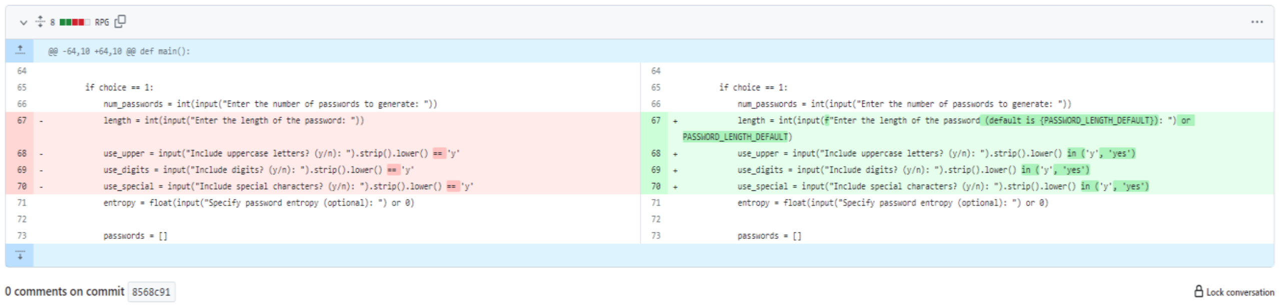
Clean Code Development (CCD)

1. Error Handling

The code includes error handling in the “main()”, including catching “ValueError” and then printing meaningful error messages to the user. So, the user could find what error exactly happened.

2. User Interaction

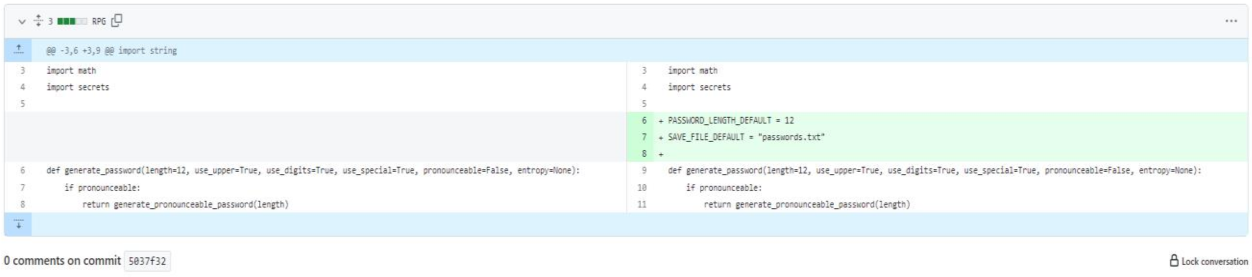
The user interaction is handled in the main function, making it clear how the script interacts with the user. Simplify Boolean expressions for better readability. For example, instead of “strip().lower() == 'y'”, consider using “strip().lower() in ('y', 'yes')”.



```diff
@@ -64,10 +64,10 @@ def main():

        if choice == 1:
            num_passwords = int(input("Enter the number of passwords to generate: "))
-           length = int(input("Enter the length of the password: "))
-           use_upper = input("Include uppercase letters? (y/n): ").strip().lower() == 'y'
-           use_digits = input("Include digits? (y/n): ").strip().lower() == 'y'
-           use_special = input("Include special characters? (y/n): ").strip().lower() == 'y'
+           length = int(input(f"Enter the length of the password (default is {PASSWORD_LENGTH_DEFAULT}): ") or PASSWORD_LENGTH_DEFAULT)
+           use_upper = input("Include uppercase letters? (y/n): ").strip().lower() in ('y', 'yes')
+           use_digits = input("Include digits? (y/n): ").strip().lower() in ('y', 'yes')
+           use_special = input("Include special characters? (y/n): ").strip().lower() in ('y', 'yes')
            entropy = float(input("Specify password entropy (optional): ") or 0)

            passwords = []
```

3. Consistent Naming

Descriptive and consistent naming throughout the code not only makes it easier to read your code but also means that everyone can easily understand what your code means. In addition, make your code more maintainable for yourself by aiding in understanding the purpose of each variable, function, and other objects. On the other hand, some strings that are used multiple times in your code could be defined as constants for better maintainability (e.g., "passwords.txt" in my code).



