NATIONAL UNIVERSITY OF SINGAPORE

Semester 1, 2019/2020

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 **NINE(9)** questions and comprises TEN(10) 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.
9. You cannot import any packages or functions unless it's specified.
10. Other than the RPG questions, your function should return values instead of printing them.

STUDENT NUMBER:________________________

(This portion is for examiner's use only)

Question	Max. Marks	Score	Check
Q1	10		
Q2	14		
Q3	7		
Q4	7		
Q5	7		
Q6	10		
Q7	20		
Q8	10		
Q9	15		
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. **The type of your answer is important**, e.g. the integer 5 is different from ‘5’ or 5.0.

Evaluate the Following:	**Answer:**
`not True or True and False or not True`	
`'abc' < 'abd' < 'abe'`	
`4 ** (3 > 2) + 1`	
`2*2 ** 2*2`	
`27 // 6 % 2`	
`(lambda x:x*2)(lambda x:x[::2])('abcdefg')`	
`list((lambda x,y,z:x(y,z))(map,lambda x:x*2,[1,2,3]))`	
`list(map(lambda x:x%3,list(range(6))))`	
`5 & 2`	
`{1:2,3:4,5:6,1:3}[1]`	

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**

```
flag = True
while i < 20 and flag:
    if ((20-i)%5 == 0):
        flag = not flag
    i += 2
print(i)
```

```
def count4(n):
    cnt = [0] * 4
    for i in range(n+1):
        if i%2 == 0:
            if i%3 == 0:
                cnt[0] += 1
            elif i%4 == 0:
                cnt[1] += 1
        elif i%3 == 0:
            cnt[2] += 1
        else:
            cnt[3] += 1
    return cnt
print(count4(30))
```

```
x = 1
y = 300
while (x < 300 and y > 10):
   x *= 3
   y //= 3
print(x,y
```

```
def tup2(x,y):
    cnt = 0
    for i in range(1,y,x):
        for j in range(0,y,i):
            cnt += 1
    return cnt
print(tup2(3,12))
```

```
def rec2(stg, acc):
    if stg:
        tmp = rec2(stg[1:],'1'+acc)
        return tmp + '0'
    else:
        return acc
print(rec2('1234','5'))
```

```
def f2(n,g):
    if n > 1:
        return f2(n-1,lambda x: g(g(x)))
    return g(n)
print(f2(4, lambda x: x+1))
```

```
aList = [1,2,3,4,5,6]
def setn(n, lst):
    if n == 0:
        lst[0] = -100
    else:
        setn(n-1,lst[1:])
setn(3,aList)
print(aList)
```

Question 3 Sum of Digit Squares (Iterative) (7 marks)

Given a non-negative integer n, your task is to **sum the square of the digits** (**SDS**) of n. For example, if n = 2324,

$$SDS(2324) = 2^2+3^2+2^2+4^2 = 33$$

Write an ***iterative*** version of the function `sumDigitSqrVerI(n)`. You can assume that your input is any non-negative integers. Your outputs have to be integers. Here are some sample output:

```
>>> sumDigitSqrVerI(123456)
91
>>> sumDigitSqrVerI(987654321)
285
```

```
def sumDigitSqrVerI(n):
```

Question 4 Sum of Digit Squares (Recursive) (7 marks)

Your Task

Write a ***recursive*** version of the function `sumDigitSqrVerR(n)` with the same functionality in Question 3.

```
def sumDigitSqrVerR(n):
```

Question 5 Sum of Digit Squares (Ultimate Version) (7 marks)

Write a function `sumDigitSqrVerUltimate(n)` ***without*** using recursion or iteration. It means you cannot call your own function and you cannot use any loop such as `for` or `while`.

```
def sumDigitSqrVerUltimate(n):
```

Question 6 Happy Numbers (10 marks)

Let n = 7, the sum of digit squares (SDS) of n is 49. And the SDS of 49 is 97, and the SDS of 97 is 130, and repeat…

$$7 \xrightarrow{SDS} 49 \xrightarrow{SDS} 97 \xrightarrow{SDS} 130 \xrightarrow{SDS} 10 \xrightarrow{SDS} 1$$

We call a number n a ***happy number**** if it becomes 1 after some iteration(s)! The number 836 is a happy number and 930 is not because,

$$836 \xrightarrow{SDS} 109 \xrightarrow{SDS} 82 \xrightarrow{SDS} 68 \xrightarrow{SDS} 100 \xrightarrow{SDS} 1$$

$$930 \xrightarrow{SDS} 90 \xrightarrow{SDS} 81 \xrightarrow{SDS} 65 \xrightarrow{SDS} 61 \xrightarrow{SDS} 37 \xrightarrow{SDS} 58 \xrightarrow{SDS} 89 \xrightarrow{SDS} 145 \xrightarrow{SDS} 42 \xrightarrow{SDS} 20 \xrightarrow{SDS} 2$$

Some facts about the happy numbers:

- For any number n for $0 \le n < 10$, only 1 and 7 are happy numbers.
- Therefore, the numbers 2, 3, 4, 5, 6, 8 and 9 will never reach 1 (or 7) no matter how many times you apply SDS onto any one of them.
- Also, you can assume that, for any number $n > 9$, the cycle will produce a number that is smaller than 10 eventually if you keep applying SDS repeatedly.

Write a function `isHappyNumber(n)` to check if n is a happy number. Return `True` if n is a happy number and `False` otherwise. You can assume the number n is greater than 0. You can assume the functions in the previous questions work and call them.

