# Intro to CS in Python

## Homework Assignment 5

## Functions

## General guidelines:

a. Do exactly as required in the questions.
b. Individual work and submission – paired submission is not allowed.
c. Include in your submission a file containing all the interpreter output printed throughout the solution process of your homework. Submissions that do not include a comprehensive SHELL log file including all the research and debugging leading up to the exercise solution will not be graded.

Important note: The submission deadline is as published in the Moodle submission box. Any submission later than the deadline may not be graded. If it is graded, the grade will not exceed 95% (Meaning: Late submissions have an automatic grade reduction of 5%, very late submissions will not be graded at all).

Note: In all the coding questions below separate the code into **three files**: The first containing **definitions and constants**, the second containing **functions** and the third containing the **main program**. Don't forget to `import` the files correctly.

## Question 1:

You are building a basic banking application that allows users to perform various operations like opening accounts, depositing money, withdrawing money, and logging transactions. Your task is to implement the required functions based on the specifications provided.

Note: For the sake of simplicity and user security, the program is not expected to save the balance of the accounts, nor is the list of accounts saved. Therefore, the transactions do not test the validity of the account number; the functions will only print messages to inform the user of the success of their transactions. The conscientious student will complete the missing parts as a private project, but this is not to be submitted as homework.

Implement the functions `deposit`, `withdraw`, `open_account`, and `log_transaction` to complete the banking application. The `open_account` function should generate a unique account number based on the account type (defaulting to "Savings" if none is specified) and print a message indicating the account creation. The `deposit` and `withdraw` functions should print messages confirming the transactions. The `log_transaction` function should create a log entry for each transaction.

(Continued on the next page…)

## Function Specifications:

1. **Deposit Function**:
   - **Name**: `deposit`
   - **Parameters**:
     - `account_number` (int): The account number to deposit into.
     - `amount` (float): The amount to deposit.
   - **Action**: Prints a message indicating the deposit.
   - **Expected Output Format**:
     - "`Depositing {amount} NIS into account {account_number}.`"
2. **Withdraw Function**:
   - **Name**: `withdraw`
   - **Parameters**:
     - `account_number` (int): The account number to withdraw from.
     - `amount` (float): The amount to withdraw.
   - **Action**: Prints a message indicating the withdrawal.
   - **Expected Output Format**:
     - "`Withdrawing {amount} NIS from account {account_number}.`"
3. **Open Account Function**:
   - **Name**: `open_account`
   - **Parameters**:
     - `account_type` (str, optional): The type of account to open. Defaults to "Savings" if not provided.
   - **Action**: Generates an automatic account number based on the account type, ensuring account numbers for savings accounts start at 100000 and for checking accounts at 5000000. Prints a message indicating the account type and number. The counters used to generate account numbers should be encapsulated and inaccessible from outside the function. **Returns the account number.**
   - **Expected Output Formats**:
     - "`Opening a {account_type} account with account number {account_number}.`"
4. **Log Transaction Function**:
   - **Name**: `log_transaction`
   - **Parameters**:
     - `action` (str): The type of transaction (e.g., "deposit", "withdraw").
     - `details` (flexible set of arguments): Variable details about the transaction.
   - **Action**: Prints a log message combining the action and details.
   - **Expected Output Format**:
     - "`Log: {action} transaction details: {details}`"

Use this main program to test the functions (to be submitted as well):

```python
def main():
    print("Welcome to the Simple Banking Application!")
    running = True  # Loop condition
    account_number1 = None
    account_number2 = None

    while running:
        print("\nMenu:")
        print("1. Open Savings Account")
        print("2. Open Checking Account")
        print("3. Deposit Money")
        print("4. Withdraw Money")
        print("5. Exit")

        choice = input("Choose an option (1-5): ")

        if choice == '1':
            # Open a Savings account
            account_number1 = open_account()
            log_transaction("open", "Savings", account_number1)

        elif choice == '2':
            # Open a Checking account
            account_number2 = open_account("Checking")
            log_transaction("open", "Checking", account_number2)

        elif choice == '3':
            if account_number1 is None and account_number2 is None:
                print("You must open an account first!")
                continue
            account_number = int(input("Enter your account number: "))
            amount = float(input("Enter amount to deposit: "))
            deposit(account_number, amount)
            log_transaction("deposit", account_number, amount)

        elif choice == '4':
            if account_number1 is None and account_number2 is None:
                print("You must open an account first!")
                continue
            account_number = int(input("Enter your account number: "))
            amount = float(input("Enter amount to withdraw: "))
            withdraw(account_number, amount)
            log_transaction("withdraw", account_number, amount)

        elif choice == '5':
            print("Exiting the application. Goodbye!")
            running = False  # Change condition to exit the loop
        else:
            print("Invalid option. Please try again.")

# Call the main function to run the program
if __name__ == "__main__":
    main()
```

