SCHOOL OF COMPUTING

ASSESSMENT FOR
Special Term I, 2014/2105

Solutions for CS1010FC — PROGRAMMING METHODOLOGY

June 2015 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 **EIGHTEEN (18) 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 matriculation number below.**

MATRICULATION NO: _____

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

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Total		

Question 1: Python Expressions [30 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 = {1:2, 2:4, 3:6, 4:7}
for k in a:
    if k%2 == 1:
        del a[k]
print(a)
```

```
RuntimeError:  dictionary changed size during iteration
```

This question tests that students know that they should not be modifying data structures while iterating through them.

B. [5 marks]

```
b = [[1,2], [2,3], [3,4]]
for k,v in b:
    if k == 2:
        v = 4
    if k % 2 == 1:
        v = 0
print(b)
```

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

This question tests that the students knows that when we iterate and map the outputs into variables, we are not modifying the original iterated list. To do the modifications within `a`, we should have done:

```
for k in b:
    if k[0] == 2:
        k[1] = 4
    if k[0] % 2 == 1:
        k[1] = 0
```

C. [5 marks]

```
x = list(range(5))
def foo(y):
    try:
        y = 10
        for i in x:
            y = y/x
    except:
        print(y)
    else:
        print(x)
    finally:
        print("Done!")
foo(x)
```

```
10
Done!
```

This question tests that the students understands Exceptions. What happens is that `y` is set to `10` and then when we try to divide `y` by zero, an division by zero exception is thrown and we go to `print(y)`, which will print 10 because `y` was never reassigned. The `finally` statement will then be run to print `Done!`.

As for partial credit, 2 points for `10` and 3 points for `Done!`.

D. [5 marks]

```
def cool(a, *b):
    beans(a+b)
def beans(*x):
    print(*x)
cool((1,2),(3,4),5)
```

```
(1, 2, (3, 4), 5)
```

This question tests if the student understands `*args` for both function argument and for function call.

-1 point for the students mistakenly thought that `*args` will apply to the print statement to give `1 2 (3, 4) 5`.

E. [5 marks]

```
a = 0
while a < 10:
    print(a)
    a = a % 5
    a = (a+1)**a
print(a)
```

```
0
1
2
9
625
```

This question tests that the student can trace a simple `while` loop correctly. As for partial credit, 2 points will be given for `0` to `9` and 3 points will be given to `625`.

F. [5 marks]

```
x = 5
y = 2
def f(x,y):
    if f(x,y) == f(y,x):
        return y
    elif x == y:
        return x
    elif f(y,y):
        return x
    else:
        return y
print(f(y,x))
```

This code will end up in an infinite loop. This question exists to test the student's ability to trace through code.

Question 2: Number Sum Mania [26 marks]

A positive integer $n \geq 2$ can be expressed as the sum of a number of positive integers smaller than n. For example,

$$\begin{aligned}
2 &= 1+1 \\
3 &= 1+2 \\
&= 1+1+1 \\
4 &= 1+3 \\
&= 2+2 \\
&= 1+1+2 \\
&= 1+1+1+1 \\
5 &= 1+4 \\
&= 1+1+3 \\
&= 2+3 \\
&= 1+2+2 \\
&= 1+1+1+2 \\
&= 1+1+1+1+1
\end{aligned}$$

The function `num_sum` returns the number of ways that an integer can be expressed as the sum of a number of positive integers. From the above examples, it should be clear that:

```
num_sum(2) = 1
num_sum(3) = 2
num_sum(4) = 4
num_sum(5) = 6
```

A. Write the function `num_sum`. **BIG HINT:** `num_sum` is extremely similar to `count_change`, which was discussed in Lecture 4 and reproduced in the Appendix. [6 marks]

```
def num_sum(n):
    def helper(n,k):
        if n == 0:
            return 1
        elif n < 0 or k == 0:
            return 0
        else:
            return helper(n,k-1) + helper(n-k,k)
    return helper(n,n-1)
```

-2 points for each missing base case.

B. Write the function `sum_set` that will return a list of the lists of possible number combinations for the integer sums. **Hint:** Think about how to modify the answer for Part (A). [8 marks]

Sample execution:

