Lab 8: Coding Practice

This lab is designed to help you prep for P8, as well as give practice with concepts we've seen a lot of students struggling with. In particular, we'll practice refactoring, optimizing, mutating lists, and using different types of for loops.

Exercises

Refactoring

Being able to see repetition in code and rewrite as a function is key to being able to write more code in less time; programmers call this rewriting "refactoring."

The math is not important for our purposes, but the following three snippets all compute a geometric mean (https://en.wikipedia.org/wiki/Geometric_mean):

```
nums = [1.1, 0.9, 1.2, 1.3, 0.95]
mult = 1
for x in nums:
    mult *= x
result = mult ** (1/5)
result
```

```
nums = [0.8, 0.7, 0.9, 0.95]
mult = 1
for x in nums:
    mult *= x
result = mult ** (1/4)
result
```

```
nums = [1.5, 1.4, 1.5]
mult = 1
for x in nums:
    mult *= x
result = mult ** (1/3)
result
```

Your job is to (a) look for repetition in the above code and (b) identify what parts change between the examples. Then write a single function that can be used in all three cases. Your general strategy should be to fill the body of the function with the common logic (from step a) and capture the parts that change as parameters (from step b). Give it a try!

Optimizing

As we start dealing with bigger datasets, our programs might start taking a long time to run if we aren't careful. In such cases, might might need to optimize our code, to make it run more quickly.

Optimization often involves avoiding doing the same work more than once, especially when the repeated work is slow. Try running the following code in a cell. It should take about 20 seconds to run.

```
import time

# don't change this
def get_winners():
    # make this function slow, for the sake of the exercise
    time.sleep(0.1)
    return [999, 100, 150, 11, 555]

# can you make this function faster?
def count_winners(nums):
    count = 0
    for num in nums:
        if num in get_winners():
            count += 1
    return count

# don't change this either
nums = []
for i in range(200):
    nums.append(i)

print("Winners:", count_winners(nums))
```

The code repeatedly calls a slow function, `get_winners` (it wouldn't be slow normally, but we made it slow on purpose). Your job is to modify the `count_winners` function so that it does produces the same result while calling the slow function less often.

Lists: remove vs. pop

In each of the exercises, complete the code so that it prints the same thing twice. For example, the first exercise should print `[4,3,2,1,0]` twice.

```
nums = [5,4,3,2,1,0]
nums.pop(0)
print(nums)
nums = [5,4,3,2,1,0]
nums.remove(????)
print(nums)
```

```
nums = [5,4,3,2,1,0]
nums.pop(-1)
print(nums)
nums = [5,4,3,2,1,0]
nums.remove(????)
print(nums)
```

```
nums = [5,4,3,2,1,0]
nums.pop(????)
print(nums)
nums = [5,4,3,2,1,0]
nums.remove(1)
print(nums)
```

```
nums = ["A", "B", "C"]
nums.pop(1)
print(nums)
nums = ["A", "B", "C"]
nums.remove(????)
print(nums)
```

```
nums = [-2,-1,0,1,2]
nums.pop(????)
print(nums)
nums = [-2,-1,0,1,2]
nums.remove(-2)
print(nums)
```

```
nums = [-2,-1,0,1,2]
nums.pop(-3)
print(nums)
nums = [-2,-1,0,1,2]
nums.remove(????)
print(nums)
```

Type Conversion: Version 1

Complete the following code so it prints 600 at the end:

```
values = ["100", "200", "300"]
total = 0

for ???? in values:
    print("looping over value:", x)
    ???? += int(x)
print("total:", total)
```

Type Conversion: Version 2

Complete the following code so it prints 600 at the end:

```
values = ["100", "200", "300"]

for ???? in range(len(????)):
    print("looping over index:", i)
    values[i] = int(values[i])

print("total:", sum(values))
```

Copying

Try running the following code:

```
import copy

def median(nums):
    nums.sort()
    if len(nums) % 2 == 1:
        return nums[len(nums) // 2]
    else:
        v1 = nums[len(nums) // 2]
        v2 = nums[len(nums) // 2 - 1]
        return (v1+v2) / 2

values = [3,2,1]
print("The values are", values)
print("The median is", median(values))
print("The values are still", values)
```

It currently prints this:

```
The values are [3, 2, 1]
The median is 2
The values are still [1, 2, 3]
```

But it should print this:

```
The values are [3, 2, 1]
The median is 2
The values are still [3, 2, 1]
```

Modify the `median` function so it uses one of the functions in the copy module to get a copy of nums before sorting.

Project Hints

You'll need to do some conversions very similar to what we ask you to do in the section *Type Conversion: Version 2*. We also create a scenario in the project where you'll need to create a copy of your data, somewhat similar to what you need to do in the *Copying* section.