Iterables & Loops p.II

While Loops

Exercises

Level - Easy

Exercise 1-1

1. Ask the user for a number, n.
2. Use a while loop to count from 1 to n.
3. Print each number.

Exercise 1-2

1. Ask the user for a number, n.
2. Use a while loop to count from n down to 1.
3. Print each number.

Exercise 1-3

1. Ask the user for a number, n.
2. Use a while loop to sum all numbers from 1 to n.
3. Print the total.

Exercise 1-4

1. Ask the user for a number, n.
2. Use a while loop to sum all even numbers from 1 to n (2, 4, 6 …).
3. Print the total.

Exercise 1-5

1. Ask the user for a starting number, n.
2. Use a while loop to double the value of n until it's greater than 1000.
3. Print each doubled value.

Level - Moderate

Exercise 2-1

1. Ask the user for a number, n.
2. Use a while loop to print numbers from 1 to n but skip any number that's divisible by 5 (Tips: You can use the continue statement).

Exercise 2-2

1. Select a number between 1 and 10.
2. Use a while loop to ask the user to guess the number.
3. The loop should continue until the user guesses correctly.
4. Once the user has guessed the number, print how many tries it took.

Exercise 2-3

1. Select a number between 1 and 10.
2. Give the user only 3 attempts to guess the number using a while loop.
3. If they guess correctly within the 3 attempts, print a success message.
4. If they run out of attempts without guessing correctly, print a failure message.

Exercise 2-4

1. Use a while loop to create a calculator that performs the 7 mathematical operations (addition, subtraction …)
2. Ask the user to specify the operation in the following format: “x operator y”.
   *For example, “8 + 9” or “10 - 7” or “9**5” …*
3. After performing an operation, ask the user if they want to continue.
   a. If the user decides to continue, ask for a new operation.
   b. If the user decides not to continue, exit the loop.

Tips: You can use the `eval()` *Python built-in function.*
It will transform a string into Python code.
Thus, `eval("5-9")` *will return -4.*

Exercise 2-5

1. Ask the user for a number, n.
2. Generate the Collatz sequence starting at n, using a while loop to apply the following rules until you reach the number 1:
   a. If the number is even, divide it by 2.
   b. If the number is odd, multiply it by 3 and add 1.
3. Print the sequence.

Exercise 2-6

1. Select a number between 1 and 100.
2. Create two variables that represent the lower and higher boundaries.
3. Allow the user to guess the number (ask one of your classmates).
4. After the user provides a guess, check:
    a. If the guess is correct, congratulate the user and end the game.
    b. If the guess is lower than the number:
        i. Update the lower boundary variable to be one greater than the user's guess (since the user's guess is too low and we want to narrow the range).
        ii. Inform the user that their guess was too low.
    c. If the guess is higher than the number:
        i. Update the higher boundary to be one less than the user's guess (since the user's guess is too high and we want to narrow the range).
        ii. Inform the user that their guess was too high.
5. Provide a hint to the user by telling them the updated boundaries: "The number is between the <low boundary> and <high boundary>."
6. Continue until the user guesses correctly.

Exercise 2-7

1. Create a list of integers that can have any length.
2. Write a while loop that computes the sum of the numbers of the list one by one.
3. The loop should continue adding numbers until a negative number is found.
4. If there’s no negative number in the list, the loop continues until all the numbers of the list have been summed.

Level - Hard

Exercise 3-1

1. Ask the user for a plaintext message (only lowercase letters for simplicity).
2. Using a while loop, encrypt the message by shifting each character's position by its index in the message.

   *For example, “python” becomes “pzvkss” because p is at index 0, y is at index 1 so z is its replacement (... y **z** ...)t, t is at index 2 so v is its replacement (... t u **v** ...), h is at index 3 so k is its replacement (... h i j **k** ...) and so on.*

3. Print the encrypted message.
4. Create another while loop to decrypt the message.

Tips: Use ' abcdefghijklmnopqrstuvwxyz' to find the corresponding letter to replace.

Exercise 3-2

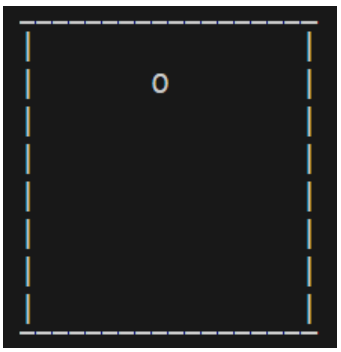
1. Ask the user for a positive number.
2. Use the Babylonian method with while loops to approximate the square root of the number.
   a. Start with an arbitrary guess x0. A good initial guess is half of the number n for which you want to find the square root.
   b. Set a small tolerance value, e.g., 0.0001, to determine the accuracy of the result.
   c. Compute the next approximation: x1 = (x0 + n/x0) / 2.
   d. If the absolute difference between x1 and x0 is smaller than the tolerance, stop the iteration.
   e. Otherwise, set x0 to x1 and repeat the process.
   f. x1 will be an approximation of the square root of n.
3. Print the approximated square root and the number of steps to reach the approximation.

Exercise 3-3

1. Ask the user to input a number n.
2. Find the smallest number that can be divided by each of the numbers from 2 to n without any remainder. Use a while loop to check each number incrementally until you find the smallest multiple.
3. Print the result and the number of steps.

Level - Very Hard

Implement a text-based 2D animation of a ball bouncing around inside a box in the console.



1. Choose a width and height for your "screen" or "box". This will define the area in which the ball can move.
2. Start the ball at the center of this box.
3. Define two directions: horizontal and vertical. Each can have a value of 1 (moving right or down) or -1 (moving left or up).
4. Clear the console to create the effect of animation frames.

   Tips: You can do that by importing the os library at the beginning of the exercise and using the system() function inside of the loop.

   ```
   import os
   os.system('cls')
   ```

5. Print rows of the box using whitespaces. When you reach the row where the ball should be, print the ball's position using a different character (e.g., "O").
6. Based on the ball's current position and direction, determine its next position:
   a. If the ball hits the left or right edge, reverse its horizontal direction.
   b. If the ball hits the top or bottom edge, reverse its vertical direction.
7. Continue updating the ball's position and redrawing the screen in a loop to create the animation effect.

   Tips: You can use an infinite while loop by using `while True:`.
   This way, the ball will never stop bouncing unless you stop the program !

8. Introduce a small delay (e.g., 0.1 seconds) between frames to make the animation visible.

   Tips: You can do that by importing the time library at the beginning of the exercise and using the sleep() function inside of the loop.

   ```
   import time
   time.sleep(0.1)
   ```