```
>>> sum_set(2)
[[1, 1]]

>>> sum_set(3)
[[1, 1, 1], [2, 1]]

>>> sum_set(4)
[[1, 1, 1, 1], [2, 1, 1], [2, 2], [3, 1]]

>>> sum_set(5)
[[1, 1, 1, 1, 1], [2, 1, 1, 1], [2, 2, 1], [3, 1, 1], [3, 2], [4, 1]]

>>> sum_set(6)
[[1, 1, 1, 1, 1, 1], [2, 1, 1, 1, 1], [2, 2, 1, 1], [2, 2, 2], [3, 1, 1, 1],
[3, 2, 1], [3, 3], [4, 1, 1], [4, 2], [5, 1]]
```

```
def sum_set(n):
    result = []
    def helper(n,k,current):
        if n == 0:
            result.append(current)
        elif n<0 or k==0:
            return
        else:
            copy = list(current)
            copy.append(k)
            helper(n,k-1,current)
            helper(n-k,k,copy)
    helper(n,n-1,[])
    return result
```

One student submitted a brute force solution where he generated every possible list with `n` elements, each up to `n` and then `filter` on `lambda x: sum(x) == n`. This solution is really inefficient, but correct, so he was awarded full credit.

C. Write the function `sum_set_product` that will return a list of the products of the integer sums produced by `sum_set`, i.e. multiply together the components of each integer sum. You can assume that you have access to the function `sum_set` even if you cannot do Part (B). [6 marks]

Sample execution:

```
>>> sum_set_product(2) # 1x1
[1]

>>> sum_set_product(3) # 1x1x1 and 2x1
[1, 2]

>>> sum_set_product(4)
[1, 2, 3, 4]

>>> sum_set_product(5) # Note that 4x1 = 2x2x1 so 5 elements, not 6
[1, 2, 3, 4, 6]

>>> sum_set_product(6)
[1, 2, 3, 4, 5, 6, 8, 9]
```

```
def sum_set_product(n):
    result = []
    for s in sum_set(n):
        product = 1
        for e in s:
            product *= e
        if product not in result:
            result.append(product)
    result.sort() # prettify - not necessary
    return result
```

-2 points for failure to check for repeats.

D. Write the function `has_prime_sum` that will return `True` for an integer n if it can be expressed as a sum of **2** prime numbers, or `False` otherwise. Assume that you have access to the function `is_prime` that will return `True` if an integer is prime. [6 marks]

Sample execution:

```
>>> has_prime_sum(2)
False

>>> has_prime_sum(3)
False

>>> has_prime_sum(4) # 2+2
True

>>> has_prime_sum(5) # 2+3
True

>>> has_prime_sum(6) # 3+3
True

>>> has_prime_sum(11) # Not possible!
False
```

```
def has_prime_sum(n):
    for i in range(2,n//2):
        if is_prime(i) and is_prime(n-i):
            return (i,n-i)
    return False
```

Question 3: Foobar – An Adventure in Obscurity [24 marks]

A. Consider the following function `foo`. It should be clear (if it is not already) that it is a sort of some kind. Explain **how** `foo` does what it does (basically, read and explain how the code does the sort).

