Python

FINAL PROJECT

Simple User Management System

Prepared by

Thou Sokleng

Table of Contents

Simple User Management System using OOP and CRUD in Python

I. Project description

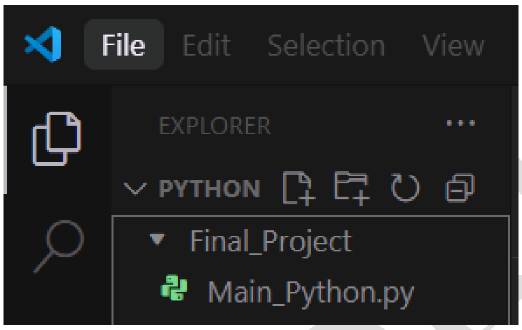
1. Purpose

- This project is build in purpose to manage user information like name, user id and email by administrator to Create, Read, Update and Delete user data using Object-Oriented Programming (OOP) in Python.

2. Project Tasks

2.1 Project Setup

- Create a new Python file with name **Main_Python.py** by use development environment **vsCode**.



2.2 Design the Classes

- Create a **User** class to represent a system user with:
  - **user_id**
  - **name**
  - **email**

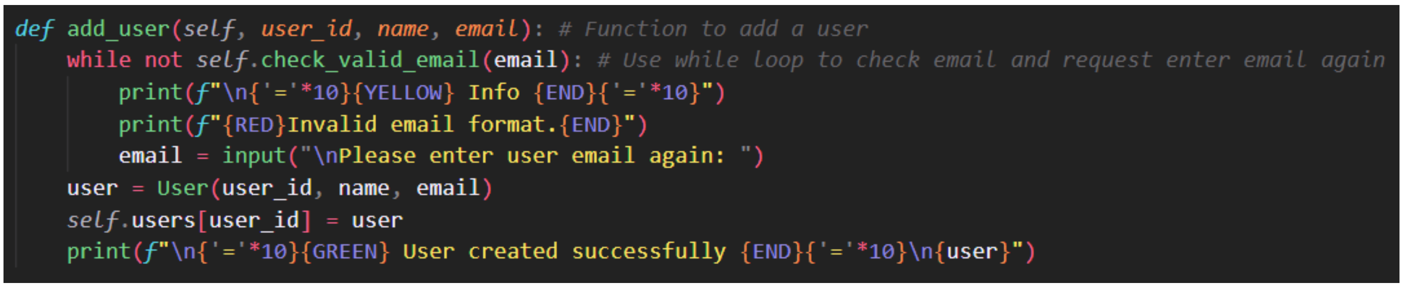
```python
class User:
    def __init__(self, user_id, name, email): # Create user object
        self.user_id = user_id
        self.name = name
        self.email = email

    def __str__(self): # String representation of the user object
        return (
            f"- User ID: {self.user_id} \n- Name: {self.name} \n- Email: {self.email}"
        )
```

- Create a **UserManager** class to handle user operations:
  - Define a function with dictionary to store users information

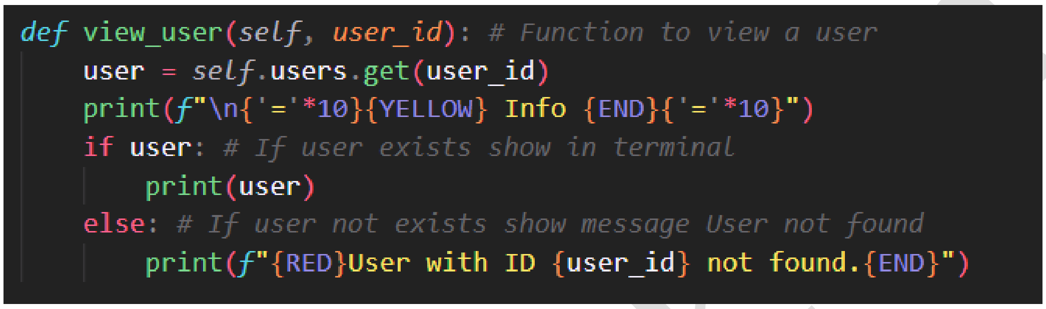
```python
class UserManager:
    def __init__(self): # Dictionary to store users information
        self.users = {}
```

- Define function to add a **user** and use with **while-loop** to check email.



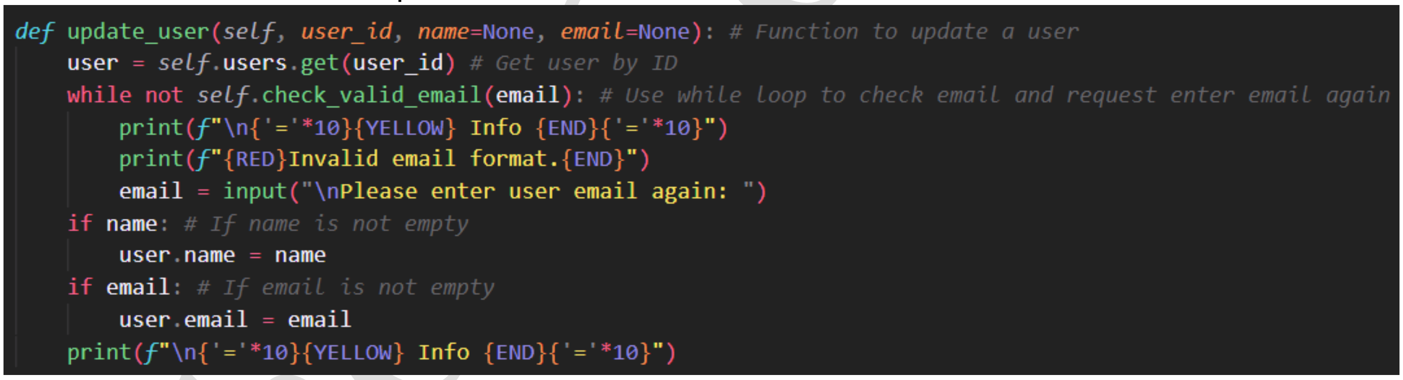
```python
def add_user(self, user_id, name, email): # Function to add a user
    while not self.check_valid_email(email): # Use while loop to check email and request enter email again
        print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
        print(f"{RED}Invalid email format.{END}")
        email = input("\nPlease enter user email again: ")
    user = User(user_id, name, email)
    self.users[user_id] = user
    print(f"\n{'='*10}{GREEN} User created successfully {END}{'='*10}\n{user}")
```

- Define function to **view** user a user. In this function we use if condition to check user If user **exists** show in terminal and if user not exists show message User **not found**.



