# CS-141 Midterm Exam Review

October 11, 2013

Presented by the RIT Computer Science Community

http://csc.cs.rit.edu

1. Write a recursive function that reverses a string (e.g. “Don’t get sick” yields “kcis teg t’noD”).

```
def reverse( string ):
    if string == '' :
        return string
    else:
        return reverse( string[1:] ) + string[0]
```

Other solutions are possible.

2. Perform a substitution trace on
`reverse('Cinco-fone')`

```
reverse('Cinco-fone')
reverse('inco-fone') + 'C'
reverse('nco-fone') + 'i' + 'C'
reverse('co-fone') + 'n' + 'i' + 'C'
reverse('o-fone') + 'c' + 'n' + 'i' + 'C'
reverse('-fone') + 'o' + 'c' + 'n' + 'i' + 'C'
reverse('fone') + '-' + 'o' + 'c' + 'n' + 'i' + 'C'
reverse('one') + 'f' + '-' + 'o' + 'c' + 'n' + 'i' + 'C'
reverse('ne') + 'o' + 'f' + '-' + 'o' + 'c' + 'n' + 'i' + 'C'
reverse('e') + 'n' + 'o' + 'f' + '-' + 'o' + 'c' + 'n' + 'i' + 'C'
'e' + 'n' + 'o' + 'f' + '-' + 'o' + 'c' + 'n' + 'i' + 'C'
'enof-ocniC'
```

3. What does the following evaluate to?

```
def writeThatDown( n ):
    if n < 5:
        return n
    return (2 * n)

def he( n ):
    temp = n + 180
    if temp > 185:
        return temp
    return n

def putstheFernback( n ):
    return -n

n = 20
n = he(putstheFernback(writeThatDown( n ) ) )
print( n )
```

-40

4. Define a function that takes an input string and rotates the sequence of letters in each word by n. For example: shift_left("DEADBEEF", 3) will produce the output string "DBEEFDEA". shift_left("Giant Robot", 4) will produce "t RobotGian"
shift_left("X", 5) will produce "X"
You should be able to shift a string by a value greater than the length of the string[1].
*Assume a function `len( str )` which returns the length of a string is provided.*

   (a) Design: Give brief description on how your function should accomplish this.
   Our implementation will grab the first part of the new string by slicing all characters at an index after the offset. The second part of the string will be all of the characters up to the offset point. We will then concatenate the two strings.
   Making it possible for the offset to be greater than the length of the string can be accomplished by making `offset = offset mod len(string)`

   (b) Testing: Provide 3 test cases, using specific values for the input string and amount of shifting and what the expected output should be for each.
   shift_left("", 5) should return ""
   shift_left("A",5) should return "A"
   shift_left("ABADCAFE",300) == shift_left("ABADCAFE",4)
   shift_left("FOO", 1) should return "OOF"

   (c) Implement the function in Python.

```python
def shift_left(string, offset):
    if len(string) == 0:
        return string
    else:
        offset = offset % len(string)
        first = string[offset:]
        last = string[:offset]
        return first + last
```

   (d) Implement the function `shift_right()`, which rotates letters in the opposite direction.

```python
def shift_right(string, offset):
    return shift_left(string, -offset)
```

[1]The % operator, which finds the remainder of a division operation will be useful here.

5. Write a function that takes in a file name, and returns the average size of a word in that file. The files will only have 1 word per line, for example:

No
soup
for
you!

which has an average length of: 3.25
*Assume a function* `len( str )` *which returns the length of a string is provided.*

```
def average_wordlength(filename):
    characters = 0
    words = 0
    for line in open(filename):
        words += 1
        characters += len(line)
    return characters/words
```

6. Write a function that takes in a string representation of a number and returns the sum of all of digits in the string.
For example sum( '11111' ) returns 5.
*Assume a function* `len( str )` *that returns the length of a string is provided.*
*Further assume there's a function* `int( str )` *which, given a string representation of an integer, returns its integer value.*

   (a) Recursively.

```
def sumRec( numbers ):
    if len( numbers ) == 0:
        return 0
    else:
        return int(numbers[0]) + sumRec( numbers[1:] )
```

   (b) Iteratively

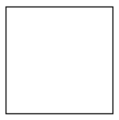
```
def sumIt( numbers ):
    total = 0
    while numbers != '':
        total += int( numbers[0] )
        numbers = numbers[1:]
    return total
```

   (c) How would you test this function?

