Name : ______________________________

Grading TA: ______________________________

- Integrity: By taking this exam, you pledge that this is your work and you have neither given nor received inappropriate help during the taking of this exam in compliance with the Academic Honor Code of Georgia Tech. Do NOT sign nor take this exam if you do not agree with the honor code.
- Devices: If your cell phone, pager, PDA, beeper, iPod, or similar item goes off during the exam, you will lose 10 points on this exam. Turn all such devices off and put them away now. You cannot have them on your desk.
- Academic Misconduct: Academic misconduct will not be tolerated. You are to uphold the honor and integrity bestowed upon you by the Georgia Institute of Technology.
  - Keep your eyes on your own paper.
  - Do your best to prevent anyone else from seeing your work.
  - Do NOT communicate with anyone other than a proctor for ANY reason in ANY language in ANY manner.
  - Do NOT share ANYTHING during the exam. (This includes no sharing of pencils, paper, erasers).
  - Follow directions given by the proctor(s).
  - Stop all writing when told to stop. Failure to stop writing on this exam when told to do so is academic misconduct.
  - Do not use notes, books, calculators, etc during the exam.
- Time: Don't get bogged down by any one question. If you get stuck, move on to the next problem and come back once you have completed all of the other problems. This exam has 8 questions on 10 pages including the title page. Please check to make sure all pages are included. You will have 50 minutes to complete this exam.

I commit to uphold the ideals of honor and integrity by refusing to betray the trust bestowed upon me as a member of the Georgia Tech community. I have also read and understand the requirements outlined above.

Signature: ______________________________

Question	Points	Score
1. Vocabulary	9	
2. Fill a Dictionary	3	
3. Lists/Tuples/Strings Differences	3	
4. List Contents	5	
5. Problem Reading	5	
6. Robot Drawing	8	
7. Get Positive Number	8	
8. Encode / Decode	10	
Total:	51	

1. *(9 points)*
   For each of the following vocabulary terms, write a concise 1-2 sentence definition. Be brief, and to the point.

   (a) [3 pts] exception

   Solution: An error that occurs at runtime. Can be handled using the try/except construct.

   (b) [3 pts] formatting operator

   Solution: The % sign is used to "format" strings. You provide a "format" string, the % sign, and then a value (or tuple of values) to insert into the appropriate location within the string.

   (c) [3 pts] increment

   Solution: increment - The process of increasing a variable, typically by one. aVar = aVar + 1

2. *(3 points)*
   Examine the code below. Write the contents of the dictionary after the code runs. You may either list a table of key/value pairs, or write what would be printed to the screen if you executed `print myD` at the python prompt.

```
myD = {}

for i in range(5):
   myD[i] = i*20
```

Solution:

```
Key Value
0 0
1 20
2 40
3 60
4 80
```

Grading:
3pts = table completely correct. (They can also write out a dictionary using python syntax if they get it correct...)
2 pts - Forget one table entry.
1 pt - Get just the keys or just the values.

3. *(3 points)*
List the differences and similarities between strings, lists and tuples:

Solution: Grading: +1 for each correct answer. +0.5 for any half answers.

Similarities: Strings, Lists and Tuples are all compound data types.
You can index and slice strings/tuples and lists.

Differences:
Strings are collections of characters, list/tuples are collections of any type.
Strings/Tuples are immutable, while Lists are mutable.

4. *(5 points)*
Read each of the following short pieces of code carefully, and write down exactly what would be printed when it was executed:

(a) [2 pts] Code:

```
list1=[True, 3.5, 'asdf']
list2=list1
list2[0]=1301
print list1
print list2
```

Solution:

```
[1301, 3.5, 'asdf']
[1301, 3.5, 'asdf']
```

(b) [2 pts] Code:

```
list1=[True, 3.5, 'asdf']
list2=list1[:]
list2[0]=1301
print list1
print list2
```

Solution:

```
[True, 3.5, 'asdf']
[1301, 3.5, 'asdf']
```

(c) [1 pt] Code:

```
list1 = [True, 3.5, 'asdf']
list2 = list1[1:]
print list2
```

Solution:

```
[3.5,'asdf']
```

5. *(5 points)*
The following code is supposed to print every line from foo.txt, but it does not work correctly. Write what the error is (what incorrect behavior does it exhibit), and how to fix the code.

```
def one_problem():
    fIn = file("foo.txt")
    line = fIn.readline()
    while(line != ""):
        line = fIn.readline()
        print line
    fIn.close()

one_problem()
```

Solution:

The function will NOT print the first line it reads. This is because it reads the first line in, then reads the 2nd line in before it starts to print anything out. Two ways to fix this: 1. Reverse the two lines inside the while loop. 2. Add an extra print line just before the while loop.

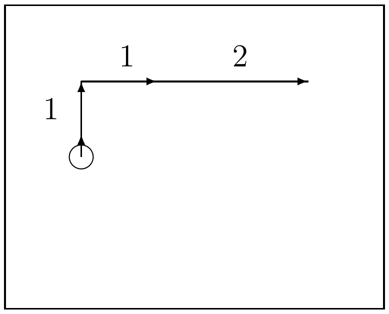
Grading:
+5 - Fully correct answer.
+3 - Knew what the problem was, but did not solve it successfully.
+3 - had correct solution, but did not indicate what incorrect behavior the code caused.

