CS 112, Lab 10 – Exercise (due Sat. 7/10, 11:59pm)
Exceptions

Files:

- Create your own file with our convention (*userID_2xx_L10.py)*.
- You should also download tester file from blackboard for testing.
- Run the tester as always: `python3 testerL10E.py gmason76_2B5_L10.py just these funcs`

As this is an **Exercise**, you can read any and all materials, ask us questions, talk with other students, and learn however you best learn in order to solve the task. Just create your own solution from those experiences, and turn in your work.

Reminder: you can run individual inputs, such as `get(['a','b','c'],0)` by loading your file into interactive mode with the **–i** flag. You should be using this technique on projects as well.

```
demo$ python3 –i gmason_2B5_L10.py
>>> get (['a', 'b', 'c'], 0 )
'a'
```

Exceptions provide a new form of control flow that also addresses code that crashes. We are able to identify questionable code by placing it in a try block, and then we can selectively catch and respond to different kinds of errors based on the type of issue that occurs.

You might also write code in a style of "ask for forgiveness" rather than "look before you leap" by just attempting the code (in a try block) that you know will easily work on good input, and then just write the response to failure afterwards.

In python, exception handling is particularly cheap (efficient); it's considered normal to use exception handling in many ways throughout our python code.

Turning It In

Add a comment at the top of the file that indicates your name, userID, G#, lab section, a description of your collaboration partners, as well as any other details you feel like sharing. Please also mention what was most helpful for you. Once you are done, run the testing script once more to make sure you didn't break things while adding these comments. **Remember, if you didn't use try/except blocks on a question when required, you won't get credit for it!** If all is well, go ahead and turn in just your one .py file you've been working on over on BlackBoard to the correct lab assignment. We have our own copy of the testing file that we'll use, so please don't turn that in (or any other extra files), as it will just slow us down.

What can I use?

As long as you don't import anything, there are no restrictions on what functions to use on this lab – use this time to learn how to raise exceptions, how to catch exceptions, and how to use exception handling to deal with special cases.

`get(xs, index, response=None):` Consider a list of values, **`xs`**. Find and return the value at location **`index`**. If **`index`** is invalid for **`xs`**, return **`response`** instead. Remember that .get() for dictionaries works gracefully for non-existing keys. Here we are implementing a get() function for list type.

- Parameters:
  - **`xs`** :: list of values of any length.
  - **`index`** :: an integer, specifying which value in the list to return.
  - **`response`** :: a python value, to be returned when **`index`** is invalid for the list.
- Return value: a value from **`xs`** at **`index`** or the pre-set **`response`**.
- Requirement: you must use try-except blocks in your solution!
- Examples:
  - `get(['a','b','c'],0)` → `'a'`
  - `get(['a','b','c'],3)` → `None`
  - `get(['a','b','c'],4,"oops")` → `'oops'`

`classify(input_string):` It takes as input a string with a mixed group of integers and words separated by spaces. It returns a tuple of two lists: a list of integers and a list of non-empty words. Use exceptions to distribute words and integers to their respective lists. Note: do **NOT** include empty string in the return list; any string that does not include an integer **only** (e.g. '$10' or '3.00') should be classified as words. **Hint**: check what happens when you call int() on strings like '3.00'.

- Parameter: **`input_string`** :: string with a group of integers and words separated by spaces.
- Return value: a tuple of a list of integers(int) and a list of words(string).
- Requirement: you must use try-except blocks in your solution!
- Examples:
  - `classify("I have 35 apples")` → `([35], ['I', 'have', 'apples'])`
  - `classify("Today is Sunday")` → `([],['Today', 'is', 'Sunday'])`
  - `classify("That will be $10 please")` → `([],['That','will','be','$10','please'])`

`shelve(inventory, product_list):` Given a dictionary, **`inventory`**, and a list of tuples **`product_list`**, we need to update the inventory with products listed in **`product_list`**: if a product is already in store, update the amount; if the product is not in store, add a new entry in **`inventory`** for it. We might also call this function to update **`inventory`** when the stock amount is decreased. But if the amount of any product is below zero, we must raise a **`ValueError`** with the **`"negative amount for product"`** message.

- Parameters:
  - **`inventory`** :: a dictionary with product name (string) keys and product amount (int) values.
  - **`product_list`** :: a list of tuples, each tuple has two members: a string as the product name and an integer as the amount.
- Returns: **`None`**. You should make update to **`inventory`** in-place.
- Raises: ValueError when the amount of a product is below zero.
- Examples:

```
>>> d = {"apple":50, "pear":30, "orange":25}
>>> ps = [("apple",20),("pear",-10),("grape",18)]
>>> shelve(d,ps)
>>> d
{'pear': 20, 'grape': 18, 'orange': 25, 'apple': 70}
>>> shelve(d,[("apple",-1000)])
Traceback (most recent call last):
…
ValueError: negative amount for apple
```