Phiphat Pinyosophon

August 05 , 2024

Foundation of Programming: Python

Assignment 06

https://github.com/pinyosophon/python110-Summer2024

FUNCTION AND CLASSES

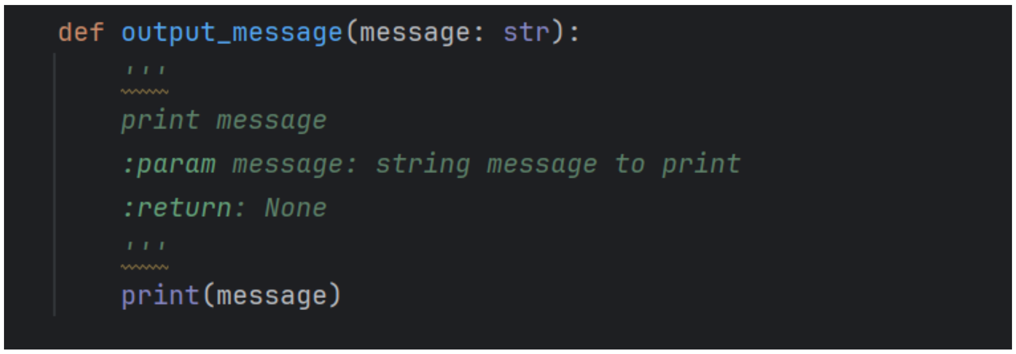
Introduction

Often in programming, we'll have to repeat certain process over and over again as part of a code we're writing, whether it be printing text, arithmetic, or asking for input data from user, we call these reusable block of code used to perform their related action "function". In this week assignment, we're to simplify our code from previous week by creating custom functions then replace the code we wrote with them, and organize them in classes.

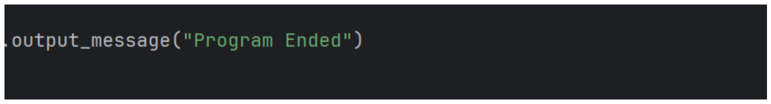
FUNCTION

As mentioned before, functions are reusable block of codes we create to perform certain actions we need. While we're writing function, there are some good practices we should follow:

1. The code defining funciton must exist before it can be called.
2. It should be defined after imports, constants, variables.
3. function is followed by parentheses.



```
def output_message(message: str):
    '''
    print message
    :param message: string message to print
    :return: None
    '''
    print(message)
```



