Problem 1.

In this question, you will practice with mutable and immutable objects in Python.

Part (a)

There is only one way for the string `s` to become something else:

```
s = "old string"
s = RHS
```

Write a function that computes the lowercase version of a string (reminder: `https://docs.python.org/3/library/stdtypes.html#str.lower`, which you can use inside the function), and then change the string `s` using your function.

Demonstrate what happens using `pythontutor.com`

Part (b)

There are (at least) two ways for the list `L` to become something else:

- `L = g(L)`
- `f(L)`, where `f` changes the contents of `L`.

Start with the string `L = [1, 2, 3]`. Change the list in the two ways above. Print `id(L)` before and after the change, and print `L` after the change. Explain what happens. Demonstrate what happens using `pythontutor.com`.

Write `g` in such a way that the `id(L)` changes after the assignment `L = g(L)`, and then also write a function `g1` which does not change the `id(L)` after the assignment `L = g1(L)`. Demonstrate what happens using `pythontutor.com`.

Part (c)

You can compute a shallow copy of a dictionary using `d.copy()`. Repeat Part (b), but with a dictionary `d` instead of a list `L`.

Part (d)

A shallow copy of a dictionary is fine as long as none of the values are mutable. Demonstrate an issue with Part (c) when a shallow copy is used.

Part (e)

You can compute a deep copy of a dictionary using `copy.deepcopy(d)`. (Need to import deepcopy) Repeat Part (d), but with a deep copy of the dictionary `d` instead of a shallow copy.

Reference: `https://www.programiz.com/python-programming/shallow-deep-copy`

Problem 2.

Here is the Binary Search code from the notes.

```
def binary_search(L, e):
   low = 0
   high = len(L)-1
   while high-low >= 2:
      mid = (low+high)//2 #e.g. 7//2 == 3
      if L[mid] > e:
         high = mid-1
      elif L[mid] < e:
         low = mid+1
      else:
         return mid
   if L[low] == e:
      return low
   elif L[high] == e:
      return high
   else:
      return None
```

Part (a)

Demonstrate that the code works on sorted lists of size 10 by inputting sample inputs and making sure that the code runs.

Part (b)

Modify the function to return the *number of iterations* that the while loop runs for. (Reminder: a function can return a tuple: the first element might be the index, and the second element the number of times that the while loop ran for)

Part (c)

In the worst case, binary search does not "return early" because `L[mid]` is not equal to `e`. What would be a way to construct a list L such that that never happens?

Part (d)

For list sizes n being 10, 100, 1000, 10000, ..., 10000000, output the number of iterations that the while loop takes, in the worst case, by constructing lists such that the number of iterations is as large as it can be for the given list size.

Part (e)

You can get the number of seconds since Jan 1, 1970 using `time.time()` after using `import time`. This means you can measure the time a function took to run using

```
import time
start = time.time()
#run function
end = time.time()
print(end-start)
```

Compare the runtime of "linear search" (using L.index()) and binary search on input sizes 10, 100, 1000, 10000, ..., 10000000, in the worst case, and output the results.

Make your code as nice as possible: for example, write a function that measures the runtime rather than copy-pasting the code above multiple times.

Problem 3.

Continue working on questions from Lab 8 you have not finished. Note: there is a solutions session here if you are stuck: `https://www.youtube.com/watch?v=-qiwIp4w6Ks`