CMPUT 175 - Lab 5: Exceptions

Submit by: Oct 19, 2020 *Demo in your lab section starting*: Oct 20, 2020

Goal: Learn about inheritance in object-oriented programming, and practice handling exceptions.

For a simple tutorial on inheritance in Python, see:
https://www.w3schools.com/python/python_inheritance.asp

Exercise 1:

Inheritance is a fundamental concept in object-oriented programming. It allows us to define a new class that has all of the methods and properties of another class – just like a child inherits traits from a parent, the new subclass inherits properties and behaviours from its base class. It is also possible for the new subclass to have new properties and behaviours in addition to those it inherited, or to override some of the inherited properties and/or behaviours with its own unique ones.

To create a subclass in Python, you create a new class definition as normal, and then add the name of the base class (the one you want to inherit from) in brackets after the new subclass name. For example, let's define a Cake class and a BirthdayCake class, where the BirthdayCake class inherits all properties and behaviours from the Cake class and also has a unique behaviour of its own (putting birthday candles on top):

```
class Cake:
    def __init__(self, flavour, frosted):
        self.flavour = flavour
        self.frosted = frosted

    def bake(self, temp):
        print('Cake is baking at', temp, 'degrees Celsius.')


class BirthdayCake(Cake):
    def putOnCandles(self, numCandles):
        if numCandles > 0:
            print(numCandles, 'candles lit on cake; ready to be blown out.')
```

You can create objects that are instances of the BirthdayCake class, just the same way as you do for any class in Python. For example:

```
grandmasCake = BirthdayCake('chocolate', 'True')
grandmasCake.bake(180)
grandmasCake.putOnCandles(90)
```

When an instance of BirthdayCake is created, the flavour and whether it is frosted or not must be specified, just like for any cake. Also like any cake, it should be baked. But not all cakes have candles put on it – just birthday cakes.

Every type of exception that we use in Python is a subclass of the built-in Exception class. In this exercise, you will make your own custom exception.

1. Download and save **lab5_inheritance.py** from eClass. Write a class definition to create a new type of exception. The class name should be `TransitionError`.

    a. This class should be a subclass of the Exception class.

    b. This class should have its own `__init__` method to override the init method (and its attributes) that it inherited from the Exception class.
    - This new method will have two parameters (besides the self parameter): `beginning` and `final`.
    - Inside the method, you should assign `beginning` and `final` to the new attributes `previousState` and `nextState`, respectively.
    - Inside the method, you will also create a third new attribute: `msg`, which is a string explaining that we cannot transition from the `previousState` to the `nextState`.

    c. This class should have a new method called `printMsg()`. It simply prints the value of the `msg` attribute.

2. Test your `TransitionError` class by running the main function already written for you in lab5_inheritance.py. **What is the difference between the second and third try statement? i.e. why does it print different results?**

Sample run:
```
Checking first person...
Pebbles Flintstone started life as a(n) baby and ended as a(n) adult
This is the transition we expect as people grow old

Checking second person...
Benjamin Button started life as a(n) adult and ended as a(n) baby
Transition error: Not normal to transition from adult to baby

Checking second person again...
General error

Goodbye...
```

Exercise 2:

You are tasked with writing a program that reads in information about client bank accounts (where all of the clients, luckily, have unique names), and then allows the user to enter transactions (withdrawals or deposits) for any of those accounts. You will need to catch and handle specific exceptions, as described below.

1. Create a new file, **lab5_accounts.py**. In this file, create a `main()` function that asks the user for the name of the file to read the account data from. If the file does not exist, your main function should handle the resulting **OSError** exception by printing an error message and exiting the entire program gracefully (i.e. without displaying the call stack and exception information).

   Sample run when non-existing filename is entered:

```
Enter filename > sillyfile.txt
File error: sillyfile.txt does not exist
Exiting program...goodbye.
```

   If the file exists, `main` should open the file in read mode and then pass the resulting TextIOWrapper object (i.e. the file handler) as an argument to the `readAccounts` function. The `main` function should then call the `processAccounts` function with the appropriate input argument.

2. Create a function called `readAccounts(infile)` to read all of the information in the account text file into a dictionary, one line at a time. Each line in the text file should contain a (unique) name, then a greater than symbol (>), then a decimal number representing the account's opening balance. If the opening balance is not a number, this function should catch the **ValueError** exception that results when you try to convert a non-number to a float. As part of handling that exception, a warning message should be displayed (see sample run below). After catching the exception, the function should continue on to the next line in the file, without adding anything about the invalid account to the dictionary. Once all of the valid account information has been added to the dictionary, this function should return the dictionary.

   Sample run for invalid opening balance:
   Input file example:
   Bob>234.70
   Mike Smith>Not_A_Float!

```
Warning! Account for Mike Smith not added: illegal value for balance
```

3. Create a function called `processAccounts(accounts)`, where the accounts parameter is a dictionary. This function should ask the user to enter the name of a client, and the amount of money to deposit or withdraw from that client's account. After each successful transaction, the amount of money left in the client's account should be displayed. If the client name does not exist as a key in the dictionary, the resulting **KeyError** that is raised should be handled by this function. For example, if Aladdin is entered by the user, and he is not in the dictionary of clients, the following message would be displayed and the user should be re-prompted to enter another client name:

```
Warning! Account for Aladdin does not exist. Transaction cancelled.
```

Once a valid client name has been entered, the user is asked to enter a transaction amount (positive for deposit, negative for withdrawal). This amount, if valid, will be added to the client's opening balance and displayed. You should update the value in the dictionary, but you do not need to update the text file. If the user enters a value that cannot be converted to a float number, the resulting **ValueError** is caught by this function so that the following message is displayed:

```
Warning! Incorrect amount. Transaction cancelled.
```

After successfully handling the error, the function allows the user to continue entering client names and transaction amounts until s/he enters **Stop**.

4. Download and save a copy of accounts.txt from eClass. Test that all exceptions described above are handled properly. Some of those exceptions (but not all) are shown in the sample run below.

Sample Run for complete program:

```
Enter filename > accounts.txt
Warning! Account for Mike Smith not added: illegal value for balance
Warning! Account for George not added: illegal value for balance
Enter account name, or 'Stop' to exit: bob
Warning! Account for bob does not exist. Transaction cancelled
Enter account name, or 'Stop' to exit: Bob
Enter transaction amount for Bob: 30
New balance for account Bob: 264.7
Enter account name, or 'Stop' to exit: Zoya
Enter transaction amount for Zoya: hello
Warning! Incorrect amount. Transaction cancelled
Enter account name, or 'Stop' to exit: Anna
Enter transaction amount for Anna: -20.78
New balance for account Anna: 3239.77
Enter account name, or 'Stop' to exit: Bob
Enter transaction amount for Bob: 5.05
New balance for account Bob: 269.75
Enter account name, or 'Stop' to exit: Stop
Exiting program...goodbye.
```