Sanghee Kang
March 1st, 2022
IT FND 110 Foundations of Programming: Python
Assignment 07
GitHub URL: https://github.com/bsb218218/IntroToProg-Python-Mod07

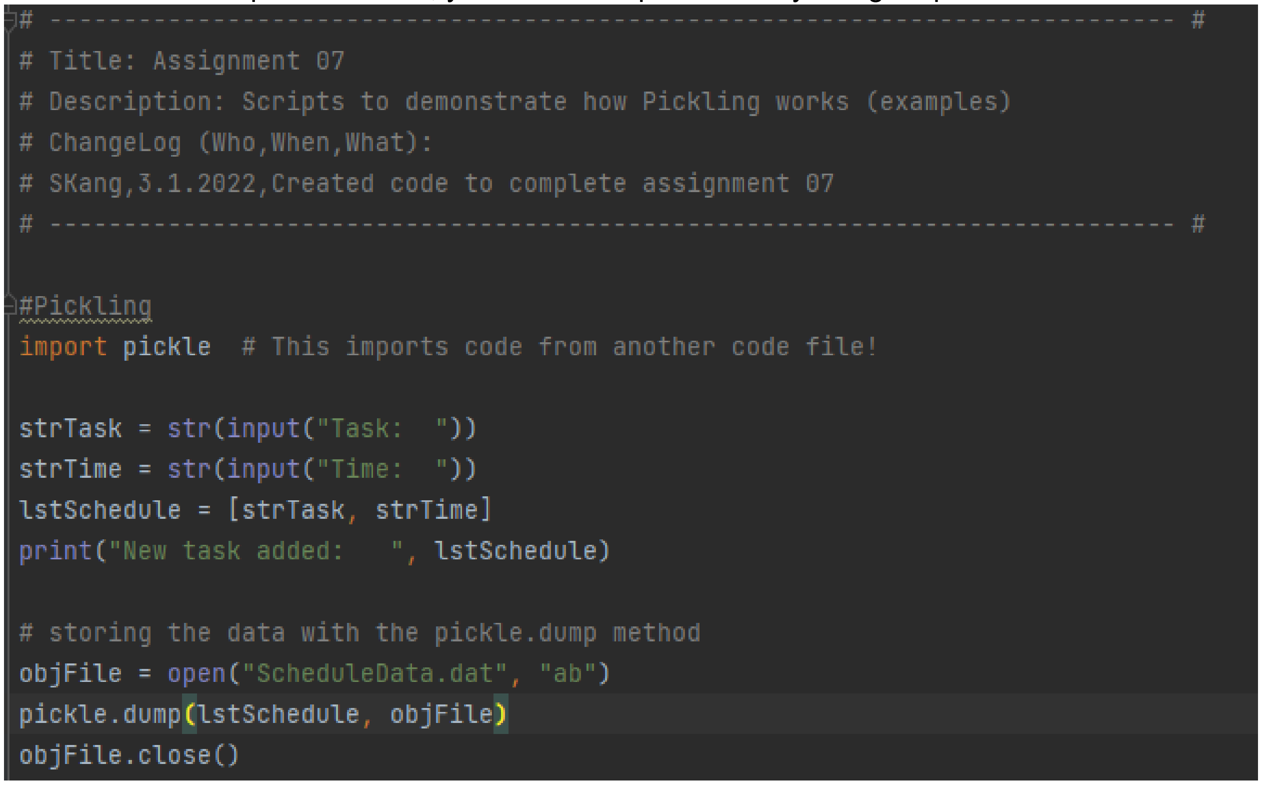
Demonstrating Pickling and Error Handling

1. Introduction

In this report, I will explain what pickling and structured error handling are and show how they work in Python by presenting a few example codes.

2. Pickling

Pickling refers to the process of converting any kind of python objects into byte streams (source: https://www.tutorialspoint.com/python-pickling). In other words, you use the pickle module to save data in binary format, instead of a plain text. Figure 1 shows an example code of pickling. In order to use the pickle module, you need to import it first by using “import”.

