SCHOOL OF COMPUTING

ASSESSMENT FOR
Special Term I, 2016/2017

Solutions for CS1010X — PROGRAMMING METHODOLOGY

June 2017 Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **(??) questions** and comprises **(??) pages**.
3. Weightage of questions is given in square brackets. The maximum attainable score is 100.
4. This is a **CLOSED** book assessment, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
5. Write all your answers in the space provided in this booklet.
6. **Please write your student number below.**

STUDENT NO: ____________________

(this portion is for the examiner's use only)

Question	Marks	Remark
Total		

Question 1: Python Expressions [25 marks]

There are several parts to this problem. Answer each part **independently and separately**. In each part, one or more Python expressions are entered into the interpreter (Python shell). Determine the response printed by the interpreter for the final expression entered. If the interpreter produces an error message, or enters an infinite loop, explain why.

A. [5 marks]

```
a = [4,2,7]
b = [2,4,1]
while a and b:
    print(a,b)
    if a[0] > b[0]:
        b.append(a.pop(0))
    elif b[0] > a[0]:
        a.append(b.pop(0))
print(a)
print(b)
```

This code will get stuck in an infinite loop with the following configuration:

```
a = [2, 7]
b = [2, 4, 1, 4]
```

This is to test that the student is able to trace through a `while` loop and understand the list operations `append` and `pop`.

B. [5 marks]

```
a = [1,2,3]
b = [a,3,4]
a[2] = b
b[0][0] = b
a[0][1] = 99
a[2][0] = 5
print(a)
```

```
[[5, 99, 4], 2, [5, 99, 4]]
```

This is to test that the student is able to trace through list references correctly after mutation.

C. [5 marks]

```
try:
    y = 12945
    while y > 10:
        x = [1]
        x.extend(str(y))
        y = sum(list(map(lambda x: int(x),x)))
except:
    print("Got error!")
finally:
    print(x,y)
```

```
[1, ’2’, ’2’] 5
```

This is to test the the student can trace through a `while` loop correctly. The `try` clause is just a red herring.

-2 for not getting the `’2’` right, i.e. some students didn’t realize that they are strings.

D. [5 marks]

```
a = {3:2,11:4}
def foo(d):
    d[7] = 11
    d[2] = 5
    del d[3]
    return tuple(d.items())
print(foo(a))
```

```
((11, 4), (2, 5), (7, 11))
```

The correct answer is that the student cannot be sure what the answer is because the ordering for `items()` is not defined.

-2 points for getting one possible answer right, but not stating that the answer is not well-defined. Random fact: only one student got this question completely correct.

E. [5 marks]

```
a = {(1,2):3, (3,4):5}
for k,v in a.items():
    a[[v,k[0]]] = k[1]
b = list(a.values())
b.sort(reverse=True)
print(b)
```

```
TypeError:  unhashable type:  'list'
```

This is to ensure that students understand that dictionary keys should be immutable and hashable types.

Question 2: Fun with Filters [27 marks]

When we use the Python `filter` function, we need to apply a predicate function. In this problem, we will explore filters.

A. [**Warm Up**] Implement the function `is_fibonacci` that will return `True` if an input integer *n* is a Fibonacci number, or `False` otherwise.

```
>>> a = [1,2,4,1,11,6,7,8,23,5,8,13]
>>> list(filter(is_fibonacci,a))
[1, 2, 1, 8, 5, 8, 13]
```

[4 marks]

```
def is_fibonacci(n):
    a,b = 0,1
    while a<=n:
        if a==n:
            return True
        a,b = b,a+b
    return False
```

Quite a large number of students provided a very inefficient but correct solution. They got full credit.

B. Suppose we use `is_fibonacci` to filter a list of *n* elements, each $\leq m$, what is the order of growth in time and space for resulting operation? Explain. [4 marks]

Time: $O(n \log m)$, since `is_fibonacci` is $\log m$ for each element and there are *n* elements.

