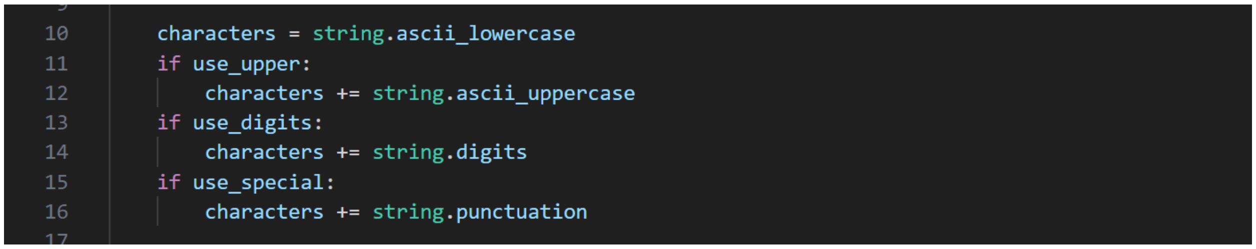
Functional Programming

1. Only final data structures.

My code does not use mutable data structures. Strings and lists are created and modified, but there are no in-place modifications. Strings and tuples in Python are examples of immutable data structures. Once a string or tuple is created, its content cannot be modified. Any operation that seems to modify it actually creates a new object. The use of immutable data structures (strings, tuples) contributes to immutability.

For example, I use "**characters"** which is a string, which is immutable. My code doesn't modify the existing string; instead, it creates a new string by concatenating different character sets based on the conditions.

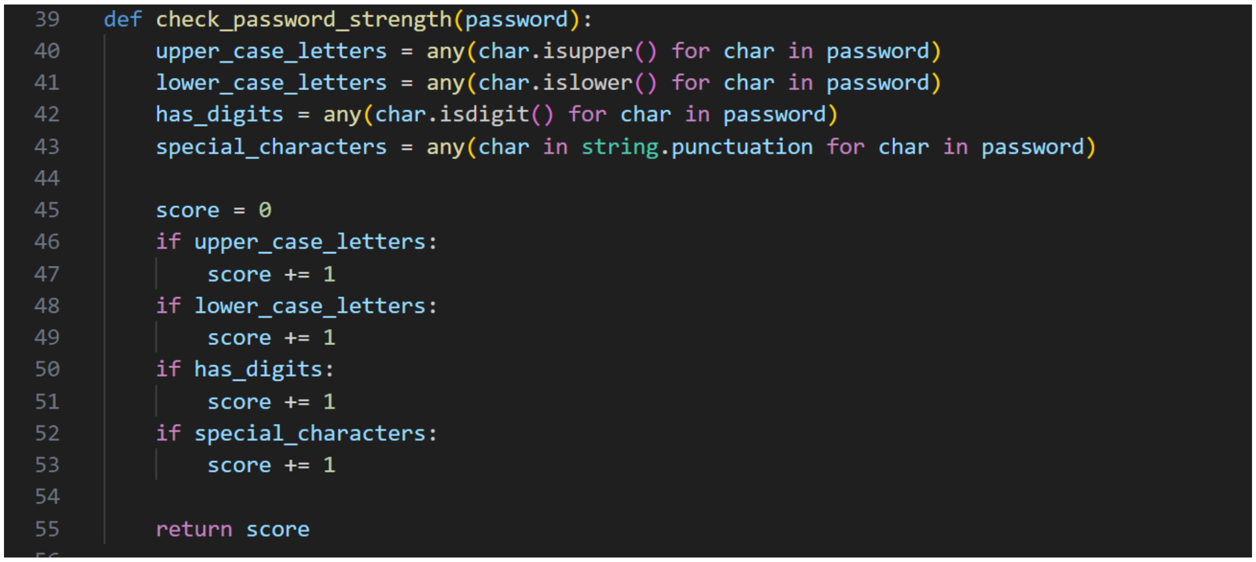


```
    characters = string.ascii_lowercase
    if use_upper:
        characters += string.ascii_uppercase
    if use_digits:
        characters += string.digits
    if use_special:
        characters += string.punctuation
```

2. Side-effect-free functions.

I use Functions like "**generate_password"**, "**generate_pronounceable_password"**, "**check_password_strength**", and "**generate_passphrase**" which are pure functions and don't have side effects. They take input parameters and return results without modifying external state.