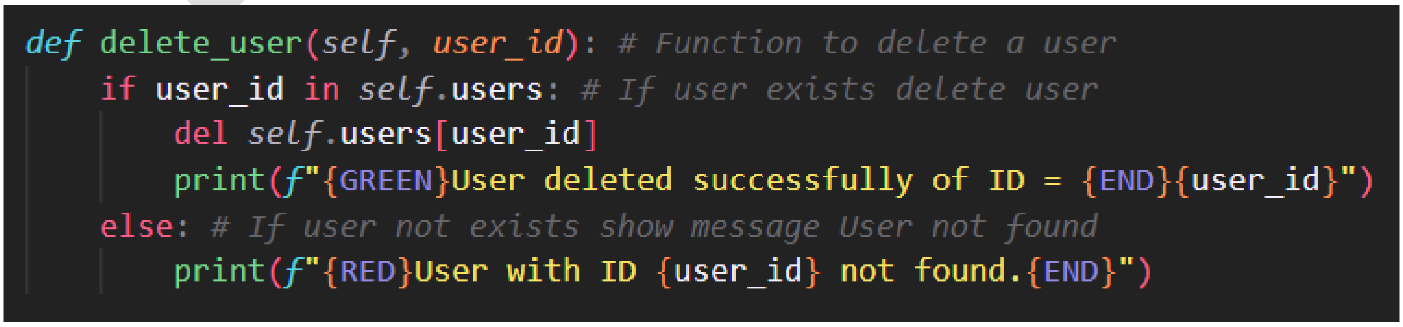
```python
def view_user(self, user_id): # Function to view a user
    user = self.users.get(user_id)
    print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
    if user: # If user exists show in terminal
        print(user)
    else: # If user not exists show message User not found
        print(f"{RED}User with ID {user_id} not found.{END}")
```

- Define function to **update** a user information. In this function best on use-id to verify and use while-loop to check email format with if condition to update new info.



```python
def update_user(self, user_id, name=None, email=None): # Function to update a user
    user = self.users.get(user_id) # Get user by ID
    while not self.check_valid_email(email): # Use while loop to check email and request enter email again
        print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
        print(f"{RED}Invalid email format.{END}")
        email = input("\nPlease enter user email again: ")
    if name: # If name is not empty
        user.name = name
    if email: # If email is not empty
        user.email = email
    print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
```

- Define function to delete a user. In this function use if condition to check user-id exist in system or not. If user exist process delete but if not exist show message User not found.



```python
def delete_user(self, user_id): # Function to delete a user
    if user_id in self.users: # If user exists delete user
        del self.users[user_id]
        print(f"{GREEN}User deleted successfully of ID = {END}{user_id}")
    else: # If user not exists show message User not found
        print(f"{RED}User with ID {user_id} not found.{END}")
```

- Define function to check email format valid or not. In this function I use import module **re** in **vsCode**.

```python
def check_valid_email(self, email): # Function to check if email is valid
    pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(pattern, email) is not None
```

```python
import re
```

2.3 Create Menu System to operation with user

- Create an instance of **UserManager** or call **Class** to use.

```python
if __name__ == "__main__":
    users_action = UserManager() # Create an instance of UserManager
```

- Use while-loop to show all menu until user exit program.

```python
while True: # Use loop to show menu until user exits
    print(f"\n{'='*10}{YELLOW} Menu {END}{'='*10}")
    print("1 - Add User")
    print("2 - View Users")
    print("3 - Update User")
    print("4 - Delete User")
    print("5 - Exit")
    choice = input("Please select an option (1-5): ")
```

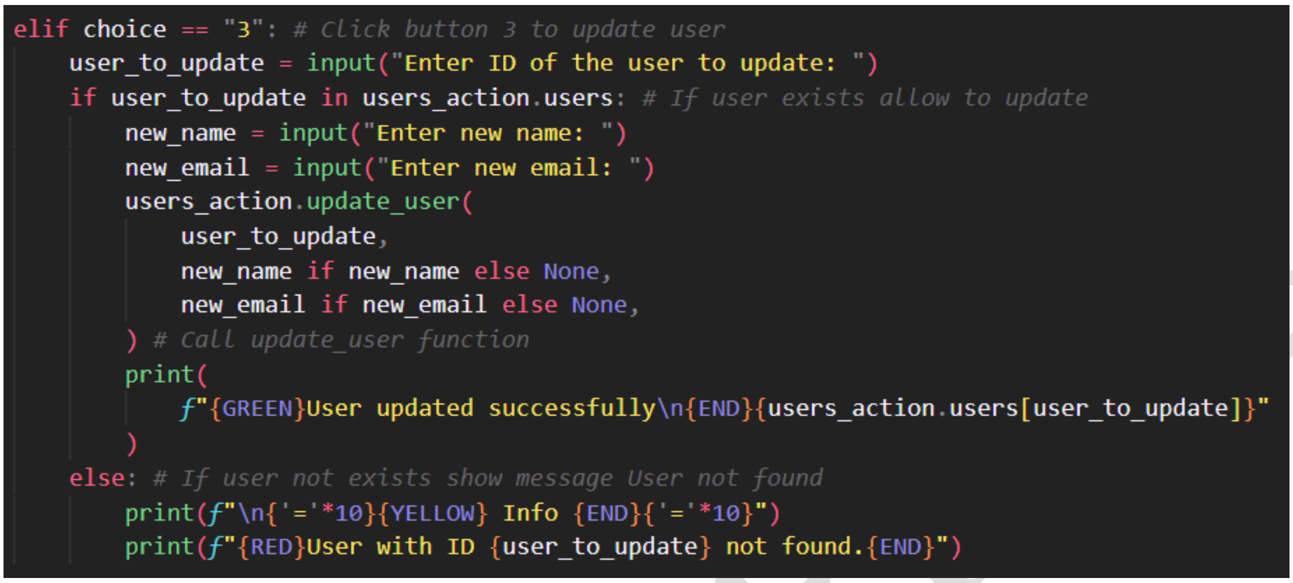
- Create condition when choose option 1 to add new user by request enter name, email and user-id. Then call add_user function and when user enter worng email format it will request enter again.

```python
if choice == "1": # Click button 1 to add user
    name = input("Enter the user's name: ")
    email = input("Enter the user's email: ")
    user_id = input("Enter the user's id: ")
    users_action.add_user(user_id, name, email) # Call add_user function
```

- Create condition when choose option 2 to view user by request enter user-id to check and call view_user function.

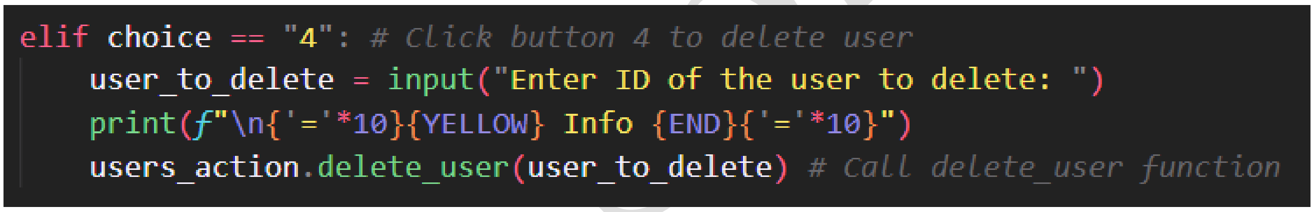
```python
elif choice == "2": # Click button 2 to view user
    user_id = input("Enter the user's id to view: ")
    users_action.view_user(user_id) # Call view_user function
```

- Create condition when choose option 3 to update user by user-id. Then if user-id valid program will allow to update info (name and email) and call update_user function, but if user-id not exist in system show message User not found.



