during the installation process? • How do you plan to use Python in your projects? 2. Homework: • Explore Python's official documentation and write a brief summary of any two features that interest you. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.3	Course Name: Python Programming Topic: Naming Conventions	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the importance of naming conventions in Python. b. Identify and apply different naming conventions for variables, functions, and classes. c. Recognize the impact of naming conventions on code readability and maintainability.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen

Teaching Development	**1. Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• Why do you think naming conventions are important in programming?
	• Can you think of a situation where a good name made code easier to understand?
	b. Introduction to Naming Conventions:
	• Define naming conventions and their role in improving code clarity and collaboration.
	
	2. Development (30 minutes)
	a. Importance of Naming Conventions (10 minutes):
	• Discuss how consistent naming improves code readability and helps others understand your code.
	• Explain the role of naming conventions in collaborative projects and code maintenance.
	b. Common Naming Conventions (20 minutes):
	• Variable and Function Names:
	• Use lowercase letters.
	• Separate words with underscores (snake_case).
	• Example:
	def calculate_area(radius):
	area = 3.14 * radius * radius
	return area
	• Class Names:
	• Use capitalized words without spaces (CamelCase).
	• Example:
	class Circle:
	def __init__(self, radius):
	self.radius = radius
	• Constants:
	• Use uppercase letters with underscores to separate words (UPPER_SNAKE_CASE).

	• Example: • MAX_CONNECTIONS = 10 4. Private Variables: • Prefix with an underscore to indicate that they are intended for internal use. • Example: _ internal_counter = 0 5. Avoiding Reserved Words**:** • Discuss the importance of avoiding Python reserved keywords (e.g., if, else, for, class). 3. Exercise (5 minutes) – • Ask students to create a small program that: • Defines a class with a name following CamelCase. • Includes methods with names in snake_case. • Demonstrates the use of a constant and a private variable. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize Key Points: • Overview of naming conventions and their benefits for code clarity and collaboration. • Importance of following consistent naming practices in Python. 2. Suggested Readings from Textbooks: • BOOK 1: *Think Python* by Allen B. Downey: Chapter 6: "Functions and Variables" (pp. 100-120) • BOOK 2: *Learning Python* by Mark Lutz: Chapter 9: "Modules and Packages" (pp. 220-240) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • Why are naming conventions important for code readability? • What are some examples of naming conventions used for different types of identifiers? 2. Homework: • Write a Python script that defines a class and at least two functions, applying proper naming conventions throughout. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.4	Course Name: Python Programming Topic: Identifiers and indentations	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Define and use valid identifiers in Python. b. Understand the importance of indentation in Python syntax. c. Apply correct indentation practices in code.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen.
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you think an identifier is in programming? • Why do you think indentation might be important in Python? b. Introduction to Identifiers and Indentation: • Define identifiers and their role in programming. • Briefly explain how indentation is used in Python to define code blocks. 2. **Development** (30 minutes) a. Identifiers (15 minutes): • Definition: Identifiers are names used to identify variables, functions, classes, and other objects in Python. • Rules for Identifiers: o Must start with a letter (a-z, A-Z) or an underscore (_). o Can contain letters, numbers (0-9), and underscores. o Cannot start with a number. o Cannot be a keyword (e.g., if, else, for, etc.). • Examples: valid_variable = 10 private_var = "Hello" number1 = 25 • Invalid Examples: 1st_variable = 5 # Invalid: starts with a number my-variable = 10 # Invalid: uses hyphen b. Indentation (15 minutes): • Definition: Indentation refers to the spaces at the beginning of a line of code that indicate a block of code. • Importance: In Python, indentation is used to define the scope of loops, functions, and conditionals. • Example of Correct Indentation: if True: print("This is indented correctly.") • Example of Incorrect Indentation: if True: print("This will cause an IndentationError.") # Incorrect • Best Practices: o Use 4 spaces for indentation (recommended by PEP 8). o Be consistent with your indentation style (spaces vs. tabs).

	3. Exercise (5 minutes) –
	• Ask students to write a small program that includes:
	• At least two valid identifiers.
	• An if statement that demonstrates correct indentation.
	Example:
	`age = 18`
	`if age >= 18:`
	`  print("You are an adult.")`
	`else:`
	`  print("You are a minor.")`
	
	Use Nearpod to collect responses and discuss the answers.

Closure	1. Summarize Key Points: o Overview of valid identifiers and their rules. o Importance of indentation for defining code blocks in Python. 2. Suggested Readings from Textbooks: o BOOK 1: *Think Python* by Allen B. Downey: Chapter 3: "Identifiers and Indentation" (pp. 40-50) o BOOK 2: *Learning Python* by Mark Lutz: Chapter 5: "Control Flow" (pp. 111-130) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	**1.** Reflective Questions: • What are the rules for creating valid identifiers in Python? • How does indentation affect the structure and readability of Python code? **2.** Homework: • Write a Python script that defines three identifiers and includes at least one function with proper indentation. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.5	Course Name: Python Programming Topic: Data Type	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Identify and describe different data types in Python. b. Create and manipulate variables of various data types. c. Understand type conversion and its importance.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen

Teaching Development	1. Introduction (5 minutes)
	a. Pre-Discussion Questions:
	• What do you think data types are in programming?
	• Why do you think different types of data are important?
	b. Introduction to Data Types:
	• Define data types and explain their role in storing and manipulating data in Python.
	
	2. **Development** (30 minutes)
	a. Basic Data Types (15 minutes):
	• Discuss and demonstrate the following data types:
	o Integers: Whole numbers (e.g., x = 10)
	o Floats: Decimal numbers (e.g., y = 3.14)
	o Strings: Text (e.g., name = "Alice")
	o Booleans: True or False values (e.g., is_active = True)
	• Live coding example:
	num = 42 # Integer
	pi = 3.14 # Float
	message = "Hello, World!" # String
	is_valid = False # Boolean
	print(num, pi, message, is_valid)
	b. Composite Data Types (10 minutes):
	• Introduce lists and dictionaries:
	o Lists: Ordered collections (e.g., my_list = [1, 2, 3])
	o Dictionaries: Key-value pairs (e.g., my_dict = {"name": "Alice", "age": 30})
	• Live coding example:
	my_list = [1, 2, 3, "Python"]
	my_dict = {"name": "Alice", "age": 30}
	print(my_list)
	print(my_dict["name"])
	c. Type Conversion (5 minutes):
	• Explain type conversion and why it's necessary (e.g., converting strings to integers).
	• Live coding example:
	age_str = "25"

	age_int = int(age_str) print(age_int, type(age_int)) # Output: 25 <class 'int'> 3. Exercise (5 minutes) – • Ask students to create a program that: • Initializes variables of at least three different data types. • Demonstrates type conversion by converting one variable type to another. • Prints the results along with their types. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize Key Points: • Overview of basic and composite data types in Python. • Importance of type conversion in programming. 2. Suggested Readings from Textbooks: • BOOK 1: *Think Python* by Allen B. Downey: Chapter 4: "Data Types" (pp. 60-80) • BOOK 2: *Learning Python* by Mark Lutz: Chapter 4: "Using Python" (pp. 91-110) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the different data types in Python, and how are they used? • Why might you need to convert data types in a program? 2. Homework: • Write a Python script that takes user input for two numbers (as strings), converts them to integers, adds them, and prints the result. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.6	Course Name: Python programming Topic: Variables	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and purpose of variables in Python. b. Define and initialize variables using different data types. c. Utilize variables in expressions and understand scope.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen

Teaching Development	1. **Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• What do you think a variable is in programming?
	• Can you give examples of where you might use variables?
	b. Introduction to Variables:
	• Define a variable and explain its role in storing data.
	• Discuss the concept of identifiers and the rules for naming variables in Python.
	2. **Development** (30 minutes)
	a. Creating and Initializing Variables (10 minutes):
	• Demonstrate how to create and assign values to variables:
	x = 10 # Integer
	name = "Alice" # String
	price = 19.99 # Float
	is_active = True # Boolean
	• Discuss dynamic typing in Python and how variable types can change:
	x = 10
	x = "Now I'm a string"
	b. Variable Types and Operations (10 minutes):
	• Explain basic data types in Python:
	o Integers, floats, strings, booleans
	• Show how to perform operations using variables:
	a = 5
	b = 10
	sum_result = a + b
	print(sum_result) # Output: 15
	• Demonstrate string concatenation and arithmetic operations:
	greeting = "Hello, " + name
	print(greeting) # Output: Hello, Alice
	c. Variable Scope (10 minutes):
	• Explain the concept of variable scope:
	o Local vs. global variables
	• Show examples:
	global_var = "I'm global!"

	def my_function(): local_var = "I'm local!" print(local_var) my_function() print(global_var) 3. Exercise (5 minutes) – • Pair students and ask them to write a program that: • Initializes three different variables of different types. • Prints each variable and its type using the type() function. Discuss their solutions and clarify any misunderstandings
Closure	1. Summarize Key Points: • Overview of variables, their purpose, and how to create them. • Importance of understanding variable types and scope. 2. Suggested Readings from Textbooks: • BOOK 1: *Think Python* by Allen B. Downey: Chapter 2: "Variables" (pp. 20-30) • BOOK 2: *Learning Python* by Mark Lutz: Chapter 3: "Introducing Python" (pp. 75-90) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What is a variable, and why is it important in programming? • How does variable scope affect the visibility of a variable? 2. Homework: • Write a Python script that takes user input for a name and age, and then prints a message that includes both variables. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.7	Course Name: Python Programming Topic: Strings in Python	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and characteristics of strings in Python. b. Perform basic operations on strings, including creation, indexing, slicing, and modification. c. Utilize built-in string methods effectively for various string manipulations.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen

Teaching Development	1. **Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• What is a string in programming?
	• Can you give examples of where you might use strings?
	b. Introduction to Strings:
	• Define strings and explain their purpose in Python as a sequence of characters.
	• Discuss characteristics: immutable, can be single or double-quoted, support escape sequences.
	
