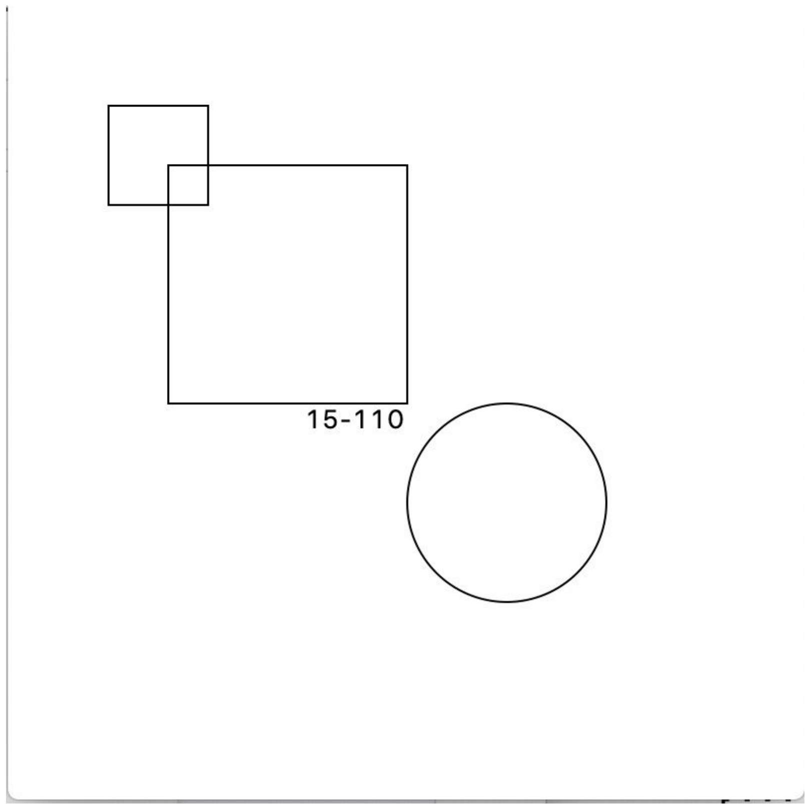
15-110 Practice Exam 2

Show work when needed, it can be used for partial credit! Also note that these questions are a rough estimate and are compiled by TA's who have not seen the exam. Topics covered in class are fair game even if they are not on this exam.

Short Answer/Multiple Choice/Fill in the blank:

Graphics
The box below is the canvas, with a width and height of 400px. Each small box is 50px by 50px. You may assume a tkinter canvas has already been run.

```
def drawCT1(canvas, w, h):
        a = 300
        canvas.create_rectangle(w/4, h/4, 50, 50, fill = "", outline = "black")
        canvas. create_rectangle(w/5, h/5, 200, 200, fill="", outline = "black")
        canvas.create_oval(w/2, h/2, a, a)
        canvas.create_text(w/2, h/2, text = "15-110", anchor = "ne")

drawCT1(canvas, 400, 400)
```



Possibly the Worst Hash Function in Existence (That's Not Just a Constant Function)

Someone has the brilliant idea of using the length of a string as its hash function. For example, hash("a") -> 1, while hash("aaaa") -> 4. Why might this be a bad idea?
There are too many collisions! hash("a") == hash("b"), and so on. If we only want one element per hash, we're up a creek. Even if we had lists of elements per hash, the lists might become so long that there would be no point in hashing in the first place.

How Fast Does It Run?

Compute the Big-O runtime of each of the following functions. Tip: listing the runtime of each line in the function may help.

```
def bigO1(s):                                   Big-O
    x = 0                                       _____
    for c in s:                                 _____
        nextC = chr(ord(c) + 1)                 _____
        x += s.count(nextC)                     _____
    return x                                    _____
```

Overall Runtime: O(n^2)
Line by Line: O(1), O(n), O(1), O(n), O(1)

```
def bigO2(L):                                   Big-O
    n = len(L)                                  _____
    newL = []                                   _____
    for i in range(n**3):                       _____
        if i % 5 == 0:                          _____
            newL.append(i)                      _____
    return newL                                 _____
```

Overall Runtime: O(n^3)
Line by Line: O(1), O(1), O(n^3), O(1), O(1), O(1)

