# CS-119 Lab #10

# Expected Learning Objectives

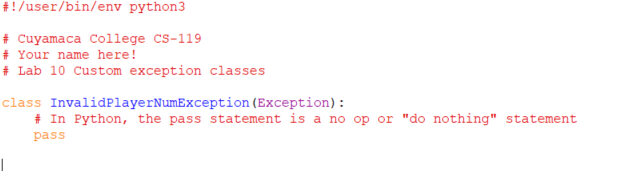
* Python exception handling
* Working with classes and inheritance
* Review of general Python coding (sequential code, selection structures, etc.)

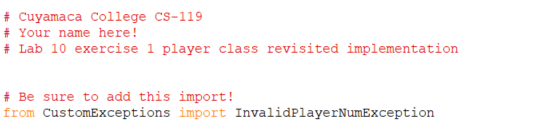
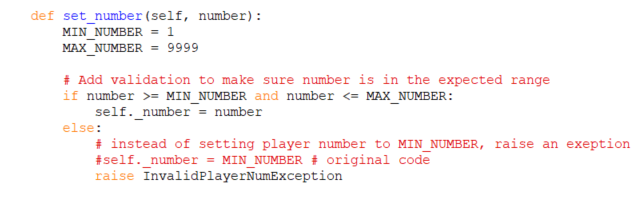
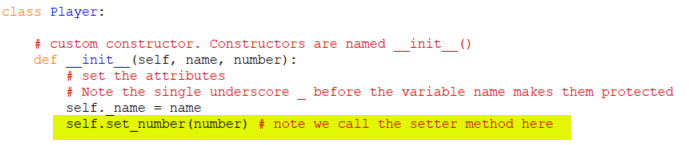
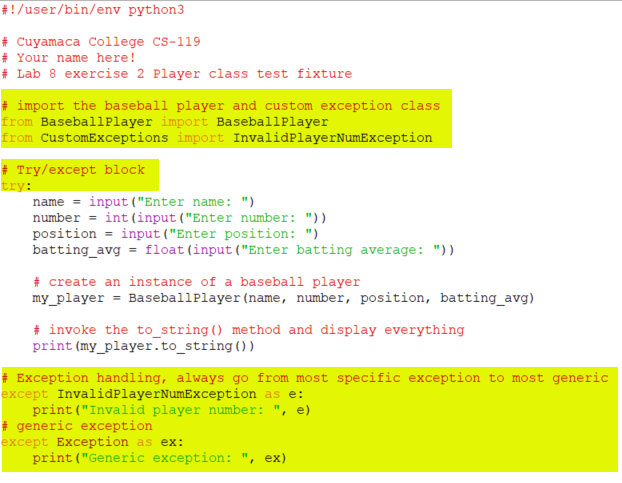
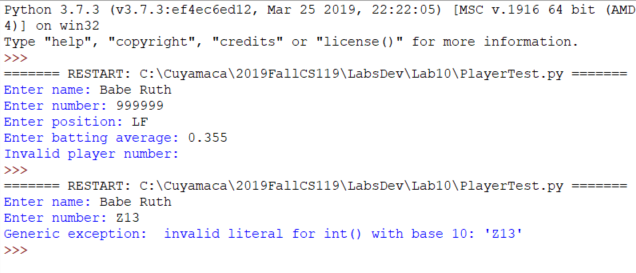
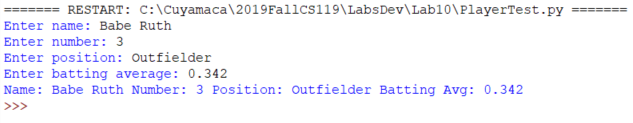
## Overview

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:  
   

1. Modify the Player class. Add an import for the custom exception:  
   
2. Modify the set\_number method as shown below:  
   
3. Modify the constructor with the code modification shown below:  
   
4. 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!  
   
5. 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:  
     
   
6. 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** |  |