



**UGANDA INSTITUTE OF INFORMATION AND COMMUNICATIONS TECHNOLOGY
END OF SEMESTER TWO EXAMINATIONS
ACADEMIC YEAR 2025/2026**

PROGRAMME(S): DIPLOMA IN SOFTWARE ENGINEERING (DSWE)

YEAR OF STUDY: ONE SEMESTER TWO

COURSE: COMPUTER PROGRAMMING II

COURSE CODE : CSC2104

DATE: SUNDAY 28TH, JUNE 2026

TIME: 9:00 AM – 12:00 PM

DURATION: 3 HOURS

VENUE: SEMINAR ROOM, ELECTRONICS LAB, ITU, SMART IDEAS LAB (HUB)

INSTRUCTIONS:

- i. This is a practical examination. Create a folder on the desktop in your name and registration number where you will be saving your work
- ii. Attempt only TWO (2) questions
- iii. All questions carry equal marks

QUESTION 1:

An Institute is experiencing challenges in managing student admissions due to increasing application numbers. Currently, student details are collected manually using paper forms, making it difficult to maintain accurate records and efficiently register multiple students. To improve efficiency, the institute administration requires a command-line Python application that captures student admission details, displays a registration summary, and continuously registers students until the user decides to stop.

Required Tasks (Total: 50 Marks)

(a) Using any IDE, create a file named "**student_admission**" to develop your Python solution. (2 Marks)

(b) **Recursive Function Definition:** (4 Marks)

Define a Python function named **register_student()** that will handle the entire student admission process. The function should be capable of calling itself recursively to allow registration of multiple students. (6 Marks)

(c) **Institute Information Collection:**

Within the function, prompt the user to enter the following institutional details and store them in appropriate variables:

- institute_name
- department

(d) **Student Information Collection:** (8 Marks)

Prompt the user to enter the following student details:

- first_name ,second_name ,last_name ,gender , dob (Date of Birth) ,nationality ,course_applied

(e) **Guardian and Educational Background Collection:**

Prompt the user to enter the following details:

Guardian Information

- guardian_name ,guardian_phone ,guardian_email

Educational Background (8 Marks)

- secondary_school (A-Level)
- secondary_school1 (O-Level)

(f) **Registration Summary Display** (10 Marks):

After capturing all details, display a structured admission matching the following format:

(g) **Recursive Admission Processing:** (12 Marks)

After displaying the registration summary, prompt the user for YES or NO so that the program can run recursively.

QUESTION 2:

A Savings and Credit Co-operative Organization (SACCO) in Kabale is facing operational bottlenecks due to a rapid increase in small business loan applications. Loan officers currently verify applicant eligibility manually using paper forms and basic calculators. This manual workflow causes significant processing delays, calculation errors and transparency disputes among members.

To automate and standardize this process, the SACCO management requires a command-line Python application. The application must evaluate member eligibility based on strict institutional policies, calculate

maximum allowable loan limits, process multiple applicants sequentially and output a structured audit summary.

SACCO Credit Policy:

- **Savings Tenure:** A member must have an active savings history with the SACCO for a minimum of 6 months.
- **Debt Exposure:** A member's existing outstanding loan balance must be less than UGX 500,000.
- **Credit Limit:** Qualified members are eligible to receive a maximum loan amount equivalent to exactly three (3) times their verified monthly income.

Required Tasks (Total: 50 Marks)

- (a) **Environment Setup:** Create a file named “*loan_management*” to develop your Python solution.
(2 Marks)
- (b) **Eligibility Logic Function:** Define a Python function named “*check_eligibility*” that accepts only two arguments: “*months_saved*” (integer) and “*loan_balance*” (float).
(6 Marks)
- (c) **Loan Quantum Function:** Define a Python function named “*calculate_loan*” that accepts only one argument: “*monthly_income*” (float). The function must calculate and return the maximum permissible loan amount based on the credit limit policy.
(4 Marks)
- (d) **Data Structure Initialization:** Initialize an empty dictionary named “*sacco_members*” to store the processed applicant records.
(2 Marks)
- (e) **Data Collection Loop & Ingestion:** Implement a dynamic loop structure to collect data for the test profiles provided in the table below. For each iteration, the program must execute the following operations:
(20 Marks)

Member ID	Monthly Income (UGX)	Loan Balance (UGX)	Months Saved
1001	250,000	0	17
1002	1,000,000	700,000	3
1003	2,500,000	300,000	10
1004	450,000	510,000	2
1005	750,000	20,000	9