## Execution example:

The text that is not highlighted is text outputted from the given main program.

```
Welcome to the Simple Banking Application!

Menu:
1. Open Savings Account
2. Open Checking Account
3. Deposit Money
4. Withdraw Money
5. Exit
Choose an option (1-5): 1
Opening a Savings account with account number 100000.
Log: open transaction details: Savings, 100000

Menu:
1. Open Savings Account
2. Open Checking Account
3. Deposit Money
4. Withdraw Money
5. Exit
Choose an option (1-5): 2
Opening a Checking account with account number 5000000.
Log: open transaction details: Checking, 5000000

Menu:
1. Open Savings Account
2. Open Checking Account
3. Deposit Money
4. Withdraw Money
5. Exit
Choose an option (1-5): 3
Enter your account number: 100000
Enter amount to deposit: 200
Depositing 200.0 NIS into account 100000.
Log: deposit transaction details: 100000, 200.0

Menu:
1. Open Savings Account
2. Open Checking Account
3. Deposit Money
4. Withdraw Money
5. Exit
Choose an option (1-5): 4
Enter your account number: 5000000
Enter amount to withdraw: 50
Withdrawing 50.0 NIS from account 5000000.
Log: withdraw transaction details: 5000000, 50.0

Menu:
1. Open Savings Account
2. Open Checking Account
3. Deposit Money
4. Withdraw Money
5. Exit
Choose an option (1-5): 1
Opening a Savings account with account number 100001.
Log: open transaction details: Savings, 100001
```

```
Menu:
1. Open Savings Account
2. Open Checking Account
3. Deposit Money
4. Withdraw Money
5. Exit
Choose an option (1-5): 5
Exiting the application. Goodbye!
```

## Question 2:

The East Indian Trade Company operates a fleet of various ships, each designed for specific roles in cargo transport across the seas. The company aims to maximize efficiency and profitability by selecting the most suitable ship for each journey.

The fleet includes:

1. **Galleon**:
   - **Capacity:** 100 cubic meters
   - **Cruising Speed:** 5 knots
   - **Crew Size:** 30
   - **Description:** A large multi-deck ship often used for long voyages and capable of carrying significant cargo.
2. **Caravel**:
   - **Capacity:** 60 cubic meters
   - **Cruising Speed:** 10 knots
   - **Crew Size:** 15
   - **Description:** A small, highly maneuverable ship favored for exploration and swift transport of goods.
3. **Trawler**:
   - **Capacity:** 40 cubic meters
   - **Cruising Speed:** 8 knots
   - **Crew Size:** 6
   - **Description:** Originally designed for fishing, these ships can also be used for transporting cargo efficiently along coastal routes.

Your task is to create a Python script that evaluates which ship is best suited for transporting cargo based on trip duration, fuel consumption, food requirements, and potential revenue.

The program should begin by prompting the user to input the value of the cargo per cubic meter in NIS, the distance of the hop in nautical miles, the price per liter of fuel in NIS, the price per kilogram of food in NIS and the hourly wages per crew member in NIS. With these inputs, the program will then evaluate each ship in the fleet based on its specifications, including capacity and cruising speed. For each ship, the program will calculate the trip duration, total fuel required based on cruising speed, and total food required for the crew, all while considering that the cargo volume will be the remainder of the ship’s capacity after accounting for the space needed for fuel and food.

