UGANDA CHRISTIAN UNIVERSITY
A Centre of Excellence in the Heart of Africa

FACULTY OF ENGINEERING, DESIGN AND TECHNOLOGY
DEPARTMENT OF COMPUTING AND TECHNOLOGY
ADVENT 2025 OOP IN-CLASS trail qtns

Object Oriented Programming in Python

DATE TO BE UNDERTAKEN: Friday **3rd Oct 2025**

PROGRAM: BSIT 2:1

VENUE: N12

Task 1: Mini Game Battle

Create a Python class called `Player` with the following:

- **Attributes:** `name`, `health` (default = 100), `attack_power`
- **Methods:**
  - `attack(other)` → subtracts `attack_power` from `other.health`
  - `heal(amount)` → increases health
  - `is_alive()` → returns `True` if `health > 0`

Steps:

1. Create 2 players with different attack powers.
2. Simulate a fight round by round until one player dies.
3. Print a battle log after each round.

Stretch: Add random **critical hits** (double damage) or a simple **defense mechanism**.

Task 2: Library System

Create two classes:

- **`Book(title, author, available=True)`**
- **`Library`** with methods:
    - `add_book(book)`
    - `borrow(title)` → marks book unavailable if free
    - `return_book(title)` → makes book available again
    - `show_available_books()` → lists available books

Steps:

1. Add at least 5 books.
2. Borrow 2 books.
3. Return 1 book.
4. Display available books after each step.

Stretch: Add **search by author** and prevent borrowing the same book twice.

Task 3: School with Students

Create two classes:

- **`Student(name, age, grades=[])`**
    - Method: `add_grade(score)`
    - Method: `average()` → returns average grade
- **`School(name)`**
    - Attributes: list of students
    - Methods:

        - `add_student(student)`
        - `top_student()` → returns student with highest average

Steps:

1. Create 3 students and add multiple grades.
2. Add them to a school.
3. Print the student with the highest average.

Stretch: Support **multiple subjects** with separate grade lists.

Task 4: ATM Simulation

Create two classes:

- **`BankAccount(owner, balance=0)`**
- **`ATM`** with methods:
    - `insert_card(account)`
    - `withdraw(amount)`
    - `deposit(amount)`
    - `check_balance()`
    - `eject_card()`

Steps:

1. Create 2 accounts.
2. Simulate deposits, withdrawals, and transfers.
3. Print balances after each action.

Stretch: Add **PIN verification** or **daily withdrawal limits**.

Task 5: Geometry Challenge

Extend your **Point** and **Circle** classes.

- **Point(x, y):**
    - `distance_to(other)` → returns distance between two points
    - `midpoint(other)` → returns a new `Point`
- **Circle(center: Point, radius):**
    - `area()`
    - `contains(point)` → returns `True` if point is inside circle

Steps:

1. Create a circle and a few points.
2. Check which points lie inside the circle.

Stretch: Add `intersects(other_circle)` → returns True if circles overlap.

Task 6: Employee Management System

Create two classes:

- **`Employee(name, salary, department)`**
    - Method: `give_raise(amount)`
- **`Company(name)`**
    - Attributes: list of employees
    - Methods:
        - `add_employee(emp)`
        - `average_salary()`

        - `list_department(dept)`

Steps:

1. Add 5 employees in different departments.
2. Give some raises.
3. Print average salary and department lists.

Stretch: Find the **highest-paid employee** per department.

Task 7: Ride-Hailing Simulation

Create four classes:

- `Driver(name, car, earnings=0)`
- `Rider(name, wallet)`
- `Car(brand, plate_number)`
- `Trip(driver, rider, km, price_per_km)` → computes fare

Steps:

1. Create 2 drivers, 2 riders, and cars.
2. Simulate at least 2 trips.
3. Deduct fare from rider wallet and add to driver earnings.

Stretch: Add **surge pricing** or **driver ratings**.

Task 8: Parking Lot Manager

Create three classes:

- `Car(brand, plate_number)`

- `Ticket(car, entry_time)`
- `ParkingLot(capacity)`

ParkingLot methods:

- `park(car)` → issues a `Ticket` if space available
- `exit(ticket, exit_time)` → calculates fee (e.g., hourly rate)
- `status()` → shows free/occupied spots

Steps:

1. Create a parking lot with limited spots.
2. Park multiple cars.
3. Exit some cars and calculate fees.

Stretch: Add **spot sizes (small/large)** and allocate cars accordingly.

Task 9: Shopping Cart System

Create two classes:

- **`Item(name, price)`**
- **`Cart()`**
    - Methods:
        - `add_item(item, qty)`
        - `remove_item(item, qty)`
        - `total()` → returns total cost
        - `summary()` → prints all items with quantities and subtotals

Steps:

1. Create at least 3 items.
2. Add them to the cart with different quantities.
3. Remove 1 item and recalculate total.

Stretch: Prevent negative quantities and add a method `clear()` to empty the cart.

Task 10: Movie Theater Booking

Create two classes:

- **`Movie(title, seats)`**
- **`BookingSystem()`**
    - Methods:
        - `add_movie(movie)`
        - `book_seat(title)` → decreases seat count if available
        - `available_movies()` → shows movies with seats left

Steps:

1. Add 3 movies with seat counts.
2. Book several seats from different movies.
3. Show which movies are still available.

Stretch: Prevent overbooking and add a `cancel_booking(title)` method.

Task 11: Attendance Register

Create a class **`Register(date)`** with methods:

- `check_in(student_name)` → adds student to list

- `is_present(student_name)` → returns True if student is in the list
- `list_students()` → prints all names checked in

Steps:

1. Create a register for today's date.
2. Add at least 5 students.
3. Check attendance for 2 names and display all present students.

Stretch: Prevent duplicate check-ins for the same student.

Task 12: Simple Logger (Rolling Buffer)

Create a class **`Logger(capacity)`** with methods:

- `log(message)` → adds a message
- `recent(n)` → returns the last n messages (newest first)
- `count()` → returns total stored messages

Steps:

1. Create a logger with capacity 5.
2. Log 8 messages.
3. Show the last 3 messages and the count.

Stretch: Add levels (`INFO`, `WARN`, `ERROR`) and a method `filter(level)` to list only those.