Topics covered in the lecture:3

- A data Type
- Object Oriented Programming - Introduction
- Examples of User Types
- Defining a Class
- Objects of a Class
- Class Methods and Attributes
- Queue
- Dunder methods
- Object Comparisons
- Copying Objects

Objective Questions:

1. What is the output of the following program?

```
class Mystery:
    def __init__(self, value=0):
        self.value = value
    def increment(self):
        self.value += 1
        return self.value

e1 = Mystery()
e2 = e1
print(e1.increment(), e2.increment(), e1.value)
```

   A) 1, 2, 1
   B) 1, 1, 1
   C) 1, 2, 2
   D) Error

2. What is the output of the following code?

```
class MyClass:
    def __init__(self, name):
        self.name = name

    def change_name(self, name):
        name = name

obj = MyClass("OldName")
obj.change_name("NewName")
print(obj.name)
```

3. What is the output?

```
class MyClass:
    def __init__(self):
        pass

    def display(self):
        print("Hello")

obj = MyClass()
obj.display(5)
```

A) No error
B) Error due to `obj.display(5)`
C) Error in `__init__`
D) Syntax Error

4. What is the output?

```
class Stack:
    def __init__(self):
        self.data = []

    def push(self, item):
        self.data.append(item)

    def pop(self):
        return self.data.pop()

s = Stack()
s.pop()
```

   A) No error
   B) Error due to `s.pop()`
   C) Error in push method
   D) Syntax Error

5. What is the role of the `__init__` method in a Python class?
   A) Initializes the Python interpreter.
   B) Initializes the object's attributes.
   C) It is called when the class is deleted.
   D) It defines a private attribute.

6. What is the output of the following code?

```
class MyClass:
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return self.value

obj = MyClass(10)
print(obj)
```

a) AttributeError: 'int' object has no attribute 'value'
b) TypeError: **str**() should return a string
c) No error
d) SyntaxError

7. What is the output of the following code?

```
class Test:
    def __init__(self, value):
        self.value = value

    def __eq__(self, other):
        return self.value == other.value

obj1 = Test(5)
obj2 = Test(5)
print(obj1 == obj2)
```

a) False
b) True
c) TypeError
d) None

8. What is the issue in the code?

```
class Car:
    def __init__(self, brand):
        self.brand = brand

    def __eq__(self, other):
        return self.brand == other.brand
```

```
car1 = Car("Toyota")
car2 = Car("Honda")
print(car1 == "Honda")
```

a) TypeError: 'str' object cannot be compared with 'Car' object
b) SyntaxError
c) Logical error
d) No issue, it will print False

9. What will be printed?

```
class Counter:
    def __init__(self, value=0):
        self.value = value

    def increment(self):
        self.value += 1

    def __str__(self):
        return str(self.value)

counter1 = Counter()
counter1.increment()
counter2 = counter1
counter2.increment()
print(counter1)
```

a) 0
b) 1
c) 2
d) Error

Programming Questions:

1. Implement a class `Stack` with methods for `push`, pop, `peek`, and `is_empty`. Use this class to reverse a string.

2. Write a function `is_balanced` that takes a string of parentheses and checks if it is balanced. Use a stack to solve this problem.

3. Write a Complex class that supports basic operations like addition, subtraction, and equality checks. Extend the class to include a method for calculating the modulus of the complex number.

4. Write a function `is_palindrome(s: str)` that checks if the string `s` is a palindrome by using a stack. A palindrome is a word that reads the same forward and backward (ignoring spaces, punctuation, etc.).

5. Given an array, write a function next_greater(lst) that finds the next greater element for each element in the list. You can assume the list contains distinct numbers. Use a stack class to solve this efficiently. Example: For the input [4, 5, 2, 10], the output should be [5, 10, 10, None].
6. Implement a queue using two stacks. Implement `enqueue`, `dequeue`, `peek`, and `is_empty` operations, ensuring the queue follows FIFO behavior using only stack operations.
7. Write a class PriorityQueue that simulates a priority queue. Implement the following methods:
    - enqueue(item, priority) – Adds an item to the queue with the specified priority.
    - dequeue() – Removes and returns the item with the highest priority.
    - peek() – Returns the item with the highest priority without removing it.

Example:

```
class PriorityQueue:
     # Define methods here

# Example usage
pq = PriorityQueue()
pq.enqueue("Task 1", 1)
pq.enqueue("Task 2", 2)
pq.enqueue("Task 3", 0)
print(pq.dequeue())  # "Task 2"
```

Scenario-Based Questions:

1. **Bank Account Simulation:**

   Write a `BankAccount` class that allows you to manage a bank account. Implement:

   - `deposit(amount)`: Adds an amount to the balance.
   - `withdraw(amount)`: Deducts an amount if funds are sufficient; otherwise, return `"Insufficient funds"`.
   - `check_balance()`: Returns the current balance.

2. **Library System:** Implement a Library class to manage a collection of books. Each book has attributes like title, author, and ISBN.
   - Methods:
     - add_book(book) – Adds a book to the library.
     - remove_book(ISBN) – Removes a book from the library.
     - search_book(title) – Searches for books by title.

        - issue_book(ISBN) – Issues a book to a member.
        - return_book(ISBN) – Returns a book to the library.

   Write a program where users can search for books, issue, and return them. Also, simulate managing a collection of books

3. **Student Grades Management:**
   Create a Student class with attributes:
        - name (String)
        - roll_number (String)
        - grades (List of grades)
    - Methods:
        - add_grade(grade) – Adds a grade to the student’s record.
        - get_average_grade() – Calculates the average grade of the student.
        - get_highest_grade() – Returns the highest grade.
    - Scenario: Write a program where teachers can add grades for students and retrieve their average or highest grade.

4. **Movie Ticket Booking System:**
    - Design a MovieTicket class with attributes:
        - movie_name (String)
        - show_time (String)
        - price (Float)
        - seats_available (Integer)
    - Methods:
        - book_ticket(number_of_seats) – Books a specified number of seats.
        - cancel_ticket(number_of_seats) – Cancels a specified number of seats.
        - get_available_seats() – Returns the number of seats available for booking.

- Scenario: Simulate a ticket booking system where users can book or cancel movie tickets for various shows.

5. **Hotel Reservation System**
   - Implement a HotelRoom class with the following attributes:
     - room_number (String)
     - room_type (String, e.g., "Single", "Double", "Suite")
     - price_per_night (Float)
     - available (Boolean)
   - Methods:
     - book_room() – Books the room if available.
     - check_out() – Marks the room as available again after checkout.
     - get_total_cost(nights) – Returns the total cost for staying the specified number of nights.
   - Scenario: Simulate a hotel reservation system where customers can book rooms and check out

6. **Task Scheduler:** Create a TaskQueue class that simulates a simple task scheduler using the First-Come, First-Served (FCFS) algorithm. Implement methods to enqueue tasks with processing times and dequeue them for processing.

7. **Customer Support Queue Management:** In a busy customer service center, support tickets are filed continuously. Each ticket has a unique ticket_id, a customer_name, and an issue_description. Tickets are processed on a first-come, first-served basis to ensure fairness. Implement a SupportTicket class that represents each ticket, including a __str__ method to display ticket details. Use a Queue to manage these support tickets, with methods to add new tickets, resolve the next one in line, and check the number of pending tickets.