Tutorial VI - Using Modules and Packages

Programming with Python

Scrambled Word Game

In this task, we will implement a simple game where the user has to guess a scrambled word. Again, we will use the methods we have learned so far: string methods, list methods, random module, and conditional statements.

```
# TODO: Write the code for a small game
# The game should work as follows:
# - The computer selects a word from a list of words
# - The computer scrambles the word and shows it to the user
# - The user has to guess the word and has three guesses
# - The user gets feedback on whether their guess is correct or not
# - The computer tells the user how many guesses they used

# List of words to choose from for the game
words = ["python", "programming", "computer", "algorithm", "database"]

# Hint: Use the random.choice() function to select a word from the list and try to come
 ↪  up with the rest of the code yourself!

import random

def scramble_word(word):
    # Convert the word to a list of characters, shuffle them, and join back into a string
    chars = list(word)
    # Your code to shuffle the list here!
    return ''.join(chars)

# Your code here
```

Custom Modules

Here we will practice how to create our own modules and use them in our scripts. Our objective is to create a small module that contains a few functions for a calculator.

```
# TODO: Create a new module called calculator.py
# The module should contain at least four functions:
# add, subtract, multiply, and divide
# Afterward, the following code should work:

import calculator as calc

assert calc.add(2, 3) == 5
assert calc.subtract(5, 1) == 4
assert calc.multiply(4, 7) == 28
assert calc.divide(10, 2) == 5

print("Wonderful, all tests passed!")
```

Random Package

Do you remember the random number guessing game from previous tutorials? We will now use the random package to implement a small version of it without any game master.

```
# TODO: Implement a random number guessing game.
# The game should work as follows:
# - The computer selects a random number between 1 and 10
# - The user has to guess the number and has three guesses
# - The computer tells the user whether their guess is too high, too low, or correct
# - The computer should also print how many guesses the user made before guessing the
 ↪  number correctly
# - It should also ask the user if they want to play again
# Your code here
```

Regular Expressions

In the last task, we will practice how to use regular expressions to match patterns in text. Imagine you are an overworked consultant and you have the task to extract all dates from a text.

```
# TODO: Try to extract all dates in the text below:
text = """
Here is a date: 12.03.2024. Here is another one: 01/01/2024. And here is the third one:
 ↪  2024-01-01.
Let's add more: 15-04-2023, 16/05/2023, and 17.06.2023.
In July, we have 18/07/2023 and 19-07-2023.
August brings us 20.08.2023 and 21/08/2023.
September dates include 22-09-2023 and 23.09.2023.
October has 24/10/2023 and 25-10-2023.
November features 26.11.2023 and 27/11/2023.
Finally, December rounds out the year with 28-12-2023 and 29.12.2023.

Continuing with more dates:
01-01-2025, 02/02/2025, 03.03.2025, 04-04-2025, 05/05/2025, 06.06.2025,
07-07-2025, 08/08/2025, 09.09.2025, 10-10-2025, 11/11/2025, 12.12.2025.

Even more dates:
13-01-2026, 14/02/2026, 15.03.2026, 16-04-2026, 17/05/2026, 18.06.2026,
19-07-2026, 20/08/2026, 21.09.2026, 22-10-2026, 23/11/2026, 24.12.2026.

And some more:
25-01-2027, 26/02/2027, 27.03.2027, 28-04-2027, 29/05/2027, 30.06.2027,
01-07-2027, 02/08/2027, 03.09.2027, 04-10-2027, 05/11/2027, 06.12.2027.
"""

# Your code here
```

A better Password Strength Checker

In this task, we will implement an improved password strength checker. We can use and practice all methods we have learned so far: string methods, regular expressions, and conditional statements.

```
# TODO: Implement a new password strength checker
# Check for:
# - Minimum length of 8 characters
# - At least one uppercase letter
# - At least one lowercase letter
# - At least one digit
# - At least one special character (!@#$%^&*)
# Return "Weak" if no criteria is met, "Medium" if at least 3 criteria are met, "Strong"
 ↪  if all criteria are met

# Test your function
print(check_password_strength("abc123"))  # Should return "Weak"
print(check_password_strength("Str0ngP@ssw0rd"))  # Should return "Strong"
print(check_password_strength("Password123"))  # Should return "Medium"
```

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 Github 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.