- i. Initialize an empty list named “*member_information*” inside the loop.
- ii. Prompt the user via the console input to enter the following values, ensuring appropriate numeric data type casting:
 - “*member_id*” (String: unique key)
 - “*monthly_income*” (float)
 - “*loan_balance*” (float)
 - “*months*” (integer)

- iii. Append the variables “*monthly_income*”, “*loan_balance*” and “*months*” into the “*member_information*” list in that exact sequence.
- iv. Populate the “*sacco_members*” dictionary by assigning the “*member_id*” as the key and the “*member_information*” list as its corresponding value.
- (f) **Batch Evaluation & Reporting:** Iterate over the “*sacco_members*” dictionary to parse each record. For each member, unpack the profile details, evaluate eligibility using the “*check_eligibility*” function defined above and if approved, compute the credit allocation using the “*calculate_loan*” function defined above. Print an audit summary statement to the console matching these exact formats, replacing the placeholders in the brackets with the actual values: **(16 Marks)**

Approved Applicant:

“Member [member_id] has been saving with the SACCO for [months] months; has a monthly income of UGX. [monthly_income] and their loan balance is UGX. [loan_balance] and they qualify for a loan of up to UGX. [maximum_loan_amount].”

Rejected Applicant:

“Member [member_id] has been saving with the SACCO for [months] months; has a monthly income of UGX. [monthly_income] and their loan balance is UGX. [loan_balance] and they do not qualify for a loan.”

QUESTION 3:

A Mobile Money agent wants a Python program to calculate commissions earned from transactions, and the Commission Rates are as follows: -

Deposits

1. Below UGX 50,000 1% commission
2. UGX 50,000 and above 2% commission

Withdrawals

1. Below UGX 50,000 3% commission
2. UGX 50,000 and above 5% commission

Required Tasks (50 Marks)

(a) Environment Setup (2 Marks)

Create a file named commission

(b) Function Creation (10 Marks)

Create a function called `calculate_commission()` that accepts:

- `transaction_type`
- `amount`

The function should calculate and return the correct commission.

(c) List Creation (2 Marks)

Create an empty list called commissions.

(d) Data Entry and Processing (24 Marks)

Using a loop, allow the user to enter details for 5 transactions.

For each transaction, enter:

- Transaction Type (deposit or withdrawal)
- Amount
- Network (MTN or Airtel)

For each transaction:

- Calculate the commission using the function.
- Store the commission in the commissions list.
- Count the number of MTN transactions.
- Count the number of Airtel transactions.

(e) Calculations (6 Marks)

Find:

- The lowest commission
- The highest commission
- The average commission from the commissions list.

QUESTION 4:

Nile General Stores Ltd operates a retail chain across Jinja district specializing in household merchandise and electronics. Last financial year, the enterprise suffered severe financial losses due to a manual paper-led inventory system, which caused duplicated invoices and untracked stock shrinkages.

The managing director, Mr. Kisekka, requires a backend prototype for a centralized inventory management system. Because he plans to scale operations to Mbale and Mukono districts, the codebase must be highly modular, maintainable and reusable. You have been tasked with building an Object-Oriented Python prototype that tracks product prices, secures stock quantities from unauthorized external manipulation and generates automated inventory reports.

Required Tasks (Total: 50 Marks)

- Environment Setup:** Create a file named “*inventory_management*” to develop your Python solution. **(2 Marks)**
- Class Definition:** Define a Python class named “*Product*”. **(2 Marks)**
- Object Initialization:** Implement a constructor that accepts and maps three arguments upon object instantiation: “*name*”, “*price*” and “*quantity*”. **(8 Marks)**
 - **Visibility constraint:** “*name*” and “*price*” must be public attributes, while “*quantity*” must be a private attribute to prevent direct, external data corruption.
- Getter Implementation:** Implement an instance method called “*get_quantity*” that safely returns the current private stock count of the product. **(6 Marks)**

- (e) **State Manipulation:** Implement an instance method called “*reduce_stock*” that accepts an integer value called “*amount*”. This method must reduce the private stock count by the specified amount. (4 Marks)

- **Industry Best Practice:** Include a basic check to prevent stock from dropping below zero. (2 Marks)

- (f) **Data Population:** Instantiate exactly four product objects using the verified baseline data below: (8 Marks)

product	price	quantity
Mobile Phone	500,000	25
Watch	100,000	37
Blender	80,000	15
Microwave	150,000	10

- (g) **Transaction Simulation:** Execute simulated sales by calling the “*reduce_stock*” method on the created objects using the following quantities: (8 Marks)

- Deduct 7 units from the Mobile Phone stock.
- Deduct 19 units from the Watch stock.
- Deduct 3 units from the Blender stock.
- Deduct 2 units from the Microwave stock.