```
def foo(lst):
    if len(lst) <= 1:
        return lst
    else:
        pivot = lst[0]
        smaller = []
        bigger = []
        for i in range(1,len(lst)):
            if lst[i] <= pivot:
                smaller.append(lst[i])
            else:
                bigger.append(lst[i])
        return foo(smaller)+[pivot]+foo(bigger)
```

[4 marks]

One of the key skills that students need to develop is how to read and understand what a piece of code is doing.

This is an implementation of *Quicksort*, so `foo` returns a new list consisting of the elements of `lst` sorted in ascending order. First, it should be clear that Quicksort uses *recursion* and the base case is either a one element list of empty list. Then, what it does is that it takes the first element of the list as a pivot and runs through the rest of the list to find the elements that are smaller or equal and larger and recursively sorts those like mergesort.

-2 points for failure to mention recursion or calling of `foo` on the sub-lists.

B. What is the order of growth in time and space *in the average case* (i.e. elements uniformly distributed at start) for the function `foo` in Part (A) in terms of n, the number of elements in the list `lst`? Explain. [4 marks]

Time: $O(n \log n)$. Analysis is the same as that for mergesort. There are likely $O(\log n)$ levels. Each level requires approximately n comparisons.

Space: $O(n)$. Recursion causes copies of the original list to be kept in memory. Careful examination will reveal that the total is something in the order of $n + \frac{n}{2} + \frac{n}{4} + \cdots \approx 2n = O(n)$. However, $O(nlogn)$ is acceptable since casual reasoning will yield $\log n$ levels each with no more than n memory.

C. The function `foo` returns a new sorted list and leaves `lst` unmodified. Write the function `bar` that will sort the elements in the same way as `foo`, but that will directly modify `lst` and return it instead. [4 marks]

```
def bar(lst):
    result = foo(lst)
    lst.clear()
    lst.extend(result)
    return lst
```

While grading, we realized that this question was somewhat ambiguous. While the intention clearly was to implement Quicksort that modifies the input list, some students interpreted it as any sort that modifies the input list. Some implemented bubble sort. One gave:

```
def bar(lst):
    lst.sort()
    return lst
```

They all got full credit.

D. **[for to while]** Re-write the following `for` loop using a `while` loop:

```
for i in range(1,len(lst)):
    if lst[i] <= pivot:
        smaller.append(lst[i])
    else:
        bigger.append(lst[i])
```

[4 marks]

```
i = 1
while i < len(lst):
    if lst[i] <= pivot:
        smaller.append(lst[i])
    else:
        bigger.append(lst[i])
    i += 1
```

-2 points for forgetting either `i=1` or `i += 1`.

E. **[In-place Sort]** Is `foo` an *in-place* sort? Explain your answer. [4 marks]

No. This is a giveaway question to test if the student understands what an in-place sort is. Clearly `foo` is not an in-place sort since it creates a whole bunch of new lists. An in-place sort will have order of growth in space of $O(1)$ since it will not require any extra memory.

F. **[Stable Sort]** Is `foo` a *stable* sort? If yes, explain your answer. If no, explain how we can modify `foo` to make the sort stable. [4 marks]

No. Existing `foo` is not a stable sort, but it's simple to fix. Just change

```
    if lst[i] <= pivot:
```

to:

```
    if lst[i] < pivot:
```

One student got full credit for suggesting that `foo` implement bubble sort.

Question 4: PM Can Sudoku, So Can You – Or So You Hope [16 marks]

PM Lee revealed at a recent talk that he writes code in his leisure and the following is an excerpt of PM Lee's Sudoku solver:

```
int InBlock[81], InRow[81], InCol[81];
const int BLANK = 0;
const int ONES = 0x3fe;  // Binary 1111111110

int main(int argc, char* argv[])
{

int i, j, Square;

for (i = 0; i < 9; i++)
for (j = 0; j < 9; j++) {
    Square = 9 * i + j;
    InRow[Square] = i;
    InCol[Square] = j;
    InBlock[Square] = (i / 3) * 3 + ( j / 3);
  }

for (Square = 0; Square < 81; Square++) {
  Sequence[Square] = Square;
  Entry[Square] = BLANK;
  LevelCount[Square] = 0;
 }

for (i = 0; i < 9; i++)
  Block[i] = Row[i] = Col[i] = ONES;

ConsoleInput();
Place(SeqPtr);
printf("\n\nTotal Count = %d\n", Count);

return 0;

}
```

A. What is the content of the arrays `InRow`, `InCol` and `InBlock`, after the following snippet of code is run:

```
int InBlock[81], InRow[81], InCol[81];

. . .

for (i = 0; i < 9; i++)
for (j = 0; j < 9; j++) {
    Square = 9 * i + j;
    InRow[Square] = i;
    InCol[Square] = j;
    InBlock[Square] = (i / 3) * 3 + ( j / 3);
  }
```

[6 marks]

