SCHOOL OF COMPUTING

ASSESSMENT FOR
Special Term I, 2018/2019

Solutions for CS1010X — PROGRAMMING METHODOLOGY

June 2019 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 **TWENTY (20) 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 assessment.
5. Write all your answers in the space provided in this booklet.
6. You should make effort to write your answer legibly and neatly to avoid any possible confusion in interpretation. Use pencil where applicable (to plan and present your Python code) to ease any amendment along the way.
7. **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 [24 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. All codes have been tested with no error to the Python interpreter.

A. [4 marks]

```
a = list(range(11))
b = a.copy()
c = a.copy()
b[6] = 77
print(list(map(lambda x: x[0]+x[3], (b[0::2], c[2::2]))))
```

```
[77, 10]
```

This is to test that student understands slicing and `map`.

B. [4 marks]

```
a = [4,3,2,1]
b = [1,4,3,2]
c = [2,1,4,3]
d = [3,2,1,4]
a[0], a[1], a[2] = a, c, b
b[0], b[1], b[2] = a, b, c
c[0], c[1], c[2] = c, b, d
d[0], d[1], d[2] = c, d, a
print(a[0][1][0][2][1][2][2][2][3])
```

```
3
```

This is to test that the student is able to trace through self-referencing list references.

C. [4 marks]

```
def testing(x, y):
    try:
        try:
            if y == 0:
                raise ZeroDivisionError("attempt to divide by zero")
            else:
                z = x / y
        except ZeroDivisionError as e:
            print(e)
        except:
            print("unexpected error")
        else:
            return z + x / yy
    except ValueError as e:
        print("ValueError", e)
    except NameError as e:
        print("Name Error:", e)
    except:
        print("unexpected error")
    finally:
        print("that's all")

ten = 10
a = 1234
testing(a, ten)
```

```
Name Error:  name 'yy' is not defined
that's all
```

This is to test that the student understands exception handling syntax.

D. [4 marks]

```
x = lambda x: x+2
y = lambda x: x*3
z = lambda z: lambda y: lambda x: z(y(x))
print(z(x)(y)(4))
```

```
14
```

This is the last chance to allow the student to demonstrate mastery over the lambdas given that many still didn't get it after the midterms and re-midterms.

E. [4 marks]

```
a = {}
for i in range(5,0,-1):
    a[2**i] = str(i)*i
print(list(filter(lambda x: x[1]>4,map(lambda x: (x[1],x[0]),a.items()))))
```

```
[('55555', 32), ('4444', 16), ('333', 8)]
```

In the past the order for dictionaries was not defined (i.e. dictionary does not guarantee any order when the elements are returned). However, as of 3.7, dictionaries return by insertion order.

F. [4 marks]

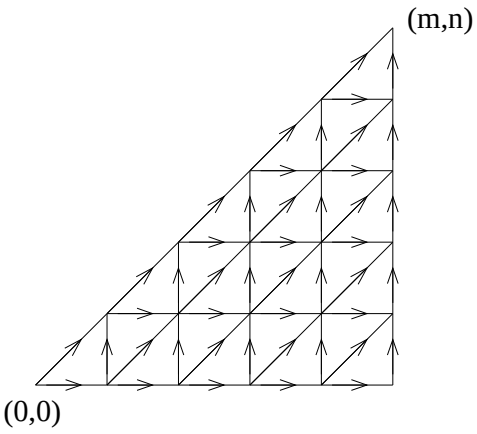
```
x = [1,2]
y = [3,4]
def bar(x,y):
    x.extend([5])
    y = y + 2*x
bar(y,x)
print(x,y)
```

```
[1, 2] [3, 4, 5]
```

This question checks that the student has an handle on scoping and aliasing.

Question 2: Lost in Triangle Land [26 marks]

Most cities are organized in what's called a Manhattan grid, aka squares. You have just taken up a job as the assistant to the city planner for a new city that is shaped like a triangle with sides m and n, $m \geq n$. The roads (which strangely enough are all one-way streets) are set up as follows:



A. [Warm Up] Write a function `ways(m,n)` that returns the number of ways that we can get from (0,0) to (m,n).

[6 marks]

Note: This problem can probably be solved mathematically, i.e. a formula will work, but the formula will not work for the later parts of this question which builds upon what you do here. Consider yourself warned.

