Lab: Dealing with Errors

This lab session is about error handling in Python. Programming errors are inevitable and dealing with them is a skill you must have in your arsenal. At the end of this session, you will:

- Apply exception handling constructs to gracefully deal with errors.
- Use PDB to debug your Python programs.

Activity 1: Exception handling and Assertions

- Take a moment to recap the usage of **try…except**, **try…except…else**, and **try…finally** exception handling constructs.
- What are assertions? How do you assert in Python?

1. Write a program that accepts two integer numbers (say, X and Y) from the user and computes the following value: (X + Y) / X^Y. Your program must handle possible exceptions scenario (e.g. x=0, non-numeric value given as input, etc) and display appropriate message.

2. Create a list of any 10 members. Write a program that prompts the user for the index of the element to access and displays that element. Handle possible exception scenario.

3. First off, define four functions for each of the four basic arithmetic operations in a file named myOwnModule.py. Now, write a program (in another file) to accept two numbers from the user and display the results of the four operations. Your program must use the functions you defined in myOwnModule.py. Handle possible exception scenario (e.g. wrong module name to import, division by zero, etc) and display appropriate message.

4. Define a function that accepts two numbers (say, A and B) and computes the value: sin(A)Cos(B) / $\sqrt{B}$. You must ascertain that B is a positive number. Handle possible error scenario.

5. Create dictionary mapping of some details of your favorite actor/actress (such as name, age, nationality, etc). Now, prompt the user for which details s/he would like to access and display the corresponding detail. Handle possible error scenario.

Activity 2: Debugging

In this activity, you will experiment with the PDB debugger to navigate through a program and debug it in case of any error.

1. Use any one of the above program for this. Run the program through the PDB debugger:
   a. Option 1: `python/python3 –m pdb <yourProgramName>`

   b. Option 2: embed debugging code into your program wherever you want tracing to begin:
   ```
   import pdb
   pdb.set_trace()
   ```

2. Display the source code using the ___ command. How many lines are displayed by default?

3. Navigate through each line of your program using the ____ command.

4. If you have variables, you can print the values of each variable by using the **print (p)** command. Try it.

5. Set **breakpoints** at two different lines. What purpose do you think breakpoints serve?

6. What do you think is the use of the **clear** command?

7. The **help (h)** command displays more pdb commands. Get help on the **continue (c)** command. What is its purpose?

8. Get help on the **step (s)** and **return (r)** commands. You can see their usage in a function-based program.

9. Figure out what the **run** or **restart** command achieves.

10. How about the **quit (q)** command?