```
InRow:                      InBlock:
[0,0,0,0,0,0,0,0,0,         [0,0,0,1,1,1,2,2,2,
 1,1,1,1,1,1,1,1,1,          0,0,0,1,1,1,2,2,2,
 2,2,2,2,2,2,2,2,2,          0,0,0,1,1,1,2,2,2,
 3,3,3,3,3,3,3,3,3,          3,3,3,4,4,4,5,5,5,
 4,4,4,4,4,4,4,4,4,          3,3,3,4,4,4,5,5,5,
 5,5,5,5,5,5,5,5,5,          3,3,3,4,4,4,5,5,5,
 6,6,6,6,6,6,6,6,6,          6,6,6,7,7,7,8,8,8,
 7,7,7,7,7,7,7,7,7,          6,6,6,7,7,7,8,8,8,
 8,8,8,8,8,8,8,8,8]          6,6,6,7,7,7,8,8,8]

InCol:
[0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8,
 0,1,2,3,4,5,6,7,8]
```

B. Write the equivalent Python code for the code snippet in Part (A). `InRow`, `InCol` and `InBlock` should be lists in place of the original C arrays. [5 marks]

```
InBlock = [0]*81
InRow = [0]*81
InCol = [0]*81
for i in range(9):
    for j in range(9):
        Square = 9 * i + j
        InRow[Square] = i
        InCol[Square] = j
        InBlock[Square] = int(i / 3) * 3 + int( j / 3)
```

-2 points if student misses out the `int` operation for the `InBlock` list. C division of integers does not support floating point operations. Alternatively, `int(i / 3)` may be substituted with `i // 3`.

C. [**Fizzbuzz**] Write a C program that prints the numbers from 1 to 100, with one number in each row, except for the following:

- for the multiples of three, print `"Fizz"` instead;
- for the multiples of five, print `"Buzz"` instead; and
- for numbers which are multiples of both three and five, print `"FizzBuzz"`.

[5 marks]

```
int main()
{
    int i;
    for (i=1; i<=100; i++){
        if (i%3 == 0) {
            if (i%5 == 0) {
                printf("Fizzbuzz\n");
            } else {
                printf("Fizz\n");
            }
        } else if (i%5 == 0) {
            printf("Buzz\n");
        } else {
            printf("%d\n", i);
        }

    }
}
```

Alternatively, `i%15 == 0` must come first in the `if-else` conditions (or -2 points).

```
int main()
{
    int i;
    for (i=1; i<=100; i++){
        if (i%15 == 0) {
            printf("FizzBuzz\n");
        } else if (i%3 == 0) {
            printf("Fizz\n");
        } else if (i%5 == 0) {
            printf("Buzz\n");
        } else {
            printf("%d\n", i);
        }

    }
}
```

Question 5: Computer Science – A Liberal Art? [4 marks]

Steve Jobs once said,

> *"I think everybody in this country should learn how to program a computer. To learn a computer language because it teaches you how to think. It's like going to law school. I don't think anybody should be a lawyer, but I think going to law school could actually be useful because it teaches you how to think in a certain way. In the same way, that computer programming teaches you, in a slightly different way, how to think. And, so I view Computer Science as a liberal art. It should be something that everybody takes a year in their life... one of courses they take is learning how to program."*

You have learnt how to program in CS1010FC. Do you agree with Jobs? Explain.

The student will be awarded points as long as he/she is coherent and doesn't say something obviously wrong.

Appendix

The following are some functions that were introduced in class:

Count Change [Lecture 4]

```
def cc(amount, kinds_of_coins):
    if amount == 0:
        return 1
    elif amount < 0 or kinds_of_coins == 0:
        return 0
    else:
        return cc(amount, kinds_of_coins-1)
             + cc(amount - first_denomination(kinds_of_coins), kinds_of_coins)

def first_denomination(kinds_of_coins):
  ... <left as an exercise>

def count_change(amount)
  return cc(amount,5)
```

— 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 ! –