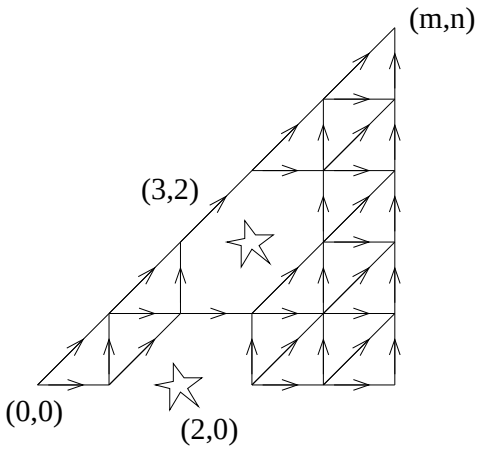
This question tests the student's mastery of recursion. The key is to realize that the solution is just to add 3 sub-problems and to set the right base cases/boundary conditions.

```
def ways(m,n):
    if m==0 and n==0:
        return 1
    elif n>m or m<0 or n<0:
        return 0
    else:
        return ways(m-1,n) + ways(m-1,n-1) + ways(m,n-1)
```

B. **[Blocked Roads]** Triangle City is like Paris. The workers like to go on strike and junctions are occasionally blockaded because of the strikes.

Write a function `blocked_ways(m,n,blocked)` where `blocked` is a list of tuple pairs of the blocked junctions that returns the number of ways that we can get from (0,0) to (m,n) for this scenario. [4 marks]

An example of what the road network looks like when `blocked = [(2,0), (3,2)]` is shown below.



This question is a give away given that it is similar to a past year exam question. The goal of this question is to test that that student knows the transformation to get here from Part(A). If Part(A) is wrong, but the additional 2 lines of code are correct, the student will get full credit.

```
def blocked_ways(m,n,blocked):
    if (m,n) in blocked:
        return 0
    elif m==0 and n==0:
        return 1
    elif n>m or m<0 or n<0:
        return 0
    else:
        return blocked_ways(m-1,n,blocked) + \
        blocked_ways(m-1,n-1,blocked) + blocked_ways(m,n-1,blocked)
```

C. What is the order of growth in time and space for the C function `blocked_ways` you implemented in Part(B) in terms of n and m? Explain. Note that if your Part(B) is too far from correctly done, no credit will be awarded for this part. [4 marks]

Time: $O(3^{n+m})$, worst case is no blockage and it's a three-way tree with some pruning. Note that $O(3^m)$ is not accepted as it is of a slower growth than $O(3^{n+m})$. We have assumed ℓ, the number of elements in `blocked` to be some constant. Otherwise, you can write it as $O(3^{n+m}\ell)$ too.

Space: $O(n+m)$ or $O(m)$ as $m \geq n$, proportional to the maximum depth of the recursion.

2 marks for each correct answer.

D. **[Memoization]** Improve the efficiency of the code you wrote in Part(B) with memoization. We will call the new function `memoized_blocked`. [6 marks]

```
def memoized_blocked(m,n,blocked):
    d = {}
    for m,n in blocked:
    d[(m,n)]= 0

    def helper(m,n):
        if (m,n) in d:
            return d[(m,n)]
        elif m==0 and n==0:
            return 1
        elif n>m or m<0 or n<0:
            return 0
        else:
            ans = helper(m-1,n) + helper(m-1,n-1) + helper(m,n-1)
            d[(m,n)] = ans
            return ans
    return helper(m,n)
```

E. **[We Love ERP]** Triangle City suffers from congestion and is learning from Singapore, Triangle City imposes a toll for every road. It costs \$1 to use a horizontal or vertical road segment and \$3 to use a diagonal segment.

Write a function `cheapest_path(m,n,blocked)` that returns the cost of the cheapest path to get from (0,0) to (m,n), where `blocked` is a list of tuple pairs of the blocked junctions. You can assume that there exists at least one path to the specified destination. [6 marks]

```
def cheapest_path(m,n,blocked):
    if (m,n) in blocked:
        return 4*(m+n+2)
    elif m==0 and n==0:
        return 0
    elif n>m or m<0 or n<0:
        return 4*(m+n+2)
    else:
        return min(1+cheapest_path(m-1,n,blocked),3+ \
            cheapest_path(m-1,n-1,blocked),1+cheapest_path(m,n-1,blocked))
```

This is a test the student has the understanding and mental agility to transform the basic solution in Part(B) to a solution for a similar problem. `4*(m+n+2)` is chosen arbitrary large enough for our purpose, or you may use `float('inf')`.

