CSE 101: Introduction to Computational and Algorithmic Thinking

Exam 2 Study Questions

Programming Questions: Short Answer

1. Give the output for each of the following code fragments. Note that there are no syntax errors in any of this code.

   a.
   ```
   a = 'WolfieSeawolf'
   for i in range(1,7,2):
        print(a[i])
   ```

   a. ofe

   b.
   ```
   b = 10
   b *= 2 + 2
   print(b)
   ```

   b. 40

   c.
   ```
   day = 14
   if day > 12:
       print('A')
   elif day < 16:
       print('F')
   if day > 20:
       print('Q')
   else:
       print('Z')
   ```

   c. A Z

   d.
   ```
   total = 0
   while total < 10:
       for i in range(3):
           total += i
       total *= 2
       print(total)
   ```

   d. 6 18

   e.
   ```
   x = 3
   for i in range(3):
       y = 2**x
       print(y)
   ```

   e. 8 8 8

f.
```
a = 5
b = 4
for i in range(4):
    if a > b:
        a -= 1
    else:
        b += 1
print(a)
print(b)
```

f. 4 7

g.
```
a = 6
b = 4
for i in range(3):
    if a > b:
        a -= b
    b += a
print(a)
print(b)
```

g. 2 10

h.
```
for i in range(1,3):
    for j in range(2,4):
        print(str(i*j))
```

h. 2 3 4 6

i.
```
for i in range(3,5):
    for j in range(1,i):
        print(str(i+j))
```

i. 4 5 5 6 7

j.
```
def func(n):
    return 2*n + 1

print(func(func(3)))
```

j. 15

k.
```
nums = [5, 2, 8, -4, 2, 9, -4, -2, 0, 7]
i = 0
while nums[i] < 6:
    print(nums[i])
    i += 1
```

k. 5 2

l.
```
nums = [3, 1, 0, 2]
for i in range(4):
    print(nums[nums[i]])
```

l. 2 1 3 0

m.
```
nums = [5, 2, 9, 7, 4, 3, 6, 0, 1]
print(nums[2:5])
```

m. `[9, 7, 4]`

n.
```
nums = [5, 2, 9, 7, 4, 3, 6, 0, 1]
print(nums[4:])
```

n. `[4, 3, 6, 0, 1]`

o.
```
nums = [5, 2, 9, 7, 4, 3, 6, 0, 1]
print(nums[:3])
```

o. `[5, 2, 9]`

p.
```
nums = [5, 2, 9, 7, 4, 3, 6, 0, 1]
print(nums[2:-2])
```

p. `[9, 7, 4, 3, 6]`

2. What string is stored in the variable `name` after the following code has been executed?

```
name = 'wolfie the seawolf'
name.upper()
```

The string still contains `'wolfie the seawolf'`. Why? Because strings are immutable (unchangeable). If we actually wanted to replace `name` with `'WOLFIE THE SEAWOLF'`, we would need to type this:

```
name = name.upper()
```

Divide and Conquer

3. Suppose 600 people were standing in line at a baseball park and we wanted to program a robot to find our friend, who is standing somewhere in line.

   a. Assume that the people were not sorted in any way and were arranged in random order on the line. If the robot used the *sequential search* algorithm to find the intended person, what would be the *minimum* number of people the robot might have to question before it found the intended person?

   a. 1

   b. Assume that the people were sorted by their last names. If the robot used the *sequential search* algorithm to find the intended person, what would be the *maximum* number of people the robot might have to question before it found the intended person?

   b. 599 or 600

   c. Assume that the people were sorted by their last names. If the robot used the *binary search* algorithm to find the intended person, what would be the *minimum* number of people the robot might have to question before it found the intended person?

c. 1

d. Assume that the people were sorted by their last names. If the robot used the *binary search* algorithm to find the intended person, what would be the *maximum* number of people the robot might have to question before it found the intended person?

d. approx. 10

Programming Problems: Iteration (Non-recursive Functions)

You are strongly encouraged to complete the review programming problems from earlier in the course if you haven't done so yet!

