Final Exam

CS97: Principles and Practices of Computing

Thursday, December 14, 2017

1.	2.
3.	4.

Name:____

ID:____

Rules of the game:

- **Write your name and ID number above.**
- The exam is closed-book and closed-notes.
- Please write your answers directly on the exam. Do not turn in anything else.
- The exam ends promptly at 11am.
- Read questions carefully. Understand a question before you start writing.
- Relax!

1. (2 points each) Consider this function:

```
def f(x):
    if x > 10:
        y = x
    elif x > 0:
        y = 2 * x
    else:
        y = 0
    return y
```

   (a) Provide a positive integer n such that f(n) returns 10, or say NONE if no such positive integer exists.

   5

   (b) Provide a positive integer n such that f(n) returns 11, or say NONE if no such positive integer exists.

   11

2. (2 points each) Consider this function named g, which identical to the function f above but with the keyword elif replaced by the keyword if:

```
def g(x):
    if x > 10:
        y = x
    if x > 0:
        y = 2 * x
    else:
        y = 0
    return y
```

   (a) Provide a positive integer n such that g(n) returns a different number than f(n), or say NONE if no such positive integer exists.

   11 (or any integer greater than 10)

   (b) Provide a positive integer n such that g(n) returns the same number as f(n), or say NONE if no such positive integer exists.

   1 (or any positive integer less than or equal to 10)

3. (5 points each) In this problem you will implement several versions of a function `sumToK(k, l)`, which takes an integer `k` and a list of integer lists `l` and returns a count of the number of integer lists in `l` that sum to exactly `k`. For example,

```
sumToK(10, [[1,2], [1,2,3,4], [5,5,0], [8,2,3], [-15, 25]])
```

returns `3`, since 3 of the 5 inner lists in the given list sum to exactly 10.

Note: In all versions below, you may use Python's `sum` *function, which sums the elements of an integer list, to sum each inner list. You may also use other built-in Python functions (e.g.,* `max`*,* `min`*,* `len`*, etc.).* **However, carefully read the requirements for each implementation in the questions below to ensure you adhere to them as well.**

(a) Implement `sumToK` as a single recursive function.

```
def sumToK(k, l):
    # your code goes here
```

```
def sumToK(k, l):
    if l == []:
        return 0
    elif sum(l[0]) == k:
        return 1 + sumToK(k, l[1:])
    else:
        return sumToK(k, l[1:])
```

(b) Now implement `sumToK` as a single function that uses a `for` loop instead of recursion.

```
def sumToK(k, l):
    # your code goes here
```

```
def sumToK(k, l):
    count = 0
    for innerL in l:
        if sum(innerL) == k:
            count += 1
    return count
```

(c) Now implement `sumToK` as a single function that uses a `while` loop instead of a `for` loop.

```
def sumToK(k, l):
    # your code goes here
```

```
def sumToK(k, l):
    count = 0
    i = 0
    while i < len(l):
        if sum(l[i]) == k:
            count += 1
        i += 1
    return count
```

(d) Finally, implement `sumToK` without using either loops or recursion. Instead make use of one or more of `map`, `filter`, and `reduce`. *Note that you may still use the* `sum` *function and other built-in Python functions.*

```
def sumToK(k, l):
    # your code goes here

def sumToK_F(k, l):
    return len(list(filter(lambda innerL: sum(innerL) == k, l)))
```

4. (2 points each) Consider the following code:

```
>>> myPair = [1, 2]
>>> swap(myPair)
>>> myPair
```

For each definition of swap below, what will the Python interpreter print after executing the last line of code above? Circle the right answer among the five choices.

(a)
```
def swap(p):
    p[0] = p[1]
    p[1] = p[0]
    return
```

i. `[1, 1]`
ii. `[1, 2]`
iii. `[2, 1]`
iv. `[2, 2]`
v. None of the above

(b)
```
def swap(p):
    p = [p[1], p[0]]
    return
```

i. `[1, 1]`
ii. `[1, 2]`
iii. `[2, 1]`
iv. `[2, 2]`
v. None of the above

(c)
```
def swap(p):
    first = p[0]
    second = p[1]
    first = second
    second = p[0]
    return
```

i. `[1, 1]`
ii. `[1, 2]`
iii. `[2, 1]`
iv. `[2, 2]`
v. None of the above