Problem 1

Give the time cost of each line of code or snippet, using big-Oh notation. Assume that $n$ represents the length of the list, or other data structure.

Some of these require some thought about what Python has to do. If you are uncertain about your answer, then write down your thoughts - then ask! We’ll discuss common questions in class.

```python
x = 10

a = b  # do we need to know what type b is???

some_list.append(100)

foo = other_list[1:]

for i in some_list:
    print(i)
```
Problem 2

For each function below, give the time cost of each individual line of code (for one iteration of the line); write it next to the line (in big-Oh notation). Then, in a second column, write how many times that line of code runs (also in big-Oh). Assume that n is the size of the data passed to the function - such as the length of a list.

```python
def pretty_print_list(vals):
    print("{"")
    first = True
    for v in vals:
        if not first:
            print(", ")
        first = False
        print(v)  # assume O(1) (not always true)
    print("}\")

def find_dups(vals):
    for i in range(len(vals)):
        for j in range(len(vals)):
            if i != j:
                if vals[i] == vals[j]:
                    print(i,j)

def bubble_sort_is_terrible(vals):
    for loop_count in range(len(vals)):
        for i in range(len(vals)-1):
            if vals[i] > vals[i+1]:
                Vals[i],vals[i+1] = vals[i+1],vals[i]
```

Problem 3

For each function below, give the time cost of each individual line of code in big-Oh notation. This time, though, give the total cost for that line, across all iterations. (That is, an O(1) line, which runs O(n) times, should be listed as O(n).)

Make sure to account for the cost of other functions that these functions call!

```python
def max_min(vals):
    assert len(vals) > 0
    max = vals[0]
    min = vals[0]
    other_vals = vals[1:]
    for v in other_vals:
        if v < min:
            min = v
        if v > max:
            max = v
    return (min, max)

def wrapper(vals):
    bigOh_1_func(vals)
    for v in vals:
        bigOh_n_func(vals)

def list_to_str(vals):
    retval = ""
    for v in vals:
        new_retval = retval + str(vals)
        retval = new_retval
    return retval
```
Problem 4

Both functions below run in $O(n)$ time. However, from a casual inspection, you might think that they run in $O(n^2)$. Explain why they run in $O(n)$.

If you don’t know the answer, that’s OK - ask for help! (And write down as much as you know.)

def func1(vals, val1, val2):
    pos1 = -1
    pos2 = -2
    for i in range(len(list)):
        if vals[i] == val1:
            pos1 = i
        if vals[i] == val2:
            pos2 = i
        if pos1 < pos2:
            slice = vals[pos1:pos2]
            for v in slice:
                print(v)

def func2(vals):
    for i in range(len(vals)):
        if vals[i] > 0:
            for v in vals[:i+1]:
                print(v)
            break