4. Write Python code that would take a list of integers called `ages` and **print** the count of how many of the values are between 18 and 30, inclusive. Do not write a function.

```
count = 0
for a in ages:
    if 18 <= a <= 30:
        count += 1
print(count)
```

5. Write Python code that would take a list of strings called `brands` and **create** a new string `initials` consisting of the capitalized first letter of each string in `brands`. For example, if `brands` were `['Dell','apple','` then the `initials` string would be `'DASP'`. Do not write a function.

```
initials = ''
for b in brands:
    initials += b[0].upper()
```

6. Write no more than ten lines of Python code that would **print** all the different two-digit positive integers that could be formed using the four digits 3, 4, 5, 6. (i.e., 33, 34, 35, ..., 66)

```
for i in range(3,7):
    for j in range(3,7):
        print(i*10+j)
```

7. Sometimes some words like "localization" or "internationalization" are so long that writing them many times in one text is quite tiresome. Let's consider a word *too long* if its length is strictly more than 10 characters. All *too long* words should be replaced with a special abbreviation.

   This abbreviation is made like this: we write down the first two and the last two letters of a word, and between them we write the number of letters between the first two and the last two letters.

   Thus, "localization" will be abbreviated "lo8on", and "internationalization" will be abbreviated "in16on"".

   Write a function `abbreviate(word)` that takes a single word and, if it is 10 characters in length or more, returns its abbreviation according to the above scheme. If the word has fewer than 10 characters, the function simply returns the word unchanged.

```
def abbreviate(word):
    if len(word) < 10:
        return word
    else:
        return word[:2] + str(len(word[2:-2])) + word[-2:]
```

8. Write a Python function `compute(expr)` that takes a string argument containing a single-digit positive integer, followed by an arbitrary number of spaces, followed by a mathematical operator (+, , /, *), followed by an arbitrary number of spaces, followed by a another single-digit positive integer. The function uses string operations to extract the two numbers and operator, performs the desired operation, and returns the result.

   Examples:

   - `compute('3 +8')` would return `11`
   - `compute('9 - 4')` would return `5`
   - `compute('7/ 2')` would return `3.5`
   - `compute('6*2')` would return `12`

```
def compute(expr):
    num1 = int(expr[0])
    i = 1
    while expr[i] == ' ':
        i += 1
    operator = expr[i]
    num2 = int(expr[-1])
    if operator == '+':
        return num1 + num2
    elif operator == '-':
        return num1 - num2
    elif operator == '*':
        return num1 * num2
    else:
        return num1 / num2
```

9. Write a Python function `count_ones(bstring)` that takes a string argument representing a binary number (0s and 1s). First, your function should verify that the string actually represents a binary number. That is, the function should verify that the number contains only 0s and 1s. If that is not that case, your program should return `None` to indicate an error. Otherwise, your function should count how many 1s are in the string and return that count.

```
def count_ones(bstring):
    count = 0
    for ch in bstring:
        if ch not in '01':
            return None
        elif ch == '1':
            count += 1
    return count
```

10. Write a Python function `count_words(text)` that takes a string argument and returns a dictionary that maps each word to its frequency in the argument. In other words, suppose `text` contained the word “happy” 3 times. Then the returned dictionary would contain the key/value pair `"happy":3`. A punctuation mark should be considered as belonging to a particular word.

    A more comprehensive example: for the sentence

    > In the loveliest town of all, where the houses were white and high and the elms trees were green and higher than the houses, where the front yards were wide and pleasant and the back yards were bushy and worth finding out about, where the streets sloped down to the stream and the stream flowed quietly under the bridge, where the lawns ended in orchards and the orchards ended in fields and the fields ended in pastures and the pastures climbed the hill and disappeared over the top toward the wonderful wide sky, in this loveliest of all towns Stuart stopped to get a drink of sarsaparilla.