Question 3: Implementing Collections with Overloading [29 marks]

Mathematically, a set is a collection of objects where there are no duplicates. A multiset is a collection of objects but duplicates are allowed. In this question, we will explore the implementation of these 2 data structures. Note that Python natively implements set but you are not allowed to use the `set` primitive in this question.

A. Clearly, these 2 objects are related and we decided to implement `Set` as a subclass of `MultiSet`. Explain the reasoning of our decision to do so. [2 marks]

A set is a multiset with constraints applied, while a multiset might not be a set. When subclassing, the subclass needs to be a proper subset of the superclass.

Credit will be given if students talk about one being more restrictive or something like that. Some students were writing random stuff.

Our `MultiSet` class implements the following methods:

1. `add(item)` adds new item to the multiset.
2. `extend(seq)` adds all the elements of a specified sequence `seq` to the multiset.
3. `contains(item)` returns `True` if the multiset contains the specified item. Returns `False`, otherwise.
4. `count(item)` returns the number of instances of the specified item in the multiset.

The following is a sketch of our implementation for `MultiSet`:

```
class MultiSet:

    def __init__(self):
        self.stuff = []

    def add(self,item):
        <T1>

    def extend(self,seq):
        <T2>

    def contains(self,item):
        <T3>

    def count(self,item):
        <T4>
```

B. **[Warm up]** Please provide possible implementations for T1, T2, T3, and T4.[4 marks]

<T1>:
[1 marks]

```
self.stuff.append(item)
```

<T2>:
[1 marks]

```
for e in seq:
    self.add(e)
```

<T3>:
[1 marks]

```
return item in self.stuff
```

<T4>:
[1 marks]

```
return self.stuff.count(item)
```

C. Two multisets are equivalent if they have the same number of equivalent objects for each object in the multisets. Add a method that will allow us to test two multisets for equivalence using ==. [5 marks]

```
    def __eq__(self,other):
        my_count = {}
        for e in self.stuff:
            if e not in my_count:
                my_count[e] = 0
            my_count[e] += 1

        for e in other.stuff:
            if e not in my_count:
                return False
            my_count[e] -= 1

        return not my_count or max(my_count.values())==0
```

Alternate solution:

```
    def __eq__(self,other):
        x = list(self.stuff)
        y = list(other.stuff)
        x.sort()
        y.sort()
        return x == y
```

This question tests if the student understands how to override `__eq__` to implement equivalence for objects.

Need to check both sides. Just checking that one contains the other is not sufficient. -2 marks for failing to check one side. This is a very, very common mistake. -1 mark if student names `__eq__` wrongly.

D. Implement the `Set` class as an subclass of `MultiSet` with what you think is the minimal number of lines of code. Note that if `==` is used to check `Set()==MultiSet()`, we should get `False`. Explain why your solution is minimal. [7 marks]

```
class Set(MultiSet):

    def add(self,item):
        if item not in self.stuff:
            self.stuff.append(item)

    def __eq__(self,other):
        if not isinstance(other,Set):
            return False
        else:
            return super().__eq__(other)
```

Overriding 1 out of the 5 original methods seems minimal enough, but note that we don't have to override `extend` only because we implemented `extend` in terms of `add`. If the student's implementation in Part (B) didn't do so, `extend` would also have to be overriden. No penalty for this.

To achieve `MultiSet()!=Set()`, we will need to override `__eq__` and use either `isinstance` or `type` to check.

2 points for `add` and/or `extend`.
3 points for `__eq__`. If students implemented `__eq__` in Part (C) such that no overriding is needed, they get these 3 points for free.
2 points for a reasonable explanation along the lines that overriding `add` and `extend` is sufficient to enforce the set invariant of no repeats.
-1 point for solutions that are clearly not minimal.

