Maggie Valencia
March 4, 2023
IT FDN 110 A
Assignment07 Part I

Assignment 07 - Exception Handling in Python

In Python, handling errors is done with the use of the **Try-Except** block. This is an important mechanism that puts guardrails around your program to prevent it from crashing or having unintended consequences when a human provides wrongful inputs. When building programs, take the time to think of and implement all potential scenarios in which a program might not function or could crash the program.

There are several topics covered in this document related to error handling, these include: (a) Exceptions; (b) Handling Errors with Built-In Exceptions and (b) User-defined exceptions.

Exceptions

In Python, there are Built-In Exceptions that are raised when the execution of a program is disrupted due to some type of error in the logic of the program. In the example in figure 2, **FileNotFoundError** exception is raised when the file **'MyData.txt'** cannot be found. When this happens the program simply terminates and stops working.

```
# ---------------------------------------------- #
# Title: Error Handling Example 01 (Built-In)
# Description: A simple example of error handling
# ChangeLog: (Who, When, What)
# MValencia,2.26.2022,Created Script
# ---------------------------------------------- #

# Declare my variables
lstRow = []
strFile = None
objFile = None


# Process the data
strFile = input("Enter file name: ")
objFile = open(strFile, "r")
print(objFile)
objFile.close()

# Presentation
print(objFile)
```

Figure 1 - Opening File

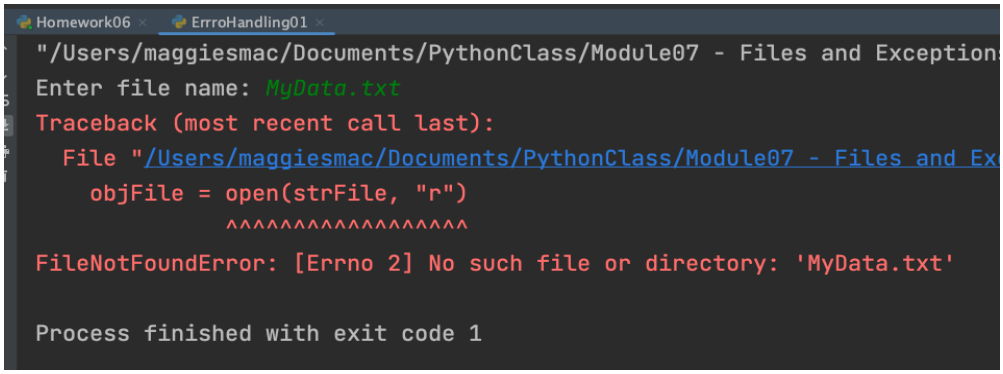

Figure 2 - Exception FileNotFoundError

Handling Errors with Built-In Exceptions

System errors that are generated by Python are oftentimes difficult for a human to understand. In Python, the **Try-Except** block is a way for a programmer to pass more user-friendly messages to help a user recover from an error. In the example below, in figure 3, by using the **except** clause and the **FileNotFoundError** exception, a more user-friendly message is passed to help recover from an error. There are many other Exception classes pre-built in Python that handle many types of errors, most of these are defined under the **Exception** class, such as the ArithmeticError for calculations errors, OSError for I/O errors or SyntaxError, when invalid syntax is used, that a programmer can include these in their code with **Try-Except** to anticipate all the different ways things can go wrong.

```
# ------------------------------------------------ #
# Title: Error Handling Example 01 (Built-In)
# Description: A simple example of error handling
# ChangeLog: (Who, When, What)
# MValencia,2.26.2022,Created Script
# ------------------------------------------------ #

# Declare my variables
lstRow = []
strFile = None
objFile = None

# Process the data
strFile = input("Enter file name: ")
try:
   objFile = open(strFile, "r")
   print(objFile)
   objFile.close()
except FileNotFoundError:
   print("File does not exist in directory\n"
         "\tPlease provide the path or create a new file called,", strFile, "...try again")

# Presentation
print(objFile)
```

