CS-119 Lab #10

Expected Learning Objectives

- Python exception handling
- Working with classes and inheritance
- Review of general Python coding (sequential code, selection structures, etc.)

Overview

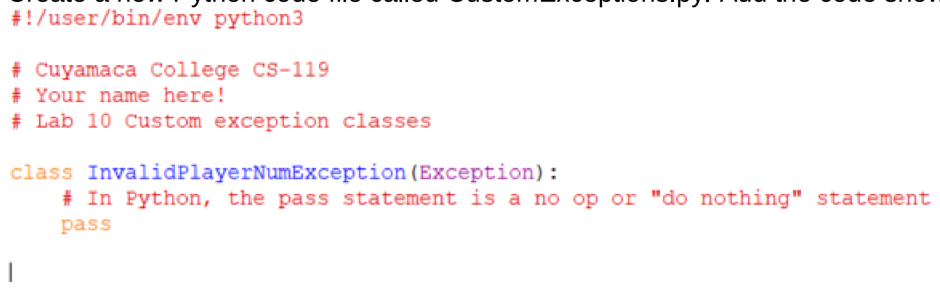
We've seen from the textbook chapter that exception handling is a vital tool that helps us write more robust, fault tolerant code. Up until now, we've put very little emphasis on error handling in our code.

Exercise 1 Player Exercise (Revisited)

For this exercise, you are going to modify the Player class created in Lab 8 and add a custom exception class for an invalid player number, throw the exception when the player number is invalid, and catch/handle it in the Python client code. You don't need to do any class models or pseudocode. The focus here will be strictly on creating a custom exception class, throwing and catching exceptions. As a best practice, and to avoid "reinventing the wheel", use the built-in Python exception classes wherever possible. Custom classes should be created to enforce business rules that aren't handled by any of the built-in exception classes.

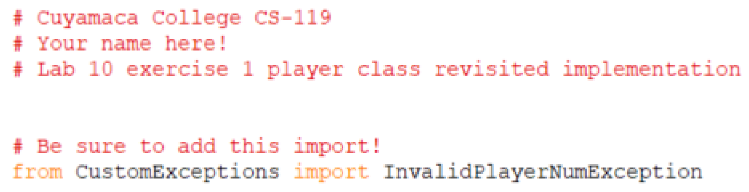
1. Copy the player files from lab 8 to your Lab 10 folder. Lab 8 ***must*** be completed before starting this!

2. Create a new Python code file called CustomExceptions.py. Add the code shown below:



```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 10 Custom exception classes

class InvalidPlayerNumException(Exception):
    # In Python, the pass statement is a no op or "do nothing" statement
    pass

|
```

3. Modify the Player class. Add an import for the custom exception:



```
# Cuyamaca College CS-119
# Your name here!
# Lab 10 exercise 1 player class revisited implementation


# Be sure to add this import!
from CustomExceptions import InvalidPlayerNumException
```

4. Modify the set_number method as shown below:

```
def set_number(self, number):
    MIN_NUMBER = 1
    MAX_NUMBER = 9999

    # Add validation to make sure number is in the expected range
    if number >= MIN_NUMBER and number <= MAX_NUMBER:
        self._number = number
    else:
        # instead of setting player number to MIN_NUMBER, raise an exeption
        #self._number = MIN_NUMBER # original code
        raise InvalidPlayerNumException
```

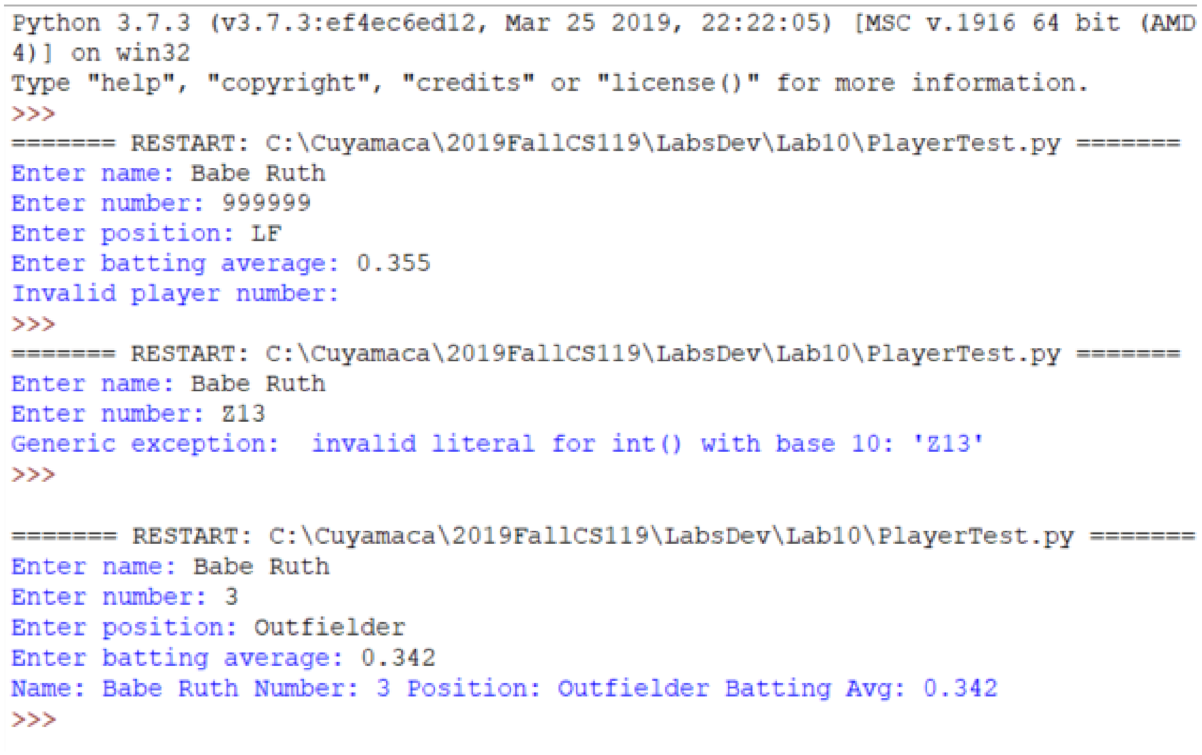
5. Modify the constructor with the code modification shown below:

```
class Player:

    # custom constructor. Constructors are named __init__()
    def __init__(self, name, number):
        # set the attributes
        # Note the single underscore _ before the variable name makes them protected
        self._name = name
        self.set_number(number) # note we call the setter method here
```