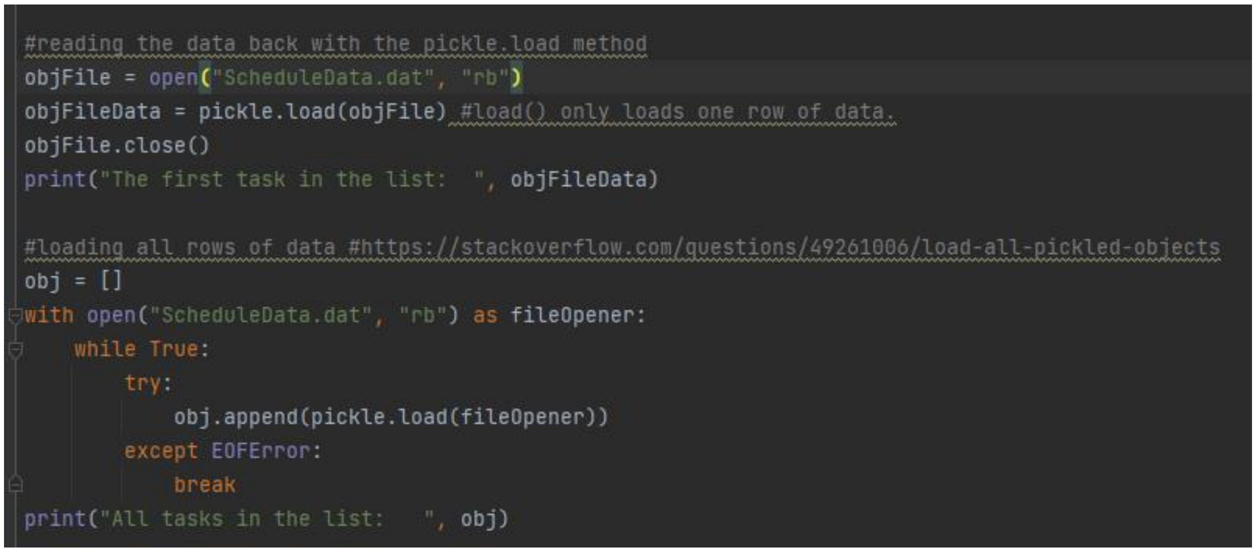


```
# ---------------------------------------------------------------------------- #
# Title: Assignment 07
# Description: Scripts to demonstrate how Pickling works (examples)
# ChangeLog (Who,When,What):
# SKang,3.1.2022,Created code to complete assignment 07
# ---------------------------------------------------------------------------- #

#Pickling
import pickle  # This imports code from another code file!

strTask = str(input("Task:  "))
strTime = str(input("Time:  "))
lstSchedule = [strTask, strTime]
print("New task added:   ", lstSchedule)

# storing the data with the pickle.dump method
objFile = open("ScheduleData.dat", "ab")
pickle.dump(lstSchedule, objFile)
objFile.close()
```

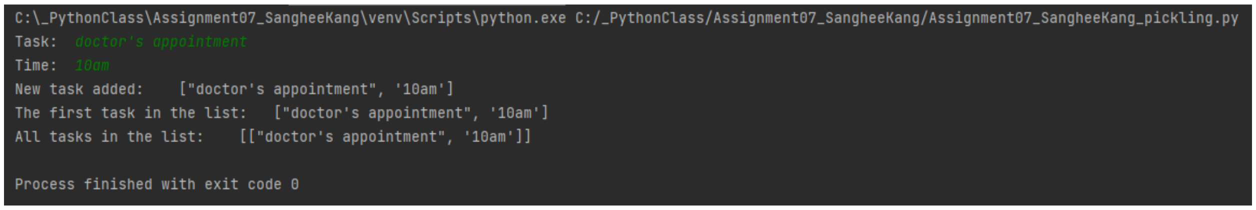


```python
#reading the data back with the pickle.load method
objFile = open("ScheduleData.dat", "rb")
objFileData = pickle.load(objFile) #load() only loads one row of data.
objFile.close()
print("The first task in the list:  ", objFileData)

#loading all rows of data #https://stackoverflow.com/questions/49261006/load-all-pickled-objects
obj = []
with open("ScheduleData.dat", "rb") as fileOpener:
    while True:
        try:
            obj.append(pickle.load(fileOpener))
        except EOFError:
            break
print("All tasks in the list:   ", obj)
```

Figure 1. An example code to demonstrate pickling (top and bottom)

Using the code, I collect schedule data (i.e., task and time) from the user and want to save the data into a binary file, not a plain text file. First, the code captures the data from the user and then opens the file using ‘open()’ and ‘ab’. Afterwards, the code stores the data using the pickle.dump method. You can read the data back with the pickle.load method. You first open the file using ‘open()’ and ‘rb’, then you load the data using the pickle.load() method. But note that this code only loads the first row of the data. In order to load all objects from the data, you can use with open() as, the while loop, try-except functions, and the append() as shown in Figure 1. I did my own research in order to find a way to load all data from the file (source: https://stackoverflow.com/questions/49261006/load-all-pickled-objects). When you run the code, the code prompts you to enter a task and a time. I input “doctor’s appointment and 10am” in a row. Then the codes yield the following:

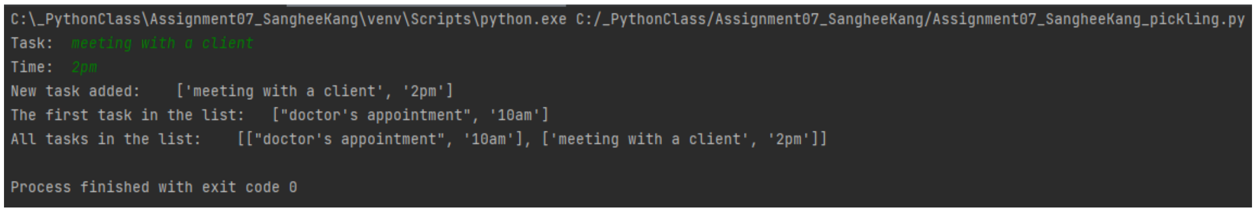


```
C:\_PythonClass\Assignment07_SangheeKang\venv\Scripts\python.exe C:/_PythonClass/Assignment07_SangheeKang/Assignment07_SangheeKang_pickling.py
Task:  doctor's appointment
Time:  10am
New task added:    ["doctor's appointment", '10am']
The first task in the list:   ["doctor's appointment", '10am']
All tasks in the list:    [["doctor's appointment", '10am']]

Process finished with exit code 0
```

Figure 2. Output of the first running of the code

When you run the code one more time, then your data file will have two tasks saved. While pickle.load() would show only the first object, the last part of the code will show you all objects saved in the file (Figure 3).