	2. **Development** (30 minutes)
	a. Creating and Accessing Strings (10 minutes):
	• Show how to create strings:
	string1 = "Hello, World!"
	string2 = 'Python Programming'
	• Explain indexing:
	print(string1[0]) # Output: H
	print(string1[-1]) # Output: !
	• Discuss the concept of string length:
	print(len(string1)) # Output: 13
	b. String Slicing and Concatenation (10 minutes):
	• Demonstrate slicing:
	print(string1[0:5]) # Output: Hello
	print(string1[7:]) # Output: World!
	• Explain string concatenation:
	combined = string1 + " " + string2
	print(combined) # Output: Hello, World! Python Programming
	c. Built-in String Methods (10 minutes):
	• Introduce common string methods:
	○ lower(), upper(), strip(), replace(), split(), and join():
	print(string2.lower()) # Output: python programming
	print(string2.upper()) # Output: PYTHON PROGRAMMING
	print(string1.replace("World", "Python")) # Output: Hello, Python!
	words = string2.split() # Output: ['Python', 'Programming']

	3. Exercise (5 minutes) – • Pair students and ask them to write a program that: • Takes a user-input string. • Counts the number of vowels in the string. • Replaces spaces with underscores and prints the modified string. Discuss their solutions and clarify any misunderstandings.
Closure	1. Summarize Key Points: • Overview of strings, their creation, and basic operations. • Importance of built-in string methods for manipulation. 2. Suggested Readings from Textbooks: • BOOK 1: *Think Python* by Allen B. Downey: Chapter 8: "Strings" (pp. 135-145) • BOOK 2: *Learning Python* by Mark Lutz: Chapter 6: "Strings" (pp. 215-230) Spend 5 minutes to wrap up and consolidate the learnings.
Evaluation	1. Reflective Questions: • What are the main characteristics of strings in Python? • How do you perform string concatenation and slicing? 2. Homework: • Write a Python script that takes a sentence as input, reverses it, and prints the result. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.8	Course Name: Python Programming Topic: Slicing	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of slicing in Python and its syntax. b. Perform slicing operations on various data structures, including strings, lists, and tuples. c. Utilize slicing for practical tasks, such as extracting sublists or substrings.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you think slicing means in the context of programming? • Have you ever needed to extract a portion of a string or list? b. Introduction to Slicing: • Define slicing and explain its importance in manipulating sequences in Python. • Introduce the general syntax for slicing: sequence[start:end:step]. 2. **Development** (30 minutes) a. Basic Slicing Syntax (10 minutes): • Explain each part of the slicing syntax: o start: starting index (inclusive) o end: ending index (exclusive) o step: increments between each index • Provide examples using a string: text = "Hello, World!" print(text[0:5]) # Output: Hello print(text[7:]) # Output: World! print(text[:5]) # Output: Hello print(text[::2]) # Output: Hlo ol! **b. Slicing Lists and Tuples (10 minutes):** • Demonstrate slicing on lists: numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9] print(numbers[2:5]) # Output: [3, 4, 5] print(numbers[:-3]) # Output: [1, 2, 3, 4, 5, 6] print(numbers[::3]) # Output: [1, 4, 7] • Show how slicing works with tuples: my_tuple = (10, 20, 30, 40, 50) print(my_tuple[1:4]) # Output: (20, 30, 40) **c. Practical Applications of Slicing (10 minutes):** • Discuss common use cases for slicing, such as: o Extracting substrings or sublists o Reversing a sequence:

	print(text[::-1]) # Output: !dlroW ,olleH • Hands-on Activity: Ask students to practice slicing on strings and lists. Provide a few examples for them to solve. 3. Exercise (5 minutes) – • Pair students and ask them to write a program that: • Creates a list of ten numbers. • Slices the list to extract the first half, the last half, and every second element. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize Key Points: • Overview of slicing syntax and its applications in Python. • Importance of slicing for effective data manipulation. 2. Suggested Readings from Textbooks: • BOOK 1: *Think Python* by Allen B. Downey: Chapter 10: "Lists" (pp. 162-173) • BOOK 2: *Learning Python* by Mark Lutz: Chapter 6: "Lists" (pp. 232-250) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What is the syntax for slicing in Python? • How can slicing be useful when working with strings and lists? 2. Homework: • Write a Python script that takes a sentence as input, slices it to extract the first and last three words, and prints them. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 1.9	Course Name: Python Programming Topic: IDLE in python	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the purpose and features of IDLE as an Integrated Development Environment (IDE) for Python. b. Navigate the IDLE interface and utilize its features for writing and executing Python code. c. Perform basic tasks such as creating, saving, and running Python scripts within IDLE.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Live demonstrations on a shared screen
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you use for writing and executing Python code? • Have you heard of IDLE? What do you think it is? b. Introduction to IDLE: • Define IDLE (Integrated Development and Learning Environment) and its purpose for Python programming. • Briefly discuss its history and why it's a good choice for beginners. 2. **Development** (30 minutes) a. Navigating the IDLE Interface (10 minutes): • Launch IDLE and show the main components: o Shell window o Editor window • Explain the purpose of each component. b. Writing and Running Python Code (10 minutes): • Demonstrate how to write code in the editor: print ("Hello, World!") • Show how to save a script: o File > Save As... • Explain how to run the script: o Run > Run Module (or F5) c. Utilizing IDLE Features (10 minutes): • Discuss features such as: o Syntax highlighting o Autocompletion o Debugging tools (basic) • Demonstrate how to use the interactive shell for quick tests: x = 5 y = 10 print(x + y)

	3. Exercise (5 minutes) – • Pair students and ask them to: • Create a new Python file in IDLE. • Write a program that calculates the area of a rectangle given width and height. • Save and run the program. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize Key Points: • Overview of IDLE and its features. • Importance of IDLE for beginners in Python programming. 2. Suggested Readings from Textbooks: • BOOK 1: *Think Python* by Allen B. Downey: Chapter 1: "The Beginning" (pp. 1-15) • BOOK 2: *Learning Python* by Mark Lutz: Chapter 1: "Introduction" (pp. 1-10) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the key features of IDLE that make it useful for Python programming? • How do you create and run a Python script in IDLE? 2. Homework: • Write a Python script in IDLE that takes user input for a number and prints whether it is even or odd. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.1	Course Name: Python Programming Topic: Control Statement	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of control statements in Python. b. Use if, elif, and else statements effectively c. Explain the importance of control statements in programming.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	**1. Introduction (5 minutes)** **a.** Pre-Discussion Questions: • What are control statements? • Why do we need control statements in programming? **b.** Introduction to Control Statements: • Explain the concept of control statements. • Discuss the benefits of using control statements (e.g., decision-making, flow control). • Link to introductory video: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ **2. Development (30 minutes)** a. Using if, elif, and else Statements (15 minutes): • Explain the syntax and structure of if, elif, and else statements. • Example: Write a simple program to check if a number is positive, negative, or zero. • Hands-on: Ask students to write a program that checks whether a student has passed based on their score. • Reference video: https://www.youtube.com/watch?v=0sF0B1q4Uq8 b. Importance of Control Statements (15 minutes): • Explain how control statements help in making decisions in code. • Discuss real-world applications and case studies where control statements are crucial. • Highlight the importance of readability and maintainability of code. 3. Exercise (5 minutes) – • Activity: • Pair students and ask them to create a program that determines whether a given year is a leap year. • Discuss their answers and clarify any misconceptions Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points:

	• Definition and importance of control statements. • Syntax and usage of if, elif, and else statements. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 4: "Conditionals" (pp. 40-55) • BOOK 2: Learning Python by Mark Lutz: Chapter 4: "Conditionals" (pp. 112-125) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are control statements, and why are they important? 2. Homework: • Write a short note on how control statements enhance the functionality of programs Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.2	Course Name: Python Programming Topic: Operators	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Identify different types of operators in Python. b. Demonstrate the use of operators in expressions. c. Explain the significance of operators in programming.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. Introduction (5 minutes) a. Pre-Discussion Questions: • What do you understand by operators in programming? • Can you name some operators you have used before? b. Introduction to Operators: • Define operators in Python. • Discuss the importance of operators for performing operations on data. • Link to introductory video: https://www.youtube.com/watch?v=H1f8U3uW8yY **2. Development (30 minutes)** a. Types of Operators (15 minutes): • Arithmetic Operators: Explain +, -, *, /, //, %, **. • Comparison Operators: Explain ==, !=, >, <, >=, <=. • Logical Operators: Explain and, or, not. • Assignment Operators: Explain =, +=, -=, *=, /=. • Identity and Membership Operators: Explain is, is not, in, not in. • Hands-on: Provide examples for each type and ask students to practice basic operations. b. Using Operators in Expressions (15 minutes): • Discuss operator precedence and associativity. • Provide examples illustrating how to use operators in complex expressions. • Hands-on: Ask students to create expressions using different types of operators and evaluate them. • Reference video: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ 3. Exercise (5 minutes) – • Activity: • Ask students to pair up and create a simple calculator program using different operators. • Discuss their approaches and correct any misconceptions. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: ○ Overview of different types of operators in Python. ○ Importance of operators in performing operations on data. 2. Suggested Readings from Textbooks**:**

	• BOOK 1: Think Python by Allen B. Downey: Chapter 2: "Variables, Expressions, and Statements" (pp. 15-26) • BOOK 2: Learning Python by Mark Lutz: Chapter 5: "Operators and Expressions" (pp. 147-164) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the different types of operators in Python, and how do they differ? 2. Homework: • Write a short note on the importance of operator precedence in programming. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.3(a)	Course Name: Python Programming Topic: For Loop	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the syntax and structure of for loops in Python. b. Utilize for loops to iterate over different data structures. c. Apply for loops to solve practical problems.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen

Teaching Development	1. **Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• What do you know about loops in programming?
	• Have you used loops to repeat tasks? If so, how?
	b. Introduction to For Loops:
	• Define for loops and explain their purpose in iterating over sequences.
	• Discuss the benefits of using for loops for repetitive tasks.
	• Link to introductory video: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ
	
	2. **Development** (30 minutes)
	a. Syntax and Structure (10 minutes):
	• For example:
	for item in iterable:
	# code block
	• Discuss the concept of iterables (lists, tuples, strings, etc.).
	• Provide examples of for loops iterating over lists and strings.
	b. Using Range Function (10 minutes):
	• Explain the range() function and how it generates a sequence of numbers.
	• Example:
	python
	for i in range(5):
	print(i)
	• Hands-on: Ask students to write a loop that prints numbers from 1 to 10.
	c. Nested For Loops (10 minutes):
	• Introduce the concept of nested for loops.
	• Example:
	python
	for i in range(3):
	for j in range(2):
	print(i, j)
	• Discuss practical applications, such as working with multi-dimensional data.

