EECE 230 Introduction to Computation and Programming, Sections 4, 5, 6, 7, 10, 11, and 12
Quiz I

March 2, 2019

- The duration of this exam is 2 hours and 55 minutes. Keep in mind that you need around 10 minutes at the end of the duration of the exam to submit your answers. It is your responsibility to make sure your files are correctly submitted.
- The exam consists of 4 problems for 190 points
- You can use all the material in the exam zip file on moodle (lecture slides, source code, programming assignments, and solutions). Once you download the zip file, moodle will be disconnected.
- You are also allowed to bring to the exam two double-sided handwritten cheatsheets .
- At the end of the exam, moodle will reopen for exam submission. If you would like to submit your work before the end of the exam, please talk to the proctors for instructions.
- You are asked to submit a single zip file containing your `Python` files (ending with `.py` extension). Failure to do so may lead to a failing grade on the exam. It is your responsibility to make sure your files are correctly submitted.
- You are **NOT** allowed to use the **web**. You are not allowed to use **USB's** or files previously stored on your machine.
- If you get caught violating the above rules or if you communicate with a person other than the exam proctors during the exam, you will immediately get zero and you will be referred to the appropriate disciplinary committee.
- Cell phones and any other unauthorized electronic devices are absolutely not allowed in the exam rooms. They should be turned off and put away.
- The problems are of varying difficulty. Below is a rough ordering estimate of the problems in order of increasing difficulty.
  - Level 0 (20 points): Problem 1.a
  - Level 1 (95 points): Problems 1.b, 1.c, 2, and least efficient solution of Problem 3
  - Level 2 (65 points): Efficient solutions of Problem 3, Problem 4.a, and nonefficient solution of Problem 4.b
  - Level 3 (10 points): Efficient solution of Problem 4.b
- Detailed comments are worth partial credit.
- Plan your time wisely. Do not spend too much time on any one problem. Read through all of them first and attack them in the order that allows you to make the most progress.
- Good luck!

Problem 1 (60 points). Relax: EECE 230 is fun and easy

a) **(20 points)** Write a Python program which asks the user to enter an integer n. If $n = 1$, your program should print "EECE 230 is fun". If $n = 2$, your program should print "EECE 230 is easy". If $n = 3$, your program should print "EECE 230 is fun and easy". Else, your program should print "Invalid input".

Sample input/output:

```
Enter a an integer:1
EECE 230 is fun

Enter a an integer:2
EECE 230 is easy
```

```
Enter a an integer:3
EECE 230 is fun and easy.

Enter a an integer:10
Invalid input
```

Any correct solution is worth full grade.

Submit your solution in a file called Prob1a.py including your name and ID number.

b) **(20 points)** Write a Python program which keeps on asking the user the question "Is EECE 230 fun and easy?" until the user enters "YES", "Yes", or "yes", in which case the program should print "Thank you for your input" and then stop.

Sample input/output:

```
Is EECE 230 fun and easy? Yes
Thank you for your input
```

```
Is EECE 230 fun and easy? NO

Is EECE 230 fun and easy? Maybe

Is EECE 230 fun and easy? 2

Is EECE 230 fun and easy? YES
Thank you for your input
```

Any correct solution is worth full grade.

Submit your solution in a file called Prob1b.py including your name and ID number.

c) **(20 points)** Write a function `funAndEasy(s)`, which given a string `s`, returns a modified version of `s` in which the first occurrence of "EECE 230" in `s` is replaced with "EECE 230 (which is fun and easy)". If "EECE 230" does not appear in `s`, `funAndEasy(s)` should return a string equal to `s`. If "EECE 230" appears more than once in `s`, only the first occurrence would be altered.

You are asked to solve this part the easy way using the `str.find` method associated with the string type. The `str.find` method (which we didn't cover in class) works as follows: if `s` and `t` are strings, `s.find(t)` returns the index of the first occurrence of `t` in `s`, i.e., the minimum index i such that $s[i : i + m] == t$, where $m = len(t)$. If `t` is not a substring of `t`, `s.find(t)` returns -1. For instance, `"abcde".find("abc")` returns 0, `"xyzabcxabcz".find("abc")` returns 3, and `"abxc".find("abc")` returns -1.

Test program:

```
print(funAndEasy("I am taking EECE 230 this semester"))
print(funAndEasy("... EECE 230 ... EECE 230 ..."))
print(funAndEasy("60 points so far"))
```

Output:

```
I am taking EECE 230 (which is fun and easy) this semester
... EECE 230 (which is fun and easy) ... EECE 230 ...
60 points so far
```

Any correct solution is worth full grade.