Figure 3 - Built in Exceptions

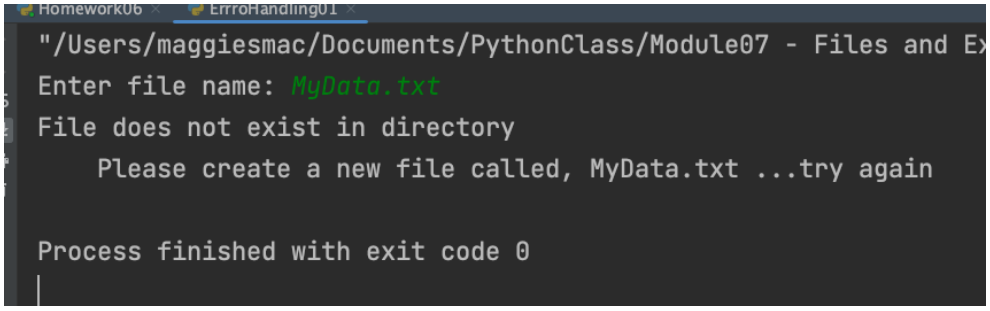

figure 4 - User Friendly Exception Error

User-defined exceptions

User-defined exceptions in Python are created by programmers, see Figure 5. These are not built-in to the Python Exception class and are a useful way to place constraints on the input values that we want a program to accept. Examples are when we are looking for a specific range of numbers from a user, only wanting integers or strings as inputs or some other specific input to ensure the program works as expected.

In this example in Figure 5, we define AgeInvalidError under the Exception class if for example we don't want an user to enter a value that is less than 0 - we can put these parameters in and when a user enters a negative number a message is printed asking for a valid value:

```
# Processing  ------------------------------------------------------------- #
class AgeInvalidError(Exception):
    """  Age or year value must be valid """
    def __str__(self):
        return 'Invalid correct age range or year'

class Processor:
    """  Performs Processing tasks """

    @staticmethod
    def calculate_year_born(age):
        '''  Calculates the year someone is born

        :param age:
        :return: none
        '''
        try:
            from datetime import date  # importing date class from datetime module
            todays_date = date.today()  # creating the date object of today's date
            if age <= 0:
                raise AgeInvalidError()
            year_born = todays_date.year - age
            print("The person was born in", year_born)
            print("================================")
            print()
        except AgeInvalidError:
            print("Enter the correct age.")
```

Figure 5 - User Defined Exception

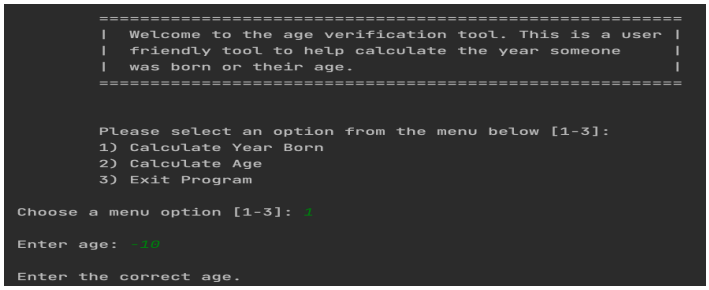

Figure 6 - Incorrect Age Value

Learning Sites

I picked the following learning sites on the web because I found the information useful in how exception handling was defined and examples provided. The built-exceptions page in Python.org website does a good job going through the built-in exceptions and the logic of how they are organized. The two videos demonstrate easy examples around error handling.

Sites

Tutorialspoint - Exception Handling
Learning Python - Exception Handling
Learning Python - Errors and Exceptions
Python.org - Built-in Exceptions

Videos

Python Programming Tutorial - 28 - You are the only Exception
Python Tutorial: Using Try/Except Blocks for Error Handling

Program Running in PC

```
        ============================================================
        |  Welcome to the age verification tool. This is a user |
        |  friendly tool to help calculate the year someone     |
        |  was born or their age.                               |
        ============================================================


        Please select an option from the menu below [1-3]:
        1) Calculate Year Born
        2) Calculate Age
        3) Exit Program

Choose a menu option [1-3]: 1

Enter age: 20

The person was born in 2003
================================


        Please select an option from the menu below [1-3]:
        1) Calculate Year Born
        2) Calculate Age
        3) Exit Program

Choose a menu option [1-3]: 2

Enter year born: 2020

The person is 3 years old
================================

        Please select an option from the menu below [1-3]:
        1) Calculate Year Born
        2) Calculate Age
        3) Exit Program

Choose a menu option [1-3]: 3

Thank for use the age tool, goodbye!

Process finished with exit code 0
```

Figure 7 - Running in PC

Running in Terminal

Figure 8 - Running in Terminal