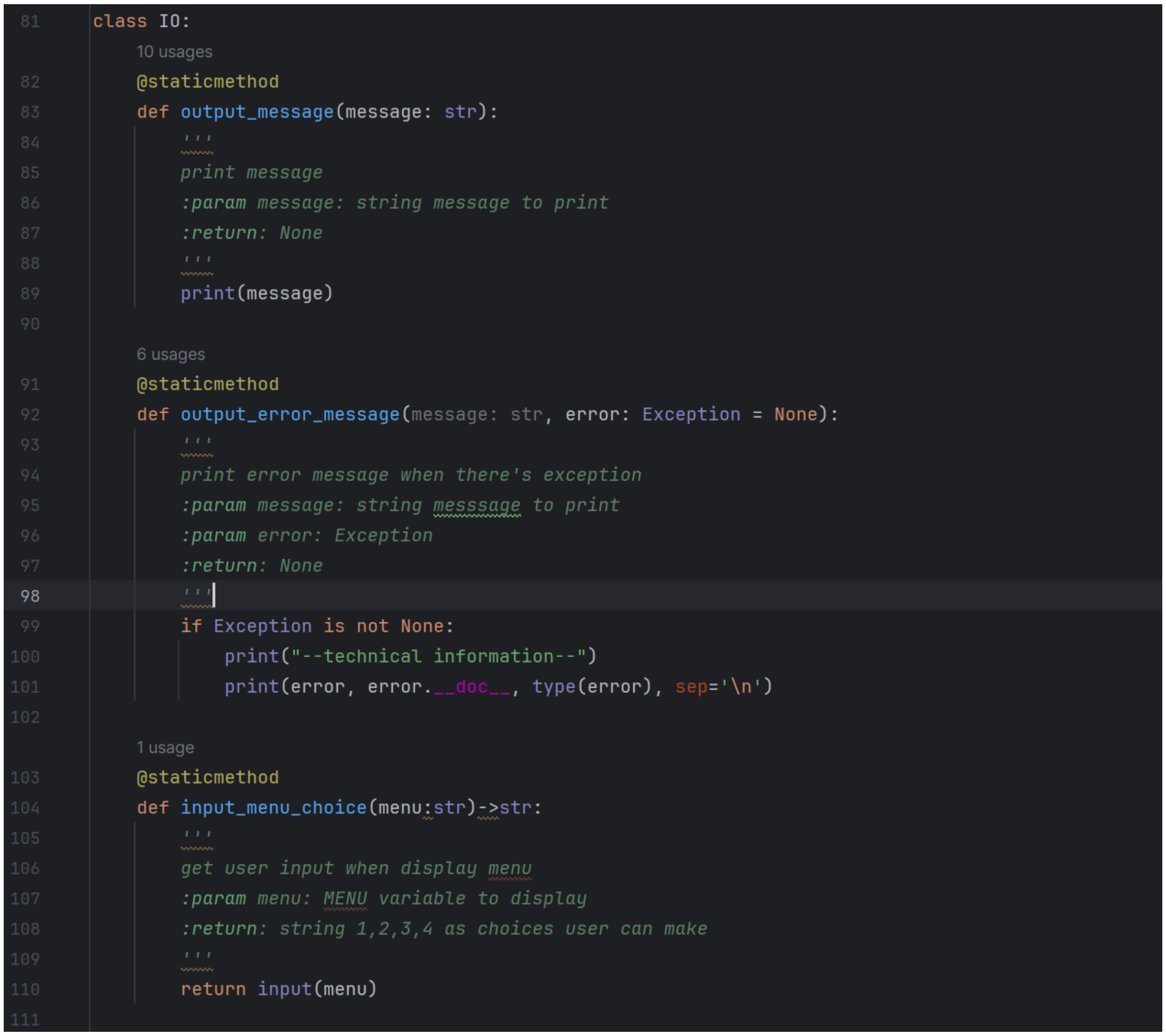
```
output_message("Program Ended")
```

Figure01: Example of function and its usage.

In ***Figure01***, we can see an example of a function written to replace simple print() function. One of the benefits to do it this way instead of use print() function is that if there's anything we need to add onto this print function, we can do so and it will affect everything that's using this output_message function.

CLASS

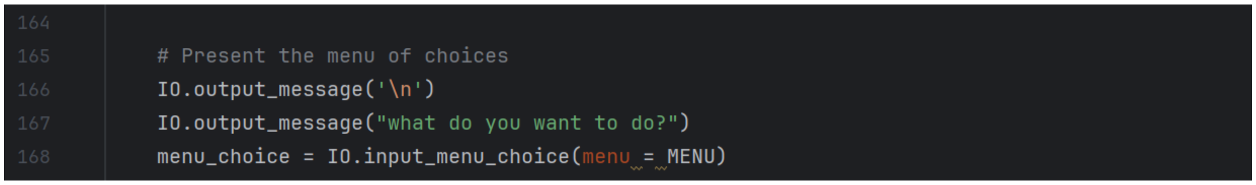
In Python, "class" is used to organize functions we created. It provides a mean to bundling data and similar type of functions together. You can look at it as a template for creating objects. In Figure02, is an example of how a class was used. We created many different type of function dealing with inputting and outputting data



```
class IO:
    10 usages
    @staticmethod
    def output_message(message: str):
        '''
        print message
        :param message: string message to print
        :return: None
        '''
        print(message)

    6 usages
    @staticmethod
    def output_error_message(message: str, error: Exception = None):
        '''
        print error message when there's exception
        :param message: string messsage to print
        :param error: Exception
        :return: None
        '''
        if Exception is not None:
            print("--technical information--")
            print(error, error.__doc__, type(error), sep='\n')

    1 usage
    @staticmethod
    def input_menu_choice(menu:str)->str:
        '''
        get user input when display menu
        :param menu: MENU variable to display
        :return: string 1,2,3,4 as choices user can make
        '''
        return input(menu)

```

Figure02: example of class with functions in it.