For example, given "**check_password_strength**" function:

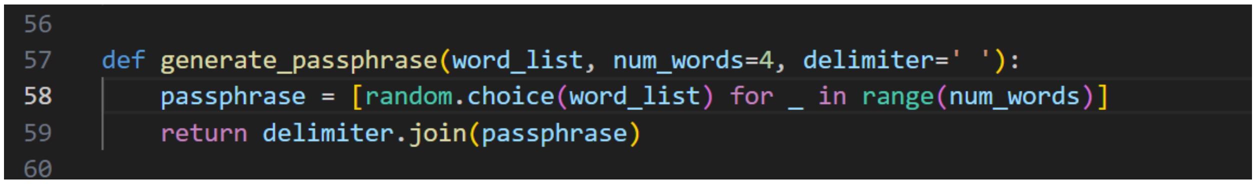


```python
def check_password_strength(password):
    upper_case_letters = any(char.isupper() for char in password)
    lower_case_letters = any(char.islower() for char in password)
    has_digits = any(char.isdigit() for char in password)
    special_characters = any(char in string.punctuation for char in password)

    score = 0
    if upper_case_letters:
        score += 1
    if lower_case_letters:
        score += 1
    if has_digits:
        score += 1
    if special_characters:
        score += 1

    return score
```

- The function doesn't modify any external state or variables. It doesn't interact with the global state or modify any variables outside its scope.
- The function doesn't perform any input/output operations like reading or writing to files, databases, or the console.
- The function doesn't have any observable side effects; it only returns the calculated strength score.

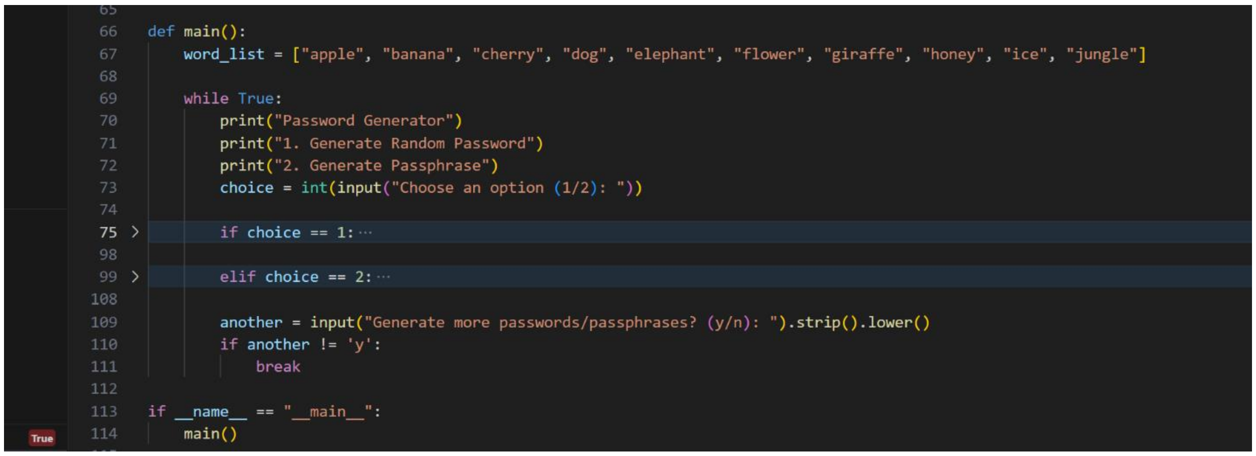
3. The use of higher-order functions.

