Clean Code Cheat Sheet

1. Refactoring:

- Instead of trying to clean dirty code, it is sometimes better to refactor the code to have a fresh start

```
+ #Initialize passwords notebook with placeholder info
+ notebook = {
+     'Websites': {
+         'amazon.com':{
+         'Username': 'johndoe',
+         'Password':'pass123'
+         }
+         },
+         'Emails': {
+             'gmail': {
+             'Username': 'johndoe@gmail.com',
+             'Password':'pass321'
+             }
+             }
+         }
+
+ while True:
+     #Continously print the notebook
+     print('\nCurrent credentials in notebook:')
+     for cat, name in notebook.items():
+         print(f'---Category: {cat}---')
+         for key, value in name.items():
+             print(f' --{key}--')
+             for user, passw in value.items():
+                 print(f'   -{user}: {passw}')
+
+     #Print the menu and ask for input
+     print('Menu: \n1. Add New Entry \n2. Remove Entry \n3. Add New Category \n4. Remove Category \n5. Exit')
+
+     choice = input('Enter your choice: ')
+
```

```
+ #Notebook as class
+ class Notebook:
+     def __init__(self, book) -> None:
+         self.book = book
+ 
+     def __str__(self) -> str:
+         return f'{self.book}'
+ 
+     def print_book(self):
+         print('\nCurrent credentials in notebook:\n')
+         for categ, name in self.book.items():
+             print(f'---Category: {categ}---')
+             for key, value in name.items():
+                 print(f' --{key}--')
+                 for user, passw in value.items():
+                     print(f'   -{user}: {passw}')
+ 
+     def add_entry(self):
+         # Add new entry
+         category = input('Enter the category (e.g., Websites, Emails): ')#Take out of class?
+         # Check if category exists
+         #Take check out of class?
+         if category in self.book:
+              site = input('Enter the name of the website or service: ')
+              username = input('Enter the username: ')
+              password = input('Enter the password: ')
+              # Add the new entry to the dictionary
+              self.book[category][site] = {'Username': username, 'Password': password}
+              print('New entry added successfully.')
+         else:
+              print(f'Category {category} not found')
```

2. Always try to explain yourself in code:

- As the name suggests, the code must be easily comprehended using naming conventions and comments. These comments must be straightforward and simple.

```
def encrypt_and_write(self, key):
    # Convert dictionary to JSON string and encrypt with user's key
    # Convert dictionary to JSON
    json_string = json.dumps(self.book)
    # Encrypt the JSON
    cipher_suite = Fernet(key)
    encrypted_data = cipher_suite.encrypt(json_string.encode())
    # Write encrypted data to file
    with open('Notebook.txt', 'wb') as file:
        file.write(encrypted_data)
    # Clear the data in the encrypted_data variable, for security reasons
    encrypted_data = b''
```

3. Error handling:

- Errors should be handled gracefully to avoid abruptly exiting the program, and must provide reasonable information to the user

```
        else:
            print(f'Entry {site} exists already')
    else:
        print(f'Category {category} not found')
```

4. Choose descriptive and unambiguous names:

- When naming methods and variables, one must try to make them generally descriptive, enhancing code readability.

```
def print_book(self):
    # Iterate through the dictionary to print it
    print('\nCurrent credentials in notebook:\n')
    for categ, name in self.book.items():
        print(f'---Category: {categ}---')
        for key, value in name.items():
            print(f' --{key}--')
            for user, passw in value.items():
                print(f'    -{user}: {passw}')
```

5. Related code should appear vertically dense:

- To increase readability, it is recommended to make code dense when the methods or functions are related; and sparse when there's logic or where classes end.

```
def remote_entry(self):
    # Remove entry
    category = input('Enter the category of the entry you want to remove: ')
    # Check if category exists
    if category in self.book:
        site = input('Enter the name of the entry you want to remove: ')
        if site in self.book[category]:
            del self.book[category][site]
            print(f'Entry in category {category} removed successfully.')
        else:
            print(f'Entry {site} not found')
    else:
        print(f'Category {category} not found.')

def remove_category(self):
    # Remove Category
    category = input('Enter the category you want to remove: ')
    # Check if category exists
    if category in self.book:
            del self.book[category]
            print(f'Entry in category {category} removed successfully.')
    else:
        print(f'Category {category} not found.')

def check_for_file(self):
    # Check if the 'Notebook.txt' file exists in the current directory
    file_path = 'Notebook.txt'
    if os.path.exists(file_path):
        print(f'The file {file_path} exists.')
        return 1
    else:
        print(f'The file {file_path} does not exist.')
        return 0
```

6. Keep it simple stupid (KISS):

- Methods and functions should do simple tasks, so that code is straightforward, without unnecessary complexity

```
def generate_key():
    # Ask the user to enter a secure password for the Notebook
    password = input('Please enter the master password\n')
    # Salt the password to avoid rainbow tables, ideally it should be different each time
    salt = b'salt_123'
    # Derive cryptographic key from the user password
    kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    iterations=100000,
    salt=salt,
    length=32
    )
    key = base64.urlsafe_b64encode(kdf.derive(password.encode()))
    return key
```

7. Use explanatory variables:

- Names of variables should indicate what they are being used for.

```
        if category in self.book:
            site = input('Enter the name of the website or service: ')
            # Check if entry exists
            if site not in self.book[category]:


    def check_for_file(self):
        # Check if the 'Notebook.txt' file exists in the current directory
        file_path = 'Notebook.txt'
```

8. Encapsulate boundary conditions:

- Validate file or data existence in order to prevent unexpected behavior or errors.

```
    def check_for_file(self):
        # Check if the 'Notebook.txt' file exists in the current directory
        file_path = 'Notebook.txt'
        if os.path.exists(file_path):
            print(f'The file {file_path} exists.')
            return 1
        else:
            print(f'The file {file_path} does not exist.')
            return 0
```

9. Use pronounceable names:

- We are coding for humans, so variable and method names should be pronounceable.

```
    def print_book(self):
        ...                 print(f'    -{user}: {passw}')

    def add_entry(self):
        ...           print(f'Category {category} not found')

    def add_category(self):
        ...          print(f'Category {category} added successfully.')

    def remote_entry(self):
        ...          print(f'Category {category} not found.')

    def remove_category(self):
        ...          print(f'Category {category} not found.')
```

10. Do one thing:

- Our methods and functions should preferably do a simple task, and do it well, so that they may be easily implemented.

```
def mock_notebook(self):
        # Create new Notebook.txt file with mock data
        print('Creating new Notebook')
        notebook = Notebook({
        'Websites': {
            'amazon.com':{
            'Username': 'johndoe',
            'Password':'pass123'
            }
            },
            'Emails': {
                'gmail': {
                'Username': 'johndoe@gmail.com',
                'Password':'pass321'
                }
                }
            }
        )
```