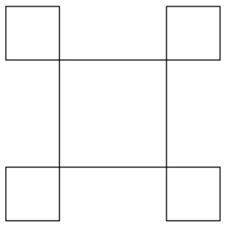
   - Empty string
   - String length = 1
   - string length > 1

7. Assuming the turtle is facing East, write the Python code to draw the following picture given the proper depth as input:

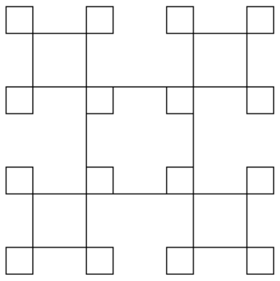
- depth = 0
  No output
- depth = 1



- depth = 2



- depth = 3



```
def drawSqaures( length, depth ):
    if depth <= 0:
        return
    count = 4
    while count > 0:
        turtle.forward( length )
        turtle.left( 90 )
        drawSqaures( length/2, depth-1 )
        turtle.right( 180 )
        count -= 1
```

8. Raymond and Connor have an issue. Raymond is a sadist and loves destroying or changing strings. Connor, on the other hand, loves keeping a count of things. You will be writing a function to satisfy both of their desires.

   (a) Write a `strhead()` function, which takes a string and returns the character in the first index of the string.

```
def strHead(str):
    return str[0]
```

   (b) Write a `strtail()` function, which takes a string and returns everything in the string except for the first index of the string (everything after the head).

```
def strTail(str):
    return str[1:]
```

   (c) Write a tail recursive function `coRec`, which takes a string and a character and returns the number of times the character appears in the string. For example, `coRec("Eric is enjoying the weather.", 'i')` should return `3`. Do not use the `len()` function.

```
def coRec(string,char,counter=0):
    if(string==''):
        return counter
    if(strHead(string)==char):
        return coRec(strTail(string),char,counter+1)
    else:
        return coRec(strTail(string),char,counter)
```

   (d) Write a iterative function `rmIter`, which takes the same parameters above and returns the string with all instances of the character removed. For example, `rmIter("Chris has hats.", 'a')` should return `"Chris hs hts."`

```
def rmIter(str,char):
    newString = ''
    for i in range(0,len(str)):
        if(str[i]!=char):
            newString=newString + str[i]
    return newString
```

(e) Write an iterative function (using a for/while loop, etc.) `rmNco`, which takes a string and a character, removes all instances of the character in the string, prints the resulting string, and lastly returns the number of times a character was removed. You may use the `len()` function. For example, `rmNco("We collectively enjoy things", 'o')` prints `"We cllectively enjy things"`, and returns 2.

```
def rmNco(str,char):
    count = 0
    endString = ''
    for i in range(0,len(str)):
        if(str[i]!=char):
            endString=endString+str[i]
        else:
            count++
    print(endString)
    return count
```

(f) What are the complexity of the functions you wrote for parts 8c, 8d, and 8e?
All three functions run in O(N) time

9. Write a function which performs a basic string compression, which uses counts of repeated characters. For example the string *abbbccccaaa* would be compressed to: *a*1*b*3*c*4*a*3.
*Assume a function* `len( str )` *which returns the length of a string is provided.*
*assume a function* `int( str )` *which, given a string representation of an integer, returns its integer value.*
*Assume a function* `str( int )` *which, given an integer, returns the string representation of this integer, is provided.*

```
def compress( string ):
    new = ''
    curChar = string[0]
    count = 0
    for char in string:
        if char == curChar:
            count += 1
        else:
            new = new + curChar + str(count)
            curChar = char
            count = 1

    new = new + curChar + str(count)
    return new
```

10. Sally is being plagued by an army of lookalike suiters, each of which presents her with an enticing but unbearable meal upon arrival. The dishes all smell amazing, however, so she can't help but try each one. A dish will never fail to disappoint her, but fortunately, some of the suiters shared recipes and created identical concoctions. Once Sally tastes a meal once, she can immediately smell out instances of the same dish and send their bearers away.

