15110 Summer 2018
Problem Set 3

Name: _____

Andrew ID: _____

Instructions

- Type or neatly write the answers to the following problems.
- Save or scan this file as a pdf and submit to Gradescope

Exercises

1. [2 points] Program Logic

   The following Python function prints out the heart disease risk for a person based on age and body mass index (BMI).

```
def heart_risk(age, bmi):
    if age < 45:
        if bmi < 22:
            risk = "low"
        else:
            risk = "medium"
    else:
        if bmi < 22:
            risk = "medium"
        else:
            risk = "high"
    print(risk)
```

   a. What would the function print for a 45 year old person with a bmi of 25?

   b. Find a pair of age and bmi values such that the function would print `"low"`?

   c. Draw a flowchart that shows the flow of control through the various conditions in the function. You can use lecture slides as a reference.

d. Suppose that the writer of the code made an indentation error and wrote the code as follows:

```
def heart_risk(age, bmi):
    if age < 45:
        if bmi < 22:
            risk = "low"
        else:
            risk = "medium"
    else:
        if bmi < 22:
            risk = "medium"
        else:
            risk = "high"
            print(risk)
```

What would the function print for a 45 year old person with a bmi of 21 in this case?

2. [2 pts] Revisit the function `sum(n)` from the Lecture Notes
   a. Write a Python function `sum2(lst)` by modifying `sum(n)` so that it takes a list of integers as an input, as opposed to an integer `n`, and returns the sum of the numbers in the input list. Your `for` loop should iterate over list indices produced by `range`.

   b. Note that we have learned that a `for` statement can be used to iterate over any sequence of values, not only those produced by a `range`. Write an alternative version of your function called `sum3(lst)` from part a that uses a `for` loop but does not use `range`.

   c. Now, write a function `sum4(lst)` that uses a while loop to add the all the values in the input list.

3. [2 points] Consider the following function that finds the minimum of integers given in a list.

```
def findmin(lst):
    min_num = lst[0]
    i = 1
    while i < len(lst):
        if lst[i] < min_num:
            min_num = lst[i]
        i = i + 1
    return min_num
```

   a. Show how this function computes `findmin([7,4,6,-1,4,19])` by creating a table that shows the values of each variable at the end of each iteration of the loop. We have started the table for you. The first line shows the initial values of the variables before the first iteration of the loop. The second line shows the values of the variables after the first iteration of the loop.

list	min_num	i
[7,4,6,-1,4,19]	7	1
[7,4,6,-1,4,19]	4	2

   b. What happens if we call this function with the empty list as its argument? Why?

   c. What happens if we remove the assignment statement `i = i + 1`? Why?

d. Suppose that we want the function to return the **position(index)** of the minimum element in the list. For example, the position of the minimum element in the list [7,4,6,-1,4,19] is 3. What simple change in the code would make the function return what we want?

4. [2 points]
   a. Suppose that you are given the following Python code. Describe what the function below computes and displays on screen?

```
def table1(n):
    for i in range(1, n + 1):
        row = []
        for j in range(1, n + 1):
            row.append(i*j)
        print(row)
    return None
```

   b. Consider the following variant of the above function. What does it compute and display on screen? Suppose that it is called by using 100 for `m`. For what input value `n` would the statement `row.append(i*j)` end up being executed 100 thousand times?

```
def table2(m, n):
    for i in range(1, m + 1):
        row = []
        for j in range(1, n + 1):
            row.append(i*j)
        print(row)
    return None
```

5. [2 points] Recall the implementation of the Sieve of Eratosthenes that we discussed in class
    a. How would you modify the function `sieve(n)` to return the number of primes less than or equal to n? That is, the function must now return how many primes there are that are less than or equal to n.

    b. How would you modify the function `sieve(n)` to return the largest prime less than or equal to n?

    Hint: If you understand how `sieve(n)` works and what it returns, it should be easy to answer the questions above. The answers require only minor modifications to the existing function.

    c. The function below is an incorrect implementation for the sift operation. Explain **why** it does not really do what is stated in the comment right after the function name. *Hint: Think about the effect of the `remove` method on a list.*

```
def sift_wrong(lst,k):
    # eliminates multiples of k
    for i in range(0,len(lst)):
        if lst[i] % k == 0:
            lst.remove(lst[i])
       return lst
```