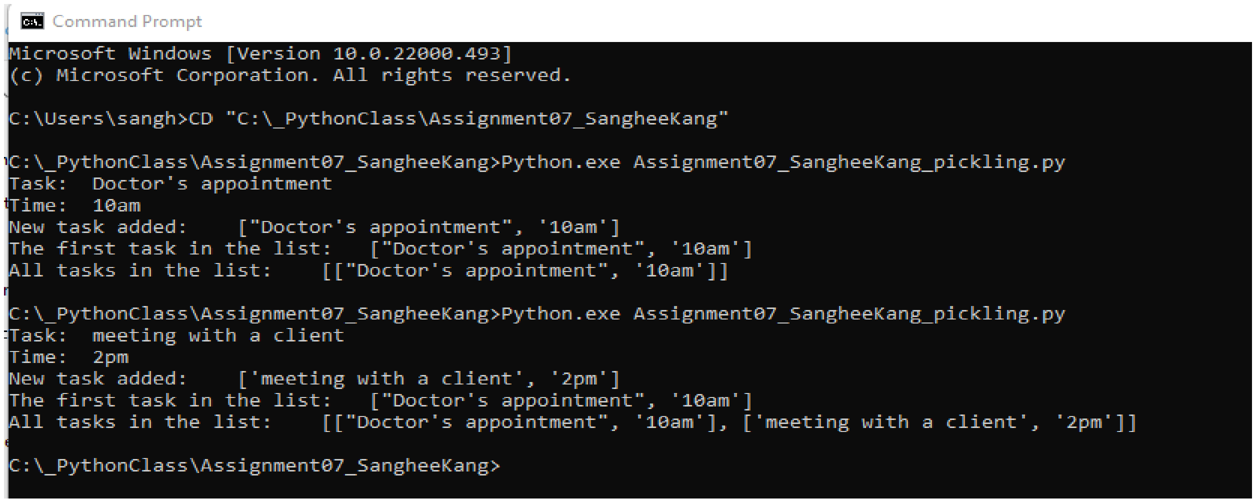


```
C:\_PythonClass\Assignment07_SangheeKang\venv\Scripts\python.exe C:/_PythonClass/Assignment07_SangheeKang/Assignment07_SangheeKang_pickling.py
Task:  meeting with a client
Time:  2pm
New task added:    ['meeting with a client', '2pm']
The first task in the list:   ["doctor's appointment", '10am']
All tasks in the list:    [["doctor's appointment", '10am'], ['meeting with a client', '2pm']]

Process finished with exit code 0
```

Figure 3. Output of the second running of the code

I ran the code in the command shell as well and the same output was resulted in (Figure 4):



```
Command Prompt
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sangh>CD "C:\_PythonClass\Assignment07_SangheeKang"

C:\_PythonClass\Assignment07_SangheeKang>Python.exe Assignment07_SangheeKang_pickling.py
Task:  Doctor's appointment
Time:  10am
New task added:    ["Doctor's appointment", '10am']
The first task in the list:   ["Doctor's appointment", '10am']
All tasks in the list:    [["Doctor's appointment", '10am']]

C:\_PythonClass\Assignment07_SangheeKang>Python.exe Assignment07_SangheeKang_pickling.py
Task:  meeting with a client
Time:  2pm
New task added:    ['meeting with a client', '2pm']
The first task in the list:   ["Doctor's appointment", '10am']
All tasks in the list:    [["Doctor's appointment", '10am'], ['meeting with a client', '2pm']]

C:\_PythonClass\Assignment07_SangheeKang>
```

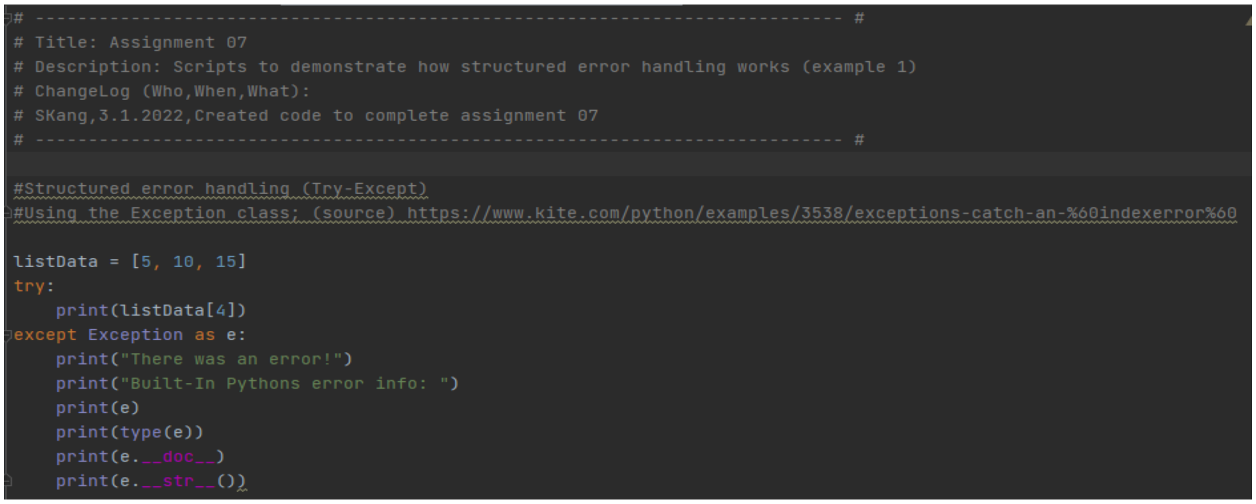
Figure 4. Output in Command Shell

3. Structured Error Handling

When using a code by users, there are many cases in which an error could occur. For such cases, developers employ error handling to catch and deal with exceptions or errors. Following the lecture content, I will explain how structured error handling work in Python by introducing four ways to handle errors or exceptions with my own example codes.

