NATIONAL UNIVERSITY OF SINGAPORE

Semester 2, 2018/2019

IT5001 - Software Development Fundamentals

Time Allowed: 2 Hours

INSTRUCTION TO CANDIDATES

1. This is a CLOSED book assessment. You are allowed to bring one A4 size cheat sheet.
2. This assessment paper contains **SEVEN(7)** questions and comprises THIRTEEN(13) printed pages.
3. Answer *ALL* questions within the spaces provided in this booklet.
4. You are allowed to use the back of the paper but please remember to state "P.T.O."
5. *Cross out any draft* or otherwise we will mark the poorer answers.
6. Please write your student number below, but NOT your name.
7. The light gray lines in the program box are for your indentations.
8. For coding questions, the length and style contribute to the final marks. For example, you cannot get full marks if your code is unnecessarily long or hard-coded.

STUDENT NUMBER:________________________

(This portion is for examiner's use only)

Question	Max. Marks	Score	Check
Q1	10		
Q2	14		
Q3	24		
Q4	22		
Q5	10		
Q6	10		
Q7	10		
Total	100		

Question 1 (10 marks) Expression Evaluation

Evaluate the following terms. If we type them into the shell, what will be the output or echo from IDLE? If any of these causes an error, please write “error” instead. **<u>The type of your answer is important</u>**, e.g. the integer 5 is different from ‘5’ or 5.0.

Evaluate the Following:	**Answer:**
`print(1**2+3-4*5)`	
`'aacb' > 'aaaba'`	
`'abc' in 'aaabbbccc'`	
`True + 3`	
`20//6*3`	
`[i**2 for i in range(4)]`	
`tuple('abcd')`	
`1 if False else 'huh'`	
`list(range(4,20,6))`	
`'abc'[5]`	

Question 2 (14 marks = 2 × 7) Code Tracing

Each row of the table is a separate piece of program/file. What is the output of each of them when we run it? If the code produces errors or runs into infinite loops, please state ‘error’ or ‘infinite loop’ respectively.

Code	**Output**

`f = lambda x,y:x*y`
`print(f(f(1,2),f(3,4)))`	

`x = 0`
`for i in range(10):`
`    x += i`
`print(x)`	

`n = 2`
`while (n%2==0) or (n%3==0):`
`    n += 1`
`print(n)`	

`d = {'a':1, 'b':2, 'c':3}`
`print(d['b']**d['c'])`	

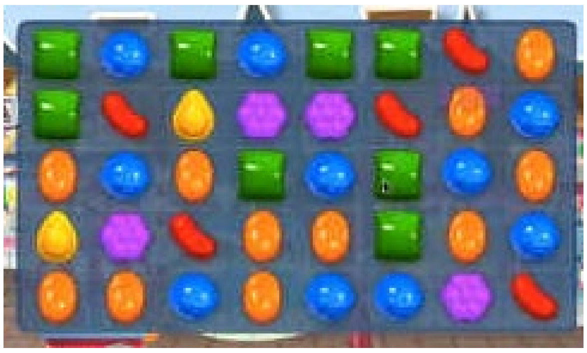
`a = (1,2,3)`
`b = [4,5,6]`
`print(list(a+b))`	

`def foo(n):`
`    a,b = 0,1`
`    for i in range(n):`
`        c = a+b`
`        a,b = b,c`
`    return b`
 	
`for i in range(7):`
`    print(foo(i))`	

`def foo(L):`
`    if not L:`
`        return 1`
`    return L[0] * foo(L[1:])`
 	
`print(foo([2,6,10]))`	

Question 3 (24 marks = 5 + 8 + 4 + 5 + 2) Candy Crush Saga

I hope everyone has tried the game *Candy Crush Saga* before. In case you have not, you got six different kinds of candies on a board with different colors, namely, red, orange, yellow, green, blue, and purple. Here is a screenshot of the game at Level 1. There are 40 candies on the board. (If two candies are in the same shape, they are in the same color.)



Ultimate Task:

Let's focus on Level 1 with 40 candies on the board. Suppose each candy is placed in random with an equal chance. What is the probability that the board has **at least THREE of each type of candy**? When this happens, we name it as a ***BINGO*** board! We will guide you through this problem step by step. **For each task, you can assume that functions in the previous tasks were correctly implemented (except Task 1 of course).**

Question 3 Task 1

Suppose we simplify the problem by representing each color with an integer from 1 to 6, e.g. red = 1, yellow = 2,… etc. In order to generate a random board, write a function gen(N) to take in an integer N and generate a list of N random numbers ranging from 1 to 6, inclusively.