Here are some sample outputs:

```
>>> print(isHappyNumber(83))
False

>>> print(isHappyNumber(849))
False

>>> print(isHappyNumber(10888))
True
```

*The *happy number* is a **real** definition in mathematics (not something we made up!). There are other definitions like *Harshad numbers*, which means “numbers with great joy” in Sanskrit (a classical Indian language).

```
def isHappyNumber(n):
```

Question 7 RPG OOP (10+10 marks)

Remember our RPG assignment? We are going to build more characters on top of that in this question. You can assume the function `randint(a,b)` (that generates a random integers x for a <= x <= b) is imported from the package `random`.

Question 7 Task 1

Write the class for a new type of character `ConfusedFighter`. He will have the same parameters (str, wis, hp and so on) as a `Fighter`, except that the class name is '`ConfusedFighter`' and his cost is only 50. When he acts, he will attack as the same manner as a Fighter, but to the opponent or any of his own team members, or even hurt himself! Luckily, he only has a %25 chance to attack his own team.

```
class ConfusedFighter(                    ):
```

Question 7 Task 2

Write the class for a new type of character **`Vampire`**. He will have the same parameters (str, wis, hp and so on) as a Fighter, except that the class name is 'Vampire' and his cost is 300. When he acts, he will attack in the same manner of a fighter, except one extra ability. Whatever damage he inflicts to his enemy, he will "suck" half of that damage and use that to increase his hp if his hp is not full, namely, he cannot exceed his maxhp.

```
class Vampire(                    ):
```

This is the skeleton code given in the assignment for your reference.

```
class Character(object):
    def __init__(self):
        self.name = ''
        self.maxhp = 1000
        self.hp = 1000
        self.str = 0
        self.maxmana = 0
        self.mana = 0
        self.cost = 9999999999
        self.alive = True
    def act(self,myTeam,enemy):
        return
    def gotHurt(self,damage):
        if damage >= self.hp:
            self.hp = 0
            self.alive = False
            dprint(self.name + ' died!')
        else:
            self.hp -= damage
            dprint(self.name +
                   f' hurt with remaining hp {self.hp}.')

class Fighter(Character):
    def __init__(self):
        super().__init__()
        self.name = 'Fighter'
        self.maxhp = 1200
        self.hp = 1200
        self.str = 100
        self.cost = 100

    def act(self,myTeam,enemy):
        target = randAlive(enemy)
        dprint(f'Hurt enemy {target} by damage {self.str}.')
        enemy[target].gotHurt(self.str)

class Mage(Character):
    def __init__(self):
        super().__init__()
        self.name = 'Mage'
        self.maxmana = 50
        self.mana = 50
        self.hp = 800
        self.cost = 200
        self.int = 400
    def cast(self,myTeam,enemy):
        self.mana -= manaCost
        target = randAlive(enemy)
        dprint(f'Strike enemy {target} with spell')
        enemy[target].gotHurt(self.int)

    def act(self,myTeam,enemy):
        if self.mana < manaCost:
            self.mana += manaRecovery
            dprint(f'Mana recover to {self.mana}.')
        else:
            self.cast(myTeam,enemy)
```

Question 8 (10 marks) Deep Map

Remember we have the following functions discussed in the lecture

```
def map(fn, seq):
    if seq == ():
        return ()
    else:
        return (fn(seq[0]), ) + map(fn, seq[1:])
```

```
def scale_tree(tree, factor):
    def scale_func(subtree):
        if is_leaf(subtree):
            return factor * subtree
        else:
            return scale_tree(subtree, factor)
    return map(scale_func, tree)
```

Write a generic function deep_map(f,tree) that will take in a tree and reproduce a tree that map the function f to each leave. You should use our version of map() instead of the Python built-in map(). Sample output:

```
>>> tree = (1,(4,5,(6,7,8),(((9,),),)))
>>> print(deep_map(lambda x:x*2,tree))
(2, (8, 10, (12, 14, 16), (((18,),),)))
>>> print(deep_map(lambda x:x*x,tree))
(1, (16, 25, (36, 49, 64), (((81,),),)))
>>> print(deep_map(lambda x:(x,),tree))
((1,), ((4,), (5,), ((6,), (7,), (8,)), ((((9,),),),)))
```

```
def deep_map(f, tree):
```

Question 9 (15 marks) Permutation

In this question, you cannot import any package or function.

Given a string s and a number n, we want to generate all the possible combination of using the characters in s with length n. For example, if `s = 'ab'`, and all the combinations of `'a'` and `'b'` with a length of 4 is:

```
>>> product('ab',4)
['aaaa', 'aaab', 'aaba', 'aabb', 'abaa', 'abab', 'abba', 'abbb', 'baaa', 'baab',
'baba', 'babb', 'bbaa', 'bbab', 'bbba', 'bbbb']
```

If `s = '12345'` and `n = 4`:

```
>>> product('12345',2)
['11', '12', '13', '14', '15', '21', '22', '23', '24', '25', '31', '32', '33', '34',
'35', '41', '42', '43', '44', '45', '51', '52', '53', '54', '55']
```

You can assume that the string only contains printable characters and there is no duplicate in the string, e.g. "`abc12a`" and "`12 abc`" are not allowed. Write a function `product()` to return a **<u>list</u>** of all permutations for any string with n > 0.

```
def product(s, n):
```

End of paper