	• Hands-on: Ask students to create a simple multiplication table using nested loops. 3. Exercise (5 minutes) – • Activity: • Pair students and ask them to write a program that uses a for loop to count the occurrences of each character in a given string. • Discuss their solutions and clarify any misunderstandings. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of for loops and their syntax. • Importance of iterating over data structures and using the range function. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 5: "Conditionals and Recursion" (pp. 97-110) • BOOK 2: Learning Python by Mark Lutz: Chapter 7: "Iterations" (pp. 197-215) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What is the purpose of a for loop? • How does the range function enhance the utility of for loops? 2. Homework**:** • Write a short program using a for loop to create a list of squares of numbers from 1 to 10 Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.3	Course Name: Python Programming Topic: Break and continue	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the purpose of break and continue statements in loops. b. Use break to exit a loop prematurely. c. Use continue to skip the current iteration of a loop.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen

Teaching Development	• **Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• What do you think happens when a loop needs to be exited early?
	• Have you ever needed to skip certain iterations in a loop?
	• b. Introduction to Break and Continue:
	• Define break and continue statements.
	• Discuss their importance for controlling the flow of loops.
	• Link to introductory video: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ
	• **Development** (30 minutes)
	• Using Break Statement (15 minutes):
	• Explain the syntax and functionality of break:
	for item in iterable:
	if condition:
	break
	• Provide a simple example:
	for i in range(10):
	if i == 5:
	break
	print(i)
	• Discuss practical applications, such as finding a specific item in a list.
	• Hands-on: Ask students to write a loop that exits when they encounter a negative number in a list.
	• Using Continue Statement (15 minutes):
	• Explain the syntax and functionality of continue:
	for item in iterable:
	if condition:
	continue
	• Provide a simple example:
	for i in range(10):
	if i % 2 == 0:
	continue
	print(i)

	• Discuss practical applications, such as skipping unwanted values. • Hands-on: Ask students to create a loop that prints only the odd numbers from 1 to 20. • Exercise (5 minutes) – **Activity:** • Pair students and ask them to write a program that uses both break and continue to process a list of numbers, printing only positive numbers and stopping when they encounter a zero. • Discuss their solutions and clarify any misunderstandings. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of break and continue statements and their usage in loops. • Importance of flow control in programming. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 6: "Lists" (pp. 121-134) • BOOK 2: Learning Python by Mark Lutz: Chapter 8: "Statements" (pp. 231-250) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • How do break and continue alter the flow of a loop? • In what situations might you prefer to use one over the other? 2. Homework: • Write a program that generates numbers from 1 to 50 and uses continue to skip multiples of 5 while using break to stop if the number exceeds 30. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.3	Course Name: Python Programming Topic: While loop	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the syntax and structure of while loops in Python. b. Use while loops to perform repeated tasks based on conditions. c. Identify and avoid common pitfalls, such as infinite loops
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen

Teaching Development	1. **Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• What do you understand by loops in programming?
	• Can you think of a scenario where a task might need to repeat until a condition is met?
	b. Introduction to While Loops**:**
	• Define while loops and their purpose in executing a block of code as long as a condition is true.
	• Discuss the importance of condition-based iteration.
	• Link to introductory video: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ
	
	2. **Development** (30 minutes)
	a. Syntax and Structure (10 minutes):
	• Explain the basic syntax of a while loop:
	python
	while condition:
	# code block
	• Provide a simple example:
	python
	count = 0
	while count < 5:
	print(count)
	count += 1
	• Discuss the importance of updating the loop variable to avoid infinite loops.
	b. Common Use Cases (10 minutes):
	• Discuss scenarios where while loops are particularly useful, such as user input validation.
	• Example: A loop that continues until the user enters a valid response.
	• Hands-on: Ask students to create a while loop that prompts the user for a number until they enter a positive integer.
	c. Avoiding Infinite Loops (10 minutes):
	• Explain what an infinite loop is and how it occurs.

	• Provide examples of bad practices that lead to infinite loops. • Discuss how to debug and handle infinite loops. • Hands-on: Ask students to modify a problematic loop to correct it and avoid infinite execution. 3. Exercise (5 minutes) – • Activity: • Pair students and ask them to write a program that uses a while loop to simulate a simple guessing game, where the user guesses a number until they get it right. • Discuss their solutions and clarify any misunderstandings) Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of while loops and their syntax. • Importance of condition-based iteration and avoiding infinite loops. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 5: "Conditionals and Recursion" (pp. 97-110) • BOOK 2: Learning Python by Mark Lutz: Chapter 7: "Iterations" (pp. 197-215) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What is the main difference between for loops and while loops? • How can you ensure that a while loop does not run indefinitely? 2. Homework: • Write a program using a while loop to print the factorial of a number provided by the user. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.3	Course Name: Python Programming Topic: Looping Statement	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of looping statements in Python. b. Use for and while loops effectively. c. Explain the importance of loops in programming.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you understand by loops in programming? • Why do we use loops in our code? b. Introduction to Looping Statements**:** • Define looping statements and their purpose in programming. • Discuss the importance of loops for repetitive tasks. • Link to introductory video: https://www.youtube.com/watch?v=6iF8Xb7Z3wQ 2. **Development** (30 minutes) a. Using For Loops (15 minutes): • Explain the syntax and structure of for loops. • Example: Iterate over a list of numbers and print each number. • Discuss the use of the range() function with for loops. • Hands-on: Ask students to write a for loop to calculate the sum of numbers from 1 to 10. b. Using While Loops (15 minutes**):** • Explain the syntax and structure of while loops. • Example: Write a program that prints numbers until a condition is met. • Discuss potential pitfalls, such as infinite loops. • Hands-on: Ask students to create a while loop that counts down from 10 to 1. 3. Exercise (5 minutes) – Activity: • Pair students and ask them to create a program that uses both for and while loops to print the first 10 Fibonacci numbers. • Discuss their approaches and clarify any misconceptions. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of for and while loops in Python. • Importance of loops for executing repetitive tasks efficiently. 2. Suggested Readings from Textbooks**:**

	• BOOK 1: Think Python by Allen B. Downey: Chapter 5: "Conditionals and Recursion" (pp. 97-110) • BOOK 2: Learning Python by Mark Lutz: Chapter 7: "Iterations" (pp. 197-215) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the differences between for loops and while loops? • When would you choose to use one type of loop over the other? 2. Homework**:** • Write a short note on the advantages of using loops in programming. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.4	Course Name: Python Programming Topic: Lists	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and structure of lists in Python. b. Perform basic operations on lists (creation, indexing, slicing, and modification). c. Utilize built-in list methods effectively
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you understand by the term "list" in programming? • Have you used lists in other programming languages? If so, how? b. Introduction to Lists: • Define what a list is and explain its purpose in Python. • Discuss the characteristics of lists: ordered, mutable, and allow duplicates. • Link to introductory video: https://www.youtube.com/watch?v=ohCDWZgNIU0 2. **Development** (30 minutes) a. Creating and Accessing Lists (10 minutes): • Explain how to create a list: my_list = [1, 2, 3, 4, 5] • Discuss indexing and how to access elements: print(my_list[0]) # Accessing the first element • Introduce negative indexing. b. Modifying Lists (10 minutes): • Explain how to modify lists (adding, removing, and updating elements): Adding: append(), insert(), extend() Removing: remove(), pop(), clear() • Example of modifying a list: my_list.append(6) my_list.remove(2) • Hands-on: Ask students to create a list and perform various modifications. c. Slicing and List Methods (10 minutes): • Explain how to slice lists to access a range of elements: sub_list = my_list[1:4] # Slicing from index 1 to 3 • Discuss common list methods: sort(), reverse(), count(), and index(). • Hands-on: Ask students to create a list of numbers, sort it, and find the index of a specific number. 3. Exercise (5 minutes) –

	• Activity: • Pair students and ask them to write a program that: • Creates a list of fruits. • Asks the user to add a fruit, remove a fruit, and display the final list. • Discuss their solutions and clarify any misunderstandings. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of lists, their creation, and operations. • Importance of lists in organizing and manipulating collections of data. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 10: "Lists" (pp. 149-160) • BOOK 2: Learning Python by Mark Lutz: Chapter 6: "Lists" (pp. 139-159) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the key characteristics of lists in Python? • How do you access and modify elements in a list? 2. Homework: • Write a program that creates a list of five numbers, calculates their average, and displays the numbers that are above the average. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.5	Course Name: Python Programming Topic: Sets in Python	Course No.: BCAMJ--301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and characteristics of sets in Python. b. Perform basic operations on sets (creation, modification, and operations). c. Utilize built-in set methods effectively
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you understand by the term "set" in programming? • Can you think of scenarios where unique collections of items are useful? b. Introduction to Sets: • Define what a set is and explain its purpose in Python. • Discuss the characteristics of sets: unordered, mutable, and do not allow duplicates. 2. **Development** (30 minutes) a. Creating and Accessing Sets (10 minutes): • Explain how to create a set: my_set = {1, 2, 3, 4, 5} • Discuss the difference between sets and lists. • Show how to access elements indirectly (since sets are unordered). b. Modifying Sets (10 minutes): • Explain how to modify sets (adding and removing elements): ○ Adding: add(), update() ○ Removing: remove(), discard(), pop(), clear() ○ Example of modifying a set: my_set.add(6) my_set.remove(2) • Hands-on: Ask students to create a set and perform various modifications. c. Set Operations (10 minutes): • Discuss common set operations: union, intersection, difference, and symmetric difference: ○ Union: set1 | set2 or set1.union(set2) ○ Intersection: set1 & set2 or set1.intersection(set2) ○ Difference: set1 - set2 or set1.difference(set2) ○ Symmetric Difference: set1 ^ set2 or set1.symmetric_difference(set2) • Hands-on: Ask students to create two sets and perform these operations, displaying the results.

