Assignment 8 – Exception Handling

Exceptions

An *exception* is a signal that an error or other unusual condition has occurred. In python, there are a number of built-in exceptions, which indicate conditions like reading past the end of a file, or dividing by zero.

To start, we'll look at some basic exceptions you've probably encountered.

Throwing an Exception – ValueError

1. Code the following:

   Screen Capture #1 (1 point)

```
#! /usr/bin/env python3

# Get a number from the user
number = int(input("Enter an integer to square: "))

# Display the number squared
print("Display the number squared: " + str(number * number))
```

```
Run test
C:\Users\Saddleback\AppData\Local\Programs\Python\Python36\python.exe C:/Users/
Enter an integer to square: six
Traceback (most recent call last):
  File "C:/Users/Saddleback/PycharmProjects/test/test.py", line 4, in <module>
    number = int(input("Enter an integer to square: "))
ValueError: invalid literal for int() with base 10: 'six'

Process finished with exit code 1
```

Handling Exceptions - try statement

2. Code the following:

   Screen Capture #2 (1 point)

```
#! /usr/bin/env python3

try:
    # Get a number from the user
    number = int(input("Enter an integer to square: "))

    # Display the number squared
    print("Display the number squared: " + str(number * number))
except ValueError:
    # Display an invalid entry message
    print("You entered an invalid integer")
```

```
Run test
C:\Users\Saddleback\AppData\Local\Programs\Python\Python36\python.
Enter an integer to square: seven
You entered an invalid integer

Process finished with exit code 0
```

Total Calculator Program

In this program, you will create a program to get a price and quantity and calculate the total. You will need to check to make sure both the price (float) and quantity (integer) entries are valid before moving to the next process.

3. Create a new python file names **total_calculator.py** (Make sure you attach this file when you submit assignment 8)

   The output should look like this:
   Code Validation – total_calculator.py (2 points)
   Screen Capture #3 (2 points)

```
C:\Users\Saddleback\AppData\Local\Programs\P
The Total Calculator program

Enter price: 9,99
Invalid decimal number. Please try again.
Enter price:  9.99
Enter quantity: $
Invalid integer. Please try again.
Enter quantity: 5

PRICE:      9.99
QUANTITY:   5
TOTAL:      49.95

Process finished with exit code 0
```

Handling Multiple Exceptions - except statements

In the previous examples, we handled the ValueError exception. In reality there are a lot more exceptions and you can handle more than one by simply stacking more exceptions.

4. Create a new **exceptions.py** file.
5. Code the following:

   Screen Capture #4 (1 point)

```python
#! /usr/bin/env python3

# Prompt for the filename
filename = input("Enter the OS filename: ")

OS = []

try:
    # Try to open the file
    with open("c:\\cimp8a\\" + filename) as file:
        for line in file:
            # Add the OS to the list
            line = line.replace("\n", "")
            OS.append(line)
except FileNotFoundError:
    # Display a file not found error
    print("Could not find the file " + filename)
except OSError:
    # Display an error reading file message
    print("File found - error reading file")
except Exception:
    # Display a generic error message
    print("An unexpected error occurred")
```

```
Run: exceptions
C:\Users\Saddleback\venv\Scripts\python.exe "C:/Users/Sa
Enter the OS filename: OS.text
Could not find the file OS.text

Process finished with exit code 0
```

The Exception Object

The exception object contains information about the exception that can help determine what caused the error that just occurred. Additionally, the exit() method of the sys module can be called to terminate the application.