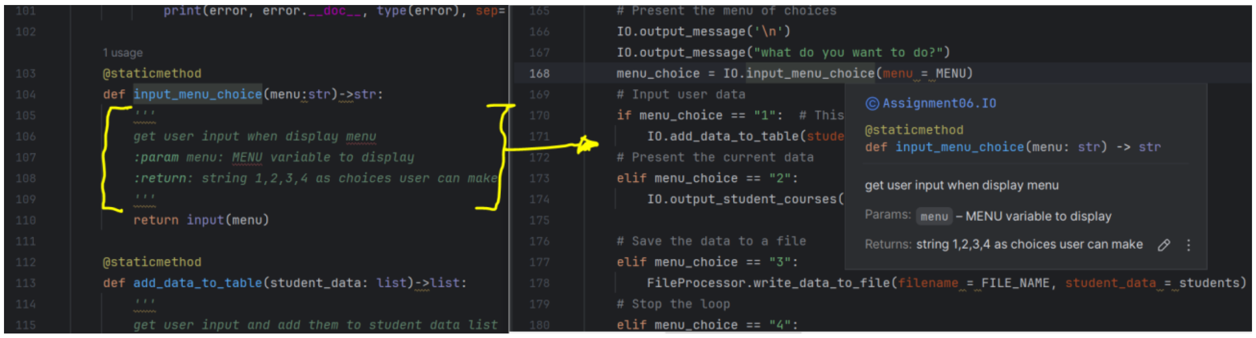
To use function in a class, we can call on it like how you can see in ***Figure03*** below:



```python

    # Present the menu of choices
    IO.output_message('\n')
    IO.output_message("what do you want to do?")
    menu_choice = IO.input_menu_choice(menu = MENU)
```

Figure03: using function organized in class.

As you can see, you put a name of a class first, followed by name of the function and then put related string/variables in the parentheses. For IO.output_message, you just need to type in what you want it to print, and for IO_input_menu_choice() you have to put in menu variable for it to work correctly. And how do you know what to put in between the parentheses? A good practice for when writing function and class would be to create comment and description for it while you create one. So when you hover over a function, it will show what the function does, what kind of data it's expecting, and what it is returning (if any).



```python
            print(error, error.__doc__, type(error), sep=

    1 usage
    @staticmethod
    def input_menu_choice(menu:str)->str:
        '''
        get user input when display menu
        :param menu: MENU variable to display
        :return: string 1,2,3,4 as choices user can make
        '''
        return input(menu)

    @staticmethod
    def add_data_to_table(student_data: list)->list:
        '''
        get user input and add them to student data list
```

```python
    # Present the menu of choices
    IO.output_message('\n')
    IO.output_message("what do you want to do?")
    menu_choice = IO.input_menu_choice(menu = MENU)
    # Input user data
    if menu_choice == "1":  # This
        IO.add_data_to_table(stude
    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(

    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(filename = FILE_NAME, student_data = students)
    # Stop the loop
    elif menu_choice == "4":
```

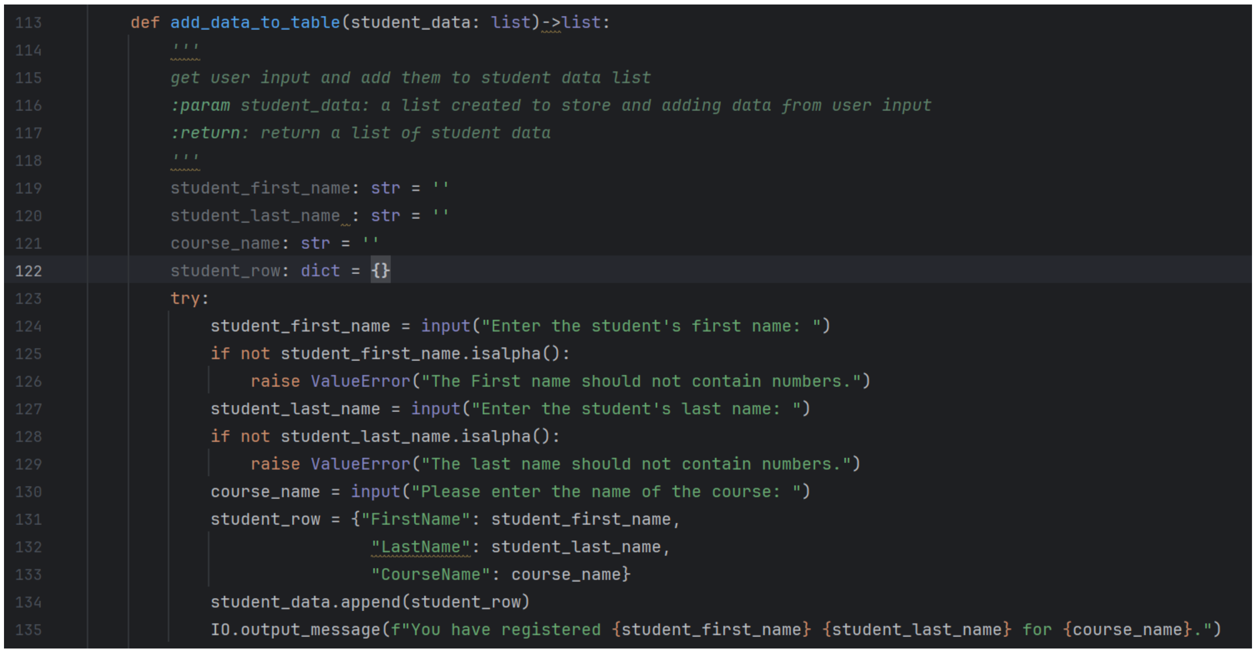
```
Assignment06.IO
@staticmethod
def input_menu_choice(menu: str) -> str

get user input when display menu
Params: menu – MENU variable to display
Returns: string 1,2,3,4 as choices user can make
```