The use of slicing is helpful in this part.

Submit your solution in a file called Prob1c.py including your name and ID number.

Problem 2 (40 points). Constructing lists and strings

a) **Pyramid-shaped list (20 points).** Write a program which asks the user to enter an integer m. Your program should first construct a height-m pyramid-shaped list

$$L = [1, 2, \ldots, m-1, m, m-1, \ldots, 1],$$

and then display L, as shown in the below test program. If $m \leq 0$, L should be the empty list.

Sample input/output:

```
Enter value of m:-1
Pyramid-shaped list: []

Enter value of m:0
Pyramid-shaped list: []

Enter value of m:1
Pyramid-shaped list: [1]
```

```
Enter value of m:2
Pyramid-shaped list: [1, 2, 1]

Enter value of m:3
Pyramid-shaped list: [1, 2, 3, 2, 1]

Enter value of m:4
Pyramid-shaped list: [1, 2, 3, 4, 3, 2, 1]
```

Any correct solution is worth full grade.

Submit your solution in a file called Prob2a.py including your name and ID number.

b) **String of stars and ones (20 points).** Write a function `buildString(m)`, which given an integer `m`, returns a string consisting of one one, then two ones, ..., m ones, separated by stars as shown in the test program below. If $m \leq 0$, the string should consist of a single star.

Test program:

```
print(buildString(-1))
print(buildString(0))
print(buildString(1))
print(buildString(2))
print(buildString(3))
print(buildString(4))
print(buildString(5))
print(buildString(6))
```

```
*
*
*1*
*1*11*
*1*11*111*
*1*11*111*1111*
*1*11*111*1111*11111*
*1*11*111*1111*11111*111111*
```

Any correct solution is worth full grade.

Submit your solution in a file called Prob2b.py including your name and ID number.

Problem 3 (30 points). Sum of two squares

In this problem, **you are NOT allowed to use the power operator **, the `math` module, or any other module.**

Write a program which asks the user to enter an integer n and checks where or not n is the sum of squares of two integers. In particular, if there exist integers x and y such that $n = x^2 + y^2$, your program should print "YES" and display any such pair x, y as shown in the output of the test program below. If no such pair of integers exists, your program should print "NO".

For instance, out of the first 31 nonnegative integers $0, \ldots, 30$, the following are sums of two squares: 0 $(0^2 + 0^2)$, 1 $(0^2 + 1^2)$, 2 $(1^2 + 1^2)$, 4 $(0^2 + 2^2)$, 5 $(1^2 + 2^2)$, 8 $(2^2 + 2^2)$, 9 $(0^2 + 3^2)$, 10 $(1^2 + 3^2)$, 13 $(2^2 + 3^2)$, 16 $(0^2 + 4^2)$, 17 $(1^2 + 4^2)$, 18 $(3^2 + 3^2)$, 20 $(2^2 + 4^2)$, 25 $(0^2 + 5^2)$, 26 $(1^2 + 5^2)$, and 29 $(2^2 + 5^2)$.

Sample input/output:

```
Enter an integer: -1
NO

Enter an integer: 1
YES: 1 = 0^2 + 1^2

Enter an integer: 5
YES: 5 = 1^2 + 2^2

Enter an integer: 6
NO

Enter an integer: 7
NO

Enter an integer: 10
YES: 10 = 1^2 + 3^2

Enter an integer: 25
YES: 25 = 0^2 + 5^2

Enter an integer: 27
NO

Enter an integer: 65426
YES: 65426 = 101^2 + 235^2

Enter an integer: 65436
NO
```

Note that if the answer the "YES", you are not asked to display all valid pairs (x, y) such that $n = x^2+y^2$; one pair is enough (In general, if (x, y) is valid, then (y, x) is also valid. For certain values of n, even if we restrict our attention to $0 \leq x \leq y$, there can be multiple solutions, e.g., $25 = 0^2+5^2 = 3^2+4^2$).

Any correct solution is worth 15/30 points. Faster solutions are worth more points. The least efficient, which worth 15/30, will take a very long time of values of n such as $n = 65436$. The second least efficient solution will take almost no time on $n = 65436$ but a very long time on values of n such as $n = 73758473987$. This is worth 20/30 points. If your solution takes few seconds on values of n such as $n = 73758473987$, you will get full grade.

(If your solution takes almost no time on values of n such as $n = 73758473987$, you will get 30 bonus points, i.e., 60/30 points on this problem. Figuring this out during the exam limited time is probably difficult, so don't try achieve this objective unless you are done with all other problems.)