Functions like “**generate_passphrase”** take another function “**random. choice**” as an argument. This is an example of a higher-order function.



```python

def generate_passphrase(word_list, num_words=4, delimiter=' '):
    passphrase = [random.choice(word_list) for _ in range(num_words)]
    return delimiter.join(passphrase)

```

Another example of my code where a higher-order function is indirectly involved, is the “**main**” function that is shown below:

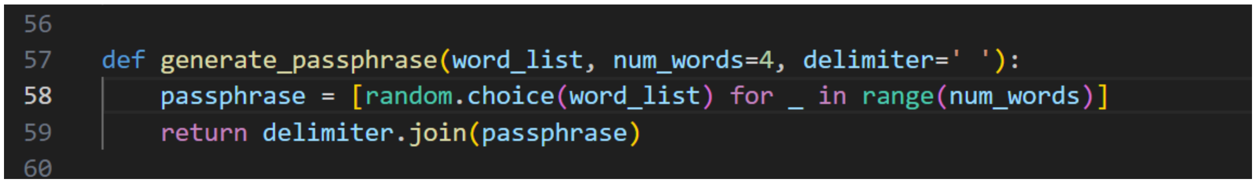


```python
def main():
    word_list = ["apple", "banana", "cherry", "dog", "elephant", "flower", "giraffe", "honey", "ice", "jungle"]

    while True:
        print("Password Generator")
        print("1. Generate Random Password")
        print("2. Generate Passphrase")
        choice = int(input("Choose an option (1/2): "))

        if choice == 1:···

        elif choice == 2:···

        another = input("Generate more passwords/passphrases? (y/n): ").strip().lower()
        if another != 'y':
            break

if __name__ == "__main__":
    main()
```

In this part of my code, “**input”** is a higher-order function because it takes a function “**strip”** as an argument. The “**strip**” function is applied to remove leading and trailing whitespaces from the user's input.

4. Functions as parameters and return values.

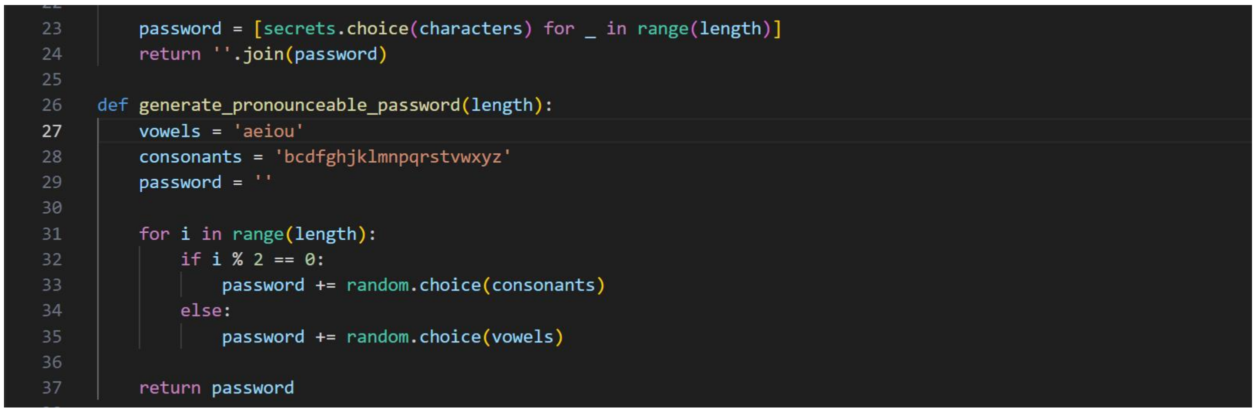
The “**generate_passphrase**” function takes a list of words and a function “**random.choice**” as parameters. It generates a passphrase based on these parameters. And the “**generate_passphrase”** function returns a string generated based on the input parameters.



```python
def generate_passphrase(word_list, num_words=4, delimiter=' '):
    passphrase = [random.choice(word_list) for _ in range(num_words)]
    return delimiter.join(passphrase)
```

5. Use closures / anonymous functions.

I use functions like “**secrets.choice”** and “**random.choice”**, which can be considered as functions passed as parameters. These functions encapsulate behavior and contribute to the closure concept.



```
    password = [secrets.choice(characters) for _ in range(length)]
    return ''.join(password)

def generate_pronounceable_password(length):
    vowels = 'aeiou'
    consonants = 'bcdfghjklmnpqrstvwxyz'
    password = ''

    for i in range(length):
        if i % 2 == 0:
            password += random.choice(consonants)
        else:
            password += random.choice(vowels)

    return password
```