	3. Exercise (5 minutes) – • Activity: • Pair students and ask them to write a program that: • Creates two sets of numbers. • Calculates and displays their union, intersection, and difference. • Discuss their solutions and clarify any misunderstandings. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of sets, their creation, and operations. • Importance of sets in handling unique collections of data. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 10: "Sets" (pp. 161-173) • BOOK 2: Learning Python by Mark Lutz: Chapter 10: "Sets" (pp. 272-287) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the key characteristics of sets in Python? • How do you perform operations on sets? 2. Homework: • Write a program that takes a list of numbers, converts it to a set to remove duplicates, and then displays the unique numbers. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.6	Course Name: Python Programming Topic: Tuples	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and characteristics of tuples in Python. b. Perform basic operations on tuples (creation, indexing, and slicing). c. Utilize built-in tuple methods effectively
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you think a tuple is in programming? • How do you think tuples differ from lists? b. Introduction to Tuples**:** • Define what a tuple is and explain its purpose in Python. • Discuss the characteristics of tuples: ordered, immutable, and allow duplicates. • Link to introductory video: https://www.youtube.com/watch?v=W8KRzmY3U2Q 2. **Development** (30 minutes) a. Creating and Accessing Tuples (10 minutes**):** • Explain how to create a tuple: my_tuple = (1, 2, 3, 4, 5) • Show how to create a tuple with a single element (using a comma). • Discuss indexing and how to access elements: print(my_tuple[0]) # Accessing the first element • Introduce negative indexing for accessing elements from the end b. Modifying Tuples (10 minutes): • Explain the immutability of tuples and how it affects modification. • Discuss how to create a new tuple based on an existing one: new_tuple = my_tuple + (6, 7) • Hands-on: Ask students to create a tuple and demonstrate how to create a new tuple with modified values. c. Tuple Operations and Methods (10 minutes): • Discuss common tuple operations: concatenation, repetition, and slicing: • Example of concatenation: combined_tuple = my_tuple + (8, 9) • Explain built-in methods such as count() and index(): my_tuple.count(2) # Count occurrences of an element my_tuple.index(3) # Find the index of an element • Hands-on: Ask students to create a tuple, use count(), and find the index of a specific element.

	3. Exercise (5 minutes) – • Activity**:** • Pair students and ask them to write a program that: • Creates a tuple of names. • Asks the user to enter a name and checks if it exists in the tuple, returning the index if found. • Discuss their solutions and clarify any misunderstandings. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of tuples, their creation, and operations. • Importance of tuples in handling fixed collections of data. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 10: "Tuples" (pp. 173-183) • BOOK 2: Learning Python by Mark Lutz: Chapter 7: "Tuples" (pp. 188-205) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the key characteristics of tuples in Python? • How do tuples differ from lists, and when might you prefer one over the other? 2. Homework: • Write a program that creates a tuple of numbers, calculates the average, and displays all numbers above the average Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 2.7	Course Name: Python Programming Topic: Dictionaries	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept and characteristics of dictionaries in Python. b. Perform basic operations on dictionaries (creation, accessing, and modifying). c. Utilize built-in dictionary methods effectively.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples on a shared screen
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What do you understand by the term "dictionary" in programming? • Can you think of scenarios where key-value pairs are useful? b. Introduction to Dictionaries: • Define what a dictionary is and explain its purpose in Python. • Discuss the characteristics of dictionaries: unordered, mutable, and stored as key-value pairs. • Link to introductory video: https://www.youtube.com/watch?v=daefaLgNkw0 . 2. **Development** (30 minutes) a. Creating and Accessing Dictionaries (10 minutes): • Explain how to create a dictionary: my_dict = {'name': 'Alice', 'age': 25, 'city': 'New York'} • Show how to access values using keys: print(my_dict['name']) # Accessing the value for the key 'name' • Discuss the importance of keys being unique. b. Modifying Dictionaries (10 minutes**):** • Explain how to modify dictionaries (adding, updating, and removing key-value pairs): • Adding/updating: my_dict['age'] = 26 # Updating value my_dict['gender'] = 'Female' # Adding new key-value pair Removing: del, pop(), clear(). • Example of modifying a dictionary: del my_dict['city'] • Hands-on: Ask students to create a dictionary and perform various modifications. c. Dictionary Methods (10 minutes**):** • Discuss common dictionary methods: ○ keys(), values(), items(): keys = my_dict.keys() values = my_dict.values() items = my_dict.items()

	• Explain how to iterate through keys and values. • Hands-on: Ask students to create a dictionary and use these methods to display keys, values, and key-value pairs. 3. Exercise (5 minutes) – • **Activity:** • Pair students and ask them to write a program that: • Creates a dictionary of products with prices. • Asks the user for a product name and displays its price, handling cases where the product does not exist. • Discuss their solutions and clarify any misunderstandings. Use Nearpod to collect responses and discuss the answers.
Closure	1. Summarize key points: • Overview of dictionaries, their creation, and operations. • Importance of dictionaries in handling structured data with key-value associations. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 11: "Dictionaries" (pp. 183-195) • BOOK 2: Learning Python by Mark Lutz: Chapter 9: "Dictionaries" (pp. 260-275) Spend 5 minutes to wrap up and consolidate the learnings
Evaluation	1. Reflective Questions: • What are the key characteristics of dictionaries in Python? • How do you access and modify elements in a dictionary? 2. Homework: • Write a program that creates a dictionary of student names and their grades, allowing the user to enter a name and receive the corresponding grade. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 3.1	Course Name: Python Programming Topic: Introduction to Functions	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of functions in Python. b. Define their own functions. c. Explain the importance of using functions in programming.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • What is a function? • Why do we use functions in programming? b. Introduction to Functions: • Explain the concept of a function. • Discuss the benefits of using functions (e.g., code reusability, modularity). • https://www.youtube.com/watch?v=9Os0o3wzS **2. Development** (30 minutes) a. Defining Your Own Function (15 minutes): • Syntax of defining a function in Python. • Example: Define a simple function to add two numbers. • Hands-on: Ask students to write their own function to multiply two numbers. • https://www.youtube.com/watch?v=9Os0o3wzS_I b. Importance of Functions (15 minutes): • Explain how functions help in organizing code. • Real-world examples and case studies. • Discuss the concept of code reusability and modularity. • https://www.youtube.com/watch?v=YXPyB4XeYLA **3.** Exercise (5 minutes) o Activity: o Pair students and ask them to define a function to find the maximum of three numbers. o Discuss their answers and correct any misconceptions.
Closure	1. Summarize key points: • Definition and importance of functions. • Syntax of defining a function. 2. Suggested Readings from Textbooks:

	• BOOK 1: Think Python by Allen B. Downey: Chapter 3: "Functions" (pp. 27-39) • BOOK 2: Learning Python by Mark Lutz: Chapter 16: "Functions" (pp. 465-491)
Evaluation	1. Reflective Questions: - What are functions, and why are they important? 2. Homework: - Write a short note on the benefits of using functions in programming

Lesson Plan No. 3.2	Course Name: Python Programming Topic: Parameters and Function Documentation	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of parameters in functions. b. Document functions using docstrings. c. Explain the importance of parameters and documentation in functions.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • What are parameters in a function? • Why is it important to document functions? b. Introduction to Parameters and Documentation: • parameters and their role in functions. • Define Introduce function documentation using docstrings. • YouTubeResource: https://www.youtube.com/watch?v=5L28TMfrQos **2. Development** (30 minutes) a. Parameters in Functions (15 minutes): • Syntax and examples of using parameters. • Example: Define a function with parameters to calculate the area of a rectangle. • Hands-on: Ask students to write a function with parameters to calculate the perimeter of a rectangle. b. Function Documentation (15 minutes): • Explain the importance of documenting functions. • Example: Add docstrings to the previously defined functions. • Hands-on: Ask students to document their perimeter function using docstrings • https://www.youtube.com/watch?v=Q8LhG9Pi5KM **3.** Exercise (5 minutes) c. Activity: • Pair students and ask them to document a function that calculates the square of a number. • Discuss their answers and correct any misconceptions.
Closure	1. Summarize key points: • Parameters in functions. • Importance of function documentation.

	2. Suggested Readings: https://www.youtube.com/watch?v=It-sOke7O64 3. Suggested Readings from Textbooks • Book 1: Think Python by Allen B. Downey: • Chapter 3: "Functions" (pp. 40-42) • Book 2: Learning Python by Mark Lutz: • Chapter 16: "Functions" (pp. 492-497)
Evaluation	1. Reflective Questions: Why are parameters and documentation important in functions? 2. Homework: Write a short note on how to document functions using docstring

Lesson Plan No. 3.3	Course Name: Python Programming Topic: Keyword and Optional Parameters	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand keyword and optional parameters. b. Explain the benefits of using keyword and optional parameters. c. Write functions using keyword and optional parameters.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What are keyword parameters? • What are optional parameters? b. Introduction to Keyword and Optional Parameters: • Define keyword and optional parameters. • Discuss their benefits and use cases. • YouTube Resource: https://www.youtube.com/watch?v=cGvNZQsVn-8 2. **Development** (30 minutes) a. Keyword Parameters (15 minutes): • Syntax and examples of using keyword parameters. • Example: Define a function with keyword parameters to calculate the volume of a box. • Hands-on: Ask students to write a function with keyword parameters to calculate the surface area of a box. • YouTube Resource: https://www.youtube.com/watch?v=JpeD8RbU4_s • b. Optional Parameters (15 minutes): • Syntax and examples of using optional parameters. • Example: Define a function with optional parameters to print a greeting message. • Hands-on: Ask students to write a function with optional parameters to print a custom message. • **YouTube Resource**: https://www.youtube.com/watch?v=JpeD8RbU4_s 3. Exercise (5 minutes)

	• Pair students and ask them to define a function with keyword and optional parameters to calculate the area of a circle (with optional precision). • Discuss their answers and correct any misconceptions
Closure	1. Summarize key points: • Keyword and optional parameters in functions. • Benefits of using these parameters. 2. Suggested Readings: https://www.youtube.com/watch?v=cGvNZQsVn-8 3. Suggested Readings from Textbooks • Book 1: Think Python by Allen B. Downey: ▪ Chapter 3: "Functions" (pp. 42-46) • Book 2: Learning Python by Mark Lutz: Chapter 18: "Advanced Function Topics" (pp. 621-628)
Evaluation	1. Reflective Questions: What are the benefits of using keyword and optional parameters? 2. Homework: Write a short note on the use cases of keyword and optional parameters in functions.

