Python Programming Examination

Instructor: Shuvam Banerji Seal
Class: XI
Subject: Computer Science with Python
Time Allotted: 3 Hours
Maximum Marks: 100

General Instructions:

- All questions are compulsory.
- This question paper is divided into three sections: A, B, and C.
- Section A carries 30 marks.
- Section B carries 35 marks.
- Section C carries 35 marks.
- Internal choice is provided in some questions.
- Programming solutions should be written in Python.
- Neatness and clarity of code will be appreciated.

Section A: Control Flow and Algorithmic Thinking (30 Marks)

1. **The Game of Nim (A simplified version):** (10 Marks)

   Two players are playing a game with a pile of 21 stones. They take turns removing either 1, 2, or 3 stones. The player who takes the last stone wins. Write a Python program that simulates a human player playing against the computer. The computer should follow a simple winning strategy: try to leave a number of stones that is a multiple of 4 for the human player.

   Your program should:

   - Start with a pile of 21 stones.
   - Alternate turns between the human and the computer, with the computer playing first.
   - Prompt the human player to enter a valid number of stones to remove (1, 2, or 3).
   - The computer should make its move based on the described strategy. If it cannot make a move to leave a multiple of 4, it should remove 1 stone.

- The game ends when there are no stones left, and the last player to move is declared the winner.

OR

The Collatz Conjecture:

The Collatz conjecture is a famous unsolved problem in mathematics. It states that for any positive integer `n`, the sequence defined as follows will always reach 1:

- If `n` is even, the next number in the sequence is `n / 2`.
- If `n` is odd, the next number is `3n + 1`.

Write a Python program that takes a positive integer as input from the user and generates the Collatz sequence for that number until it reaches 1. The program must also count and display the total number of steps it took to reach 1 and find the highest number reached during the sequence.

2. **Pascal's Triangle:** (10 Marks)

   Write a Python program that takes an integer `n` as input and prints the first `n` rows of Pascal's triangle. Pascal's triangle is a triangular array of binomial coefficients. The entry in the `n`-th row and `k`-th column is given by the formula $\binom{n}{k}$. Each number is the sum of the two numbers directly above it.

   Example for n = 5:

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

   (Hint: You can use nested loops and a list to store the values of the previous row to calculate the current row.)

3. **Special Numbers:** (10 Marks)

   Write a Python program to check if a given positive integer is a "Krishnamurthy Number". A number is a Krishnamurthy number if the sum of the factorials of its digits is equal to the number itself. For example, 145 is a Krishnamurthy number because 1! + 4! + 5! = 1 + 24 + 120 = 145.

   Your program should contain a separate function to calculate the factorial of a number.

OR

Write a Python program to find the sum of the following exponential series up to `n` terms, where the values of `x` and `n` are entered by the user:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + ... + \frac{x^n}{n!}$$

(Hint: You will need a function to calculate factorials and another for powers, or you can compute them iteratively within the loop.)

Section B: Data Structures (35 Marks)

4. **Bookshop Inventory Management (DSA with Lists):** (15 Marks)

   You are tasked with creating a simple inventory management system for a bookshop. The inventory is stored as a list of lists, where each inner list contains the book's title (string), author (string), and quantity (integer).

   `inventory = [["The Lord of the Rings", "J.R.R. Tolkien", 5], ["The Hitchhiker's Guide", "Douglas Adams", 10]]`

   Write a Python program that implements the following functions:

   - `add_book(inventory, title, author, quantity)`: Adds a new book to the inventory. If a book with the same title already exists, it should just update (add to) the quantity.
   - `search_book(inventory, title)`: Searches for a book by its title (case-insensitive) and returns a formatted string with its details (e.g., "Title: The Lord of the Rings, Author: J.R.R. Tolkien, Quantity: 5") if found, otherwise returns a "Book not found" message.
   - `sell_book(inventory, title, num_sold)`: Decreases the quantity of a book when it's sold. Ensure you handle cases where the book is not found or there isn't enough stock to sell (the quantity should not go below zero).

   OR

   Sparse Matrix Representation:

   A sparse matrix is a matrix where most elements are zero. Storing it as a 2D list is inefficient. A better way is to store only the non-zero elements. Represent a sparse matrix using a list of lists (a triplet representation), where each inner list contains `[row, column, value]` for a non-zero element.

   Write a Python program that implements the following:

   - A function `convertToSparse(dense_matrix)` that takes a standard dense matrix (list of lists) as input and returns its sparse matrix representation (triplet form).
   - A function `getElement(sparse_matrix, num_rows, num_cols, row, col)` that takes the sparse matrix, the dimensions of the original matrix, and a specific `row` and `col` index. It should retrieve the value of the element at that position. If the element's triplet is not found in the sparse representation, it should return 0.
   - Demonstrate your functions with a sample 4x4 dense matrix containing at least 10 zero elements.