3.1. Using the Exception Class

In order to handle errors or exceptions, you can use the try-except function. Figure 5 shows how you can use the try-except function. Also, this code is using the Exception class, which is used to catch any type of error (I referred to the following source but I created my own example: https://www.kite.com/python/examples/3538/exceptions-catch-an-%60indexerror%60). When you ‘try’ a code (print(listData[4]), and if there is an error, a built-in python error (using the Exception class) is shown to the user. In the following example, as the index ‘4’ is out of range, IndexError, which is a built-in python error, happened.



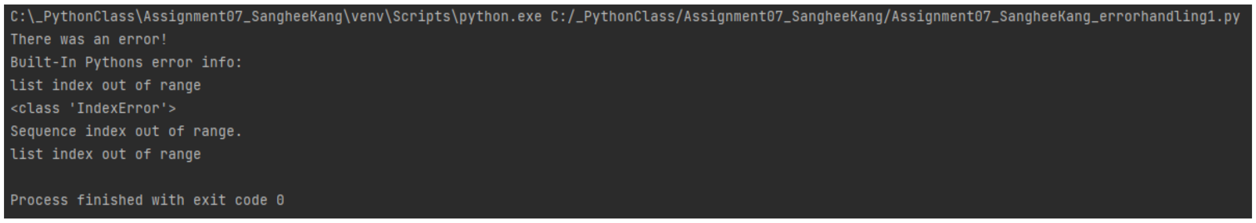
```python
# --------------------------------------------------------------------------- #
# Title: Assignment 07
# Description: Scripts to demonstrate how structured error handling works (example 1)
# ChangeLog (Who,When,What):
# SKang,3.1.2022,Created code to complete assignment 07
# --------------------------------------------------------------------------- #

#Structured error handling (Try-Except)
#Using the Exception class; (source) https://www.kite.com/python/examples/3538/exceptions-catch-an-%60indexerror%60

listData = [5, 10, 15]
try:
    print(listData[4])
except Exception as e:
    print("There was an error!")
    print("Built-In Pythons error info: ")
    print(e)
    print(type(e))
    print(e.__doc__)
    print(e.__str__())
```

Figure 5. Code: Assignment07_SangheeKang_errorhandling1

The output of the code is shown below (Figure 6). The error type is IndexError and some more information is presented.



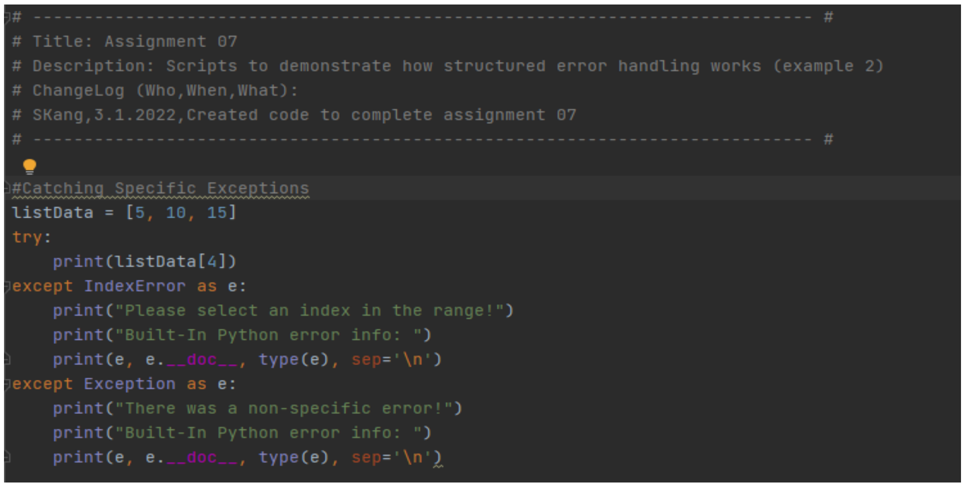
```
C:\_PythonClass\Assignment07_SangheeKang\venv\Scripts\python.exe C:/_PythonClass/Assignment07_SangheeKang/Assignment07_SangheeKang_errorhandling1.py
There was an error!
Built-In Pythons error info:
list index out of range
<class 'IndexError'>
Sequence index out of range.
list index out of range

Process finished with exit code 0
```