Lesson Plan No. 3.4	Course Name: Python Programming Topic: Passing Collections to a Function	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand how to pass collections to a function. b. Explain the benefits of passing collections to a function. c. Write functions that accept collections as parameters.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • What are collections in Python? • Why would you pass collections to a function? b. Introduction to Passing Collections: • Define collections and their types (lists, tuples, dictionaries, sets). • Discuss the benefits of passing collections to a function. • YouTube Resource: https://www.youtube.com/watch?v=HGHmW-Feo5E 2. **Development** (30 minutes) a. Passing Lists and Tuples (15 minutes): • Syntax and examples of passing lists and tuples to functions. • Example: Define a function to calculate the sum of all elements in a list. • Hands-on: Ask students to write a function to find the maximum element in a tuple. • YouTube Resource: https://www.youtube.com/watch?v=ohCDWZgNIU0 b. Passing Dictionaries and Sets (15 minutes): • Explain how to pass dictionaries and sets to functions. • Example: Define a function to count occurrences of each item in a dictionary. • Hands-on: Ask students to write a function to find the union of two sets. • YouTube Resource: https://www.youtube.com/watch?v=daefaLgNfpc **3.** Exercise (5 minutes)

	Activity: • Pair students and ask them to define a function to merge two lists into one. • Discuss their answers and correct any misconceptions.
Closure	1. Summarize key points: • Passing collections (lists, tuples, dictionaries, and sets) to functions. • Benefits of handling collections in functions. 2. Suggested Readings: • YouTube Resource: https://www.youtube.com/watch?v=ohCDWZgNIU0 3. Suggested Readings from Textbooks • Book 1: Think Python by Allen B. Downey: • Chapter 4: "Functions and Recursion" (pp. 51-63) • Book 2: Learning Python by Mark Lutz: • Chapter 16: "Functions" (pp. 478-488)
Evaluation	1. Reflective Questions: Why is it beneficial to pass collections to functions? How can you handle different types of collections in your functions? 2. Homework: Write a short note on how to handle different types of collections in functions

Lesson Plan No. 3.5	Course Name: Python programming Topic: Variable Number of Arguments	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand how to handle a variable number of arguments in functions. b. Write functions that accept variable numbers of arguments. c. Explain the benefits of using variable arguments in functions.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What are variable number of arguments? • Why might you need to use them in a function? b. Introduction to Variable Number of Arguments: • Define variable number of arguments (using *args and **kwargs). • Discuss scenarios where they are useful. • https://www.youtube.com/watch?v=biK1A0q2nYw **2. Development** (30 minutes) a. *Using args for Variable Arguments (15 minutes): • Explain how to use *args to handle a variable number of positional arguments. • Example: Define a function to calculate the sum of an arbitrary number of arguments. • Hands-on: Ask students to write a function using *args to find the average of numbers. • https://www.youtube.com/watch?v=kDyT6tFk-dA b. **Using kwargs for Keyword Arguments (15 minutes): - Explain how to use **kwargs to handle a variable number of keyword arguments. - Example: Define a function to display user information with optional details. - Hands-on: Ask students to write a function that accepts any number of keyword arguments and returns a dictionary. - https://www.youtube.com/watch?v=RPSFLk9XdVw

	3. Exercise (5 minutes) Activity: • Pair students and ask them to define a function that uses both *args and **kwargs. • Discuss their answers and correct any misconceptions.
Closure	1. Summarize key points: • Using *args for variable positional arguments. • Using **kwargs for variable keyword arguments. 2. Suggested Readings: https://www.youtube.com/watch?v=RPSFLk9XdVw 3. Suggested Readings from Textbooks: • Book 1: Think Python by Allen B. Downey: Chapter 4: "Functions and Recursion" (pp. 64-67) • Book 2: Learning Python by Mark Lutz: Chapter 18: "Advanced Function Concepts" (pp. 620-622)
Evaluation	1. Reflective Questions: What are the benefits of using *args and **kwargs in functions? Can you provide an example where these might be useful? 2. Homework: Write a function that accepts both *args and **kwargs and performs a meaningful operation using both

Lesson Plan No. 3.6	Course Name: Python Programming Topic: Scope and Functions as First-Class Citizens	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of scope in Python functions. b. Explain the idea of functions as first-class citizens in Python. c. Apply the concept of scope and first-class functions in their code.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • What is scope in programming? • How do you think functions are treated in Python? b. Introduction to Scope and Functions as First-Class Citizens: • Define local and global scope in Python. • Explain how functions are first-class citizens in Python. • https://www.youtube.com/watch?v=RBBYmcnp3Nw **2. Development** (30 minutes) a. Scope in Python (15 minutes): • Explain local vs global variables and the nonlocal keyword. • Example: Define functions to demonstrate local and global variables. • Hands-on: Ask students to write a function that modifies a global variable. • https://www.youtube.com/watch?v=JPE-9q5Jq5U b. Functions as First-Class Citizens (15 minutes): • Explain how functions can be passed as arguments, returned from other functions, and assigned to variables. • Example: Define a higher-order function that takes a function as an argument. • Hands-on: Ask students to write a function that returns another function. • https://www.youtube.com/watch?v=JPE-9q5Jq5U **3.** Exercise(5 Minutes): • Pair students and ask them to write a function that returns a function, then use it to calculate different mathematical operations. • Discuss their answers and correct any misconceptions.

Closure	1. Summarize key points: • Scope of variables in Python. • Functions as first-class citizens. 2. Suggested Readings: https://www.youtube.com/watch?v=JPE-9q5Jq5U 3. Suggested Readings from Textbooks • Book 1: Think Python by Allen B. Downey: Chapter 4: "Functions and Recursion" (pp. 64-77) • Book 2: Learning Python by Mark Lutz: Chapter 16: "Functions" (pp. 492-508)
Evaluation	1. Reflective Questions: • How does Python handle variable scope? What does it mean for functions to be first-class citizens? 2. Homework: • Write a short note on the concept of functions as first-class citizens and their scope in Python.

+

Lesson Plan No. 3.7	Course Name: Python Programming Topic: Passing Functions as Arguments and Mapping Functions in a Dictionary	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand how to pass functions as arguments to other functions. b. Map functions in a dictionary. c. Apply these concepts in practical programming scenarios.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • How can you use functions within other functions? • What is a function dictionary? b. Introduction to Passing Functions and Function Mapping: • Define how to pass functions as arguments. • Explain the concept of mapping functions in a dictionary. • https://www.youtube.com/watch?v=SOa7eN99V3U 2. **Development** (30 minutes) a. Passing Functions as Arguments (15 minutes): • Explain how to pass functions as arguments to other functions. • Example: Define a function that takes another function as a parameter and applies it to a list of numbers. • Hands-on: Ask students to create a function that applies different mathematical operations to a list using function arguments. • https://www.youtube.com/watch?v=RT2bC4P4M9E b. Mapping Functions in a Dictionary (15 minutes): • Explain how to use dictionaries to map functions to keys. • Example: Create a dictionary that maps operation names (like 'add', 'subtract') to functions. • Hands-on: Ask students to write a dictionary-based calculator • https://www.youtube.com/watch?v=8f-Jh_sLhT8 3. Exercise(5 Minutes): • Pair students and ask them to map functions for basic arithmetic operations in a dictionary and perform calculations based on user input. • Discuss their answers and correct any misconceptions
Closure	1. Summarize key points:

	• Passing functions as arguments. • Mapping functions in a dictionary 2. Suggested Readings: https://www.youtube.com/watch?v=8f-Jh_sLhT8 3. Suggested Readings from Textbooks • Book 1: Think Python by Allen B. Downey: Chapter 4: "Functions and Recursion" (pp. 78-89) • Book 2: Learning Python by Mark Lutz: Chapter 19: "Advanced Function Topics" (pp. 621-634)
Evaluation	1. Reflective Questions: • How can functions be passed as arguments? What is the benefit of mapping functions in a dictionary? Homework: • Write a short note on how passing functions as arguments and function mapping can be used in programming

Lesson Plan No. 3.8	Course Name: Python Programming Topic: Lambda Functions, Modules, and Standard Modules	Course No.: COM-501

Objectives	At the end of the lesson, the student shall be able to: a. Understand lambda functions and their use cases. b. Learn how to use and create modules in Python. c. Explore standard modules like sys, math, and time
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • What is a lambda function? • How do modules help in organizing Python code? b. Introduction to Lambda Functions and Modules: • Define lambda functions and their syntax. • Explain the role of modules and provide examples of standard modules. • https://www.youtube.com/watch?v=hT7N59j9JVs 2. **Development** (30 minutes) a. Lambda Functions (10 minutes): • Explain the syntax and use cases of lambda functions. • Example: Define a lambda function to sort a list of tuples by the second element. • Hands-on: Ask students to create lambda functions for basic operations. • https://www.youtube.com/watch?v=hT7N59j9JVsperations. b. Modules and Standard Modules (20 minutes): • Explain how to create and use modules in Python. • Example: Create a custom module and import it. • Explore standard modules sys, math, and time: • sys: Handling system-specific parameters and functions. • math: Mathematical functions and constants. • time: Time-related functions. • Hands-on: Ask students to use these standard modules in small coding exercises. • https://www.youtube.com/watch?v=1cY5kRS_6mU Exercise (5 minutes) –

	• Pair students and ask them to create a custom module with a few functions, then use it in a script. • Discuss their answers and correct any misconceptions.
Closure	1. Summarize key points: • Lambda functions and their applications. • Creating and using modules in Python. • Overview of standard modules. 2. Suggested Readings: https://www.youtube.com/watch?v=1cY5kRS_6mU 3. Suggested Readings from Textbooks • Book 1: Think Python by Allen B. Downey: Chapter 7: "Functions" (pp. 109-124) • Book 2: Learning Python by Mark Lutz: Chapter 21: "Modules" (pp. 895-930)
Evaluation	1. Reflective Questions: • What are lambda functions, and when would you use them? How do modules help in organizing Python code? 2. Homework: • Write a short note on lambda functions, creating modules, and the use of standard Python modules.

Lesson Plan No. 4.1	Course Name: Python Programming Topic: Objects and Classes	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of objects and classes in Python. b. Create and utilize objects and classes in Python. c. Explain the principles of object-oriented programming.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	1. **Introduction (5 minutes)** a. Pre-Discussion Questions: • What are objects and classes in programming? • Why is object-oriented programming important? • What are some real-world examples of objects? • Links: • Python Classes and Objects • Introduction to OOP in Python b. Introduction to Objects and Classes: • Define Class: "In Python, a class is a blueprint for creating objects. Classes encapsulate data for the object." • Show a simple "Hello, Class!" example in Python. 2. **Development** (30 minutes) a. History and Evolution of Object-Oriented Programming (10 minutes): • Discuss the origins of OOP and its pioneers like Alan Kay. • Highlight key milestones in the development of OOP. • Video Reference: History of OOP b. Key Features of Classes in Python (10 minutes): • Explain the concept of encapsulation, inheritance, and polymorphism. • Discuss the simplicity and readability of Python's class syntax. • Real-time Example: Demonstrate a simple Python class with attributes and methods. c. Creating and Using Classes (10 minutes): • Describe the process of defining a class and creating instances. • Explain the importance of the __init__ method.