Space: $O(n)$ since `is_fibonacci` is iterative and in the worst case, the list operation will return a list containing all the *n* elements in the input list.

A small number of students confused *m* and *n*.

C. [**Memoization**] Let's try to make the `is_fibonacci` filter faster using memoization.

```
>>> a = [1,2,4,1,11,6,7,8,23,5,8,13]
>>> list(filter(make_memo_fib(),a))
[1, 2, 1, 8, 5, 8, 13]
```

While the result is the same as that in Part(A), give a possible implementation for the function `make_memo_fib`, so that the resulting filtering operation for Part(B) runs faster for a large number of elements. [6 marks]

```
def make_memo_fib():
    seen = {}
    largest = [-1]
    def helper(n):
        if n in seen:
            return seen[n]
        elif largest[0] >= n:
            return False
        a,b = 0,1
        while a<=n:
            seen[a] = True
            if a==n:
                return True
            a,b = b,a+b
        largest[0] = n
        return False
    return helper
```

This memoization code is not completely straightforward to write. Requires some amount of intuition for the problem and cleverness. If seen is a list instead of a dictionary, then it will be no faster than Part(B). Basically, we need to make `is_fibonacci` run in $O(1)$ time after the first time.

The students who wrote terrible code in Part(A) had it easy here because it would be quite straightforward to memoize the code to improve performance.

-3 for students who put the answers in a list. This doesn't work because we need a $O(1)$ hash table for efficient lookup. Typically we use a dictionary.

Zero marks for the small number of students who do do really bad things by creating a list and iterating through it in a very perverse manner and also creating a dictionary and then iterating through `values()` instead of checking for the key.

D. [**Stateful Filters**] Most times we work with stateless filters, i.e. the result is independent of history. However, it is possible to have stateful filters, i.e. we can have a filter that does not allow consecutive elements to be the same. For example,

```
>>> a = [1,1,2,1,11,1,2,2,2,6,7,7,8,23,8]
>>> list(filter(make_no_consecutive_filter(),a))
[1, 2, 1, 11, 1, 2, 6, 7, 8, 23, 8]
```

Give a possible implementation for the function `make_no_consecutive_filter`. [5 marks]

```
def make_no_consecutive_filter():
    last = []
    def helper(x):
        if not last:
            last.append(x)
            return True
        elif x==last[0]:
            return False
        else:
            last[0] = x
            return True
    return helper
```

-2 for scoping error if one tries to assign a variable inside `helper` without doing a `nonlocal`.

E. [**Combining Filters**] Suppose we want to be able to combine 2 filter functions into one function as follows:

```
>>> a = [1,1,2,4,1,11,1,2,2,6,7,7,8,23,5,8,13]
>>> f = combine(lambda x: x%2==0, lambda x: x<6)
>>> list(filter(f,a))
[2, 4, 2, 2]
```

Give a possible implementation for the function `combine`. [4 marks]

```
def combine(f1,f2):
    def helper(n):
        return f1(n) and f2(n)
    return helper
```

F. Does the order of `f1` and `f2` in `combine` matter? Explain. [4 marks]

It matters if one of the filters is stateful, i.e.

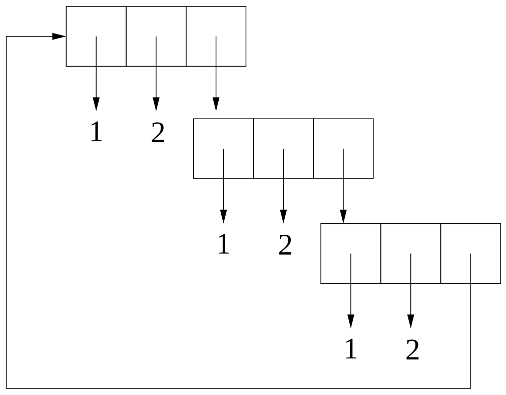
```
def make_odd_pos_filter():
    last = [False]
    def helper(x):
        last[0] = not last[0]
        return last[0]
    return helper
```