Figure04: Dont forget to write comment/description for your function.

In ***Figure04,*** it shows that this input_menu_choice expect to have a variable for menu, and it will return string. Since out menu consist of 4 choices, only those 4 choices are expected to work, and anything other than that will give error or the program will prompt user to type in new input.

Global Variables vs. Local Variables

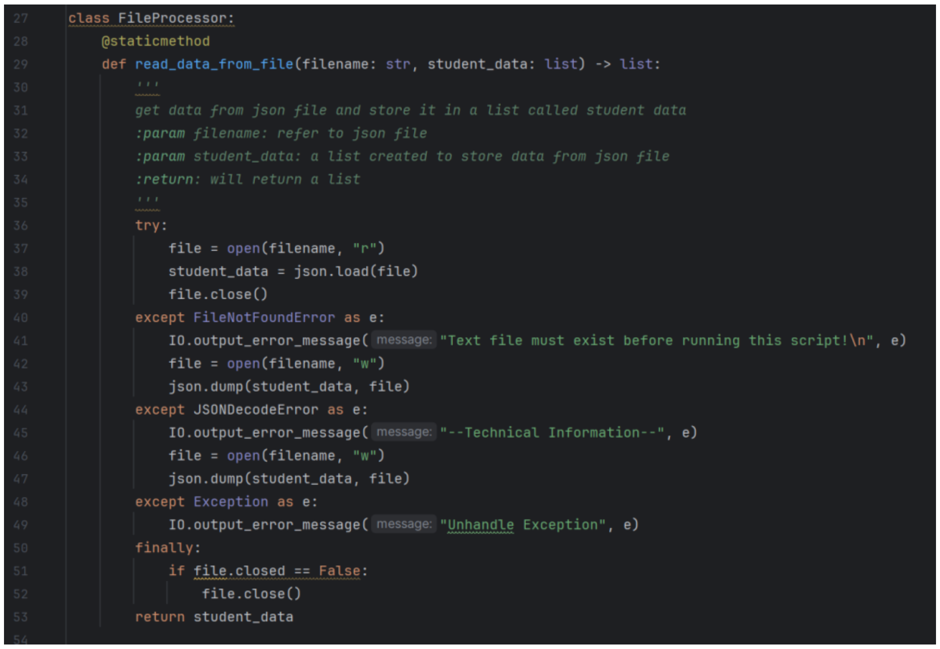
Now that we started writing our own function, we also started to deal with variables in a way we previously did not in weeks before. We now are dealing with global variables and local variables. Local variables, as the name suggests, are variables we created to use locally within a function. You will not be able to access these variables from outside their own function. For example, please take a look at example below in ***Figure05***:



```python
    def add_data_to_table(student_data: list)->list:
        '''
        get user input and add them to student data list
        :param student_data: a list created to store and adding data from user input
        :return: return a list of student data
        '''
        student_first_name: str = ''
        student_last_name : str = ''
        course_name: str = ''
        student_row: dict = {}
        try:
            student_first_name = input("Enter the student's first name: ")
            if not student_first_name.isalpha():
                raise ValueError("The First name should not contain numbers.")
            student_last_name = input("Enter the student's last name: ")
            if not student_last_name.isalpha():
                raise ValueError("The last name should not contain numbers.")
            course_name = input("Please enter the name of the course: ")
            student_row = {"FirstName": student_first_name,
                           "LastName": student_last_name,
                           "CourseName": course_name}
            student_data.append(student_row)
            IO.output_message(f"You have registered {student_first_name} {student_last_name} for {course_name}.")
```

Figure05: local variables.