	• Real-time Example: Show examples of creating and using objects in Python. 3. Exercise (5 minutes) • Activity: Pair students and ask them to write a simple Python class that represents a book with attributes like title and author. • Discuss their code and correct any syntax errors.
Closure	1. Summarize key points: • Concept of classes in Python. • Key features and advantages of using classes. • Creating and using objects in Python. 2. Suggested Readings • Book 1: "Think Python" by Allen B. Downey. • Chapter 15: Classes and Objects, pp. 185-204 • Book 2: "Learning Python" by Mark Lutz. • Chapter 25: Object-Oriented Programming, pp. 775-817
Evaluation	1. Reflective Questions: Discuss why using classes can make code more modular and reusable. 2. Homework: Write a short note on the key principles of object-oriented programming and their implementation in Python.

Lesson Plan No. 4.2	Course Name: Python Programming Topic: Principles of Object Orientation	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the core principles of object-oriented programming. b. Apply the principles of encapsulation, inheritance, and polymorphism in Python. c. Explain the benefits of using object-oriented programming.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What are the core principles of object-oriented programming? • How do these principles improve code design and maintenance? • What are some examples of encapsulation in everyday life? Links: • Principles of OOP • OOP Concepts in Python b. Introduction to OOP Principles: • Define OOP: "Object-oriented programming is a programming paradigm based on the concept of 'objects' which can contain data and code." • Show a simple example demonstrating encapsulation. 2. **Development** (30 minutes) a. Encapsulation (10 minutes): • Discuss what encapsulation is and its importance. • Show examples of using private and protected attributes in Python. • Real-time Example: Demonstrate a class with encapsulated attributes and methods. c. Inheritance (10 minutes): • Explain what inheritance is and how it allows for code reuse.

	• Show examples of creating subclasses and using inherited methods. • Real-time Example: Demonstrate inheritance with a simple superclass and subclass. d. Polymorphism (10 minutes): • Explain what polymorphism is and how it enables flexibility in code. • Show examples of method overriding and using polymorphic behaviour. • Real-time Example: Demonstrate polymorphism with different class implementations. 3. Exercise (5 minutes) • Activity: Pair students and ask them to write a Python program that demonstrates inheritance and polymorphism. • Discuss their code and correct any syntax errors.

Closure	1. Summarize key points:
	
	• Core principles of OOP: encapsulation, inheritance, and polymorphism.
	• Benefits of using OOP in programming.
	
	2. Suggested Readings:
	
	• Book 1: "Think Python" by Allen B. Downey.
	  • Chapter 18: Inheritance, pp. 241-252
	• Book 2: "Learning Python" by Mark Lutz.
	  • Chapter 25: OOP: The Big Picture, pp. 775-817

Evaluation	1. Reflective Questions: Discuss how encapsulation, inheritance, and polymorphism improve software design. 2. Homework: Write a short note on the benefits of using object-oriented programming in large-scale software development.

Lesson Plan No. 4.3	Course Name: Python Programming Topic: Creating Classes	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Create custom classes in Python. b. Define attributes and methods for classes. c. Utilize classes to create objects and perform operations.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • How do you define a class in Python? • What are attributes and methods within a class? • Can you think of any real-world objects that can be represented as classes? Links: • Creating Classes in Python • Python Classes Explained **b.** Introduction to Creating Classes: • Define Class: "A class in Python is defined using the class keyword followed by the class name and a colon." • Show a simple class definition example. **2. Development** (30 minutes) a. Defining a Class (10 minutes): • Explain the syntax for defining a class using the class keyword. • Show examples of creating classes with attributes and methods. • Real-time Example: Demonstrate a simple class with attributes like name and age. b. Attributes and Methods (10 minutes): • Explain what attributes and methods are and how they are defined within a class. • Show examples of accessing and modifying attributes and calling methods on class objects.

	• Real-time Example: Show examples of creating objects and using their methods. c. Creating Objects (10 minutes)**:** • Explain how to create objects (instances) of a class. • Show examples of creating objects and performing operations using class methods. • Real-time Example: Demonstrate creating multiple objects and interacting with them. **3.** Exercise (5 minutes) • Activity: Pair students and ask them to write a simple Python class that represents a car with attributes like make and model. • Discuss their code and correct any syntax errors.
Closure	1. Summarize key points: • Defining and using classes in Python. • Attributes and methods within classes. • Creating and using objects in Python. 2. Suggested Readings: • Book 1: "Think Python" by Allen B. Downey. • Chapter 15: Classes and Objects, pp. 185-204 • Book 2: "Learning Python" by Mark Lutz. • Chapter 25: Object-Oriented Programming, pp. 775-817
Evaluation	1. Reflective Questions: Discuss the importance of defining classes for organizing code. 2. Homework: Write a short note on the process of creating and using classes in Python.

Lesson Plan No. 4.4	Course Name: Python Programming Topic: Instance Methods	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand the purpose of instance methods in Python. b. Implement instance methods within classes. c. Utilize instance methods to manipulate object data.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: ▪ What is an instance method? ▪ How do instance methods differ from regular functions? ▪ Why are instance methods important in classes? Links: ▪ Instance Methods in Python ▪ Python OOP: Instance Methods b. Introduction to Instance Methods: ▪ Define Instance Methods: "Instance methods are functions defined inside a class that operate on instances of the class." ▪ Show simple examples of instance methods in Python. 2. **Development** (30 minutes) c. Understanding Instance Methods (10 minutes): ▪ Explain the purpose and syntax of instance methods. ▪ Show examples of defining and calling instance methods. ▪ Real-time Example: Demonstrate a simple class with instance methods to manipulate object attributes. d. Advanced Instance Methods (10 minutes): ▪ Explain how to pass arguments to instance methods. ▪ Show examples of instance methods performing calculations and returning values.

	▪ Real-time Example: Demonstrate a class with advanced instance methods for practical use cases. e. Practical Implementation (10 minutes): ▪ Show how to use instance methods to interact with object data. ▪ Real-time Example: Demonstrate a class with multiple instance methods to manage object state. 3. Exercise (5 minutes) • Activity: Pair students and ask them to write a Python class with several instance methods. • Discuss their code and correct any syntax errors.
Closure	4. Summarize key points: • Purpose and implementation of instance methods. • Syntax and usage of instance methods within classes. • Practical applications of instance methods. 5. Suggested Readings: • Book 1: "Think Python" by Allen B. Downey. • Chapter 17: Classes and Methods, pp. 221-240 • Book 3: "Core Python Programming" by Wesley J. Chun. • Chapter 13: Object-Oriented Programming, pp. 327-36
Evaluation	1. Reflective Questions: Discuss how instance methods enhance the functionality of classes. 2. Homework: Write a short note on the importance of instance methods and provide examples of their use.

Lesson Plan No. 4.4	Course Name: Python Programming Topic: File Organization and Special Methods	Course No.: BCAMJ-301

Objectives	At the end of the lesson, the student shall be able to: a. Understand the concept of file organization in Python projects. b. Implement special methods in Python classes. c. Explain the benefits of using special methods.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	**Introduction** (5 minutes) a. Pre-Discussion Questions: • What is file organization, and why is it important? • What are special methods in Python? • How do special methods enhance class functionality? Links: • Organizing Python Projects • Special Methods in Python b. Introduction to File Organization and Special Methods: • Define File Organization: "File organization refers to structuring the files and directories in a project to maintain a clean and manageable codebase." • Define Special Methods: "Special methods, also known as dunder methods, are predefined methods in Python with double underscores before and after their names (e.g., __init__, __str__)." • Show simple examples of file organization and special methods in Python. **3. Development** (30 minutes) c. Understanding File Organization (10 minutes): • Explain best practices for organizing Python projects. • Show examples of directory structures and module organization. • Real-time Example: Demonstrate a simple Python project with a well-organized file structure. d. Implementing Special Methods (10 minutes):

	• Explain the purpose and syntax of common special methods (__init__, __str__, __repr__). • Show examples of defining and using special methods. • Real-time Example: Demonstrate a class with special methods to customize object behavior. e. Advanced Special Methods (10 minutes): • Explain how to use special methods for operator overloading. • Show examples of overloading arithmetic and comparison operators using special methods. • Real-time Example: Demonstrate a class with overloaded operators using special methods. 3. Exercise (5 minutes) • Activity: Pair students and ask them to organize a simple Python project and implement a class with special methods. • Discuss their code and correct any syntax errors.
Closure	1. Summarize key points: • Importance of file organization in Python projects. • Purpose and implementation of special methods. • Practical applications of special methods in class design. 2. Suggested Readings: • Book 1: "Think Python" by Allen B. Downey. • Chapter 18: Inheritance, pp. 241-252 • Book 2: "Learning Python" by Mark Lutz. • Chapter 24: Exception Basics, pp. 717-751
Evaluation	1. Reflective Questions: Discuss how special methods can enhance class functionality and maintainability. 2. Homework: Write a short note on the importance of file organization and the role of special methods in Python classes.

Lesson Plan No. 4.7	Course Name: Python Programming Topic: Class Variables and Inheritance	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of class variables in Python. b. Implement class variables within classes. c. Explain the concept of inheritance and its types. d. Utilize inheritance to create hierarchies in Python classes.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What are class variables? • How do class variables differ from instance variables? • What is inheritance in object-oriented programming? Link: • Class Variables in Python • Inheritance in Python b. Introduction to Class Variables and Inheritance: • Define Class Variables: "Class variables are shared across all instances of a class." • Define Inheritance: "Inheritance is a mechanism in which one class acquires the property of another class." • Show simple examples of class variables and inheritance in Python. 2. **Development** (30 minutes) a. Understanding Class Variables (10 minutes): • Explain the purpose and syntax of class variables. • Show examples of defining and using class variables. • Real-time Example: Demonstrate a simple class with class variables to manage shared data. b. Implementing Inheritance (10 minutes): • Explain the concept of base class and derived class. • Show examples of single inheritance and multiple inheritance.