    from E.B. White’s *Stuart Little*, the function would return this dictionary:

```
{'loveliest': 2, 'sarsaparilla.': 1, 'higher': 1, 'high': 1, 'than': 1,
'town': 1, 'finding': 1, 'streets': 1, 'get': 1, 'stopped': 1, 'the': 17,
'Stuart': 1, 'ended': 3, 'white': 1, 'pleasant': 1, 'disappeared': 1,
'In': 1, 'wonderful': 1, 'where': 4, 'under': 1, 'flowed': 1, 'bushy': 1,
'yards': 2, 'pastures': 2, 'stream': 2, 'orchards': 2, 'of': 3, 'to': 2,
'out': 1, 'drink': 1, 'climbed': 1, 'wide': 2, 'a': 1, 'about,': 1,
'bridge,': 1, 'and': 11, 'all': 1, 'toward': 1, 'this': 1, 'sloped': 1,
'over': 1, 'towns': 1, 'all,': 1, 'sky,': 1, 'hill': 1, 'houses,': 1,
'trees': 1, 'green': 1, 'front': 1, 'houses': 1, 'down': 1, 'lawns': 1,
'were': 4, 'top': 1, 'elms': 1, 'quietly': 1, 'in': 4, 'worth': 1,
'fields': 2, 'back': 1
```

```
def count_words(text):
    result = {}
    words = text.split()
    for w in words:
        result.setdefault(w, 0)
        result[w] += 1
    return result
```

11. Write a function `election(filename)` that opens a file whose name is given as the argument and return the name of the winner. The winner is the person who received the most votes overall in all voting districts. The file contains the names of the two candidates, how many voting districts there are, and how many votes each candidate received in each voting district. For example, suppose the file `votes.txt` contains the following data:

```
Alice Bob
5
4 17
5 18
12 4
2 19
7 9
```

We see that the two candidates for office are Alice and Bob. There are 5 voting districts. Alice received 4 votes in district #1, Bob received 17 votes in district #1, and so on, all the way down to district #5, where Alice received 7 votes and Bob received 9 votes. If there is a tie, the function returns the word `'Tie'`. For the above example, the function would return `'Bob'` because Bob received 67 votes to Alice's 30.

```
def election(filename):
    votes_file = open(filename)
    names = votes_file.readline().split()
    votes1 = 0
    votes2 = 0
    num_districts = int(votes_file.readline())
    for i in range(num_districts):
        votes = votes_file.readline().split()
        votes1 += int(votes[0])
        votes2 += int(votes[1])
    if votes1 > votes2:
        return names[0]
    elif votes2 > votes1:
        return names[1]
    else:
        return 'Tie'
    votes_file.close()
```

Programming Problems: Recursive Functions

The questions below come in pairs: first you are asked to solve a problem with a non-recursive function (i.e., using loops only), and then to solve the same problem using recursion only (i.e., no loops). Sometimes thinking about a non-recursive solution to a problem can help you find a recursive solution.

12. Write a *non-recursive, iterative* function called `is_binary(n)` that returns `True` if the string argument is a valid binary number, or `False` if it is not. For example, if `n` is `"11001"`, the function will return `True`. As another example, if `n` is `"1100121101"`, the function will return `False`.

```
def is_binary(n):
    for c in n:
        if c not in '01':
            return False
    return True
```

13. Write a *recursive* function called `ris_binary(n)` that returns `True` if the string argument is a valid binary number, or `False` if it is not. For example, if `n` is `"1001"`, the function will return `True`. As another example, if `n` is `"1100121101"`, the function will return `False`. You may not use loops at all, neither in `ris_binary()` nor in any helper function(s) you might write.

```
def ris_binary(n):
    if len(n) == 0:
        return True
    elif n[0] == '0' or n[0] =='1':
        return ris_binary(n[1:])
    else:
        return False
```

14. Write a *non-recursive, iterative* function called `trib(n)` that computes the n-th "Tribonacci" number, T_n, where $T_1 = 1, T_2 = 1, T_3 = 2$ and for $n > 3$, $T_n = T_{n-1} + T_{n-2} + T_{n-3}$. If $n \leq 0$, the `trib()` function should return -1. The first several values of the Tribonacci sequence are $1, 1, 2, 4, 7, 13, 24, 44, 81, 149, \ldots$.

```
def trib(n):
    if n == 1 or n == 2:
        return 1
    elif n == 3:
        return 2
    a, b, c = 1, 1, 2  # this is called a "tuple assignment"
    # Can also be written as three separate assignment statements.
    for i in range(n-3):
        c, b, a = c+b+a, c, b
    return c
```

Below is an alternate solution that illustrates a little more clearly what is happening inside the for-loop:

```
def trib(n):
    if n == 1 or n == 2:
        return 1
    elif n == 3:
        return 2
    a, b, c = 1, 1, 2
    for i in range(n-3):
        t = a+b+c
        a = b
        b = c
        c = t
    return c
```

15. Write a *recursive* function called `rtrib(n)` that computes the n-th Tribonacci number using recursion instead of loops. You may not use loops at all, neither in `rtrib(n)` nor in any helper function(s) you might write.

