National University of Singapore
School of Computing
CS1010S: Programming Methodology

Extra Practice 2 Solutions

Question 1

Trace the following code. **(4 marks)**

```
result = 0
for i in range(5):       # this means i will be set to 0, 1, 2, 3, 4
    result += 1
print(result)   # 5

result2 = 0
for i in range(1, 5):   # this means i will be set to 1, 2, 3, 4
    result2 += 1
print(result2)  # 4
```

Question 2

Trace the following code. **(5 marks)**

```
result = 0
for i in range(1, 13, 3):   # i =    1   4   7   10
    if i % 2 == 0:
        i += 2              # i =        6       12
    else:
        result //= i    # result =  0       1
    result += i         # result =  1   7   8   20

print(result)   # 20
```

Question 3

Trace the following code. **(5 marks)**

```
a, b, c = "east", "easter", "easy"
a, b, c = c, a, b    # a = "easy", b = "east", c = "easter"

if a < b:            # No, "easy" > "east"
    a, b = b, a
else:                # Yes
    if b < c:        # Yes, "east" < "easter"
        a += b       # a is now "easyeast"
b = a + c            # b is now "easyeast" + "easter" = "easyeasteaster"
                     # c remains "easter"
```

```
print(a[:-1])    # easyeas
print(b[1:])     # asyeasteaster
print(c[::-1])   # retsae
```

Question 4

To play a game of bowling, we will store our results from each throw in an integer such as 1459. In this game, we will only play with 9 pins. 1459 means 1 pin is struck in the first shot, 4 pins in the second shot, 5 pins in the third shot and a strike in the last shot.

(a) Define a function `score` that takes in an integer and returns the total score of the game (1 pin = 1 point). Use an iterative approach. **(4 marks)**
Sample Tests:

```
>>> score(1459)
19
>>> score(999)
27
```

Solution:

```
# Using while loop
def score(pins):
    result = 0
    while pins > 0: # or while pins
        result += pins % 10
        pins //= 10
    return result

# Using for loop
def score(pins):
    result = 0
    for i in str(pins):
        result += int(i)
    return result
```

(b) What is the order of growth (in space and time) of your solution in part (a)? Explain your answer. **(2 marks)**
Solution:
O(n) time and O(1) space where n is the number of pins for both methods. This is because we iterate through each of the pin once and do a constant number of operations within, overall by using only two or three variables.

(c) Define a function `score_recursive` that does the same thing as in part a but in a **recursive** manner. **(4 marks)**
Solution:

```
def score(pins):
    if pins < 10:
        return pins
    return pins % 10 + score(pins // 10)
```

(d) What is the order of growth (in space and time) of your solution in part (c)? Explain your answer. **(2 marks)**
Solution:
O(n) time and O(n) space where n is the number of pins. There are approximately n deferred operations, making the space complexity to be O(n).

(e) Define a function **strike_count** and **strike_count_recursive** that takes in an integer and returns the total number of strikes in the game. Use iteration and recursion respectively. **(8 marks)**
Sample Tests:

```
>>> strike_count(919)
2
>>> strike_count(1234560)
0
>>> strike_count(9999)
4
```

Solution:

```
# Iteration using while loop
def strike_count(pins):
    result = 0
    while pins > 0: # or while pins
        result += int(pins % 10 == 9)
        pins //= 10
    return result

# Iteration using for loop
def strike_count(pins):
    result = 0
    for i in str(pins):
        result += int(i == "9")
    return result

# Recursion
def strike_count(pins):
    if pins < 10:
        return int(pins == 9)
    return int(pins % 10 == 9) + strike_count(pins // 10)

# Note:
# int(statement) is equivalent to 1 if statement is true and 0 otherwise.
# You may use this to save some lines.
```

(f) Now, each strike is going to be worth an extra 5 points each! Using your previously defined functions, define a new function **score_improved** that takes in an integer and returns the total score. **(4 marks)**
Sample Tests:

```
>>> score_improved(919)
29
>>> score_improved(1234)
```

```
10
>>> score_improved(12349)
24
```

Solution:

```
def score_improved(pins):
    return score(pins) + 5 * strike_count(pins)
```

Question 5

(a) We will define another maskify function to encrypt our password. Given a password of any length, we want to mask all the characters with "*". Define a function **`maskify`** that takes in a password as a string and returns the new masked word. Use **iteration**. **(4 marks)**
Sample Tests:

```
>>> maskify("password")
'********'
>>> maskify("121")
'***'
```