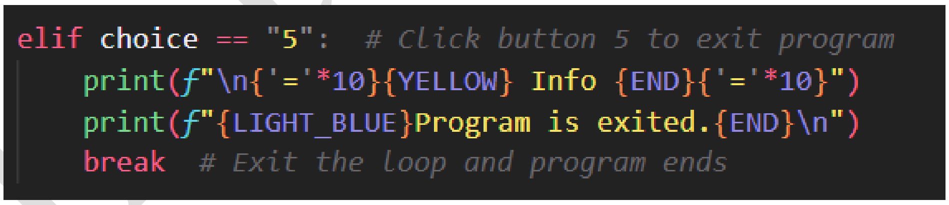
```python
elif choice == "3": # Click button 3 to update user
    user_to_update = input("Enter ID of the user to update: ")
    if user_to_update in users_action.users: # If user exists allow to update
        new_name = input("Enter new name: ")
        new_email = input("Enter new email: ")
        users_action.update_user(
            user_to_update,
            new_name if new_name else None,
            new_email if new_email else None,
        ) # Call update_user function
        print(
            f"{GREEN}User updated successfully\n{END}{users_action.users[user_to_update]}"
        )
    else: # If user not exists show message User not found
        print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
        print(f"{RED}User with ID {user_to_update} not found.{END}")
```

- Create condition when choose option 4 to delete user by user-id and call delete_user function.



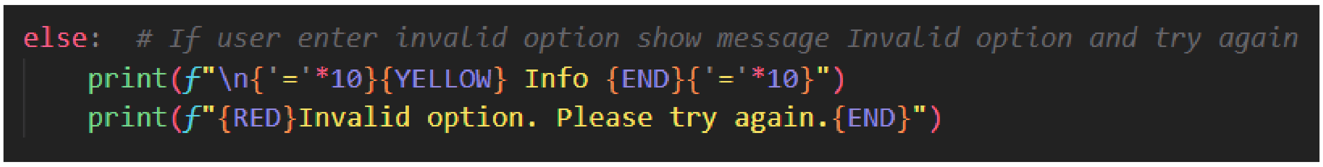
```python
elif choice == "4": # Click button 4 to delete user
    user_to_delete = input("Enter ID of the user to delete: ")
    print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
    users_action.delete_user(user_to_delete) # Call delete_user function
```

- Create condition when choose option 5 to exit program by use **break** to exit loop process.



```python
elif choice == "5":  # Click button 5 to exit program
    print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
    print(f"{LIGHT_BLUE}Program is exited.{END}\n")
    break  # Exit the loop and program ends
```

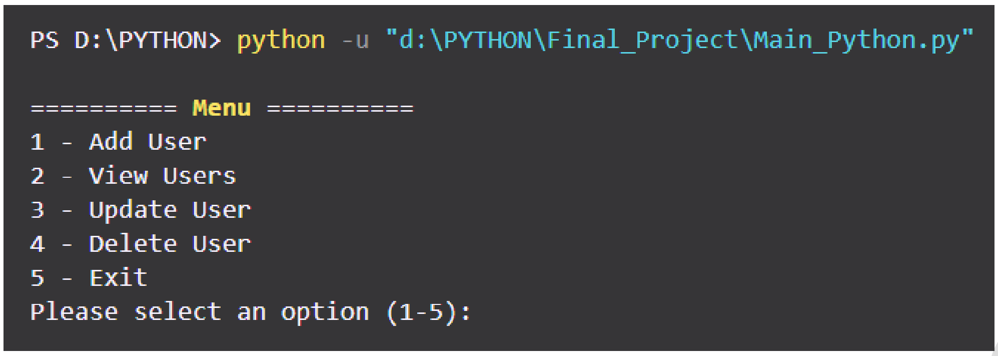
- Create condition when user enter invalied option and show message Invalid option and try enter again.



```python
else:  # If user enter invalid option show message Invalid option and try again
    print(f"\n{'='*10}{YELLOW} Info {END}{'='*10}")
    print(f"{RED}Invalid option. Please try again.{END}")
```

3. Testing

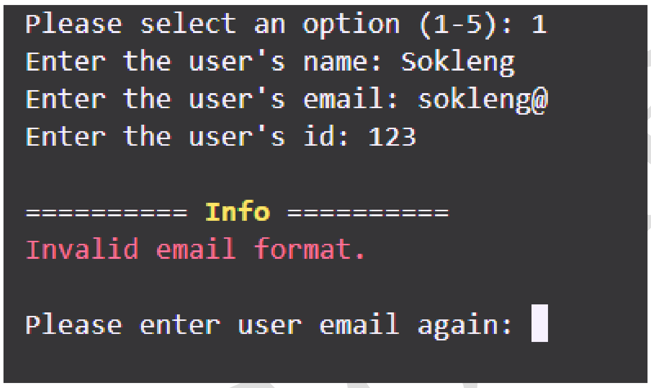
- Test run program



```
PS D:\PYTHON> python -u "d:\PYTHON\Final_Project\Main_Python.py"

========== Menu ==========
1 - Add User
2 - View Users
3 - Update User
4 - Delete User
5 - Exit
Please select an option (1-5):
```

- Test function adding user
  - Case 1: enter invalid email format

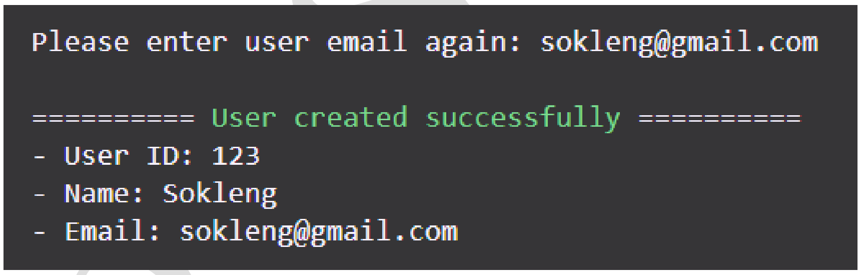


```
Please select an option (1-5): 1
Enter the user's name: Sokleng
Enter the user's email: sokleng@
Enter the user's id: 123

========== Info ==========
Invalid email format.

Please enter user email again:
```

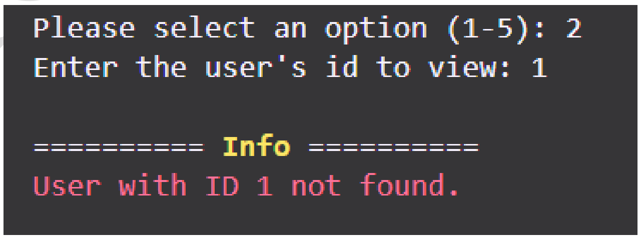
  - Case 2: enter valid email format



```
Please enter user email again: sokleng@gmail.com

========== User created successfully ==========
- User ID: 123
- Name: Sokleng
- Email: sokleng@gmail.com
```