E. **[Overloading]** We had discussed overriding in class. Overriding happens when a subclass implements a method that overrides a method in the superclass. There is another concept in OOP known as overloading. This happens when a class implements two methods with the same name, but which takes in different types of argument. Python does not support this natively, but we can still achieve it and in this question, you will tell us how it is done. Suppose we want to enhance `MultiSet` so that we can optionally create a new `MultiSet` with an existing `MultiSet` as input. The new `MultiSet` will contain the same elements, i.e. we have just copied the old `MultiSet`. Additionally, `extend` should accept a `MultiSet` as input in addition to a sequence. Implement the new `MultiSet`, but you can omit methods that are not different from that in Parts (B) and (C). [5 marks]

Sample execution:

```
>>> s1 = MultiSet()
>>> s1.add(1)
>>> s2 = MultiSet(s1)
>>> s2.extend((2,))
>>> s2.extend(s1)
>>> s2.count(1)
2
>>> s2.count(2)
1
```

```
class MultiSet:

    def __init__(self,*args):
        if args and isinstance(args[0],MultiSet):
            self.stuff = list(args[0].stuff)
        else:
            self.stuff = []

    def extend(self,seq):
        if isinstance(seq,MultiSet):
            seq = seq.stuff
        for e in seq:
            self.add(e)
```

We just need to replace `__init__` and `extend`. This is a test that the student understands `*args` and also dispatch on type.

F. [Efficiency Hacking is not Free!] When a colleague tried to use your implementation of your `MultiSet`, it was all good at the beginning, but over time, it seems like the performance was starting to degrade. He thinks that your `contain` and `count` methods are too inefficient. Show how you would improve the implementation of `MultiSet` (using a different data structure) in Part (B) to achieve $O(1)$ performance (in time) for `contain` and `count` methods. What are the limitations of your new implementation (if any). [6 marks]

Basically, we just need to replace `stuff` with a dictionary.

```
class MultiSet:

    def __init__(self):
        self.stuff = {}

    def add(self,item):
        if item not in self.stuff:
            self.stuff[item] = 1
        else:
            self.stuff[item] += 1

    def extend(self,seq):
        for e in seq:
            self.add(e)

    def contains(self,item):
        return item in self.stuff

    def count(self,item):
        if item in self.stuff:
            return self.stuff[item]
        else:
            return 0

    def __eq__(self,other):
        return self.stuff == other.stuff
```

2 points for the limitation is that this new `MultiSet` only works with items that are immutable and hashable, so you won't be able to put lists into this new implementation. Some students said that by using a dictionary, we no longer preserve ordering. This is true but irrelevant since `Set` does not need ordering to begin with and so it's not a limitation.

Question 4: Pancake Sort [18 marks]

Pancake sort is a sorting of a stack of pancakes. The stack is represented as a list with the bottom-most pancake appearing as the first entry in the list, and so on. The sizes of the pancakes are represented with integers, with a small number representing a larger size. We are to sort the stack of pancakes so that the sizes appearing in decreasing order, i.e. for the input of [5, 3, 2, 1, 6, 7, 4], we want the output [1, 2, 3, 4, 5, 6, 7].

In doing sorting, we are allowed only the flip operation. A flip at position *i* is actually reversing all the numbers from position *i* till the top of the stack. For example, the first flip below is "1, 6, 7, 4" to result in "4, 7, 6, 1". With this, we use one flip to bring a small number to the top, and another flip to bring this small number to the correct position. The sample execution is as follows:

```
[5, 3, 2, 1, 6, 7, 4] # before sorting
[5, 3, 2, 4, 7, 6, 1] # after flipping 1 to the top
[1, 6, 7, 4, 2, 3, 5] # after flipping 1 to the correct position
[1, 6, 7, 4, 5, 3, 2] # after flipping 2 to the top
[1, 2, 3, 5, 4, 7, 6] # after flipping 2 to the correct position
[1, 2, 6, 7, 4, 5, 3] # after flipping 3 to the top
[1, 2, 3, 5, 4, 7, 6] # after flipping 3 to the correct position
[1, 2, 3, 5, 6, 7, 4] # after flipping 4 to the top
[1, 2, 3, 4, 7, 6, 5] # after flipping 4 to the correct position
[1, 2, 3, 4, 7, 6, 5] # after flipping 5 to the top
[1, 2, 3, 4, 5, 6, 7] # after flipping 5 to the correct position
[1, 2, 3, 4, 5, 7, 6] # after flipping 6 to the top
[1, 2, 3, 4, 5, 6, 7] # after flipping 6 to the correct position
[1, 2, 3, 4, 5, 6, 7] # after sorting
```