Submit your solution in a file called Prob3.py including your name and ID number.

Problem 4 (60 points). Minimum distance between repeated elements

The use of slicing is helpful in this problem.

a) **(30 points) Minimum distance of a list.** Implement the function `minDistance(L)`, which given a list L of numbers, finds the minimum distance between the indices of repeated elements in L. That is, `minDistance(L)` should return the minimum value of $|i - j|$, where $0 \leq i \leq n - 1$ and $0 \leq j \leq n - 1$ are different indices in L such that $L[i] = L[j]$, and n is the length of L. If all the elements in L are distinct, your function should return n (the largest possible value of $|i - j|$ is $n - 1$).

For instance, consider the list

$$L = [17, 10, 43, \underline{14}, -3, 2, 17, \underline{14}, 5, -3, 3, 12, -11, 17].$$

Then, `minDistance(L)`$= 4$ and it is achieved by the two underlined occurrences of 14. (The element 17 appears three times in L at indices 0, 6, and 13. Thus, the minimum distance between two occurrences of 17 is $6 - 0 = 6$. The element 14 appears twice in L at indices 3, and 7. Thus, the minimum distance between the two occurrences of 14 is $7 - 3 = 4$. The only other repeated element of L is -3, which appears at indices 3 and 9. Thus, the distance between the two occurrences of -3 is thus $9 - 3 = 6$.)

In each of the following lists, the closest repeated elements are underlined.

Test program/output:

```
print(minDistance([17,10,43,14,-3,2,17]))                    | 6
print(minDistance([17,10,43,14,-3,2,10]))                    | 5
print(minDistance([17,10,43,14,-3,2,17,14,5,-3,3,12,17]))    | 4
print(minDistance([17,10,10,14,-3,14,17]))                   | 1
print(minDistance([10,10,10]))                               | 1
print(minDistance([10]))                                     | 1
print(minDistance([]))                                       | 0
```

Any correct solution is worth full grade.

Submit your solution in a file called Prob4a.py including your name and ID number.

b) **(30 points) Maximum minDistance of a contiguous sublist of L of size $n//2$.** In this part, we are given a list `L` of n numbers. We are interested in sublists `S` of `L` consisting of $n//2$ contiguous elements. As described in Part (a), each such sublist `S` has an associated minimum distance `minDistance(S)`. Out of all such `S`, we would like to find an `S` such that `minDistance(S)` is as large as possible.

In particular, you are asked to implement the function `maxMinDistance(L)`, which given a list L, finds a contiguous sublist S of L of length $n//2$ such that `minDistance(S)` is maximal. In addition to S, `maxMinDistance(L)` should return the value of `d` of `minDistance(S)`. Namely, `maxMinDistance(L)` should return the tuple `(d,S)`. If there are more than one contiguous sublist of L which achieve the maximum, any one of them is a valid answer.

For instance, consider $L = [10, 10, 14, 10, 11, 14, 17, 14, 3, -3, 3]$, thus $n = 11$ and $n//2 = 5$. Then `maxMinDitance(L)`$= (3,\ [\underline{14}, 10, 11, \underline{14}, 17])$ since L has the the following contiguous sublists of length 5:

$L_1 = [\underline{10}, \underline{10}, 14, 10, 11]$ with `minDistance`$(L_1) = 1$,
$L_2 = [\underline{10}, 14, \underline{10}, 11, 14]$ with `minDistance`$(L_2) = 2$,
$L_3 = [\underline{14}, 10, 11, \underline{14}, 17]$ with `minDistance`$(L_3) = \mathbf{3}$ (maximal),
$L_4 = [10, 11, \underline{14}, 17, \underline{14}]$ with `minDistance`$(L_4) = 2$,
$L_5 = [11, \underline{14}, 17, \underline{14}, 3]$ with `minDistance`$(L_5) = 2$,
$L_6 = [\underline{14}, 17, \underline{14}, 3, -3]$ with `minDistance`$(L_6) = 2$, and
$L_7 = [17, 14, \underline{3}, -3, \underline{3}]$ with `minDistance`$(L_7) = 2$.

Test program:

```
print(maxMinDistance([10,10,14,10,11,14,17,14,3,-3,3]))
```

Output:

```
(3, [14, 10, 11, 14, 17])
```

Any correct solution is worth 20/30 points. To get a full grade, do it efficiently: do it without triply-nested loops (or disguised triply-nested loops), and more specifically do it using only doubly-nested loops.

Submit your code in a file called Prob4b.py including your name and ID number.