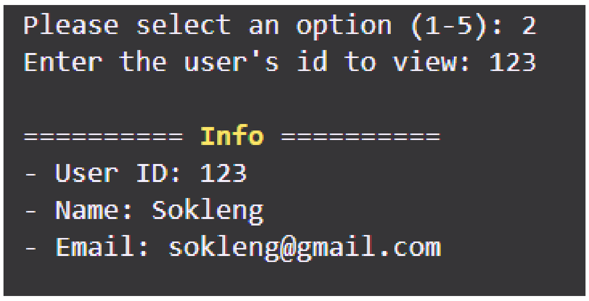
- Test function viewing user
  - Case 1: enter invalid user-id



```
Please select an option (1-5): 2
Enter the user's id to view: 1

========== Info ==========
User with ID 1 not found.
```

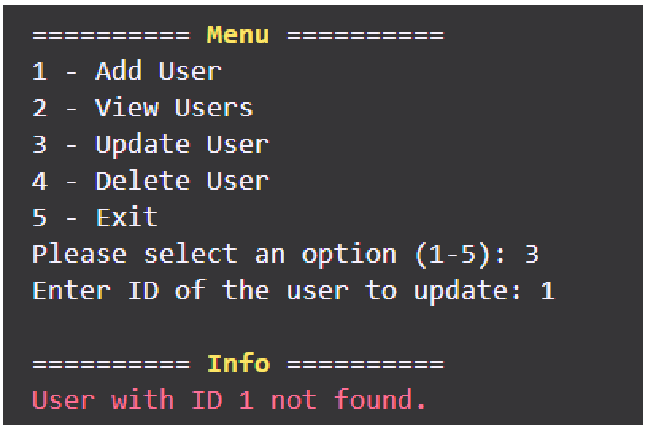
  - Case 2: enter valid user-id



```
Please select an option (1-5): 2
Enter the user's id to view: 123

========== Info ==========
- User ID: 123
- Name: Sokleng
- Email: sokleng@gmail.com
```

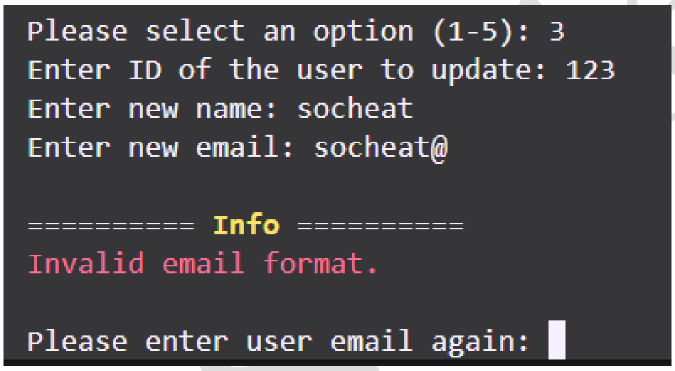
- Test function updating user
  - Case 1: enter invalid user-id



```
========== Menu ==========
1 - Add User
2 - View Users
3 - Update User
4 - Delete User
5 - Exit
Please select an option (1-5): 3
Enter ID of the user to update: 1

========== Info ==========
User with ID 1 not found.
```

  - Case 2: enter valid user-id, but input invalid email format

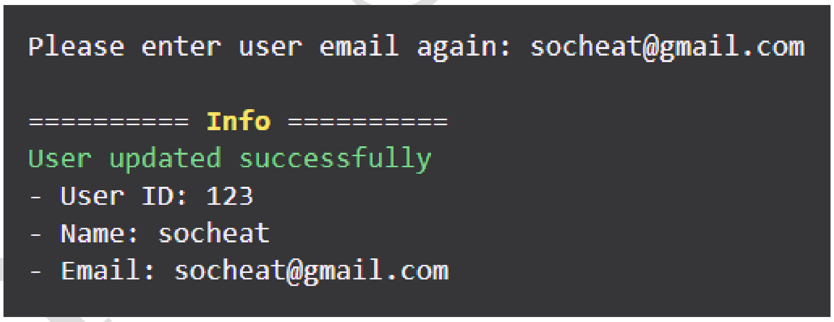


```
Please select an option (1-5): 3
Enter ID of the user to update: 123
Enter new name: socheat
Enter new email: socheat@

========== Info ==========
Invalid email format.

Please enter user email again:
```

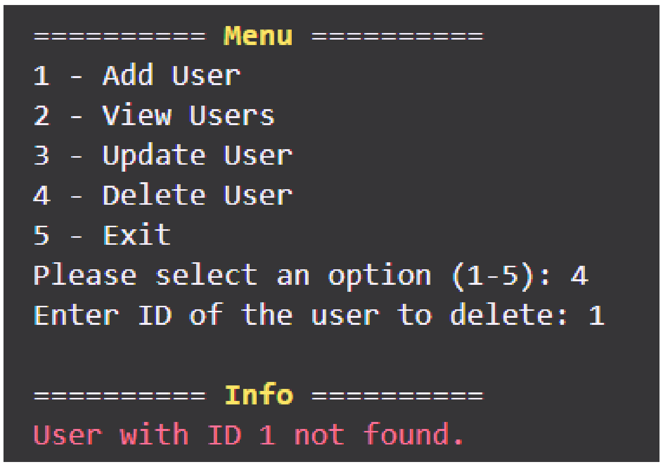
  - Case 3: enter valid email



```
Please enter user email again: socheat@gmail.com

========== Info ==========
User updated successfully
- User ID: 123
- Name: socheat
- Email: socheat@gmail.com
```

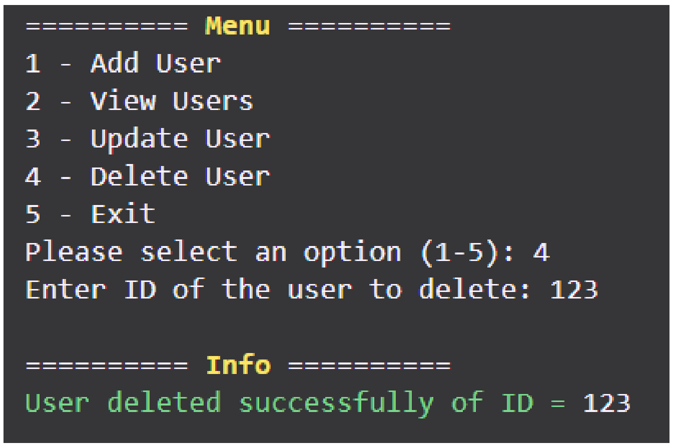
- Test function deleting users
  - Case 1: enter invalid user-id



```
========== Menu ==========
1 - Add User
2 - View Users
3 - Update User
4 - Delete User
5 - Exit
Please select an option (1-5): 4
Enter ID of the user to delete: 1

========== Info ==========
User with ID 1 not found.
```