Then:

```
>>> f1 = combine(lambda x: x%2==0, make_odd_pos_filter())
>>> f2 = combine(make_odd_pos_filter(),lambda x: x%2==0)
>>> a = [2,1,4,1,]
>>> list(filter(f1,a))
[2]
>>> list(filter(f2,a))
[2, 4]
```

Question 3: Unchained Melody [Challenging] [24 marks]

In this problem, we will work with recursive lists of a special form where the last element points to the next list and the last list in the series points back to the first list in the series. Each list is also called a *link* (of the chain). This is illustrated here:



A. The function `chain(seq,n)` creates a chain of *n* links by replicating a sequence `seq` as follows:

```
>>> chain((1,2),1)
[1, 2, [...]]

>>> chain((1,2),3)  # Illustrated above
[1, 2, [1, 2, [1, 2, [...]]]]

>>> chain((4,),4)
[4, [4, [4, [4, [...]]]]]
```

Give a possible implementation for `chain(seq,n)`. [5 marks]

```
def chain(seq,n):
    ans = list(seq)
    ans.append(None) # Placeholder
    link = list(ans)
    current = ans
    for i in range(n-1):
        new_link = list(link)
        current[-1] = new_link
        current = new_link
    current[-1] = ans
    return ans
```

-2 marks for not getting the final back link to the top of the chain correct.

B. Given a chain it would certainly be helpful to be able to count the number of links in the chain. For example:

```
>>> count_links(chain((1,2),1))
1

>>> count_links(chain([3,2],2))
2

>>> count_links(chain((4,),4))
4
```

Give a possible implementation for `count_links`. [5 marks]

```
def count_links(chain):
    count = 1
    current = chain[-1]
    while not current is chain:
        count += 1
        current = current[-1]
    return count
```

This tests that the students know how to use `is` to test for the cycle.

C. Ben Bitdiddle used a `deep_copy` function to create deep copies of some lists of lists and it worked perfectly (remember our recitation?). However, when he tried to use a list `deep_copy` function to create a deep copy of a chain, it failed. Provide a plausible explanation for this. [4 marks]

The chain loops back to the start, so a naive implementation of `deep_copy` that doesn't deal with the cycles correctly will fail. Basically, there is a need to handle cycles or there will be an infinite loop.

D. Give a possible implementation for a function `copy_chain` that will return a deep copy of a chain.

```
>>> a = chain((1,2),4)
>>> a[2][2][0] = 5
>>> b = copy_chain(a)
>>> b
[1, 2, [1, 2, [5, 2, [1, 2, [...]]]]]
```

[5 marks]

```
def copy_chain(chain):
    depth = count_links(chain)
    ans = list(chain)
    last = ans
    current = chain[-1]
    for i in range(depth-1):
        last[-1] = list(current)
        current = current[-1]
        last = last[-1]
    last[-1] = ans
    return ans
```

-3 for just finding the length and returning `chain(seq,n)`. This assumes that all the links are the same, but the sample execution already shows that we need to copy each link.

Zero for those who just copied the `deep_copy` code here. We just said in Part(C) that it doesn't work didn't we?

E. A regular chain has repeated links. It is possible for a fault to happen and one of the links might become "faulty." A faulty link has one element that is different from all the other links. Implement the function `find_fault` that will return the different element. You can assume that the chain has at least 3 links and there's only one faulty element.

```
>>> a = chain((1,2),4)
>>> a[2][2][0] = 5
>>> a
[1, 2, [1, 2, [5, 2, [1, 2, [...]]]]]
>>> find_fault(a)
5
```

[5 marks]

```
def find_fault(chain):
    depth = count_links(chain)
    found = {}
    for i in range(depth):
        if tuple(chain[:-1]) not in found:
            found[tuple(chain[:-1])] = 0
        found[tuple(chain[:-1])] += 1
        chain = chain[-1]
    keys = list(found.items())
    keys.sort(key=lambda x: x[1], reverse=True)
    keys = list(map(lambda x:x[0],keys))
    for i in range(len(keys)):
        if keys[0][i] != keys[1][i]:
            return keys[1][i]
```

Some student just did a brute force approach to find 3 links that contained one different link. -2 marks for assuming that the first link is not faulty.