6. Modify the following:

```
#! /usr/bin/env python3

import sys

# Prompt for the filename
filename = input("Enter the OS filename: ")

OS = []

try:
    # Try to open the file
    with open("c:\\cimp110\\" + filename) as file:
        for line in file:
            # Add the OS to the list
            line = line.replace("\n", "")
            OS.append(line)
except FileNotFoundError as e:
    # Display an invalid entry message
    print("FileNotFoundError:", e)
    sys.exit()
except OSError as e:
    # Display an error reading file message
    print("OSError", e)
    sys.exit()
except Exception as e:
    # Display a generic error message
    print(type(e), e)
    sys.exit()
```

Screen Capture #5 (1 point)

```
C:\Users\Saddleback\AppData\Local\Programs\Python\Python36\python.exe C:/Users/Sa
Enter the OS filename: os.text
FileNotFoundError: [Errno 2] No such file or directory: 'c:\\cimp110\\os.text'

Process finished with exit code 0
```

The Movie List Program

For this version of the application, we'll handle some of the exceptions.

Make a copy the movie-4.py file and rename to **movie-5.py**.

Code Validation – movie-5.py (4 points)
Screen Capture #6 (2 point)

```
"C:\Users\Saddleback\PycharmProjects
The Movie List program

COMMAND MENU
list - List all movies
add -  Add a movie
del -  Delete a movie
exit - Exit program

Could not find movies.csv file.
Terminating program.

Process finished with exit code 0
```

- Create a new function for exiting the application

```
#!/usr/bin/env python3
import csv
import sys

FILENAME = "movies.csv"


def exit_program():
    print("Terminating program.")
    sys.exit()
```

- In the function the reads the movies file:
  - Add a generic exception
    - Exit the program

```
def read_movies():
    try:
        movies = []
        with open(FILENAME, newline="") as file:
            reader = csv.reader(file)
            for row in reader:
                movies.append(row)
        return movies
    except FileNotFoundError as e:
        print("Could not find " + FILENAME + " file.")
        exit_program()
    except Exception as e:
        print(type(e), e)
        exit_program()
```

- In the function that writes the movies back into the file:
  - Add a generic exception
    - Exit the program

```
def write_movies(movies):
    try:
        with open(FILENAME, "w", newline="") as file:
            writer = csv.writer(file)
            writer.writerows(movies)
    except Exception as e:
        print(type(e), e)
        exit_program()
```

- In the Add movie function:
  - Add a ValueError exception for entering the movie year
    - Prompt for another entry

```python
def add_movie(movies):
    title = input("Name: ")
    while True:
        try:
            year = int(input("Year: "))
            movie = (title, year)
            movies.append(movie)
            write_movies(movies)
            print(movie[0] + " was added.\n")
            break
        except ValueError:
            print("Invalid entry for year, please try again.")
```

Screen Capture #7 (2 point)

```
"C:\Users\Saddleback\PycharmProjects\Assig
The Movie List program

COMMAND MENU
list - List all movies
add -  Add a movie
del -  Delete a movie
exit - Exit program

Command: del
Number: aaa
Invalid movie number. Please try again.
Number: 2
The Shawshank Redemption was deleted.
```

- In the function for deleting movies:
  - Add a ValueError exception for when the user enters an invalid movie number to delete
    - Prompt for another entry (while loop?)

```python

def delete_movie(movies):
    while True:
        try:
            number = int(input("Number: "))
            if number < 1 or number > len(movies):
                print("There is no movie with that number. " +
                      "Please try again.")
            else:
                movie = movies.pop(number - 1)
                write_movies(movies)
                print(movie[0] + " was deleted.\n")
                break
        except ValueError:
            print("Invalid movie number. Please try again.")
            continue

```

finally clause

The finally clause is typically used to “cleanup” after the try clause because the code in the finally will run at the end of the try or except clause if an exception is thrown.