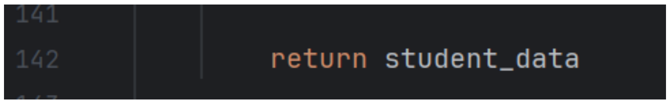
As you can see that this function has 4 local variables, student_first_name, student_last_name, course_name, and student_row. These functions will not be available to use anywhere else except from within this add_data_to_table() function. So if these variables only exist locally, we have to use other ways to access data created from within this function. We can use "return" to pass on, or return data from within a function outside of it. For example:



```python
class FileProcessor:
    @staticmethod
    def read_data_from_file(filename: str, student_data: list) -> list:
        '''
        get data from json file and store it in a list called student data
        :param filename: refer to json file
        :param student_data: a list created to store data from json file
        :return: will return a list
        '''
        try:
            file = open(filename, "r")
            student_data = json.load(file)
            file.close()
        except FileNotFoundError as e:
            IO.output_error_message( message: "Text file must exist before running this script!\n", e)
            file = open(filename, "w")
            json.dump(student_data, file)
        except JSONDecodeError as e:
            IO.output_error_message( message: "--Technical Information--", e)
            file = open(filename, "w")
            json.dump(student_data, file)
        except Exception as e:
            IO.output_error_message( message: "Unhandle Exception", e)
        finally:
            if file.closed == False:
                file.close()
        return student_data
```

Figure06a: "Returning" data created from within a function.

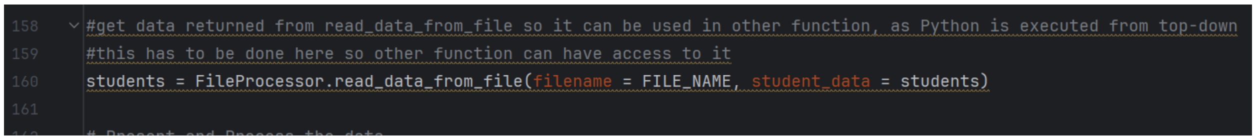
In ***Figure06a***, we created read_data_from_file() function to read and extract list from a json file. The data we get from json file is stored in a list called student_data. At the end of the function, we use this to return the data to be access from the global level:



```
        return student_data
```

