Tutorial II - Control Structures for Your Code

Programming with Python

Introduction

Just like in the previous tutorial, you will likely find solutions to most exercises online. However, I still strongly encourage you to work on these exercises independently without searching for answers. Understanding someone else's solution is very different from developing your own.

Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities. If you encounter difficulties, review the lecture materials, experiment with different approaches, and don't hesitate to ask for clarification during class discussions. Don't worry, I won't repeat this section again and again.

Decoding secret messages with methods

In this exercise, we'll practice string manipulation and slicing. You'll work with a secret message encoded using various string operations and learn how to apply some new methods like `title()`, `replace()` and `count()`. By following a series of steps, you'll gradually decode the message. Let's begin with the encoded message and work through each decoding step:

```
# Decode a secret message by following a series of instructions.
# Each instruction requires you to use different operations and methods.
# The encoded message is:
secret_message = ".tnega terces a sa slliks ym sevorpmi erutcel oN"

# a) TODO: Reverse the string
# Hint: You can use slicing to reverse a string
# Your code here

# b) TODO: Remove the period at the end
# Your code here

# c) TODO: Replace 'No' with 'This'
# Your code here

# d) TODO: Convert the string to title case
# -> Capitalize the first letter of each word
# Your code here

# e) TODO: Add an exclamation mark at the end of the sentence
# Your code here

# f) TODO: Count how many times the letter 's' appears in the decoded message (upper and lower case)
# Your code here
```

> **Tip**
>
> Use the `help()` function to get more information about a method. For example, typing `help(str.replace)` in the shell will show the documentation for the `replace()` method. To exit the documentation, press `q`.

Classifying temperatures

In this exercise, we'll practice using conditional statements to classify temperatures into different categories. We'll create a program that takes a temperature input from the user and provides a classification based on the temperature range. This exercise will help you understand how to use if-elif-else statements, handle user input, and implement a simple loop for program repetition.

```
# Use if-elif-else statements to classify the temperature
# Below 0: "Freezing"
# 0-10: "Cold"
# 11-20: "Cool"
# 21-30: "Warm"
# Above 30: "Hot"

# a) TODO: Formulate a small programm that asks the user for a temperature and then prints the correspo
# Your code here

# b) TODO: Test your code with -5, 15 and 35 as temperature
# Hint: You dont't need any new code here, just run the code from the previous step with different inpu

# c) TODO: Add a feature that asks the user if they want to continue and then repeats the programm if t
# If the user does not want to continue, the programm should end.
# Your code here
```

Number guessing game

In this exercise, we'll create an interactive number guessing game for two players. This game will help you practice using loops, conditional statements, and user input handling. You'll also learn how to implement a simple game logic and manage player turns. This exercise will reinforce your understanding of control structures and basic development concepts in Python.

```
# Create a number guessing game with 2 players. The first player is the game master and the second playe
# Start by asking for their names. Then, ask the game master to input the secret number between 1 and 20
# Make sure, that the number is not shown by adding at least 25 new lines after the input prompt. Then,
# If the guess is too high or too low, provide feedback in order to help the player and let the player g
# If the guess is correct, congratulate the player and end the game.

# a) TODO: Your task list in more detail:
# - Ask for the names of the game master and the player
# - Implement the main game loop using a while loop
# - Ask the game master for the secret number from 1-20
# - Hint: Use `print("\n" * 25)` to add 25 new lines
# - Ask the player for a guess
# - Compare the guess to the secret number
# - Provide feedback (too high, too low, or correct)
# - Keep track of the number of guesses
# - End the game when the correct number is guessed
# - Print a congratulatory message with the number of guesses taken
# - Use the given structure below to implement the game.

# Ask for the names of the game master and the player
game_master = input("Enter the name of the game master: ")
player = input("Enter the name of the player: ")

# Ask the game master for the secret number
while True:
    secret_number = int(input(f"{game_master}, enter the secret number between 1 and 20: "))
    if secret_number < 1 or secret_number > 20:
        print("Please enter a valid number.")
    else:
        break

print("\n" * 25)  # Add 25 new lines to hide the secret number

# Initialize variables
guesses = 0
correct = False
```

```
# Main game loop
while not correct:
    # Ask the player for a guess
    # Add the code here to ask the player for a guess
    guesses += 1

    # Compare the guess to the secret number
    # Add code here to compare the guess to the secret number
    # If the guess is correct, set correct to True in order to end the game.

# Print congratulatory message
# Add the code here to print a congratulatory message.

# b) TODO: Add a feature to allow both players to play again, this time switching the roles with the ga
# If the player wants to play again, the game should reset and the player should be able to play again,
# If the player does not want to play again, the game should end. Hint: Make sure to inform the players
# Your code here

# c) TODO: Bonus Task - Enhance the game by adding difficulty levels
# Implement three difficulty levels: Easy (1-10), Medium (1-20), and Hard (1-30)
# Ask the game master to choose a difficulty level before entering the secret number
# Adjust the range of possible numbers based on the chosen difficulty
# Provide a different number of allowed guesses for each difficulty level:
#   - Easy: 5 guesses
#   - Medium: 10 guesses
#   - Hard: 15 guesses
# If the player doesn't guess the number within the allowed guesses, end the game and reveal the secret
# Your code here
```

That’s it!

You can find the solutions to these exercises online in the associated GitHub repository, but we will also quickly go over them in next week’s tutorial. To access the solutions, click on the Guthub button on the lower right and search for the folder with today’s lecture and tutorial. Alternatively, you can ask ChatGPT or Claude to explain them to you. Remember, the goal is not just to complete the exercises, but to understand the concepts and improve your programming abilities.