7. Code the following:
   Screen Capture #8 (2 point)

```python
#! /usr/bin/env python3

import sys

# prompt for the file name
filename = input("Enter the OS filename: ")

OS = []

try:
    # open the file
    with open(filename) as file:
        try:
            for line in file:
                # add the os from the file to the list
                line = line.replace("\n", "")
                OS.append(line)

        except Exception as e:
            # display a generic error message
            print(type(e), e)
            sys.exit()
        finally:
            file.close()

except FileNotFoundError as e:
    # display a generic error message
    print(type(e), e)
    sys.exit()
```

raise an Exception

With the raise, you have the ability to throw a custom exception error. To raise an error:

- raise ExceptionName("*Error message*")
- raise ValueError("Invalid value")

8. Code the following:
    a. Add the "import os" so we can check the length of the file to see if it's empty.

Screen Capture #9 (1 point)

```
#! /usr/bin/env python3

import sys
import os

# prompt for the filename
filename = input("Enter the OS filename: ")
OS = []

try:
    # open the file
    with open(filename) as file:
        try:
            if os.stat(filename).st_size == 0:
                raise ValueError("The file is empty")
            for line in file:
                # add the os from the file to the list
                line = line.replace("\n", "")
                OS.append(line)
        except Exception as e:
            # display a generic error message
            print(type(e), e)
            sys.exit()
        finally:
            file.close()
except FileNotFoundError:
    # display a file not found error
    print("Could not find the file " + filename)
    sys.exit()
```

9. Run the application:
    a. You will need to create an empty file in the same folder as the python file.

Screen Capture #10 (1 point)

```
"C:\Users\Saddleback\PycharmProjects\Assig
Enter the OS filename: os.txt
<class 'ValueError'> The file is empty

Process finished with exit code 0
```

Extra Credit

To get full points for each extra credit, you must include screen captures of the running output as well as the python (.py) code files.

Extra Credit #1 – Tip Calculator (+1 Extra Credit)

Add exception handling to a Tip Calculator program.

```
Tip Calculator

INPUT
Cost of meal: ten
Must be valid decimal number. Please try again.
Cost of meal: -10
Must be greater than 0. Please try again.
Cost of meal: 52.31
Tip percent:  17.5
Must be valid integer. Please try again.
Tip percent:  20

OUTPUT
Cost of meal: 52.31
Tip percent:  20%
Tip amount:   10.46
Total amount: 62.77
```

Specifications:

- Using the code from Assignment 3, EC #1 – Tip Calculator.
- The program should accept decimal entries like 52.31 and 15.5 for the cost of the meal.
- The program should accept integer entries like 15, 20, 25 for the tip percentage.
- The program should validate both user entries. That way the user can't crash the program by entering invalid data.
- The program should only accept numbers that are greater than 0.
- The program should round results to a maximum of two decimals.

Extra Credit #2 – Wizard Inventory (+1 Extra Credit)

Add exception handling to a program that keeps track of the inventory of items that a wizard can carry. If you've done extra credit 7-1, you can add the exception handling to that program. Otherwise, you can start this program from scratch.

```
The Wizard Inventory program

COMMAND MENU
walk - Walk down the path
show - Show all items
drop - Drop an item
exit - Exit program

Could not find inventory file!
Wizard is starting with no inventory.

Command: walk
While walking down a path, you see a crossbow.
Do you want to grab it? (y/n): y
You picked up a crossbow.

Command: show
1. a crossbow

Command: drop
Number: x
Invalid item number.

Command:
```

The error message if the program can't find the items file.

```
Could not find items file.
Exiting program. Bye!
```

Specifications:

- This program should read the text file names wizard_all_items.txt that contains all the items a wizard can carry. You can find this file on Canvas.
- When the user selects the walk command, the program should randomly pick one of the items that were read from the text file and give the user the option to grab it.
- The current items that the wizard is carrying should be saved in an inventory file. Make sure to update this file every time the user grabs or drops an item.
- The Wizard can only carry four items at a time. For the drop command, display an error message if the user enters an invalid integer or an integer that doesn't correspond with an item.
- Handle all exceptions that might occur so that the user can't cause the program to crash. If the all items file is missing, display an appropriate error message and exit the program.
- If the inventory file is missing, display an appropriate error message and continue with an empty inventory file for the user. That way, the program will write a new inventory file when the user adds items to the inventory.