Figure06b: return data to be access from global level

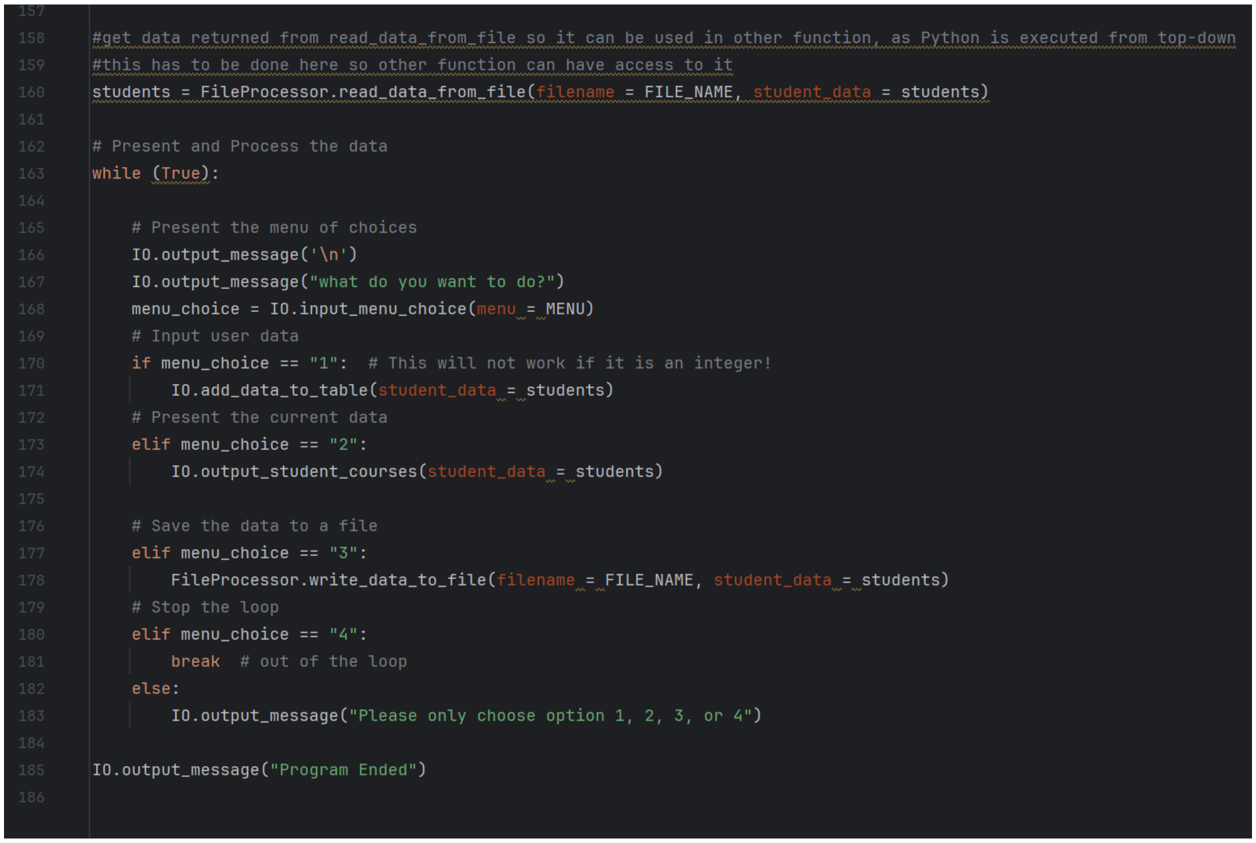
This return student_data also exist in other function as well. But before we can do that, we have to access it first. We do so by assign it like so in ***Figure07:***



```
#get data returned from read_data_from_file so it can be used in other function, as Python is executed from top-down
#this has to be done here so other function can have access to it
students = FileProcessor.read_data_from_file(filename = FILE_NAME, student_data = students)

```

Figure07: Accessing data returned from a function.

In ***Figure07***, students is a global variable we created so that we can use to transfer list data we got from executing functions. Also note that this line of code must exist before other lines that need data from students list, this is because Python is executing from top to bottom, as you can see in ***Figure08:***



```
#get data returned from read_data_from_file so it can be used in other function, as Python is executed from top-down
#this has to be done here so other function can have access to it
students = FileProcessor.read_data_from_file(filename = FILE_NAME, student_data = students)

# Present and Process the data
while (True):

    # Present the menu of choices
    IO.output_message('\n')
    IO.output_message("what do you want to do?")
    menu_choice = IO.input_menu_choice(menu = MENU)
    # Input user data
    if menu_choice == "1":  # This will not work if it is an integer!
        IO.add_data_to_table(student_data = students)
    # Present the current data
    elif menu_choice == "2":
        IO.output_student_courses(student_data = students)

    # Save the data to a file
    elif menu_choice == "3":
        FileProcessor.write_data_to_file(filename = FILE_NAME, student_data = students)
    # Stop the loop
    elif menu_choice == "4":
        break  # out of the loop
    else:
        IO.output_message("Please only choose option 1, 2, 3, or 4")

IO.output_message("Program Ended")
```

Figure08: accessing needed data BEFORE executing everything else that need it.

It should be noted that we could do this to this returned data, because this students variable here is a list type. And in Python, list type is mutable, meaning, it can be changed or modified after it has been created.

Summary

Function is a convenient way to simplify our coding, but it should be noted that, while writing function, it should exist before it can be called, and it should be defined after all the imports, constants, and variable. Also take note to be careful where you define your variables. Variables which are defined in a global level can be used anywhere in your code, while the ones you define locally within a function can only be accessed in that function, and nowhere else. Always be extra careful as local and global can have the same name, so it’s best to avoid it. You can group any related functions together in class and use it later. Take extra caution to create a line of code to access data in a function to be accessed globally before it’s needed as Python executes from top to bottom.