As the calculations are performed, the program will determine the potential revenue based on the cargo volume and assess the total costs, which will include fuel costs, food costs, and crew wages. After processing these metrics for all ships, the program will output a detailed summary for each vessel. If a ship's net profit is negative, the program will indicate that the ship is not suitable for the trip without printing the full summary. Otherwise, the summary will include the ship’s trip duration, cargo capacity, fuel and food requirements, total cost, potential revenue, and net profit. Finally, the program will identify and display the best choice of ship based on the highest net profit, concluding with a clear recommendation for the most efficient vessel for the journey.

You may assume the following constants and formulae:

- Food consumption rate: 12kg per crew member a day.
- Fuel consumption rate: For an hour of travel at a given speed S a ship of capacity C liters consumes $\frac{C \times S}{6000}$ liters of fuel.
- Every crew member gets the same wages.
- Every cubic meter contains 1000 liters.
- A kilogram of food takes up half a liter.
- A kilogram of cargo takes up 2 liters.
- A knot is 1 nautical mile an hour.

## Functions to Implement:

1. **Function to Calculate Trip Duration:**
    - **Name:** `calculate_trip_duration(distance, speed)`
    - **Parameters:**
        - `distance`: The total distance of the hop in nautical miles.
        - `speed`: The cruising speed of the ship in knots.
    - **Returns:** The duration of the trip in hours.
    - **Description:** Computes duration by dividing the distance by the speed.
2. **Function to Calculate Fuel Requirement:**
    - **Name:** `calculate_fuel_required(distance, speed, capacity)`
    - **Parameters:**
        - `distance`: The total distance of the hop in nautical miles.
        - `speed`: The cruising speed of the ship in knots.
    - **Returns:** The total amount of fuel required for the trip.
    - **Description:** Computes the fuel required based on speed, distance and ship capacity.
3. **Function to Calculate Food Requirement:**
    - **Name:** `calculate_food_required(duration, food_consumption_rate, crew_size)`
    - **Parameters:**
        - `duration`: The duration of the trip in hours.
        - `food_consumption_rate`: The amount of food consumed per crew member per hour.
        - `crew_size`: The number of crew members.
    - **Returns:** The total amount of food required for the trip.

   - **Description:** Multiplies duration by food consumption rate and crew size.
4. **Function to Calculate Revenue:**
   - **Name:** `calculate_revenue(cargo_volume, price_per_cubic_meter)`
   - **Parameters:**
     - `cargo_volume`: The volume of cargo delivered.
     - `price_per_cubic_meter`: The price at which the cargo is sold.
   - **Returns:** The total revenue generated from the cargo.
   - **Description:** Computes the revenue based on the cargo volume.
5. **Function to Calculate Total Cost:**
   - **Name:** `calculate_total_cost(fuel_cost, food_cost, wages)`
   - **Parameters:**
     - `fuel_cost`: The total cost of fuel for the trip.
     - `food_cost`: The total cost of food for the trip.
     - `wages`: The total wages for the crew over the trip duration.
   - **Returns:** The total cost incurred during the trip.
   - **Description:** Adds the costs of fuel, food, and wages.
6. **Function to Display Ship Summary:**
   - **Name:** `display_ship_summary(ship_name, cargo_volume, duration, fuel, food, total_cost, revenue, net_profit)`
   - **Parameters:**
     - `ship_name`: The name of the ship.
     - `cargo_volume`: The volume of cargo that can be transported.
     - `duration`: The total duration of the trip.
     - `fuel`: The total amount of fuel required.
     - `food`: The total amount of food required.
     - `total_cost`: The total cost incurred.
     - `revenue`: The total revenue generated.
     - `net_profit`: The profit made from the trip.
   - **Returns:** None (prints the ship summary only if net profit is non-negative).
   - **Description:** Outputs a summary of each ship's performance.

