SCHOOL OF COMPUTING

ASSESSMENT FOR
Special Term I, 2016/2017

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 **FIVE (5) questions** and comprises **SEVENTEEN (17) 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
Q1		
Q2		
Q3		
Q4		
Q5		
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)
```

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)
```

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)
```

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))
```

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)
```

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):
```

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:

Space:

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():
```

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():
```

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]
```

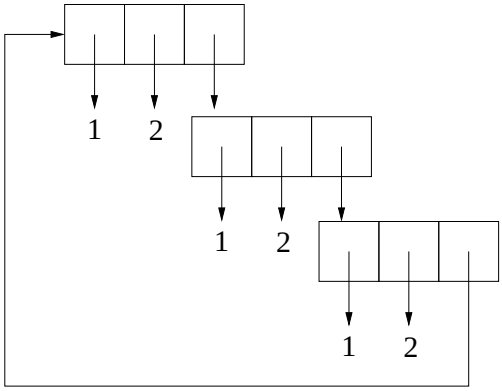
Give a possible implementation for the function `combine`. [4 marks]

```
def combine(f1,f2):
```

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

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):
```

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):
```

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]

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):
```

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):
```

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]

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:

Space:

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]

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]

T2: [2 marks]

T3: [2 marks]

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]

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!

— 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 ! –