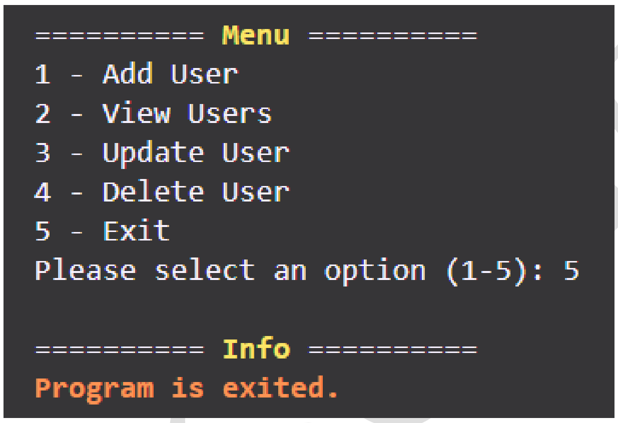
- Case 2: enter valid user-id



```
========== Menu ==========
1 - Add User
2 - View Users
3 - Update User
4 - Delete User
5 - Exit
Please select an option (1-5): 4
Enter ID of the user to delete: 123

========== Info ==========
User deleted successfully of ID = 123
```

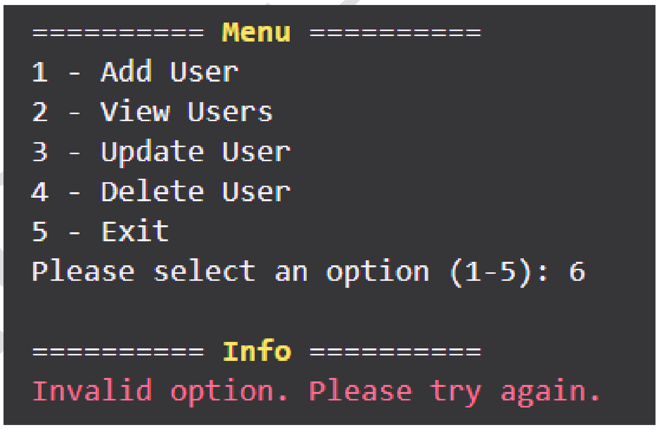
- Test exit program



```
========== Menu ==========
1 - Add User
2 - View Users
3 - Update User
4 - Delete User
5 - Exit
Please select an option (1-5): 5

========== Info ==========
Program is exited.
```

- Test input invalid option number



```
========== Menu ==========
1 - Add User
2 - View Users
3 - Update User
4 - Delete User
5 - Exit
Please select an option (1-5): 6

========== Info ==========
Invalid option. Please try again.
```

4. Add colors using colorama for better UI

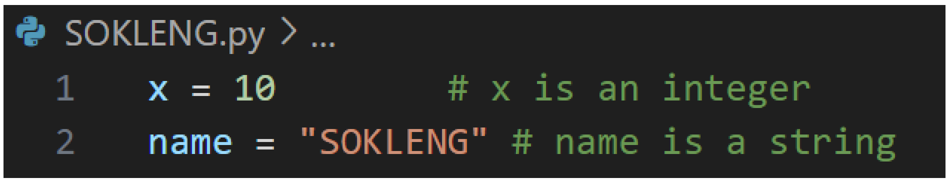
- Define color code

```python
# Color codes for terminal output
RED = "\033[0;31m"
GREEN = "\033[0;32m"
YELLOW = "\033[1;33m"
LIGHT_BLUE = "\033[1;34m"
LIGHT_GREEN = "\033[1;32m"
LIGHT_PURPLE = "\033[1;35m"
END = "\033[0m"
```

II. Summary what that I learned

1. Variable

- Variable is a symbolic name that is a reference or pointer to an object. Variables are used to store data values, and they allow you to manipulate and use these values in your programs.
- Example:



SOKLENG.py > ...

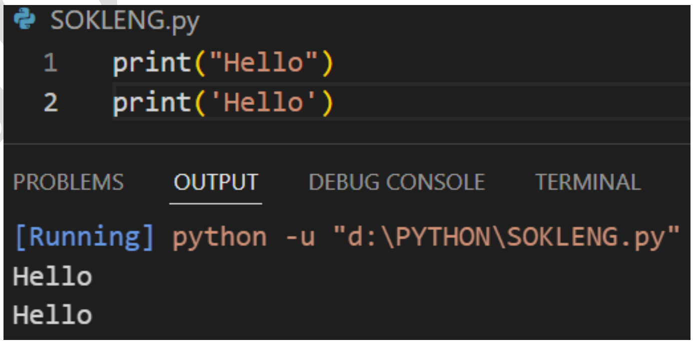
```python
x = 10          # x is an integer
name = "SOKLENG" # name is a string
```

2. Data Type

- Variables can store data of different types, and different types can do different things:
  - Text Type (str)
  - Numeric Types (int, float, complex)
  - Sequence Types (list, tuple, range)
  - Mapping Type (dict)
  - Set Types (set, frozenset)
  - Boolean Type (bool)
  - Binary Types (bytes, bytearray, memoryview)
  - None Type (NoneType)

3. String

- In Python, a string is a sequence of characters that is used to represent text data. Strings are an essential data type in Python, and they can be created using various methods.
- Example: **'hello'** is the same as **"hello"**. You can display a string literal with the print() function.



SOKLENG.py

```python
print("Hello")
print('Hello')
```

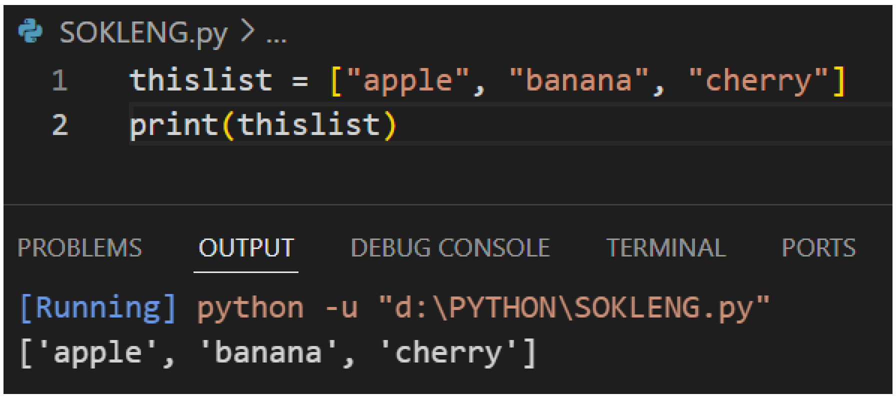
```
[Running] python -u "d:\PYTHON\SOKLENG.py"
Hello
Hello
```

4. Sequence Type

4.1 List

- Lists are used to store multiple items in a single variable like Text or number. Lists are created using square brackets **[]**.

- Example: Create a List



SOKLENG.py > ...