Sample output:

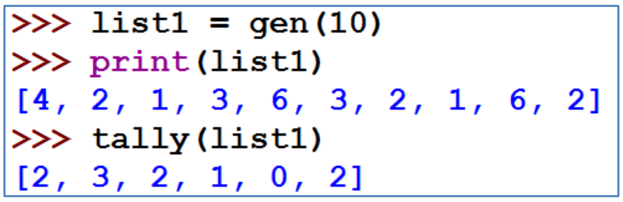
```
>>> gen(10)
[1, 2, 2, 1, 3, 3, 6, 6, 5, 1]
>>> gen(40)
[1, 4, 2, 1, 3, 5, 4, 5, 1, 5, 5, 5, 5, 2, 4, 1, 4, 6, 4, 4, 5, 5, 3
, 3, 5, 4, 2, 5, 2, 1, 5, 1, 6, 4, 1, 4, 3, 6, 1, 5]
```

```
From random import randint
def gen(N):
```

Question 3 Task 2

Write a function `tally(nlist)` who takes in a list of numbers `nlist` generated from Task 1 and count the number of times that each number appears. The function will output a list L such that `L[i]` records the number of times that the number **`i+1`** appears in the input of the function. For example, the number 1 appears two times in `list1` below so the output `L[0] = 2`.

Sample output:



```
>>> list1 = gen(10)
>>> print(list1)
[4, 2, 1, 3, 6, 3, 2, 1, 6, 2]
>>> tally(list1)
[2, 3, 2, 1, 0, 2]
```

```
def tally(nlist):
```

Question 3 Task 3

Write a function `checkBingo(nlist)` to take in a list `nlist` from **Task 1** and return True if it is a BINGO board! (And false otherwise.) Recap that you got a BINGO board if there are **at least 3 candies of each type** in a board. Note that the input is a random number list, but NOT the tally list from Task 2.

Sample output:

```
>>> list1 = gen(40)
>>> print(list1)
[1, 3, 2, 3, 3, 5, 2, 3, 1, 3, 4, 5, 2, 4, 2, 1, 4, 5, 5, 1, 2, 4,
1, 3, 6, 4, 2, 6, 5, 4, 2, 4, 3, 6, 2, 5, 6, 6, 6, 3]
>>> print(tally(list1))
[5, 8, 8, 7, 6, 6]
>>> checkBingo(list1)
True
```

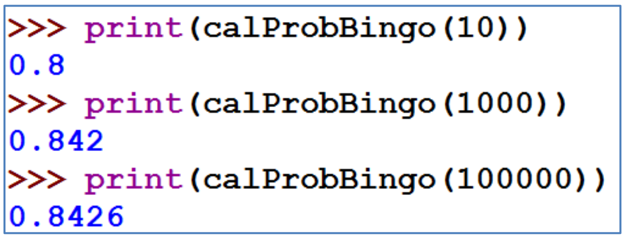
```
>>> list2 = gen(40)
>>> print(tally(list2))
[10, 6, 4, 6, 2, 12]
>>> print(list2)
[6, 6, 4, 6, 1, 1, 1, 2, 1, 6, 1, 3, 6, 4, 6, 3, 6, 4, 4, 3, 6, 2,
2, 4, 1, 3, 2, 2, 1, 6, 5, 2, 5, 6, 1, 6, 1, 4, 1, 6]
>>> checkBingo(list2)
False
```

```
def checkBingo(nlist):
```

Question 3 Task 4

Finally! Let's generate N different boards randomly to check what is the probability of having a BINGO board! Write a function `calProBingo(N)` to take in an integer N and generate N boards (with 40 candies each) and check how many of them are BINGO boards and return the probability.

Sample output:



```
>>> print(calProbBingo(10))
0.8
>>> print(calProbBingo(1000))
0.842
>>> print(calProbBingo(100000))
0.8426
```

```
def calProbBingo(N):
```

Question 3 Task 5

What is the "skill name"/methodology we used in this question, when we implement a task and assuming the previous task(s) are implemented?

Question 4 (22 marks = 4 + 5 + 5 + 8) Inheritance

In this question, you are going to use OOP and write classes for different types of vending machines.