6. *(8 points)*
Robot Drawing - Assume `turn90degrees()` has been defined as below so the robot turns right 90° and `nudge(x)` has been defined to move the robot forward x units.

```
def turn90degrees():
    turnRight(1, 1)

def nudge(x):
    forward(1, x)
```

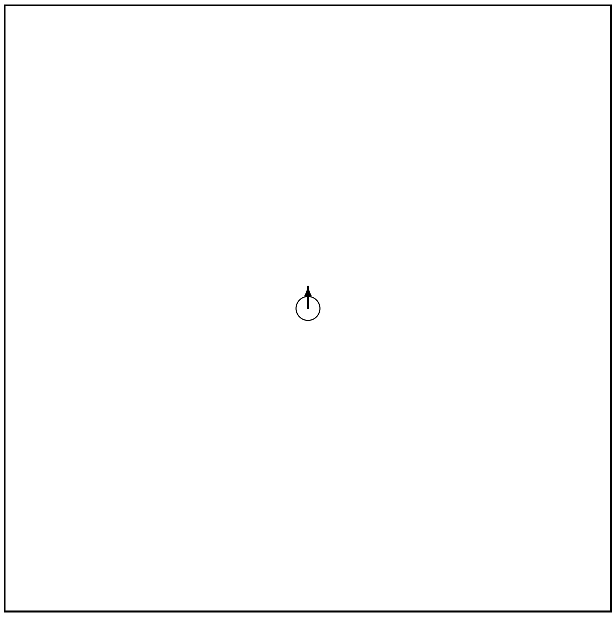
The following code makes the robot drive the trajectory drawn in the box to the right.

```
nudge(1)
turn90degrees()
nudge(1)
nudge(2)
```

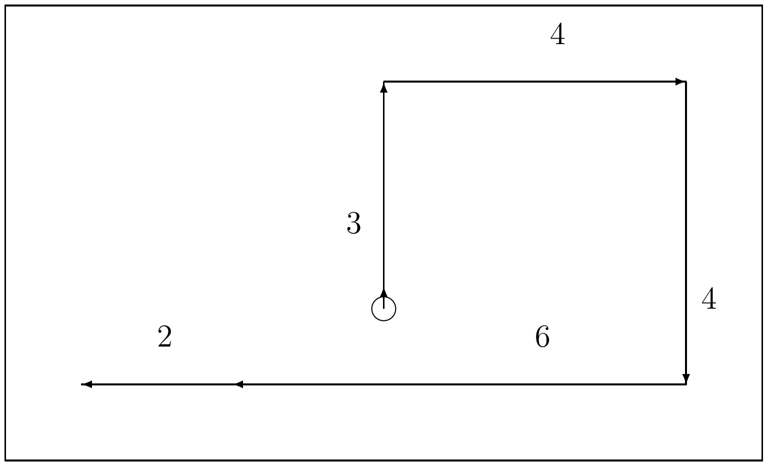


Draw the robot's trajectory when the following code is executed. Label the length of each move (nudge) using numbers as in the example above.

```
def turn90degrees():
    turnRight(1, 1)

def nudge(x):
    forward(1, x)

turns = [3,5]

for idx in [2,3,3,5,1]:
    if idx in turns:
       turn90degrees()
    nudge(idx + 1)
```



Solution:



Grading: 1 point for each correct line segment and length. 1 point for each correct turn.

7. *(8 points)*
Write a function named `getTime` that asks the user how long (in seconds) the robot should move forward. If the user enters a positive number, the function will return it as a float.

If the user enters a negative number, or zero, or anything that is not a number, the function will tell the user "Not a positive number, try again!" and then keep asking them until they do enter a positive number.

Note that your function will not actually cause a robot to move, it simply asks the user for a positive number.

Example:

```
>>> t = getTime()
Enter a positive number: Ten
Not a positive number, try again!
Enter a positive number: -5
Not a positive number, try again!
Enter a positive number: 4.5
>>>
>>>t
4.5
```

Solution:

```
def getTime():
   while True:
      userIn = raw_input("Enter a positive number:")
      try:
         userNum = float(userIn)
         if userNum > 0:
            return userNum
         else:
            print "Not a positive number, try again!"
      except:
            print "Not a positive number, try again!"
```

Grading: 2 points for prompting the user for a number and getting a string from raw_input.
2 points for trying to convert to a float
2 points for using a try/except to catch exceptions
2 points for checking that it's not negative

8. *(10 points)*
The ord function takes in a character and returns its ascii value, represented by an integer (*ord('a') returns 97*). The chr function does the opposite and takes in an integer and return the character corresponding to that value (*chr(97) returns 'a'*).

Using these built in functions, write two functions, named `decode` and `encode`. The encode function will accept a single string parameter and *return* a list of integers, one for each character in the string.

The decode function will take in a list of numbers as it's parameter and *return* a string made up of the characters represented by those numbers.

For example:

```
>>> csStr = "CS1301 rocks!"
>>> csList = encode(csStr)
>>> csList
[67, 83, 49, 51, 48, 49, 32, 114, 111, 99, 107, 115, 33]
>>> csString = decode(csList)
>>> csString
'CS1301 rocks!'
```

Solution:

```
def encode( secretMessage ):
   secretList = []
   for ch in secretMessage:
      num = ord(ch)
      secretList.append( num )
   return secretList

def decode( secretList ):
   secretMessage = ""
   for i in secretList:
      secretMessage =  secretMessage + chr(i)
   return secretMessage
```

Grading:
Encode:
1pt - Correct def statement
1pt - Correctly iterates through characters
1pt - correctly converts characters to numbers with ord()

1pt - Correctly appends number to list
1pt - returns the list

decode:
1pt - Correct def statement
1pt - Correctly iterates through numbers
1pt - correctly converts numbers to characters with chr()
1pt - Correctly appends character to string
1pt - returns the string