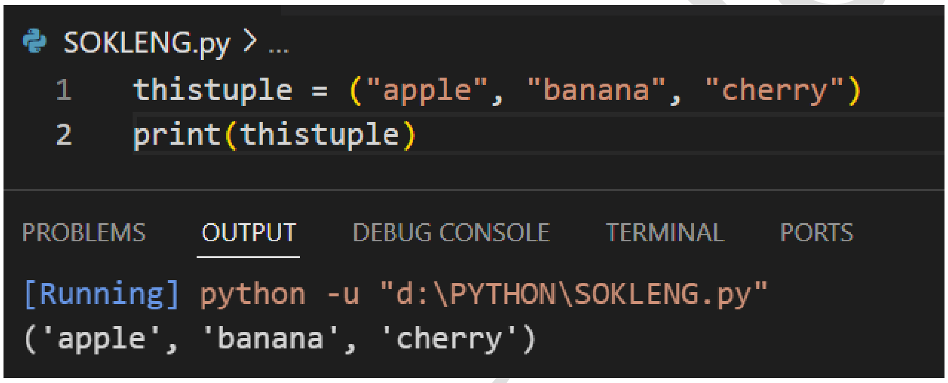
```python
thislist = ["apple", "banana", "cherry"]
print(thislist)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "d:\PYTHON\SOKLENG.py"
['apple', 'banana', 'cherry']
```

4.2 Tuple

- Tuples are used to store multiple items in a single variable. A tuple is a collection which is ordered and unchangeable.Tuples are written with round brackets **()**.
- **Example**: Create a Tuple



SOKLENG.py > ...

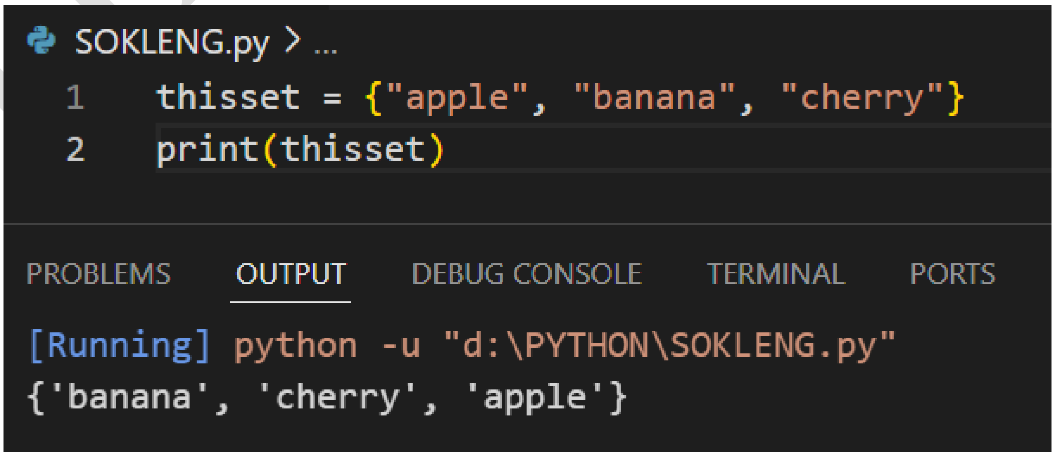
```python
thistuple = ("apple", "banana", "cherry")
print(thistuple)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "d:\PYTHON\SOKLENG.py"
('apple', 'banana', 'cherry')
```

4.3 Set

- **Sets** are used to store multiple items in a single variable. A set is a collection which is unordered, unchangeable*, and unindexed.
- **Note**: Set items are unchangeable, but you can remove items and add new items.
- **Sets** are written with curly brackets **{}**. **Example**: Create a set



SOKLENG.py > ...

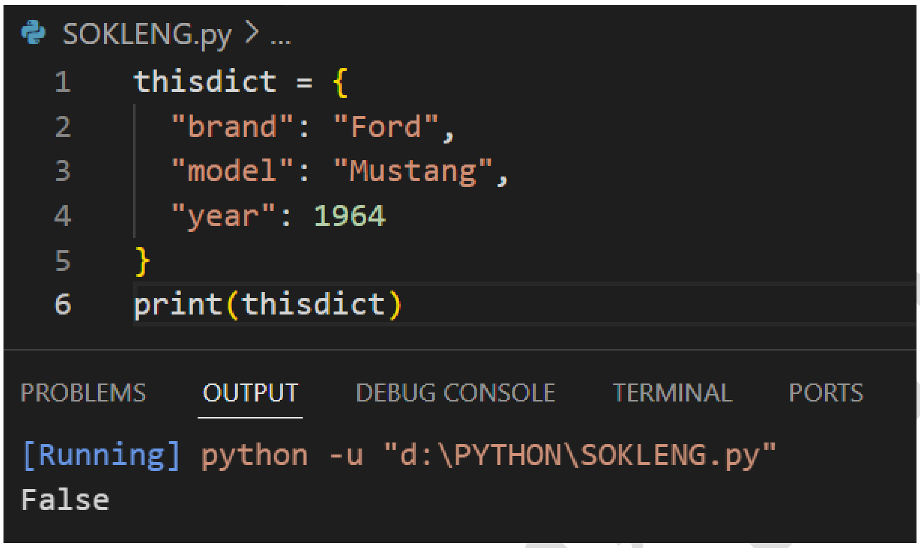
```python
thisset = {"apple", "banana", "cherry"}
print(thisset)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "d:\PYTHON\SOKLENG.py"
{'banana', 'cherry', 'apple'}
```

5. Dictionary

- Dictionaries are used to store data values in **key:value** pairs. A dictionary is a collection which is ordered*, changeable and do not allow duplicates.
- Dictionaries are written with curly brackets **{}**, and have keys and values.
- **Example**: Create and print a dictionary.


SOKLENG.py > ...
```
thisdict = {
  "brand": "Ford",
  "model": "Mustang",
  "year": 1964
}
print(thisdict)
```

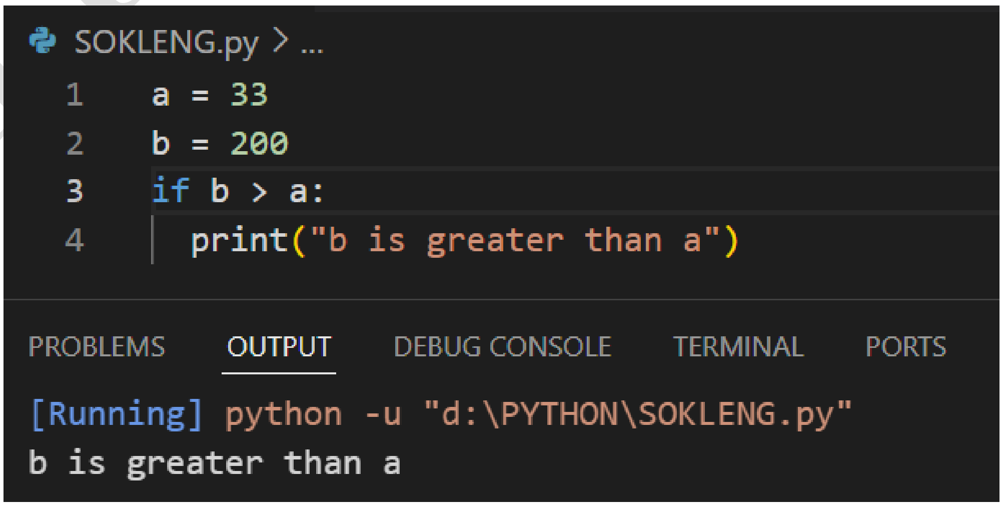
```
[Running] python -u "d:\PYTHON\SOKLENG.py"
False
```