Notes:

1. You may add more functions as required.
2. In order to print a formatted string that has a decimal number with an exact number of decimal places, use string formatting. This is done by adding an `f` (short for `format`) before the string within the `print` command. The variable is put at the correct place in curly braces followed by a colon, a dot, the number of decimal places and an `f` (short for `float`). For example, in order to print `total_cost` in the function `display_ship_summary`, use the following command:

```
print(f"Total cost: {total_cost:.2f} NIS")
```

(Continued on the next page…)

## Expected Outputs and Input Prompts

- **Welcome Message:**
  - **Header Output:**

```
Welcome to the East Indian Trade Company Fleet Management
System!
```

- **Input Prompts**:
  - **Cargo Value:**

```
Enter the value of the cargo (in NIS per kilogram):
```

  - **Distance:**

```
Enter the distance of the hop (in nautical miles):
```

  - **Fuel Price:**

```
Enter the price of fuel (in NIS per liter):
```

  - **Food Price:**

```
Enter the price of food (in NIS per kilogram):
```

  - **Wages:**

```
Enter the hourly wages for each crew member (in NIS):
```

- **Expected Outputs:**
  - The details of the trip for each ship:

```
Ship Summary for [Ship Name]:
Cargo Capacity: [Cargo Volume] cubic meters
Duration: [X] hours
Fuel required: [A] liters
Food required: [B] kg
Total cost: [C] NIS
Revenue generated: [D] NIS
Net profit: [E] NIS
```

  - If the net profit is negative:

```
The [Ship Name] is not suitable for the trip.
```

  - At the end, print the best choice of ship:

```
The best choice of ship is: [Ship Name] with a net profit of
[E] NIS.
```

## Execution examples:

Example 1:

```
Welcome to the East Indian Trade Company Fleet Management System!
Enter the value of the cargo (in NIS per kilogram): 30
Enter the distance of the hop (in nautical miles): 3000
Enter the price of fuel (in NIS per liter): 9
Enter the price of food (in NIS per kilogram): 25
Enter the hourly wages for each crew member (in NIS): 45
Galleon is not suitable for the trip.

Ship Summary for Caravel:
Cargo Capacity: 60 cubic meters
Duration: 300.00 hours
Fuel required: 30000.00 liters
Food required: 1125.00 kg
Total cost: 500625.00 NIS
Revenue generated: 505312.50 NIS
Net profit: 4687.50 NIS

Ship Summary for Trawler:
Cargo Capacity: 40 cubic meters
Duration: 375.00 hours
Fuel required: 20000.00 liters
Food required: 562.50 kg
Total cost: 295312.50 NIS
Revenue generated: 340156.25 NIS
Net profit: 44843.75 NIS

The best choice of ship is: Trawler with a net profit of 44843.75 NIS.
```

Example 2:

```
Welcome to the East Indian Trade Company Fleet Management System!
Enter the value of the cargo (in NIS per kilogram): 40
Enter the distance of the hop (in nautical miles): 1500
Enter the price of fuel (in NIS per liter): 15
Enter the price of food (in NIS per kilogram): 25
Enter the hourly wages for each crew member (in NIS): 55
Ship Summary for Galleon:
Cargo Capacity: 100 cubic meters
Duration: 300.00 hours
Fuel required: 25000.00 liters
Food required: 2250.00 kg
Total cost: 926250.00 NIS
Revenue generated: 1455000.00 NIS
Net profit: 528750.00 NIS

Ship Summary for Caravel:
Cargo Capacity: 60 cubic meters
Duration: 150.00 hours
Fuel required: 15000.00 liters
Food required: 562.50 kg
Total cost: 362812.50 NIS
Revenue generated: 888750.00 NIS
Net profit: 525937.50 NIS
```

```
Ship Summary for Trawler:
Cargo Capacity: 40 cubic meters
Duration: 187.50 hours
Fuel required: 10000.00 liters
Food required: 281.25 kg
Total cost: 218906.25 NIS
Revenue generated: 594375.00 NIS
Net profit: 375468.75 NIS

The best choice of ship is: Galleon with a net profit of 528750.00 NIS.
```

## Question 3:

In this exercise, you will simulate a simple Dungeons & Dragons (DND) battle between a player and a computer-controlled opponent. The player will choose a weapon from a selection of three options, while the computer will randomly select a weapon. Each weapon has a specific chance to hit and deals damage based on the result of rolling dice (the sides of each die have whole numbers starting from 1).