- (h) **Collection Management:** Initialize an empty Python list named “*products_list*” and populate it with the four instantiated product objects. (2 Marks)

- (i) **Automated Reporting:** Write a for loop that iterates through “*products_list*” and prints a dynamic inventory message for each product. Replace the bracketed placeholders with the actual object values: (8 Marks)

"[product_name] costs UGX [cost_of_product] and has [quantity] items left in stock."

QUESTION 5 :

Swift Traveler Coaches is a long-distance transport company operating bus routes across Uganda. During festive seasons, the company experiences operational chaos due to duplicate bookings, overbooking and inconsistent passenger records.

To solve this, management requires a reliable, modular ticketing system. The system must automatically log ticket details for financial accountability. Additionally, the code must use reusable Object-Oriented Programming (OOP) modules, as the company plans to expand its routes into South Sudan and Rwanda.

As a software developer, you are tasked with building a prototype using Python. The system must implement classes, objects, inheritance, encapsulation, polymorphism, loops and automated travel reports.

Required Tasks (Total: 50 Marks)

- (a) **Environment Setup:** Create a Jupyter Notebook file named “*transport_management.ipynb*” to develop your Python solution. (2 Marks)
- (b) **Class Creation:** Create a base class named “*Ticket*”. (2 Marks)

- (c) **Constructor Initialization:** Define a constructor that accepts one integer argument called “*num_passengers*” - the number of passengers on the ticket. Map this argument to a public instance attribute named “*passengers*”. (4 Marks)
- (d) **Encapsulation (Setter):** Inside the constructor, initialize a private instance attribute named “*price*”. Create a setter method named “*set_price*” that accepts a price value as an argument (defaulting to 50,000) and assigns it to this private attribute. Call this setter inside your constructor to set the initial price. (8 Marks)
- (e) **Encapsulation (Getter):** Create a getter method named “*get_price*” that returns the value of the private “*price*” attribute. (4 Marks)
- (f) **Inheritance & Polymorphism:** Create two child classes named “*Ordinary*” and “*VIP*” that inherit from the “*Ticket*” class. Overload or set their prices using the “*set_price*” method during initialization as follows: (12 Marks)
- Ordinary ticket price: 70,000 UGX
 - VIP ticket price: 120,000 UGX
- (g) **Data Structure:** Initialize an empty dictionary named “*tickets*” to store the created ticket objects. (2 Marks)
- (h) **Data Population (Loops & Objects):** Using a for loop or sequential assignment, populate the “*tickets*” dictionary with the following test data, using the Ticket ID as the dictionary key and the ticket object as the dictionary value: (10 Marks)

Ticket ID (Key)	Ticket Category	Number of Passengers
1001	Ordinary	1
1002	Ordinary	5
1003	VIP	1
1004	VIP	3

- (i) **Reporting:** Use a for loop to iterate through the “*tickets*” dictionary. Calculate the total cost for each ticket as (*passengers * price*) and print a report using the exact format below (replacing the brackets with actual values): (6 Marks)

“Ticket [*ticket_id*] has [*number of passengers*] passengers and its total cost is UGX. [*total cost of ticket*].”

QUESTION 6 (50 MARKS)

MTAC a training institution, intends to automate the registration and performance tracking of students enrolled in various course units. The current manual system makes it difficult to maintain accurate student records and generate performance reports.

You are required to develop a Python program that captures student information, calculates final results, determines grades, stores student records, and generates a performance report.

Grading Policy

Total Marks	81-100	70 -80	60 – 69	50 – 59	<50
Grade	A	B	C	D	F

Required Tasks

(a) Create a Python function that determines and returns a student's grade based on the total marks obtained. **(8 Marks)**

(b) Create appropriate data structures to store student records and total marks. **(4 Marks)**

(c) Develop a program that repeatedly captures the following details for each student until the user chooses to stop: **(22 Marks)**

- Registration Number, Student First Name, Middle Name, Last Name, Course Unit Code, Course Unit Name, Coursework Marks (/40), Examination Marks (/60)

The program should:

- Validate all marks entered, calculate the total marks obtained, Determine the student's grade, Store the student's record for future reporting, Keep track of all total marks entered.

(d) Using the recorded student totals, calculate and display: **(6 Marks)**

- Highest Total Mark , Lowest Total Mark , and Average Total Mark

(e) Generate and display a performance report showing, for each student: **(10 Marks)**

- Registration Number ,Student Name ,Course Unit Code ,Course Unit Name ,Coursework Marks ,Examination Marks ,Total Marks ,and Grade

The report should also display the highest, lowest, and average marks obtained by the students.

END