Solution:

```
# Closed form is NOT allowed!
def maskify(word):
    return "*"*len(word)

# Use iteration instead
def maskify(word):
    answer = ""
    for i in word:
        answer += "*"
    return answer
```

(b) State the time and space complexity of your solution. **(2 marks)**
Solution:
*O(n**2) time complexity due to string concatenation and O(n) space complexity where n is the length of the word.*

(c) Do part (a) with recursion. **(4 marks)**
Solution:

```
def maskify(word):
    if word == "":
        return ""
    return "*" + maskify(word[1:])
    # or return maskify(word[:-1]) + "*"
```

(d) State the time and space complexity of your solution. **(2 marks)**
Solution:
*O(n**2) time complexity due to string concatenation and O(n**2) space complexity where n is the length of the word.*

We can visualize the recursion tree with n levels. Note that on each level it will take up O(n) space due to string concatenation.

(e) Now, we want to put an "*" sign in between all of the letters. Define a function **slot** that does this recursively. **(6 marks)**
Sample Tests:

```
>>> slot("pass")
'p*a*s*s'
>>> slot("123")
'1*2*3'
```

Solution:

```
def slot(word):
    if len(word) <= 1: # not == 1 since word can be an empty string
        return word
    return word[0] + "*" + slot(word[1:])
```

(f) We want to insert the "*" sign now into consecutive letters that are identical to each other only. Define a function **advanced_slot** that can do this recursively. **(6 marks)**
Sample Tests:

```
>>> advanced_slot("pass")
'pas*s'
>>> advanced_slot("aaaaba")
'a*a*a*aba'
```

Solution:

```
def advanced_slot(word):
    if len(word) <= 1:
        return word
    if word[0] == word[1]:
        return word[0] + "*" + advanced_slot(word[1:])
    # else this is executed
    return word[0] + advanced_slot(word[1:])
```

Question 6

Trace the following code. **(4 marks)**

```
def weird_sum(n):
    if n == 0:
        return 0
    else:
        return n + weird_sum(n - 2)

print(weird_sum(5))
# RecursionError will occur because the base case n == 0 is never met.
```

Question 7

Trace the following code. **(4 marks)**

```
for i in range(5):
    print(i)
    i += i
# Although we are modifying i on the third line, note that i is always
# preset back to the original sequence 0, 1, 2, 3, 4.
# This means the numbers 0, 1, 2, 3, 4 will be printed but on the end
# of the for loop, we will get i = 4 + 4 = 8.
```

Question 8

You might have known about Fibonacci numbers before. But, have you known about Lucas numbers?

According to Wikipedia, we define Lucas numbers as follows.

$$L_n = \begin{cases} 2 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ L_{n-1} + L_{n-2} & \text{if } n > 1 \end{cases}$$

(a) Define a **recursive** function **lucas** that takes in a *nonnegative integer* n and returns L_n. **(2 marks)**
Sample Tests:

```
>>> lucas(1)
1
>>> lucas(10)
123
```

Solution:

```
def lucas(n):
    if n == 0:
        return 2
    elif n == 1:
        return 1
    return lucas(n-1) + lucas(n-2)
```

(b) Do part (a) **iteratively**. **(3 marks)**
Solution:

```
def lucas(n):
    a, b = 2, 1
    for _ in range(n):
        a, b = b, a + b
    return a
```

(c) **(Optional)** Define a function **lucas2** that takes in a *positive integer* n and returns

$$\frac{L_{n-1} + L_{n+1}}{5}$$

Try to call this function for $n = 1, 2, \ldots, 10$. Does **lucas2** seem familiar to you?
Solution:

```
def lucas2(n):
    # Using float division is okay but lucas2 is actually
    # a sequence of integers!
    return (lucas(n - 1) + lucas(n + 1)) // 5

print(lucas2(1))    # 1
print(lucas2(2))    # 1
print(lucas2(3))    # 2
print(lucas2(4))    # 3
print(lucas2(5))    # 5
print(lucas2(6))    # 8
print(lucas2(7))    # 13
print(lucas2(8))    # 21
print(lucas2(9))    # 34
print(lucas2(10))   # 55
# lucas2 is actually the Fibonacci sequence!
```

For more information, you may read at this Wikipedia article.

Solution compiled by Russell Saerang.