Each weapon has the following attributes:

- **Weapon Choices:**
    - **Sword:** 70% chance to hit, deals 1d8 damage (1 roll of an 8-sided die).
    - **Axe:** 60% chance to hit, deals 2d6 damage (2 rolls of a 6-sided die).
    - **Morningstar:** 80% chance to hit, deals 1d12 damage (1 roll of a 12-sided die).

The number of health points (HP) for both the player and the computer will be input by the user, with a prompt to enter their health. The computer's health will then be randomized within a specified range.

Implement a Python script that simulates the battle. In order to do so perform the following tasks:

1. **Function to Roll Dice:**
    - Create a function `roll_dice(num_rolls, num_sides)` that takes the number of rolls and the number of sides of the dice as parameters and returns the total result of the rolls.
2. **Function to Determine Hit and Damage:**
    - Create a function `attack(weapon)` that takes a weapon choice as a parameter. This function will:
        - Check if the attack hits based on the weapon's hit chance.
        - Calculate and return the damage if the attack hits.
3. **Main Program:**
    - Implement a main program that:
        - Prompts the player for their weapon choice.
        - Randomly selects a weapon for the computer.
        - Initializes health points for both the player (input by the user) and the computer (randomized).

   - Simulates an attack by both the player and the computer, showing the results of each attack and updating health points accordingly.
   - Continues the battle until either the player or the computer runs out of health points.

Notes:

1. You may add more functions as required.
2. In the beginning of your main program (after the imports) eliminate randomness by adding the following line of code: `random.seed(0)`
   **This is required before submission for successful evaluation!**

## Expected Outputs and Input Prompts

- **Battle Simulation Header:**
  - **Header Output:**

```
Welcome to the DND Battle Simulation!
```

- **Weapon Selection**:
  - **Prompt:**

```
Choose your weapon:
1. Sword (1d8 damage, 70% chance to hit)
2. Axe (2d6 damage, 60% chance to hit)
3. Morningstar (1d12 damage, 80% chance to hit)

Your choice (1-3):
```

  - **Computer's selection:**

```
Computer chooses: [description of weapon like in menu]
```

- **Health Initialization:**
  - **Prompt:**

```
Enter your health points:
```

  - **Expected Output Format:**

```
Your Health: [user input is repeated]
Computer's Health: [randomly generated between 15 and 25]
```

- **Battle Simulation:**
  - **Before the starting of the battle:**

```
--- Battle Begins ---
```

  - **Expected Output Formats for Attacks:**

- When the player attacks:

```
You attack the computer!
```

- When the computer attacks:

```
Computer attacks you!
```

- If either attack hits:

```
Hit! You dealt [damage] damage.
Computer's Health: [remaining health]
```

or

```
Hit! You are dealt [damage] damage.
Your Health: [remaining health]
```

- If either attack misses:

```
Missed!
```

- **End of Battle:**
  - When the player wins**:**

```
--- You win! ---
```

  - When the computer wins:

```
--- You lose! ---
```

**<u>Execution example:</u>**

```
Welcome to the DND Battle Simulation!

Choose your weapon:
1. Sword (1d8 damage, 70% chance to hit)
2. Axe (2d6 damage, 60% chance to hit)
3. Morningstar (1d12 damage, 80% chance to hit)
Your choice (1-3): 1

Computer chooses: Axe (2d6 damage, 60% chance to hit)

Enter your health points: 30

Your Health: 30
Computer's Health: 22
```

```
--- Battle Begins ---

You attack the computer!
Hit! You dealt 5 damage.
Computer's Health: 17

Computer attacks you!
Missed!

You attack the computer!
Hit! You dealt 3 damage.
Computer's Health: 14

Computer attacks you!
Hit! You are dealt 8 damage.
Your Health: 22

You attack the computer!
Missed!

Computer attacks you!
Hit! You are dealt 6 damage.
Your Health: 16

You attack the computer!
Hit! You dealt 7 damage.
Computer's Health: 7

Computer attacks you!
Hit! You are dealt 5 damage.
Your Health: 11

You attack the computer!
Hit! You dealt 6 damage.
Computer's Health: 1

Computer attacks you!
Missed!

You attack the computer!
Hit! You dealt 1 damage.
Computer's Health: 0

--- You win! ---
```

**Good Luck!**