   (a) Write a Python function that, given a list of "dishes," returns a list of the rejected dishes in the order they failed to fool her. (e.g. $[1, 2, 2, 3, 3, 3]$ would return [2,3,3] )

```python
def find_dupes(lst):
    result=[]
    seen=[]
    for member in lst:
        if member not in seen:
            seen.append(member)
        else:
            result.append(member)
    return result
```

   (b) If your solution to 10a was iterative, write it recursively; if it was recursive, write it iteratively.

```python
def find_recur(lst, seen=[]):
    if len(lst)==0:
        return []
    if lst[0] not in seen:
        seen.append(lst[0])
        return find_recur(lst[1:], seen)
    return [lst[0]]+find_recur(lst[1:], seen)
```

*or*

```python
def find_tail(lst, result=[], seen=[]):
    if len(lst)==0:
        return result
    if lst[0] not in seen:
        seen.append(lst[0])
    else:
        result.append(lst[0])
    return find_tail(lst[1:], result, seen)
```

   (c) What is the time complexity of your approach? Why?

   $O(N^2)$, since the `seen` list must be searched linearly

11. For the sake of this question, you find yourself to be the head programmer under Kim Jong Un's glorious rein. It also just so happens that a nation-wide track meet is being held today. Thus, the glorious leader has demanded that you write a program to keep track of information relating to all track runners present at the event.

   (a) Write a class named `TrackRunner` to keep track of all runners competing. You will need to store their `name`, `age`, and `fastestTime`.

   ```
   class TrackRunner:
       __slots__=('name','age','fastestTime')
   ```

   (b) Now write a function to make an individual TrackRunner object.

   ```
   def makeRunner(r_name,r_age,r_fastestTime):
       runner = TrackRunner()
       runner.name = r_name
       runner.age = r_age
       runner.fastestTime = r_fastestTime
       return runner
   ```

   (c) The glorious leader has decided that, on this day, no runner named `Joe` may win gold. Given a list of `TrackRunner` objects, write the function `aWinnerIsYou(r)` that finds the runner in the list `r` with the fastest time who's name is not Joe. Then print the runner's name, age and time.

   ```
   def aWinnerIsYou(r):
       best = None
       for i in range(len(r)):
           if r[i].name != 'Joe':
               if best == None:
                   best = r[i]
               elif best.fastestTime > r[i].fastestTime:
                   best = r[i]
       return best

   best = aWinnerIsYou(runnersLst)
   print(best.name, best.age, best.fastestTime)
   ```

12. Below is Python code for a function that performs an insertion sort and prints `data` after each iteration of the `for` loop.

```
def insertion_sort( data ):
    for marker in range( 1, len( data ) ):
        val = data[marker]
        i = marker
        while i > 0 and data[i-1] > val:
            data[i] = data[i-1]
            i -= 1
        data[i] = val
        print( data )
```

(a) Write out what the function will print for the input list: [3,2,7,1].

```
1 [2,3,7,1]
2 [2,3,7,1]
3 [1,2,3,7]
```

(b) What is the sort algorithm's time complexity?

$O(N^2)$

13. **Greedy algorithms.** In the game Black and White[2], the player is faced with a row of identical double-sided chips. You can probably guess what colors the two sides of each chip are. The objective is to flip as many chips as necessary so their exposed colors match that of a target pattern.
The catch? Reordering the chips is said to be Impossible by those who seem to know what they're talking about.
The *real* catch? Flipping a group of consecutive tiles can be accomplished in a single "action." If every flip takes one "action," write a function `bwMoves` that, given a starting pattern and target pattern as equal-length strings, returns the minimum number of actions required to get them to match. For instance, `bwMoves( 'BBWBBWBBBB', 'WWWWWBBWWB' )` should return 3.

*Hint: Were any of the other questions labeled with the concept they were testing?*

```
def bwMoves(start, target):
    actions=0
    first=0
    for index in range(len(start)):
        if start[index]==target[index]:
            if first!=index:
                actions+=1
            first=index+1
    if start[-1]!=target[-1]:
        actions+=1
    return actions
```

[2]Special thanks to Professor Butler for unwittingly allowing us to rip off his problem.