6. Operator

- Operators are used to perform operations on variables and values.
- Python divides the operators in the following groups:
  - Arithmetic operators (+, -, *, /, %, **)
  - Assignment operators (=, +=, -=, *=, /=)
  - Comparison operators (==, !=, >, <, >=, <=)
  - Logical operators (and, or, not)
  - Identity operators (is, is not)
  - Membership operators (in, not in)

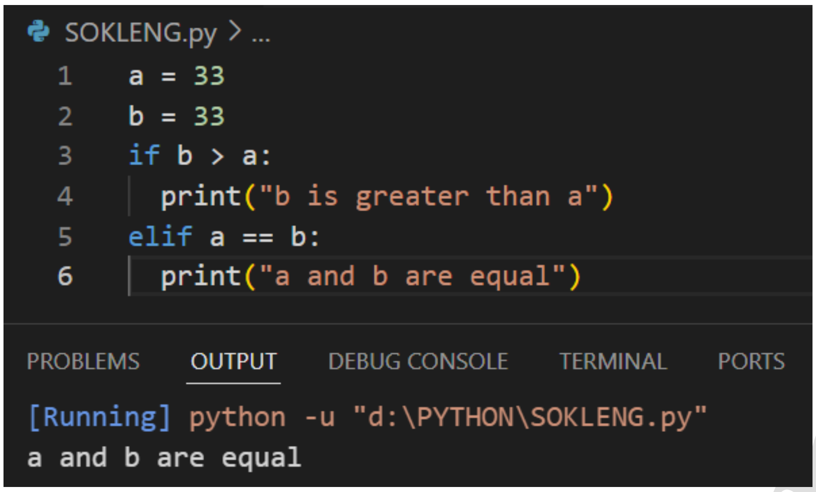
7. Condition

- These conditions can be used in several ways, most commonly in "if statements" and loops.
- An "if statement" is written by using the if keyword. Example: If statement.


SOKLENG.py > ...
```
a = 33
b = 200
if b > a:
  print("b is greater than a")
```

```
[Running] python -u "d:\PYTHON\SOKLENG.py"
b is greater than a
```

- The elif keyword is Python's way of saying "if the previous conditions were not true, then try this condition". Example:



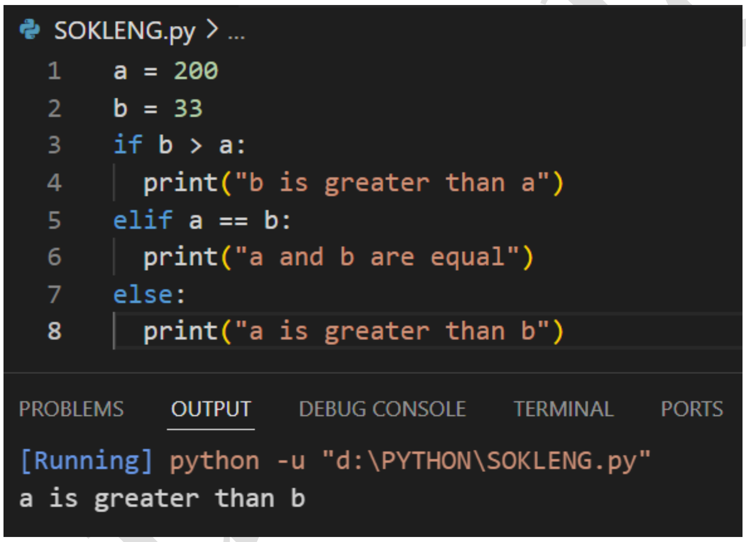
SOKLENG.py > ...

```python
a = 33
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "d:\PYTHON\SOKLENG.py"
a and b are equal
```

- The else keyword catches anything which isn't caught by the preceding conditions. Example



SOKLENG.py > ...

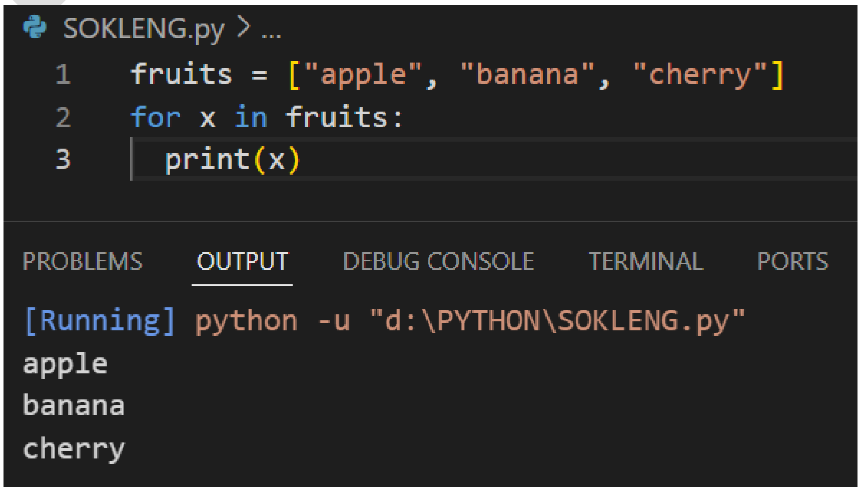
```python
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
[Running] python -u "d:\PYTHON\SOKLENG.py"
a is greater than b
```

8. Loop

- A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string). With the **for** loop we can execute a set of statements, once for each item in a list, tuple, set etc. **Example**: Print each fruit in a fruit list.