Figure 6. Output of Assignment07_SangheeKang_errorhandling1

3.2. Catching Specific Exceptions

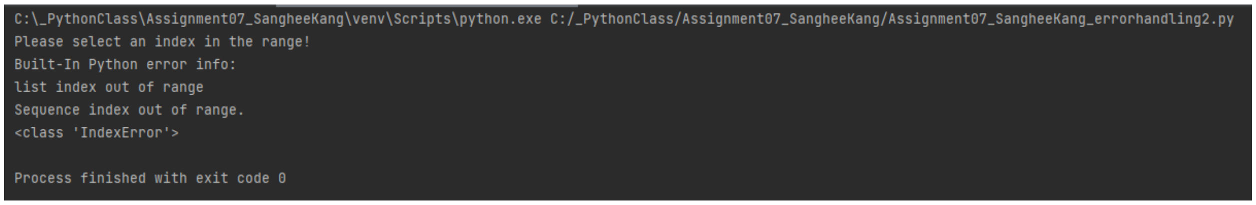
You can also catch a specific error by adding more specific exception classes (but still built-in python errors). In Figure 7, I have the same lstData and the try-except function. But this time, I used a specific error type “IndexError” in the except part (except IndexError as e:). In the case when a non-specific error occurs, I put “except Exception as e: ” as well. Figure 8 shows the result of the code.



```
# -------------------------------------------------------------------------- #
# Title: Assignment 07
# Description: Scripts to demonstrate how structured error handling works (example 2)
# ChangeLog (Who,When,What):
# SKang,3.1.2022,Created code to complete assignment 07
# -------------------------------------------------------------------------- #

#Catching Specific Exceptions
listData = [5, 10, 15]
try:
    print(listData[4])
except IndexError as e:
    print("Please select an index in the range!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
except Exception as e:
    print("There was a non-specific error!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```

Figure 7. Code: Assignment07_SangheeKang_errorhandling2



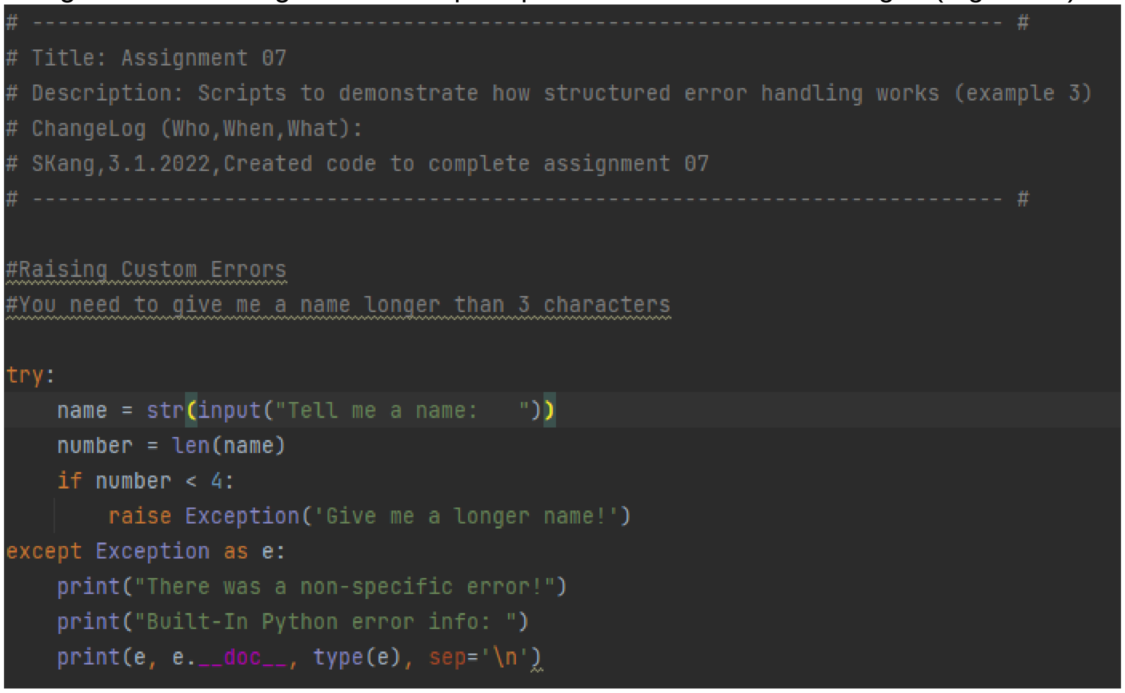
```
C:\_PythonClass\Assignment07_SangheeKang\venv\Scripts\python.exe C:/_PythonClass/Assignment07_SangheeKang/Assignment07_SangheeKang_errorhandling2.py
Please select an index in the range!
Built-In Python error info:
list index out of range
Sequence index out of range.
<class 'IndexError'>

Process finished with exit code 0
```

Figure 8. Output of Assignment07_SangheeKang_errorhandling2

3.3. Raising Custom Errors

You can raise custom errors, which are not based on conditions predetermined by Python, but based on your own custom conditions. For example, you write a code that asks the user to input a name (Figure 9). But the name should longer than 3 characters. If the user enters a short name whose length is not longer than 3 characters (e.g., Sue), an error will happen. Then, a string "Give me a longer name!" is prompted with other error messages (Figure 10).

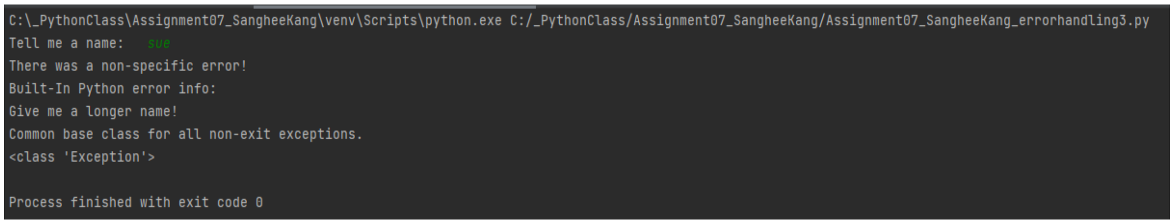


```
# ---------------------------------------------------------------------------- #
# Title: Assignment 07
# Description: Scripts to demonstrate how structured error handling works (example 3)
# ChangeLog (Who,When,What):
# SKang,3.1.2022,Created code to complete assignment 07
# ---------------------------------------------------------------------------- #