3 RPG

```
@@ -3,6 +3,9 @@ import string
```

```
import math
import secrets

def generate_password(length=12, use_upper=True, use_digits=True, use_special=True, pronounceable=False, entropy=None):
    if pronounceable:
        return generate_pronounceable_password(length)
```

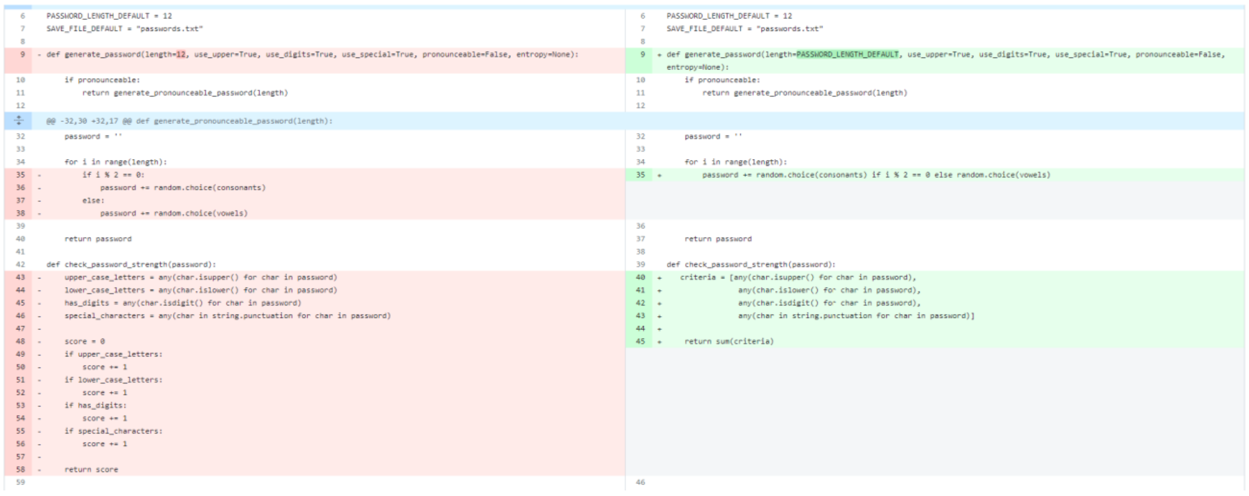
```
import math
import secrets

+ PASSWORD_LENGTH_DEFAULT = 12
+ SAVE_FILE_DEFAULT = "passwords.txt"
+
def generate_password(length=12, use_upper=True, use_digits=True, use_special=True, pronounceable=False, entropy=None):
    if pronounceable:
        return generate_pronounceable_password(length)
```

0 comments on commit 5037f32

Lock conversation

4. Reducing the Amount of Code

The code in "check_password_strength" has some complexities and could be simplified. It is replaced by readable and less confusing code.

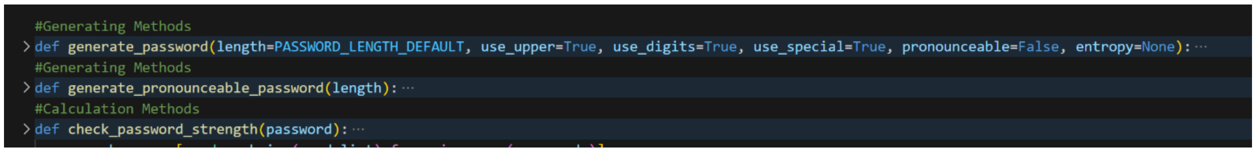


```
PASSWORD_LENGTH_DEFAULT = 12
SAVE_FILE_DEFAULT = "passwords.txt"

- def generate_password(length=12, use_upper=True, use_digits=True, use_special=True, pronounceable=False, entropy=None):
    if pronounceable:
        return generate_pronounceable_password(length)

@@ -32,30 +32,17 @@ def generate_pronounceable_password(length):
    password = ''

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

    return password

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

```

```
PASSWORD_LENGTH_DEFAULT = 12
SAVE_FILE_DEFAULT = "passwords.txt"

+ def generate_password(length=PASSWORD_LENGTH_DEFAULT, use_upper=True, use_digits=True, use_special=True, pronounceable=False,
entropy=None):
    if pronounceable:
        return generate_pronounceable_password(length)

    password = ''

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

    return password

def check_password_strength(password):
+  criteria = [any(char.isupper() for char in password),
+              any(char.islower() for char in password),
+              any(char.isdigit() for char in password),
+              any(char in string.punctuation for char in password)]
+
+   return sum(criteria)

```

5. Commenting

Providing good comments for each section, especially for the more complex ones, increases the readability of the code, and makes code maintenance much easier, as well as helps to find bugs faster.



```
#Generating Methods
def generate_password(length=PASSWORD_LENGTH_DEFAULT, use_upper=True, use_digits=True, use_special=True, pronounceable=False, entropy=None):…
#Generating Methods
def generate_pronounceable_password(length):…
#Calculation Methods
def check_password_strength(password):…
```