SOKLENG.py > ...

```python
fruits = ["apple", "banana", "cherry"]
for x in fruits:
    print(x)
```

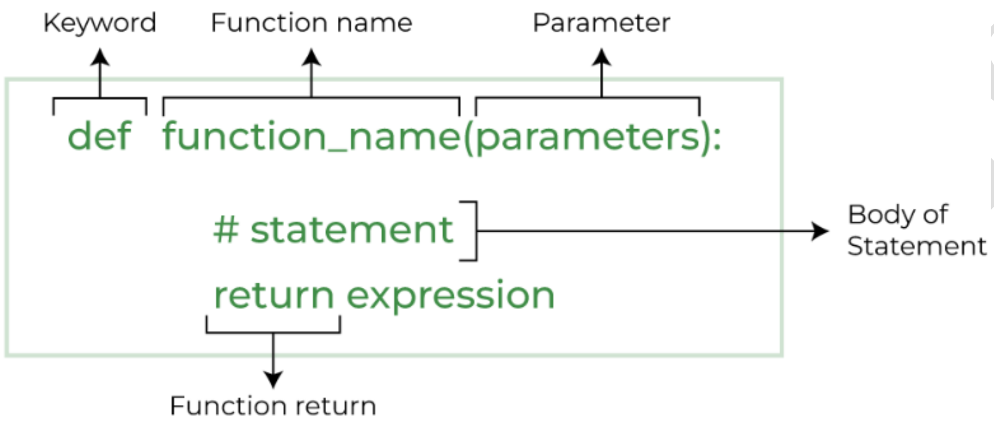
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

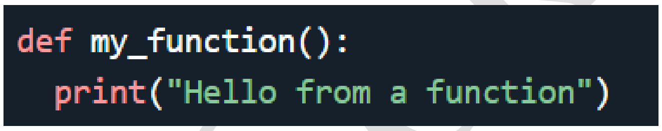
```
[Running] python -u "d:\PYTHON\SOKLENG.py"
apple
banana
cherry
```

- With the **while loop** we can execute a set of statements as long as a condition is true. Example: Print i as long as i is less than 6. The while loop requires relevant variables to be ready, in this example we need to define an indexing variable, i, which we set to 1.

9. Function

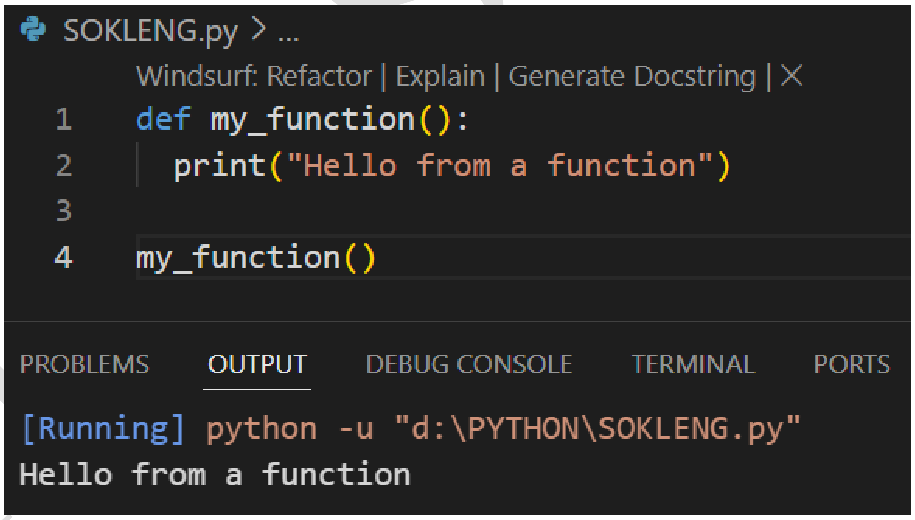
- A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result. In Python a function is defined using the **def** keyword.





```
def my_function():
  print("Hello from a function")
```

- To call a function, use the function name followed by parenthesis.



```
def my_function():
  print("Hello from a function")

my_function()
```

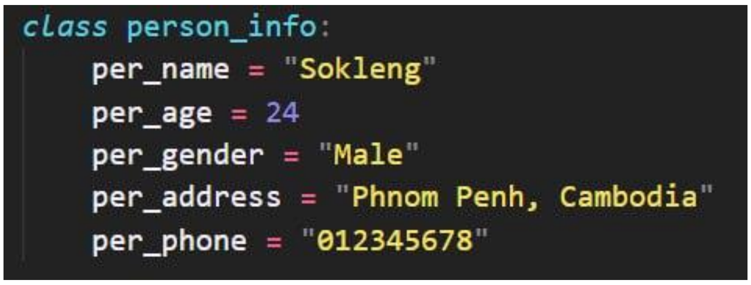
```
[Running] python -u "d:\PYTHON\SOKLENG.py"
Hello from a function
```

10. OOP

- Object-Oriented Programming (OOP) is a programming paradigm that organizes code into objects (instances of classes) rather than just functions.
- Advantages of OOP
  - Provides a clear structure to programs
  - Makes code easier to maintain, reuse, and debug
  - Helps keep your code DRY (Don't Repeat Yourself)
  - Allows you to build reusable applications with less code

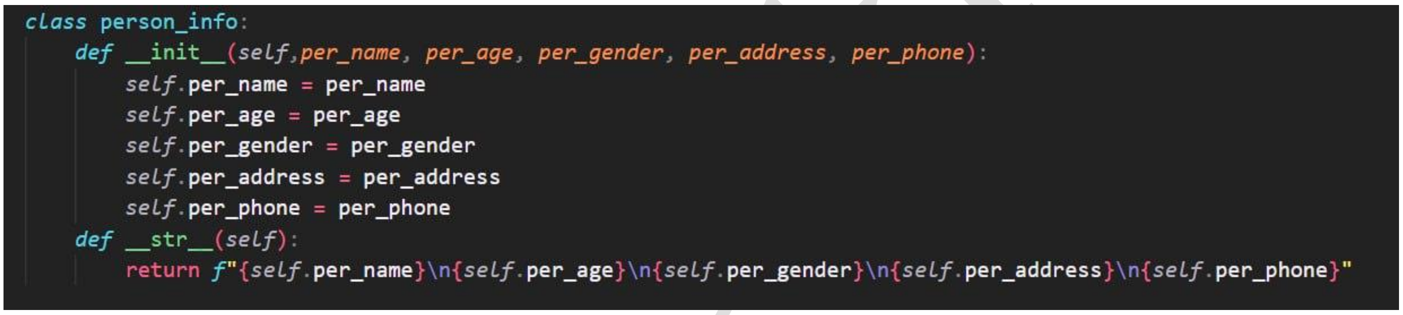
11. Class and Object

- A Class is like an object constructor, or a "blueprint" for creating objects.
- We can create object in class that created. Example



```python
class person_info:
    per_name = "Sokleng"
    per_age = 24
    per_gender = "Male"
    per_address = "Phnom Penh, Cambodia"
    per_phone = "012345678"
```

- All classes have a method called **__init__(),** which is always executed when the class is being initiated.
- Use the **__init__()** method to assign values to object properties, or other operations that are necessary to do when the object is being created.
- The **__str__()** method controls what should be returned when the class object is represented as a string. If the **__str__()** method is not set, the string representation of the object is returned. Example:



```python
class person_info:
    def __init__(self,per_name, per_age, per_gender, per_address, per_phone):
        self.per_name = per_name
        self.per_age = per_age
        self.per_gender = per_gender
        self.per_address = per_address
        self.per_phone = per_phone
    def __str__(self):
        return f"{self.per_name}\n{self.per_age}\n{self.per_gender}\n{self.per_address}\n{self.per_phone}"
```

- The **self** parameter is a reference to the current instance of the class, and is used to access variables that belong to the class.