#Raising Custom Errors
#You need to give me a name longer than 3 characters

try:
    name = str(input("Tell me a name:   "))
    number = len(name)
    if number < 4:
        raise Exception('Give me a longer name!')
except Exception as e:
    print("There was a non-specific error!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```

Figure 9. Code: Assignment07_SangheeKang_errorhandling3



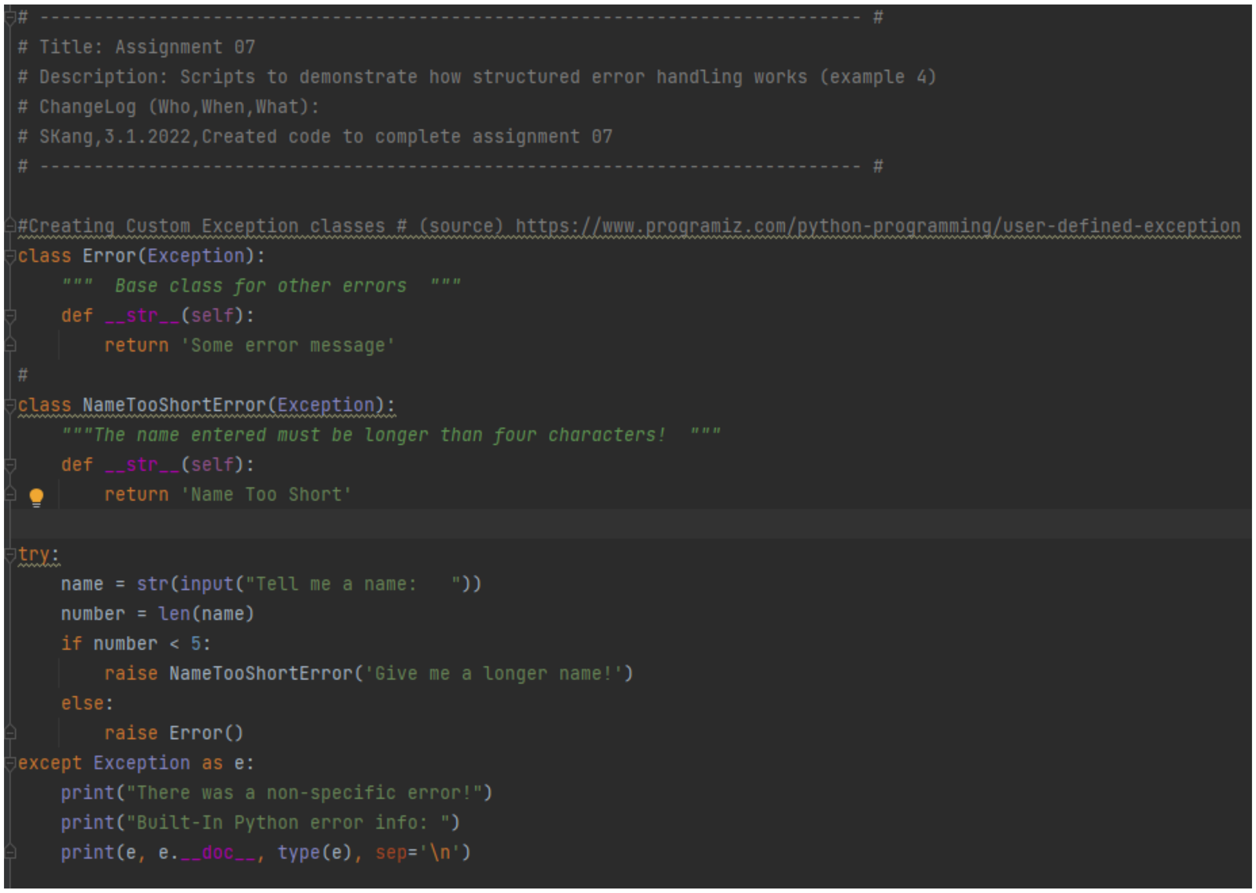
```
C:\_PythonClass\Assignment07_SangheeKang\venv\Scripts\python.exe C:/_PythonClass/Assignment07_SangheeKang/Assignment07_SangheeKang_errorhandling3.py
Tell me a name:   sue
There was a non-specific error!
Built-In Python error info: 
Give me a longer name!
Common base class for all non-exit exceptions.
<class 'Exception'>

Process finished with exit code 0
```

Figure 10. Output of Assignment07_SangheeKang_errorhandling3

3.4. Creating Custom Exception Classes

Lastly, you can create custom exception classes. In the example below (Figure 11), I basically had the same code and error, but I created Custom Except classes first and then used the try-exception function. The first class (i.e., class Error()) is for the base class for other errors. The second class is NameTooShort class, which I created to catch an error raised when the name entered by the user is not longer than three characters. For this code, I referred to the following source but I created my own example: https://www.programiz.com/python-programming/user-defined-exception). Figure 12 shows the output of the code.