	• Real-time Example: Demonstrate a simple class hierarchy with inheritance. c. Advanced Inheritance (10 minutes): • Discuss multiple inheritance and the method resolution order (MRO). • Show examples of using the super() function to call base class methods. • Real-time Example: Demonstrate advanced inheritance techniques with practical examples. 2. Exercise (5 minutes) • Activity: Pair students and ask them to write a Python program that demonstrates class variables and inheritance. • Discuss their code and correct any syntax errors.
Closure	1. Summarize key points: • Concept and implementation of class variables. • Purpose and types of inheritance in Python. • Practical applications of class variables and inheritance. 2. Suggested Readings: • Book 1: "Think Python" by Allen B. Downey. • Chapter 18: Inheritance, pp. 241-252 • Book 2: "Learning Python" by Mark Lutz. • Chapter 25: OOP: The Big Picture, pp. 775-817
Evaluation	1. Reflective Questions: Discuss how class variables and inheritance improve code design and maintainability. 2. Homework: Write a short note on the different types of inheritance in Python and their use cases.

Lesson Plan No. 4.8	Course Name: Python Programming Topic: Polymorphism, Type Identification, and Custom Exception Classes	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the concept of polymorphism in Python. b. Implement type identification techniques in Python. c. Create and use custom exception classes.
Teaching Aids (if any)	a. Slides with diagrams and definitions b. Chalkboard/Whiteboard
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What is polymorphism in object-oriented programming? • How can we identify the type of objects in Python? • What are exceptions, and why might we need custom exception classes? Links: • Polymorphism in Python • Custom Exception Classes in Python b. Introduction to Polymorphism, Type Identification, and Custom Exception Classes: • Define Polymorphism: "Polymorphism allows objects of different classes to be treated as objects of a common superclass." • Define Type Identification: "Type identification helps determine the type of an object at runtime using functions like type() and isinstance()." • Define Custom Exception Classes: "Custom exception classes are user-defined error types derived from the built-in Exception class." • Show simple examples of polymorphism, type identification, and custom exception classes in Python. 2. **Development** (30 minutes) a. Understanding Polymorphism (10 minutes): • Explain the purpose and syntax of polymorphism.

	• Show examples of polymorphism with methods and operators. • Real-time Example: Demonstrate polymorphism using a base class and multiple derived classes. b. Implementing Type Identification (10 minutes): • Explain the use of type() and isinstance() for type checking. • Show examples of type identification in functions and classes. • Real-time Example: Demonstrate a function that behaves differently based on the type of its arguments. c. Creating Custom Exception Classes (10 minutes): • Explain the need for custom exception classes. • Show examples of defining and raising custom exceptions. • Real-time Example: Demonstrate a class with custom exception handling for specific error conditions. 3. Exercise (5 minutes) • Activity: Pair students and ask them to write a Python program that demonstrates polymorphism, type identification, and custom exception classes. • Discuss their code and correct any syntax errors.

Closure	1. Summarize key points:
	
	• Concept and implementation of polymorphism.
	• Purpose and techniques of type identification.
	• Importance and usage of custom exception classes.
	
	2. Suggested Readings:
	
	• Book 1: "Think Python" by Allen B. Downey.
	  • Chapter 18: Inheritance, pp. 241-252
	• Book 2: "Learning Python" by Mark Lutz.
	  • Chapter 29: Designing with Exceptions, pp. 943-971

Evaluation	1. Reflective Questions: Discuss how polymorphism enhances code flexibility and reusability. 2. Homework: Write a short note on the benefits of using custom exception classes in Python.

Lesson Plan No.5.1	Course Name: Python Programming Topic: Introduction to I/O and Data Streams	Course No.: BCAMJ-301

Objectives	At the end of this lesson the student shell be able to: a. Understand basic concepts of I/O in Python. b. Learn about data streams and their types.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk

Teaching Development	1. **Introduction** (5 minutes)
	a. Pre-Discussion Questions:
	• What is an I/O operation in programming?
	• Example Answer: I/O operations involve reading from or writing to external resources, such as files or user inputs.
	• How are data streams used in Python?
	• Example Answer: Data streams represent a sequence of data elements, and in Python, they are used to handle input and output operations, such as reading from a file or writing to a file.
	b. Introduction to I/O and Data Streams:
	• Define I/O operations and data streams.
	• Types of data streams: Input and Output
	• https://www.youtube.com/watch?v=hT7N59j9JVs
	2. **Development** (30 minutes)
	a. Basic I/O Operations (10 minutes):
	• Explain reading from and writing to files.
	• Example: Read from a file and write to a new file
	• Python code :
	`with open('input.txt', 'r') as infile:`
	`content = infile.read()`
	`with open('output.txt', 'w') as outfile:`
	`outfile.write(content)`
	b. Data Streams (20 minutes):
	• Define and explain input and output streams.
	• Example: Using sys.stdin and sys.stdout.
	• Python code:
	• import sys
	• sys.stdout.write("Hello, World!\n")

	3. Exercise(5 Minutes) Activity - Pair students and ask them to write a script that reads from one file and writes to another. - Discuss their scripts and troubleshoot any issues.

Closure	1. Summarize basic I/O operations and data streams.
	
	2. Suggested Readings:
	
	• Book 1:"Think Python" by Allen B. Downey,
	  • Chapter 10: "Files"
	• Book 2:"Learning Python" by Mark Lutz,
	  • Chapter 18: "File I/O"

Evaluation	1. Reflective Questions: Explain basic I/O operations and data streams. 2. Homework: Write a script that performs both reading from and writing to files.

Lesson Plan No. 5.2	Course Name: Python Programming Topic: Access Modes and Writing Data to Files	Course No.: BCAMJ-301

Objectives	At the end of this lesson student shell be able to: a. Understand different file access modes. b. Learn how to write data to files using various modes.
Teaching Aids (if any)	a. Video of Facebook data center b. Use of Nearpod tool for online quiz
Teaching Development	1. Introduction (5 minutes) a. Pre-Discussion Questions: • What are the different file access modes available in Python, and what do they do? • Example Answer: The common file access modes are r (read), w (write), a (append), and b (binary). For instance, r opens a file for reading, while w opens a file for writing and creates a new file if it doesn't exist. • How would you write data to a file in Python? • Example Answer: You use Python's write() or writelines() methods on a file object. For example, file.write("Hello, world!") writes "Hello, world!" to the file. b. Introduction to Access Modes and Writing Data: • Define file access modes: r, w, a, b. • Explain writing data to files. 2. **Development** (30 minutes): a. File Access Modes (15 minutes): • Explain and demonstrate each access mode with examples. • Example: Open a file in w mode and write data to it. with • Python code open('example.txt', 'w') as file: file.write("This is a test.") b. Writing Data to Files (15 minutes): • Methods for writing data: write(), writelines(). • Example: Writing user input to a file, user_input = input("Enter text to write to file: ") with open('user_input.txt', 'w') as file: file.write(user_input)

	3. Exercise (5 minutes) – Activity: • Pair students to write a script that opens a file in different modes and writes data. • Discuss their results and provide feedback
Closure	1. Summarize Key points: • file access modes and data writing techniques. 2. Suggested Readings: • Book 1:"Think Python" by Allen B. Downey, • Chapter 11: "Files and Exceptions" • Book 2:"Learning Python" by Mark Lutz, • Chapter 18: "File I/O"
Evaluation	1. Reflective Questions: Describe file access modes and their use cases. 2. Homework: Create a script that demonstrates file writing using multiple modes

Lesson Plan No. 5.3	Course Name: Python Programming Topic: OOP Concepts	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the fundamental concepts of Object-Oriented Programming (OOP) in Python. b. Explain the key principles of OOP: encapsulation, inheritance, and polymorphism. c. Identify real-world analogies of OOP concepts.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What is Object-Oriented Programming? • How does OOP differ from procedural programming? b. Introduction to OOP Concepts: • Explain the core principles of OOP: Encapsulation, Inheritance, and Polymorphism. • Discuss real-world examples of OOP concepts. • https://www.youtube.com/watch?v=Z2q4daLcbY8 2. **Development** (30 minutes) **a.** Encapsulation (10 minutes): • Define encapsulation and its benefits. • Example: Define a class with private attributes. • Hands-on: Ask students to create a class with private attributes **b.** Inheritance (10 minutes): • Define inheritance and its benefits. • Example: Demonstrate inheritance with a base and derived class. • Hands-on: Ask students to create a class hierarchy. **c.** Polymorphism (10 minutes): • Define polymorphism and its benefits. • Example: Demonstrate method overriding. • Hands-on: Ask students to create a polymorphic method. 3. Exercise (5 minutes) • Pair students and have them design a class that demonstrates all three OOP concepts. • Discuss their designs and correct misconceptions.

Closure	1. Summarize key points: Core principles of OOP: Encapsulation, Inheritance, and Polymorphism. 2. Suggested Readings from Textbooks: • *BOOK 1:* "Think Python" by Allen B. Downey, Chapter 13: "Classes and Objects" (pp. 123-146) • *BOOK 2:* "Learning Python" by Mark Lutz, Chapter 11: "Classes and Objects" (pp. 333-366)
Evaluation	1. Reflective Questions: What are the main principles of OOP? 2. Homework: Write a short note on how encapsulation is implemented in Python.

Lesson Plan No. 5.4	Course Name: Python programming Topic: Creating Classes and Objects	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Create classes and objects in Python. b. Understand the role of the __init__ method. c. Implement class attributes and methods.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: - What is a class in Python? - How do you create an object from a class? b. Introduction to Classes and Objects: - Define what classes and objects are. - Explain the __init__ method for initializing objects. - https://www.youtube.com/watch?v=JeznW_7DlB0 2. **Development (30 minutes)** **a.** Creating a Class (15 minutes): - Syntax for defining a class and its attributes. - Example: Define a Car class with attributes and methods. - Hands-on: Ask students to define their own class, e.g., Book. **b.** Creating Objects (15 minutes): - How to instantiate objects from a class. - Example: Create instances of the Car class and use its methods. - Hands-on: Ask students to create objects from their defined class and call methods. 3. Exercise (5 minutes) - Activity: - Pair students and have them design a class with multiple objects. - Discuss their designs and correct any misconceptions.
Closure	1. Summarize key points: - Syntax for creating classes and objects. 2. Role of the __init__ method. 3. Suggested Readings from Textbooks: - BOOK 1: "Think Python" by Allen B. Downey, Chapter 14: "Classes and Objects" (pp. 147-178) - BOOK 2: "Learning Python" by Mark Lutz, Chapter 12: "Classes and Objects" (pp. 367-399)
Evaluation	1. Reflective Questions: How do you define and instantiate a class in Python? 2. Homework: Create a class for a student with attributes and methods

	Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 5.5	Course Name: Python Programming Topic: Inheritance in Python	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: • Understand the concept of inheritance in Python. • Implement single and multiple inheritance. • Demonstrate how inheritance promotes code reuse.
Teaching Aids (if any)	• PowerPoint presentation • Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes): **a.** Pre-Discussion Questions: • What is inheritance in Python? • Why is inheritance useful in programming? b. Introduction to Inheritance: • Define inheritance and its types (single, multiple, hierarchical). • https://www.youtube.com/watch?v=RZ4oaZB--_s 2. **Development** (30 minutes) a. Single Inheritance (10 minutes): • Explain single inheritance with an example. • Example: Demonstrate single inheritance using a base and derived class. • Hands-on: Ask students to implement single inheritance. b. Multiple Inheritance (10 minutes): • Explain multiple inheritance and potential issues. • Example: Demonstrate multiple inheritance with two base classes. • Hands-on: Ask students to implement multiple inheritance. c. Hierarchical Inheritance (10 minutes): • Define hierarchical inheritance. • Example: Show how multiple derived classes can inherit from a single base class. • Hands-on: Ask students to implement hierarchical inheritance. 3. Exercise (5 minutes) • Pair students and have them create a class hierarchy demonstrating different types of inheritance. • Discuss their implementations and correct any misconceptions.
Closure	1. Summarize key points: • Types of inheritance and their use cases.