I See Trees Of Green… And Other Ones Too

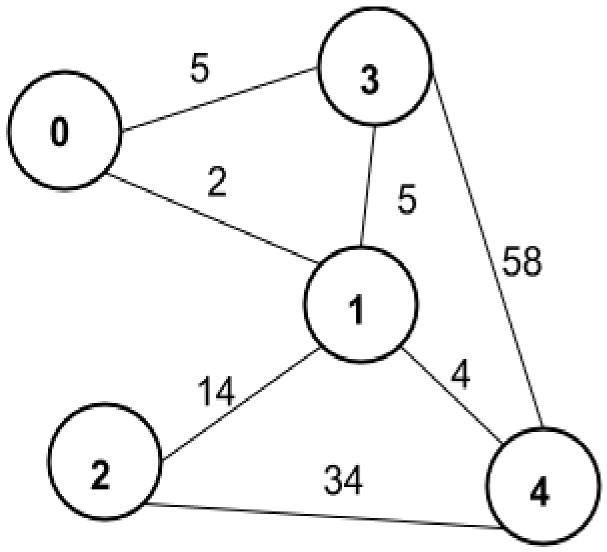
Indicate what the following function will print for the given binary tree input below.

```
def greenTrees(tree):
   numGreen = 0
   if tree["left"] != None:
      numGreen += greenTrees(tree["left"])
   if tree["value"] == "green":
      numGreen += 1
   if tree["right"] != None:
        numGreen += greenTrees(tree["right"])
    print(numGreen)
    return numGreen

tree = {"left" : {
                    "left" : {
                                "left" : None, "value" : "red",
                                "right": None
                             },
                    "value" : "green",
                    "right" : {
                                "left" : None, "value" : "green",
                                "right": None
                              }
                 },
         "value" : "green",
         "right" : {
                    "left" : None,
                    "value" : "blue",
                    "right" : {
                                "left" : None, "value" : "green",
                                "right" : None
                              }
                   }
        }

greenTrees(tree)
```

```
0
1
2
1
1
4
```

Graphs and All That

Represent the given graph as an adjacency matrix. Then, output the order in which all the nodes will be visited with BFS and DFS starting at node 1.



Adjacency Matrix:
[[None, 2, None, 5, None],
[2, None, 14, 5, 4],
[None, 14, None, None, 34],
[5, 5, None, None, 58],
[None, 4, 34, 58, None]]

BFS Output:
Any order of nodes starting with 1, eg. 1 0 2 3 4.

DFS Output:
1, 2, 4, 3, 0
Other possibilities include
0 3 4 2, 3 0 4 2, 3 4 2 0, 4 3 0 2, or 4 2 3 0

Polynomial or Nah

True or False: Some problems in P are intractable. F

True or False: NP means “not P” and therefore a problem is in NP if it is not in P.F

What is the significance of the complexity class NP-complete?
If we find a tractable solution to any NP-complete problem, then we can make all problems in NP tractable meaning that we can solve all NP problems in polynomial time.

Free Response (Code Writing):

Find My Ascii, Recursively

Write the function findMyAscii such that for an input string, recursively calculate the sum of the ascii codes of each character in the input string.

```
def findMyAscii(s):
    if s == “”:
        return 0
    else:
        return ord(s[0]) + findMyAscii(s[1:])
```

Run An Election!

Given a list of votes, return an announcement of the winner or that there was no winner.
# runElection(["Red", "Blue", "Red", "Blue", "Blue"]) -> "Blue wins!"
# runElection(["Black", "White", "White", "Black"]) -> "No winner..."

```
def runElection(votes):
    tally = {}
    for vote in votes:
        if vote not in tally:
            tally[vote] = 0
        tally[vote] += 1
    for vote in tally:
        if tally[vote] > len(votes) // 2:
            return "{} wins!".format(vote)
    return "No winner..."
```

Your Password is Safe With Me

Write a function called **IsValidLogin** with three arguments:
users a dictionary of username keys and password values (we will give this to you)
username: a string for a login name
password a string for a password.

IsValidLogin should return True if the user exists and the password is correct and False otherwise.

For example:

users =
{ "gshandar" : "ilovepitbull",
"rtn" : "feelsbadman",
"rzh1" : "grandmavibes",
“ela” : “imawesome”,
“krivers”: “stellaisthecutestdoggo”,

“srosenth”: “imsuperwoman”
  }

IsValidLogin(users, ela, “feelsbadman”) -> False
IsValidLogin(users, srosenth, “imsuperwoman”) -> True

```
def isValidLogin(users,username,password):
 for key in users:
   if key==username:
     if users[key]==password:
       return True
 return False
```