To start with, here is a basic one.

```
class VendingMachine(object):
    def __init__(self):
        self.drinkOption = []
        return

    def dispense(self,drink):
        print(f"{drink} dispensed.")

    def buyDrink(self,drink):
        print(f"{drink} is requested")
        if drink in self.drinkOption:
            self.dispense(drink)
        else:
            print(f"This machine has no {drink}")
```

To "use" this machine, we simply create an instance of it.

```
>>> vm = VendingMachine()
>>> vm.buyDrink('Water')
Water is requested
This machine has no Water
```

So, you can see that the machine is pretty "empty" and we cannot buy anything from it. Now we build a soft drink machine by inheritance:

```
class SoftDrinkVM(VendingMachine):
    def __init__(self):
        super().__init__()
        self.drinkOption = ['Coke','Sprite','Orange Juice']
```

Now you can see there are more options. So, we can now do this:

```
>>> svm = SoftDrinkVM()
>>> svm.buyDrink('Water')
Water is requested
This machine has no Water
>>> svm.buyDrink('Coke')
Coke is requested
Coke dispensed.
```

Now your first task is to implement a class `CoffeeMachine`, that can provide coffee and tea only.

Question 4 Task 1

From which class(s) should `CoffeeMachine` inherit from?

What is/are the member function(s) that you need to rewrite? Just name the function name(s) below.

Question 4 Task 2

Implement the class `CoffeeMachine` and the member function(s) from the previous question. The sample output should look something like this:

```
>>> cvm = CoffeeMachine()
>>> cvm.buyDrink('Coke')
Coke is requested
This machine has no Coke
>>> cvm.buyDrink('Coffee')
Coffee is requested
Coffee dispensed.
```

```
class CoffeeMachine(                          ):
```

Question 4 Task 3

Now, you are going to implement a `PremiumCoffeeMachine`, in which, it can dispense whatever the `CoffeeMachine` provides, **plus**, cappuccino and Latte. Moreover, there are two new functions that can change the options for sugar.

Sample output:

```
>>> cvm = PremiumCoffeeMachine()
>>> cvm.buyDrink('Coffee')
Coffee is requested
Coffee dispensed with sugar
>>> cvm.pressNoSugarButton()
No Sugar Please
>>> cvm.buyDrink('Tea')
Tea is requested
Tea dispensed without sugar
```

```
>>> cvm.pressAddSugarButton()
Add Sugar Please
>>> cvm.buyDrink('Latte')
Latte is requested
Latte dispensed with sugar
```

From which class should the class `PremiumCoffeeMachine` inherit from?

What is/are the member function(s) that you need to rewrite? Just name the function name(s) below.

Question 4 Task 4

Implement the class `PremiumCoffeeMachine` below:

```
class PremiumCoffeeMachine(                    ):
```

Question 5 (10 marks): Debugging

Mr. Silly is writing a function (on the right) to check if an item n is in a list L. We assume that L is always a list or a tuple. The function looks correct in the following sample output:

```
>>> search([1,2,3,4,5,6,7],1)
True
>>> search([1,2,3,4,5,6,7],9)
False
```

```
def search(L,n):
    if len(L) == 0:
        return False
    for i in L:
        if i == n:
            return True
        else:
            return False
```

a) Is the code correct in all situations? Please justify your answer if you think the code is correct. Otherwise, please give an example that the code will fail.

b) Correct his code with minimal changes below. You must follow his code as much as possible and keep the first three lines the same in the function body.

```
def search(L,n):
    if len(L) == 0:
        return False
    for i in L:
```

Question 6 (10 marks) Find the smallest and largest number

Write a function `findSMest(L)` to find the smallest and the largest number of the sequence L and return them as a tuple. You can assume all the elements in L are numbers and the length of L is at least 1.

Note: In this question, you are **not allowed to use any in-built function**s, especially `max()`, `min()`, `sort()` and etc.

Sample output:

```
>>> print(findSMest((5,3,1,6,999,2,7)))
(1, 999)
```

```
def findSMest(L):
```

Question 7 (10 marks) DNA Reversals

A DNA is a sequence of amino acids represented by the four characters, 'A', 'C', 'G' and 'T'. In bioinformatics, it is a standard and very popular operation to select and reverse a portion the sequence.

Write a function `DNAreversal(dna, pos, l)` to reverse the section in the DNA sequence starting with **pos** and with a length of **`l`**. You can assume the `(pos + l)` is less than the total length of the DNA.

Sample Output:

```
>>> print('AGCTTACTATACAAT')
AGCTTACTATACAAT
>>> print(DNAreversal('AGCTTACTATACAAT',2,6))
AGTCATTCATACAAT
>>>
```

The highlights and underlines are to show you where the changes are only.

```
def DNAreversal(dna,pos,l):
```

- End of Paper -