    Hint: see the lecture notes for inspiration!

```
def rtrib(n):
    if n == 1 or n == 2:
        return 1
    elif n == 3:
        return 2
    return rtrib(n-1) + rtrib(n-2) + rtrib(n-3)
```

16. Write a *non-recursive, iterative* function called `rsum_digits(n)` that computes and returns the sum of the digits of the integer argument `n`. For example, is `n` is 1920436, the function will return 25 because $1+9+2+0+4+3+6=25$.

    Hint: use `%` and `//`.

```
def sum_digits(n):
    t = 0
    while n > 0:
        t += n % 10
        n //= 10
    return t
```

17. Write a *recursive* function called `rsum_digits(n)` that computes and returns the sum of the digits of the integer argument `n`. For example, is `n` is 1920436, the function will return 25 because $1+9+2+0+4+3+6=25$. You may not use loops at all, neither in `rsum_digits()` nor in any helper function(s) you might write.

    Hint: use `%` and `//`.

```
def rsum_digits(n):
    if n == 0:
        return 0
    else:
        return n%10 + rsum_digits(n//10)
```

18. Write a *non-recursive, iterative* function called `reverse(n)` that computes and returns the "reverse" of an integer. The reverse of an integer contains the digits of the integer in reverse order. The reversal is returned as a *string*. For example, if `n` is 12345, the function will return `"54321"`. As another example, if `n` is 81920, the function will return `"02918"`.

    Hint: use `%` and `//`.

```
def reverse(n):
    result = ''
    result += str(n % 10)   # this special case is needed for n=0
    n //= 10
    while n > 0:
        result += str(n % 10)
        n //= 10
    return result
```

19. Write a *recursive* function called `rreverse(n)` that computes and returns the "reverse" of an integer. The reverse of an integer contains the digits of the integer in reverse order. The reversal is returned as a *string*. For example, if `n` is 12345, the function will return `"54321"`. As another example, if `n` is 81920, the function will return `"02918"`. You may not use loops at all, neither in `rreverse()` nor in any helper function(s) you might write.

```
def rreverse(n):
    if 0 <= n <= 9:
        return str(n)
    else:
        return str(n%10) + reverse(n//10)
```

20. Write a *non-recursive, iterative* function called `remove_consec_duplicates(s)` that removes every second consecutive instance of a repeated character, treating uppercase and lowercase letters as different characters. For example, if `s` is `"haaappppy"`, the function will return `"haappy"`.

    As another example, if `s` is `"aaAabbaBBbbbCCCcccdDDd"`, the function will return `"aAabaBbbCCccdDd"`.

    If `s` is `"aaaabbabbbbbccccccdddd"`, the function will return `"aababbbcccdd"`.

```
def remove_consec_duplicates(s):
    result = ''
    while len(s) > 1:
        result += s[0]
        if s[0] == s[1]:
            s = s[2:]
        else:
            s = s[1:]

    result += s
    return result
```

21. Write a *recursive* function called `rremove_consec_duplicates(s)` that removes every second consecutive instance of a repeated character, treating uppercase and lowercase letters as different characters. For example, if `s` is `"haaappppy"`, the function will return `"haappy"`.

    As another example, if `s` is `"aaAabbaBBbbbCCCcccdDDd"`, the function will return `"aAabaBbbCCccdDd"`.

    If `s` is `"aaaabbabbbbbccccccdddd"`, the function will return `"aababbbcccdd"`.

```
def rremove_consec_duplicates(s):
    if len(s) <= 1:
        return s
    elif s[0] == s[1]:
        return s[0] + rremove_consec_duplicates(s[2:])
    else:
        return s[0] + rremove_consec_duplicates(s[1:])
```