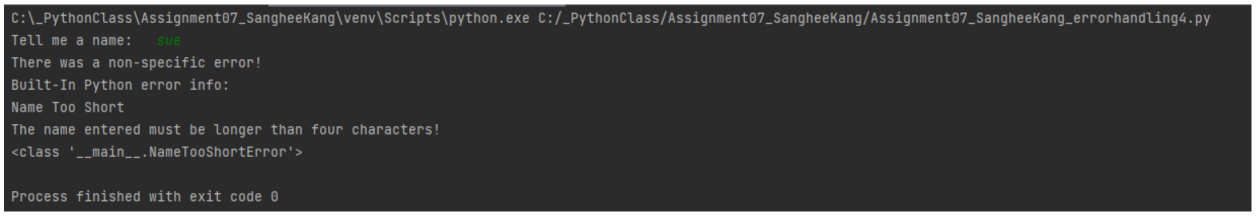


```
# ------------------------------------------------------------------------------ #
# Title: Assignment 07
# Description: Scripts to demonstrate how structured error handling works (example 4)
# ChangeLog (Who,When,What):
# SKang,3.1.2022,Created code to complete assignment 07
# ------------------------------------------------------------------------------ #

#Creating Custom Exception classes # (source) https://www.programiz.com/python-programming/user-defined-exception
class Error(Exception):
    """  Base class for other errors  """
    def __str__(self):
        return 'Some error message'
#
class NameTooShortError(Exception):
    """The name entered must be longer than four characters!  """
    def __str__(self):
        return 'Name Too Short'

try:
    name = str(input("Tell me a name:   "))
    number = len(name)
    if number < 5:
        raise NameTooShortError('Give me a longer name!')
    else:
        raise Error()
except Exception as e:
    print("There was a non-specific error!")
    print("Built-In Python error info: ")
    print(e, e.__doc__, type(e), sep='\n')
```

Figure 11. Code: Assignment07_SangheeKang_errorhandling4

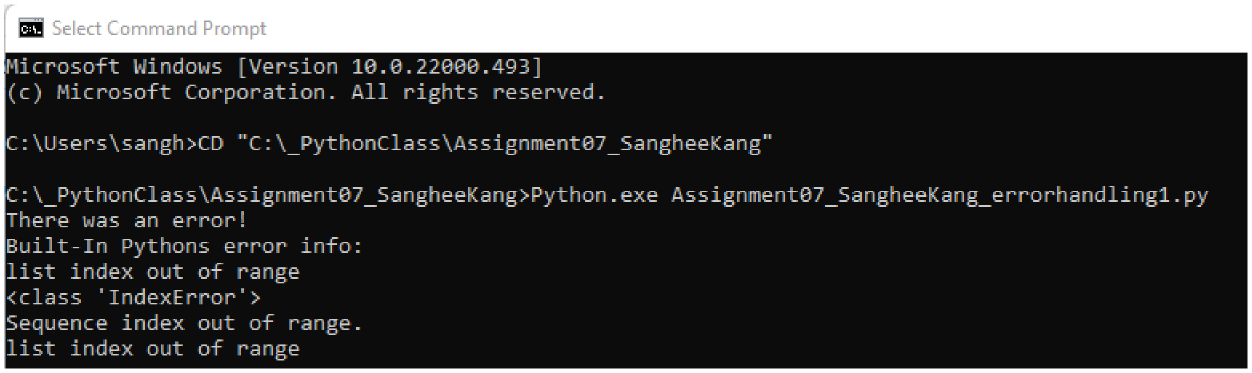


```
C:\_PythonClass\Assignment07_SangheeKang\venv\Scripts\python.exe C:/_PythonClass/Assignment07_SangheeKang/Assignment07_SangheeKang_errorhandling4.py
Tell me a name:   sue
There was a non-specific error!
Built-In Python error info:
Name Too Short
The name entered must be longer than four characters!
<class '__main__.NameTooShortError'>

Process finished with exit code 0
```

Figure 12. Output of Assignment07_SangheeKang_errorhandling4

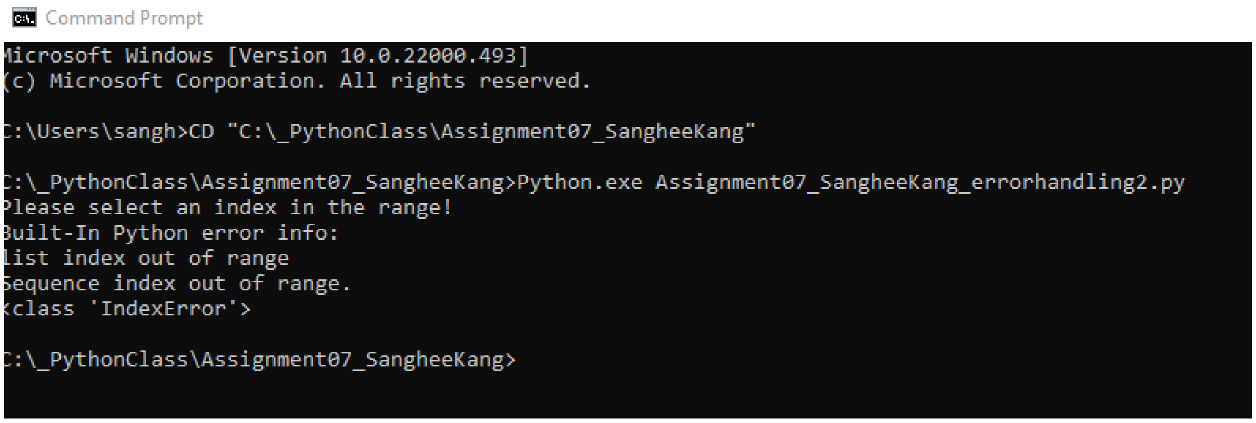
I ran the four codes in the command shell as well and obtained the same results from Pycharm:



```
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sangh>CD "C:\_PythonClass\Assignment07_SangheeKang"

C:\_PythonClass\Assignment07_SangheeKang>Python.exe Assignment07_SangheeKang_errorhandling1.py
There was an error!
Built-In Pythons error info:
list index out of range
<class 'IndexError'>
Sequence index out of range.
list index out of range
```

