SECTION A: 1.1 Is my number prime?

Description: A prime number is a natural number greater than 1 that cannot be formed by multiplying two smaller natural numbers. For example, 5 is prime because it cannot be factored into smaller numbers (other than 1 × 5). However, 4 is composite because it can be factored into 2 × 2.

Task: Write a function that takes any number as input and returns true if the number is prime and false otherwise.

Examples

- Input -> 2
- Output -> true
- Input -> 10
- Output -> false

Pseudo-Code:

1. Create a function checkForPrimeNum that accepts an integer called possiblePrimeNumber.
2. Check if possiblePrimeNumber is less than or equal to 1:
    - If true, return false because prime numbers must be greater than 1.
3. Loop through all numbers starting from 2 up to the square root of possiblePrimeNumber:
    - If possiblePrimeNumber is divisible by any of these numbers, return false (the number is composite).
4. If no divisors are found in the loop, return true because the number is prime.

Explanation:

- Function Type: The checkForPrimeNumP function is a simple Boolean function (Bool) that returns either true or false. This choice is appropriate because we are checking a condition (whether a number is prime or not).
- Parameters and Variables:
    - possiblePrimeNumber: The input integer to check if it is a prime number.
    - divisor: A variable used within the loop to check divisibility.
- Constants vs. Variables:
    - Constants: The function parameter possiblePrimeNumber is treated as a constant since its value does not change during execution.
    - Variables: divisor is a variable because it is incremented during the loop.

SECTION B: 1.3 Is my password correct? (Uses closures)

Description: A common feature in online forms is password validation. A valid password must contain at least one uppercase letter, one special symbol (e.g., @, #, $, etc.), and at least 5 characters.

Task: Write a function that checks if a given password meets the following criteria:

- Contains at least one uppercase letter.
- Contains at least one special symbol from the set: !@#$%^&*
- Has at least 5 characters

The function should return true if the password is valid and false otherwise.

Example:

- Input -> "Password123!" •
Output -> true
- Input -> "password123" •
Output -> false

Pseudo-Code:

1. Define a function correctPassword that accepts a string userPassword.
2. Create two closures:
    - hasUppercaseLetter checks if userPassword contains at least one uppercase letter.
    - hasSpecialChar checks if userPassword contains at least one special character.
3. Check if:
    - The length of userPassword is greater than or equal to 5.
    - hasUppercaseLetter returns true.
    - hasSpecialChar returns true.
4. Return true if all conditions are satisfied, otherwise return false.

Explanation:

- Function Type: correctPassword is a Boolean function that returns true or false based on password validation. Closures are used to break down the problem into smaller reusable checks.
- Parameters and Variables:
    - userPassword: The string representing the password to be validated.
    - hasUppercaseLetter, hasSpecialChar: Closures that check if the password contains necessary components.
- Constants vs. Variables:
    - Constants: The input userPassword is treated as a constant.
    - Variables: Closures like hasUppercaseLetter act as constants (not modified once defined)

SECTION C: 1.1 Longest subsequence

Description: Given an unsorted array of numbers, find the longest sequence of consecutive elements.

Task: Write a function that takes an array of numbers and returns two outputs: the longest consecutive subsequence and its length. A consecutive sequence doesn't necessarily require that our elements be in ascending or descending order.

Examples:

- Input -> [9, 4, 10, 2, 7, 3, 5, 1]
- Output1-> 1, 2, 3, 4, 5 //The subsequence is the longest sequence of consecutive elements.
- Output2-> 5 // This is the length of the subsequence

Pseudo-Code:

1. Create a function findLongestSubSeq that accepts an array unsortedNumbers.
2. Convert the array into a set to remove duplicates, then sort the set.
3. Loop through the sorted numbers, and for each number:
    - Check if the current number is consecutive with the previous number.
        - If yes: add it to the current sequence.
        - If no: compare the length of the current sequence with the longest one found so far and update accordingly.
4. Return both the longest consecutive sequence & its length.

Explanation:

- Function Type: findLongestSubSeq returns a tuple of an array and an integer. This is ideal since it gives both the longest sequence and its length.
- Parameters and Variables:
    - unsortedNumbers: The array of integers.
    - longestSeq, currentSeq: Arrays that store the consecutive numbers found during the iteration.
- Constants vs. Variables:
    - Constants: The input array unsortedNumbers is treated as a constant, and sortedUniqueNums is a constant after sorting.
    - Variables: longestSeq and currentSeq are updated during the function execution.

SECTION D : 1.1 Rock Paper Scissor (Follows OOP)

Description. Rock Paper Scissor is a game played between two people. Each player in this game forms one of three shapes. The winner will be decided as per the given rules:

- Rock vs Scissor -> Rock wins
- Rock vs Paper -> Paper wins
- Paper vs Scissor -> Scissor wins

Task: Create a Swift Playground that simulates the Rock, Paper, Scissors game. In the Playground you will have:

- A string variable will represent the user's choice (e.g., "rock", "paper", or "scissors"). • The computer will randomly select one of the three options.
- Your program will compare the user's choice with the computer's choice and print the outcome:
    - Print "You win!" if the user wins.
    - Print "Computer wins!" if the computer wins.
    - Print "It's a tie!" if both selections are the same

Pseudo-Code:

1. Create a class RockPaperScissors with 2 properties:
    - userMove: Stores the user's choice (rock, paper, or scissors).
    - computerMove: A randomly generated move selected from "rock", "paper", or "scissors".
2. Define an initializer that normalizes userMove to lowercase and assigns a random move to computerMove.
3. Inside the class, create a method gameOutcome:
    - Compare userMove and computerMove based on the game rules.
    - Return one of the following strings based on the comparison:
        - "You Win!" if the user's move beats the computer's move.
        - "Computer Wins!" if the computer's move beats the user's move.
        - "It's a Tie!" if both moves are the same.

Explanation:

- Function Type: The gameOutcome() method is a String function that returns the result of the game as a string (e.g., "User Wins!", "Computer Wins!", or "It's a tie!"). It makes sense to use a string data type because the game outcomes are naturally represented as a text message.
- Parameters and Variables:
    - userMove: This is an instance property (a string) that stores the user's move.
    - computerMove: This is an instance property that holds the randomly generated move of the computer.
    - gameOutcome(): A method that compares the userMove and computerMove to determine the winner/outcome of the game.

- Constants vs. Variables:
    - Constants: The values passed into the class initializer (i.e., userMove) and the randomly generated value for computerMove are constants in the sense that they remain unchanged after they are set.
    - Variables: Inside the method, comparisons between moves are dynamic but do not involve the creation of new variables. The properties userMove and computerMove remain fixed after initialization, following the logic of OOP encapsulation.