	• Code reuse and structure provided by inheritance. 2. Suggested Readings from Textbooks: • BOOK 1: "Think Python" by Allen B. Downey, Chapter 16: "Inheritance" (pp. 211-242) • BOOK 2: "Learning Python" by Mark Lutz, Chapter 14: "Inheritance" (pp. 426-457)
Evaluation	1. Reflective Questions: How does inheritance improve code reuse? 2. Homework: Write a program that demonstrates single and multiple inheritance. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 5.5	Course Name: Python Programming Topic: Object Initialization and Python Constructor	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand object initialization in Python. b. Explain the role of the constructor method (__init__). c. Demonstrate the creation and initialization of objects.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk
Teaching Development	1. **Introduction** (5 minutes) a. Pre-Discussion Questions: • What is object initialization? • What is the purpose of the __init__ method? b. Introduction to Object Initialization: • Explain how Python initializes objects using the __init__ method. • https://www.youtube.com/watch?v=5b2MlYDP4Wk 2. **Development (**30 minutes) a. Understanding __init__ Method (15 minutes): • Define the __init__ method and its parameters. • Example: Initialize attributes in the Person class. • Hands-on: Ask students to implement a class with an __init__ method. c. Constructor Overloading (15 minutes): • Explain constructor overloading (Python's approach with default arguments). • Example: Define multiple constructors in a class using default arguments. • Hands-on: Ask students to demonstrate constructor overloading. 3. Exercise (5 minutes) • Pair students and have them implement a class with different constructors. • Discuss their implementations and correct any misconceptions.
Closure	1. Summarize key points: • Role of the __init__ method. • Concept of constructor overloading. 2. Suggested Readings from Textbooks: • BOOK 1: "Think Python" by Allen B. Downey, Chapter 15: "Classes and Functions" (pp. 179-210)

	• BOOK 2: "Learning Python" by Mark Lutz, Chapter 13: "Object-Oriented Programming" (pp. 400-425)
Evaluation	1. Reflective Questions: What is the purpose of the __init__ method in Python? 2. Homework: Write a class with an overloaded constructor and demonstrate its usage.

Lesson Plan No. 5.7	Course Name: Python Programming Topic: Introduction and Reading a File	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Understand the basics of file handling in Python. b. Read data from a file using Python. c. Explain the importance of file handling.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • What is file handling? • Why is it important to read data from files in programming? b. Introduction to File Handling: • Explain the concept of file handling in Python. • Discuss basic file operations: opening, reading, and closing files. • https://www.youtube.com/watch?v=2zddS9d1I1E **2. Development** (30 minutes) a. Reading a File (15 minutes): • Syntax for reading files in Python. • Example: Reading the contents of a text file. • Hands-on: Students read data from a provided text file. • https://www.youtube.com/watch?v=2zddS9d1I1E b. File Methods (15 minutes): - Overview of basic file methods (read(), readline(), readlines()). - Discuss real-world scenarios for file reading. - https://www.youtube.com/watch?v=Z4tTwzK1y2Q **3.** Exercise (5 minutes) • Pair students and ask them to write a program that reads a file and prints each line. • Discuss their implementations and address any issues**.**
Closure	1. Summarize key points: • Basics of file handling and file reading in Python. • Importance of file operations. 2. Suggested Readings from Textbooks**:** • BOOK 1: Think Python by Allen B. Downey: Chapter 10: "Files" (pp. 207-221) • BOOK 2: Learning Python by Mark Lutz: Chapter 11: "Files and I/O" (pp. 345-370)

Evaluation	1. Reflective Questions: What is file handling and why is it important? 2. Homework: Write a short note on different file reading methods and their applications. Spend 5 minutes to evaluate student assimilation of the lesson contents

Lesson Plan No. 5.8	Course Name: Python Programming Topic: Writing to a File	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Write data to a file using Python. b. Understand different modes of file operations. c. Explain the significance of file writing.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples
Teaching Development	**1. Introduction (5 minutes)** a. Pre-Discussion Questions: • How do you write data to a file? • What are the different file modes in Python? • Introduction to Writing to Files: • Explain different file modes (w, a, x, etc.). • Discuss the importance of writing data to files. • https://www.youtube.com/watch?v=lJY1a1ZET9U 2. **Development (30 minutes)** a. Writing to a File (15 minutes): • Syntax for writing to files in Python. • Example: Writing a string to a text file. • Hands-on: Students write data to a new file. • https://www.youtube.com/watch?v=lJY1a1ZET9U b. File Modes and Methods (15 minutes): • Overview of file modes and their usage. • Discuss append mode and exclusive creation mode. 3. Exercise (5 minutes) • Pair students and ask them to write a program that creates a new file and writes some text to it. • Discuss their implementations and address any issues.
Closure	1. Summarize key points: • Basics of writing data to files and different file modes. • Importance of proper file handling. 2. Suggested Readings from Textbooks: • BOOK 1: Think Python by Allen B. Downey: Chapter 11: "Files" (pp. 222-237) • BOOK 2: Learning Python by Mark Lutz: Chapter 12: "Files and I/O" (pp. 371-395 Spend 5 minutes to wrap up and consolidate the learnings

Evaluation	1. Reflective Questions: How can you write data to a file, and what are the different modes available? 2. Homework: Write a short note on different file modes and their appropriate use cases.

Lesson Plan No. 5.9	Course Name: Python Programming Topic: Creating a New File	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Create new files using Python. b. Understand how to handle file creation errors. c. Explain the importance of file management
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples
Teaching Development	**1. Introduction** (5 minutes) a. Pre-Discussion Questions: • How do you create a new file in Python? • What should you consider when creating new files? b. Introduction to File Creation: • Explain the concept of file creation in Python. • Discuss common errors related to file creation and how to handle them. • Https://www.youtube.com/watch?v=lE0UGJ8s_Xk 2. **Development** (30 minutes) **a.** Creating a New File (15 minutes): • Syntax for creating and writing to a new file. • Example: Creating a new text file and writing some content to it. • Hands-on: Students create a new file and add data to it. **b.** Error Handling in File Creation (15 minutes): • Discuss common errors (e.g., file already exists, permission issues). • Demonstrate error handling using try-except blocks. • https://www.youtube.com/watch?v=d9WJs2h8B50 3. Exercise (5 minutes) • Pair students and ask them to write a program that handles file creation errors and creates a new file with specified content. • Discuss their implementations and address any issues
Closure	1. Summarize key points: - Basics of file creation and error handling. - Importance of managing file creation. 2. Suggested Readings from Textbooks**:** • BOOK 1: Think Python by Allen B. Downey: • Chapter 11: "Files" (pp. 238-252) • BOOK 2: Learning Python by Mark Lutz: • Chapter 13: "Files and I/O" (pp. 396-420)

Evaluation	1. Reflective Questions: How do you create a new file, and what are some common issues you might encounter? 2. Homework: Write a short note on best practices for file creation and management.

Lesson Plan No. 5.10	Course Name: Python Programming Topic: File Handling - File Methods	Course No.: BCAMJ-301

Objectives	At the end of the lesson the student shall be able to: a. Use various file methods in Python. b. Understand how to manipulate file contents. c. Explain the significance of file methods.
Teaching Aids (if any)	a. PowerPoint presentation b. Chalk & Talk c. Code examples
Teaching Development	**1. Introduction (5 minutes)** **a.** Pre-Discussion Questions: - What are file methods in Python? - How can file methods help in handling files? - Introduction to File Methods: - Explain common file methods (read (), write (), seek (), tell ()). - Discuss the importance of these methods in file manipulation. - https://www.youtube.com/watch?v=p7d_eWqY_JM **2. Development** (30 minutes) **a.** Overview of File Methods (15 minutes): - Detailed explanation of each file method and its usage: - read(): Reads the entire file content. - readline(): Reads a single line from the file. - readlines(): Reads all lines and returns a list. - write(): Writes a string to the file. - seek(): Moves the file pointer to a specific position. - tell(): Returns the current file pointer position. - Examples demonstrating each method. - Hands-on: Students practice using these methods with sample files. **b.** File Manipulation Techniques (15 minutes): - Combining file methods for advanced operations: - Reading and modifying file content. - Using seek() to navigate within a file and update content. - Real-world examples where file manipulation is crucial. **3.** Exercise (5 minutes) - Activity: - Pair students and ask them to create a program that demonstrates the use of multiple file methods. For example, read a file, modify its content, and write the changes to a new file. - Discuss their implementations and address any issues.
Closure	1. Summarize key points: - Overview of key file methods and their applications. - Importance of effectively using file methods for file manipulation. 2. Suggested Readings from Textbooks:

	- BOOK 1: Think Python by Allen B. Downey: Chapter 10: "Files" (pp. 222-237) - BOOK 2: Learning Python by Mark Lutz: Chapter 14: "Files and I/O" (pp. 421-445)
Evaluation	1. Reflective Questions: What are some common file methods, and how are they used? 2. Homework**:** Write a short note on how different file methods can be used in combination to handle file content efficiently. Spend 5 minutes to evaluate student assimilation of the lesson contents