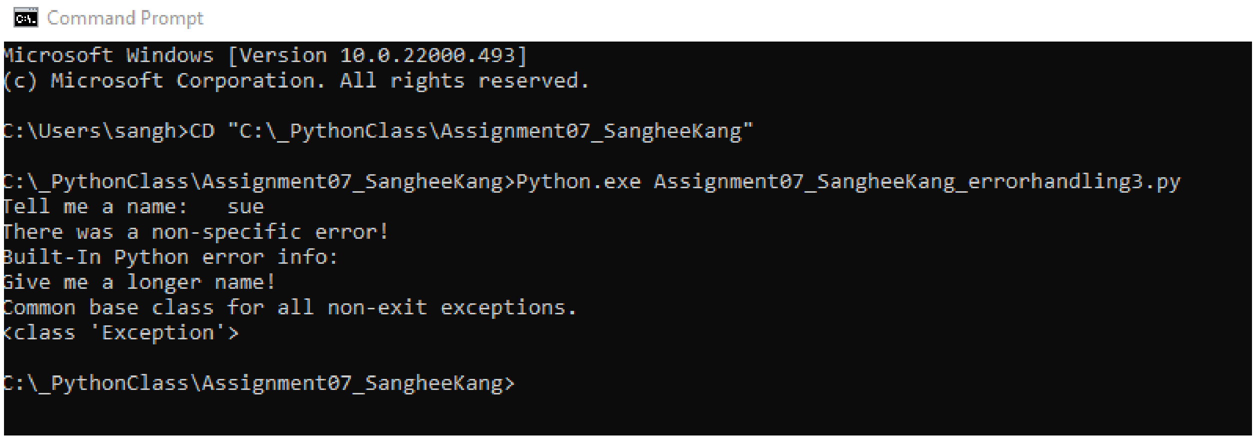


```
Command Prompt
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

:\Users\sangh>CD "C:\_PythonClass\Assignment07_SangheeKang"

:\_PythonClass\Assignment07_SangheeKang>Python.exe Assignment07_SangheeKang_errorhandling2.py
Please select an index in the range!
Built-In Python error info:
list index out of range
Sequence index out of range.
<class 'IndexError'>

:\_PythonClass\Assignment07_SangheeKang>
```

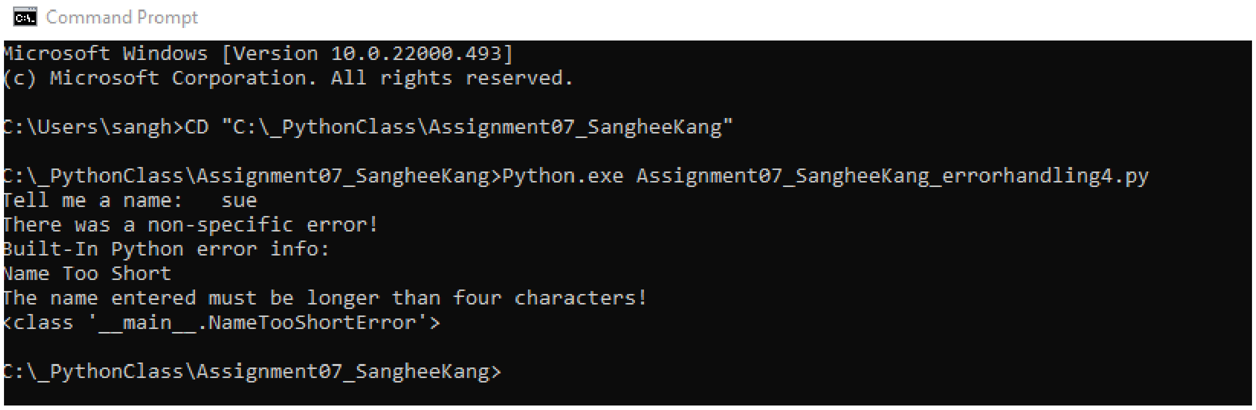


```
Command Prompt
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

:\Users\sangh>CD "C:\_PythonClass\Assignment07_SangheeKang"

:\_PythonClass\Assignment07_SangheeKang>Python.exe Assignment07_SangheeKang_errorhandling3.py
Tell me a name:   sue
There was a non-specific error!
Built-In Python error info:
Give me a longer name!
Common base class for all non-exit exceptions.
<class 'Exception'>

:\_PythonClass\Assignment07_SangheeKang>
```



```
Command Prompt
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

:\Users\sangh>CD "C:\_PythonClass\Assignment07_SangheeKang"

:\_PythonClass\Assignment07_SangheeKang>Python.exe Assignment07_SangheeKang_errorhandling4.py
Tell me a name:   sue
There was a non-specific error!
Built-In Python error info:
Name Too Short
The name entered must be longer than four characters!
<class '__main__.NameTooShortError'>

:\_PythonClass\Assignment07_SangheeKang>
```

Figure 13. Output from the Command shell

3. Summary

For this assignment, I had to do my own research to get to know more about pickling and structured error handling. By referring to what others explained about the concepts and creating my own example code, I could learn what the two concepts are and how they work in Python.