Question 4: Wrapping Your Head Around C [21 marks]

A. Consider the following C program:

```
#include <stdio.h>
void print_array(int a[], int n){
    int i;
    for (i=0; i<n-1; i++){
        printf("%d, ",a[i]);
    }
    printf("%d\n",a[n-1]);
}

int main(){
    int a[] = {4,12,-1,5,7,2,-5,11,8};
    reverse_sort(a,9);
    print_array(a,9);
    int b[] = {-4,2,-1,5,17,2,-3,11,8};
    reverse_sort(b,9);
    print_array(b,9);
    return 0;
}
```

The output of this program is:

```
12, 11, 8, 7, 5, 4, 2, -1, -5
17, 11, 8, 5, 2, 2, -1, -3, -4
```

Implement the function `reverse_sort` that achieves the desired output. [5 marks]

```
void reverse_sort(int a[], int n){
    int i,j;
    for (i=0; i<n-1; i++){
        for (j=i; j<n-1; j++){
            if (a[j]<a[j+1]) {
            int tmp = a[j];
            a[j] = a[j+1];
            a[j+1] = tmp;
            }
        }
    }
}
```

Basically, a bubble sort will work. Students need to watch their index and also use a temp variable to do the swap.

B. What is the order of growth in time and space for the sort you implemented in Part(A) in terms of the number of elements *n* for the **average** case? Explain. [4 marks]

Time: $O(n^2)$ (average case), since this is just bubble sort.

Space: $O(1)$ since this is an in-place sort!

C. Explain in simple terms what is meant by a *stable* sort. Is the function you implemented in Part(A) a stable sort? Explain. [3 marks]

A stable sort is a sort where the relative positions of elements with the same value is preserved after the sort. The implemented bubble sort is stable since swaps only take place when adjacent elements are different.

1 mark for explaining stable sort. 2 marks for accurately stating whether the code in Part(A) is a stable sort or not.

D. Consider the following C program that prints our a range of numbers in arithmetic progression:

```
#include <stdio.h>
void print_AP(int start, int stop, int step){
    int i;
    for (i=T1; i<T2; i++){
        printf("%d, ",i);
        i += T3;
    }
    printf("%d\n",i);
}
```

```
int main(){
    print_AP(1,5,1);
    print_AP(1,9,2);
    print_AP(2,10,3);
    print_AP(3,16,4);
    return 0;
}
```

The output of this program is:

```
1, 2, 3, 4
1, 3, 5, 7
2, 5, 8
3, 7, 11, 15
```

Please provide possible implementations for T1, T2, and T3. [6 marks]

T1: [2 marks]	`start`
T2: [2 marks]	`stop-step`
T3: [2 marks]	`step-1`

E. Ben Bitdiddle comes along and tells you that he doesn't like C, so he decides to implement a Python-equivalent of the C program in Part(D) as follows:

```
def print_AP(start,stop,step):
    for i in range(T1,T2):
        print(i);
        i += T3;
    print(i);
```

Do you agree with his implementation? Explain. [3 marks]

This translation from C to Python is wrong because C for loops work differently from Python for loops. `i` is reset to the next value in the sequence in each loop so `i += T3` has no effect.

Question 5: What *really* have you learnt in CS1010X? [3 marks]

What in your view are the 3 **most important** lessons that you learnt in CS1010X? Explain. Alternatively, you can draw us a picture, but it better be good!

The student will be awarded points as long as he/she is coherent and doesn't say something obviously wrong. Exactly how many points depends on the effort and thoughtfulness put into writing this mini-essay. Amuse the prof and you get full credit.

— E N D O F P A P E R —

Scratch Paper

– H A P P Y   H O L I D A Y S ! –