6. Modify your test fixture app with the highlighted code as shown below. Code in the try/except blocks needs to be indented 4 spaces. Be careful here!

```
#!/user/bin/env python3

# Cuyamaca College CS-119
# Your name here!
# Lab 8 exercise 2 Player class test fixture

# import the baseball player and custom exception class
from BaseballPlayer import BaseballPlayer
from CustomExceptions import InvalidPlayerNumException

# Try/except block
try:
    name = input("Enter name: ")
    number = int(input("Enter number: "))
    position = input("Enter position: ")
    batting_avg = float(input("Enter batting average: "))

    # create an instance of a baseball player
    my_player = BaseballPlayer(name, number, position, batting_avg)

    # invoke the to_string() method and display everything
    print(my_player.to_string())

# Exception handling, always go from most specific exception to most generic
except InvalidPlayerNumException as e:
    print("Invalid player number: ", e)
# generic exception
except Exception as ex:
    print("Generic exception: ", ex)
```

7. Test your application first with a player number out of range and then test again with a garbage number value. Note how the program responds:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD
4)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: C:\Cuyamaca\2019FallCS119\LabsDev\Lab10\PlayerTest.py =======
Enter name: Babe Ruth
Enter number: 999999
Enter position: LF
Enter batting average: 0.355
Invalid player number:
>>>
======= RESTART: C:\Cuyamaca\2019FallCS119\LabsDev\Lab10\PlayerTest.py =======
Enter name: Babe Ruth
Enter number: Z13
Generic exception:  invalid literal for int() with base 10: 'Z13'
>>>

======= RESTART: C:\Cuyamaca\2019FallCS119\LabsDev\Lab10\PlayerTest.py =======
Enter name: Babe Ruth
Enter number: 3
Enter position: Outfielder
Enter batting average: 0.342
Name: Babe Ruth Number: 3 Position: Outfielder Batting Avg: 0.342
>>>
```

8. You get to do this one on your own: Create a class called InvalidBattingException. Add this exception to the CustomExceptions file in step 2. Modify the method to set batting average to raise this exception if batting average is less than 0 or greater than 1.0. Be sure to trap for this exception in the test fixture application .

Exercise 2 Person, Employee and Student (Revisited)

For this exercise, you are going to modify the Person, Employee and Student classes you created in Lab 8 and add a custom exception classes for an invalid age (InvalidAge), years worked (InvalidYearsWorked) and units completed (InvalidUnits).

Valid age is 0 through 120.
Valid years worked is from 0 through 75.
Valid units is from 0 through 200.

Add the custom exceptions to the custom exceptions file you created in exercise 1. You don't need to do any class models or pseudocode. Like exercise 1, the focus will be on creating a custom exception class, raising and catching exceptions.

1. Copy the Person, Employee, Student and PersonTest Python code files from lab 8 to your Lab 10 folder. Lab 8 ***must*** be completed and working properly before starting this!

2. Code the new custom exceptions in CustomExceptions.

3. Modify the setter methods in Person, Student and Employee as needed to raise an exception for an invalid age, invalid years worked and invalid number of units.

4. Modify the PersonTest application to catch these exceptions. Make your error messages informative. Remember to keep the generic exception as the last one.

5. Test your application with both valid and invalid data and ensure your program responds appropriately.

Grading Criteria:

Deliverable	Points	Breakdown
Ex. 1 Player revisited	15	Coded, produces correct output
Ex. 1 Invalid batting average exception	10	Coded, produces correct output
Ex. 2 Custom exceptions	5	Coded
Ex. 2 class modifications	10	Coded, produces correct output
Ex. 2 test fixture modifications	10	Coded, produces correct output
Lab Total	**50**	