We can implement pancake sort as follows:

```
def pancake_sort(a):
    for i in range(len(a)-1):
        cur_small = i
        for j in range(i, len(a)):
            if a[j] < a[cur_small]:
                cur_small = j
        a = a[:cur_small] + reverse(a[cur_small:])
        a = a[:i] + reverse(a[i:])
    return a
```

A. **[Warm Up]** You are to implement the function `reverse` in place. First explain what you understand by in place. Next, implement `reverse` so that it is both in place and will make the above code work. [4 marks]

In place means that the function will need no extra memory, so all we can do is modify the elements in place.

```
def reverse(lst):
    for i in range(int(len(lst)/2)):
        lst[i],lst[len(lst)-i-1] = lst[len(lst)-i-1],lst[i]
    return lst
```

Note that the solution must also return the original list `lst`. In the explanation of "in-place," you should not mention it in relation to "sorting" as we are talking about "reverse" here. 0.5 is deducted for doing so.

[Repeat in C] Now, let's do the same pancake sort in C. The following is a snippet of C code that attempts to do the same.

```
#include <stdio.h>
#include <string.h>
#define SIZE 7

void print_array(int a[]) {
    for(int i=0; i < SIZE; i++)
        printf("%d ", a[i]);
    printf("\n");
}

void reverse(...){
   // you are to complete this routine
   ...
}

void pancake_sort(int num[]){
    for(int i=0; i < SIZE-1; i++){
        int cur_small = i;
        for(int j=i; j < SIZE; j++)
            if (num[j] < num[cur_small])
                cur_small = j;
        reverse(...);           // decide your parameters #1
        printf("1 flip\n");
        print_array(num);
        reverse(...);           // decide your parameters #2
        printf("2 flip\n");
        print_array(num);
    }
    return;
};
```

```
int main(void) {
    int a[] = {5, 3, 2, 1, 6, 7, 4};
    printf("Before sorting: ");
    print_array(a);
    pancake_sort(a);
    printf("After sorting: ");
    print_array(a);
    return 0;
}
```

B. The details of the C function `reverse` is missing in the above code. Decide on the appropriate input parameters and implement the function `reverse` that achieves the desired output. [6 marks]

```
void reverse(int a[], int start, int end){
    for (int i=start; i < end; i++) {
        int temp = a[i];
        a[i] = a[end];
        a[end] = temp;
        end--;
    }
}
```

Many of you "adapted" constructs available to Python but not to C. For example, "//", array[-1], etc. Another common mistake by students is not having the correct parameters for the function definition (e.g. missing "start").

Note also that the function is not to return any value (return type is "void") as given in Page 16. All updates to the values in the array are done directly to the array inside this function.

C. For the `reverse` function call marked as #1 in the `pancake_sort` routine, please write out the full statement based on your design of `reverse` function. [2 marks]

```
reverse(num, cur_small, SIZE-1);
```

Answer here should be in-line with what you gave in Part B.

D. For the `reverse` function call marked as #2 in the `pancake_sort` routine, please write out the full statement based on your design of `reverse` function. [2 marks]

```
reverse(num, i, SIZE-1);
```

Answer here should be in-line with what you gave in Part B.

E. What is the order of growth in time and space for the C function `reverse` you implemented in Part(B) in terms of the number of pancakes n to be sorted? Explain. Note that if your Part(B) is far from correctly done, no credit can be awarded for this part. [4 marks]

Time: $O(n)$, since we do a `for` loop over all the elements just once.

Space: $O(1)$, we use only a constant number of variables (in fact, just one addition variable).

2 marks for each correct answer.

Question 5: On the Algebra of Happiness [3 marks]

You were asked to watch the video of the talk given by Scott Galloway. What are the 2 most important points do you think he made? Do you agree? Why or why not? Any thoughts about how you plan to live your life to maximize your chance at happiness once you start school in August? If you would rather talk about Randy Pausch's "Last Lecture," that's fine also.

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 and obvious evidence that the student actually bothered to watch the assigned video. 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 ! –