5. **Advanced Text Analysis:** (10 Marks)

   You are given a paragraph of text. Write a Python program to perform a detailed analysis without using any external libraries.

Text: "Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with its notable use of significant indentation. Python is dynamically-typed and garbage-collected."

Your program must:

(a) Create a dictionary to store the frequency of each word (case-insensitive).

(b) Identify and print all unique words that are longer than 8 characters.

(c) Find and print the word that appears most frequently in the text.

(d) Replace all occurrences of "Python" with "PY" and print the modified text.

6. **Student Performance Report (Dictionaries and Modules):** (10 Marks)

   You have a dictionary containing student names as keys and a list of their marks in five subjects as values.

   `student_data = {"Alice": [88, 92, 80, 76, 85], "Bob": [90, 85, 84, 91, 88], "Charlie": [78, 80, 82, 79, 81]}`

   Write a Python program that iterates through this dictionary and for each student, calculates and prints:

   - Their name.
   - Their average mark.
   - The median of their marks. You **must** import and use the `median()` function from the `statistics` module for this calculation.

Section C: Advanced Problems and Applications (35 Marks)

7. **The Perceptron: Simulating a Neuron:** (18 Marks)

   A perceptron is the foundational block of neural networks. It's a simple model of a neuron that can make a binary decision (output 1 for "yes" or 0 for "no"). This is a guided question to help you build one.

   The Perceptron's Logic:

   (a) It receives multiple binary inputs (a list of 0s and 1s).

   (b) Each input has an associated `weight` (a number representing its importance).

   (c) The perceptron calculates the weighted sum of its inputs: `sum = (input1 * weight1) + (input2 * weight2) + ...`

   (d) If this `sum` is greater than a `threshold` value, the perceptron "fires" (outputs 1); otherwise, it outputs 0.

   Your Task:

   a) (12 Marks) Write a Python function `perceptron(inputs, weights, threshold)` that simulates this logic.

      - `inputs`: A list of binary values (e.g., `[1, 0, 1]`).
      - `weights`: A list of numerical weights (e.g., `[0.7, 0.5, 0.6]`).
      - `threshold`: A single numerical value.

      The function should first check if the length of `inputs` and `weights` are the same. If not, it should return an informative error message string. Otherwise, it should calculate the weighted sum and return 1 or 0 based on the comparison with the threshold.

   b) (6 Marks) Demonstrate how your perceptron function can be used to model a logical **AND gate**. An AND gate takes two inputs and outputs 1 only if both inputs are 1. Show the `weights` and `threshold` you would use, and test your function with all four possible binary inputs: `[0, 0]`, `[0, 1]`, `[1, 0]`, and `[1, 1]`.

8. **The Sieve of Eratosthenes:** (17 Marks)

   The Sieve of Eratosthenes is a highly efficient ancient algorithm for finding all prime numbers up to a specified integer `n`.

   The Algorithm:

   (a) Create a boolean list `is_prime` of size `n + 1`, and initialize all its entries to `True`. `is_prime[i]` will be `False` if `i` is not a prime, and `True` otherwise. Mark `is_prime[0]` and `is_prime[1]` as `False`.

   (b) Start with the first prime number, `p = 2`.

   (c) While `p * p <= n`:

      - If `is_prime[p]` is still `True`, then it is a prime.
      - Update all multiples of `p` greater than or equal to `p*p` to `False`. (i.e., `is_prime[p*p]`, `is_prime[p*p + p]`, ... should be `False`).

- Move to the next number `p`.

(d) After the loop, all indices `i` for which `is_prime[i]` is `True` are prime numbers.

Write a Python program that implements the Sieve of Eratosthenes to find and print all prime numbers up to a number `n` provided by the user.

OR

The Josephus Problem:

The Josephus problem is a classic mathematical puzzle of survival. There are `n` people standing in a circle, numbered 1 to `n`. The counting out begins at person number 1 and proceeds around the circle. In each step, `k-1` people are skipped and the `k`-th person is executed. The procedure is repeated with the remaining people, starting from the person next to the one who was just executed, until only one person remains.

Write a Python program that takes the number of people `n` and the step size `k` as input and returns the number of the sole survivor.

Example: If `n = 7` and `k = 3`, the people are eliminated in the order `3, 6, 2, 7, 5, 1`, and the survivor is `4`.

Hint: A list or a list-like data structure is ideal for representing the circle of people. The modulo operator will be very helpful for wrapping around the circle.