**Goals**

* To raise and handle exceptions in a program
* To use filesystem operations

You will be creating a Python program that takes in a book order on the command-line, validates it and writes the order to a file. You will raise exceptions to report errors and handle those errors by displaying them to the user.

For this lab, you are given unit tests that will verify that your Python code produces the correct results.

Start by creating a Lab 10 project in PyCharm.

**Part A – Raising Exceptions and File Existence**

Copy the test\_book\_order\_utils.py file into your project in PyCharm.

Then create a Python module called book\_order\_utils.py.

Create a function with the following signature:

**def** validate\_book\_order\_details(order\_num, title, author,

isbn, year\_pub, quantity,

cost\_cad):

This function will validate each of the order details, represented by the parameters. The table below describes the validation for each parameter and the type of exception that needs to be raised. You may use regular expressions (or other means) to do the validation. Make sure you import re if you use regular expressions.

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Validation** | **Exception** |
| order\_num | One or more integer values. Note that both 1 and 0001 would be valid. | ValueError  Order Number is invalid |
| title | One or more lower or upper case letters or spaces. | ValueError  Title is invalid |
| author | Zero or more lower or upper case letters, spaces or apostrophes. | ValueError  Author is invalid |
| isbn | Must be integers. | TypeError  ISBN must be an integer |
|  | Must be between 4 and 20 digits, inclusive | ValueError  ISBN is invalid |
| year\_pub | Must be integers. | TypeError  Year must be an integer |
|  | Must be 4 digits exactly. | ValueError  Year is invalid |
| quantity | Must be integers. | TypeError  Quantity must be an integer |
|  | Must be between 0 and 1000, inclusive | ValueError  Quantity is invalid |
| cost\_cad | Must be a floating point value with exactly 2 decimal places. | ValueError  Cost is invalid |

**Run the test\_book\_order\_details.py unit test to verify your valid\_book\_order\_details function. You will get a grade of 7.5/11 if your validation is correct. If you do not, look for errors related to ValueError or TypeError in the unit test output.**

Create a function with the following signature:

**def** calculate\_per\_book\_cost\_cad(cost\_cad, quantity):

Assume cost in Canadian dollars is a floating point value and quantity is an integer value. This function should just return a floating point value that is the cost per book in the order (i.e., the cost divided by the quantity). Note that this function will raise a ZeroDivisionError exception (done automatically by Python) when the quantity (i.e., the denominator) is zero.

**Run the test\_book\_order\_details.py unit test to verify your calculate\_per\_book\_cost\_cad function. You will get a grade of 9.5/11 if your implementation is correct (along with valid\_book\_order\_details).**

Create a function with the following signature:

**def** write\_book\_order\_details(filename, title, author,

isbn, year\_pub, quantity, cost\_cad, unit\_cost\_cad):

This function will create a file with the given filename and write the order details as follows:

BOOK ORDER

title=Intro to Python

author=Bill Smith

isbn=123456

year\_pub=2010

quantity=10

cost\_cad=$500.50

unit\_cost=$50.05

If a file with the provided filename already exists, it should raise a ValueError exception with the message “Order file name already exists!”. Use a function in os.path to check for the file existence. Make sure to import os into your module.

**Run the test\_book\_order\_details.py unit test to verify your write\_book\_order\_details function. You will get a grade of 11/11 if your implementation is correct (along with valid\_book\_order\_details and calculate\_per\_book\_cost).**

**Part B – Handling Exceptions**

Now create a Python script called book\_order.py and create and call your main function.

Your main function should do the following:

* Verify there are exactly 8 command-line arguments
* Get the following from the command-line (in this order):
  + order\_num
  + title
  + author
  + isbn
  + year\_pub
  + quantity
  + cost in Canadian dollars

Note that you need to import sys to retrieve the command line arguments

If the value of your command line argument includes spaces, you need to put quotes around it when your script is called.

* Call the validate\_book\_order\_details function in the book\_order\_utils module to validate your command-line arguments.
  + Note: Make sure to import your book\_order\_utils module
* Call the calculate\_per\_book\_cost\_cad function in the book\_order\_utils module to get the unit cost of each book in Canadian dollars. Make sure to pass in the cost in Canadian dollars as a float and the quantity as an int.
* Call the write\_book\_order\_details function in the book\_order\_utils module to write the book order to file.

Add exception handling to you main function. Put the calls to the functions in book\_order\_utils above into a try block and add exception handlers for the following exceptions:

* ValueError – Print “Value Error: “ then the message from the ValueError exception.
* TypeError – Print “Type Error: “ then the message from the TypeError exception.
* ZeroDivisionError – Print “No Books in Order”

**Run the test\_book\_order.py unit test to verify your code in your script. Modify your code until all unit tests pass. You will get a grade of 4/4 if they all pass.**

Bring your code and this paper to class next week. Work with a partner! Both of you must show the same py file but running on your own computers. Put both partners’ names in a comment at the top of the file. Bring this paper to get it signed and keep these papers until the final lesson (then hand them in and get marks for each signature). Your instructor will sign your paper after making any suggestions.

Instructor Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_