

Midterm Test

25 March 2017

Time allowed: 1 hour 45 minutes

Student No:

A								
---	--	--	--	--	--	--	--	--

Instructions (please read carefully):

1. Write down your matriculation number on the **question paper**. DO NOT WRITE YOUR NAME ON THE QUESTION SET!
2. This is **an open-sheet test**. You are allowed to bring one A4 sheet of notes (written on both sides).
3. This paper comprises **FIVE (5) questions** and **TWENTY (20) pages**. The time allowed for solving this test is **1 hour 45 minutes**.
4. The maximum score of this test is **100 marks**. The weight of each question is given in square brackets beside the question number.
5. All questions must be answered correctly for the maximum score to be attained.
6. All questions must be answered in the space provided in the answer sheet; no extra sheets will be accepted as answers.
7. The back-sides of the sheets and the pages marked “scratch paper” in the question set may be used as scratch paper.
8. You are allowed to un-staple the sheets while you solve the questions. Please make sure you staple them back in the right order at the end of the test.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like (no red color, please).

GOOD LUCK!

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 and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why. Partial marks may be awarded for workings if the final answer is wrong.

A.

```
a = 2
b = a**2 + a
c = b//a + b%a
if c%b > a:
    print("Higher")
elif c%b < a:
    print("Lower")
else:
    print("Same same!")
```

[4 marks]

B.

```
def x(y):
    def y(z):
        x = lambda x: x**2
        y = lambda x: x+2
        def z(x):
            return x(5)
        return z(y)
    return y(x)
print(x(lambda x: x+1))
```

[4 marks]

C.

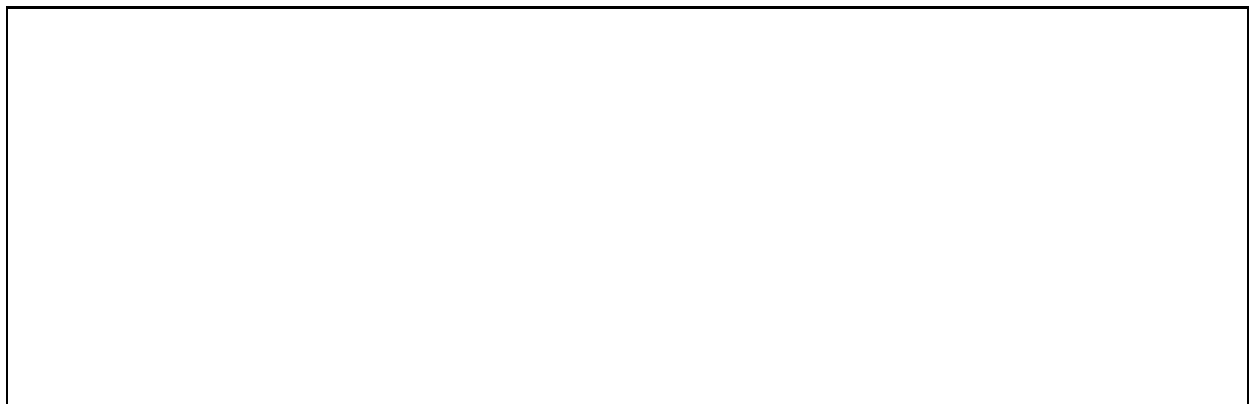
```
result = 8
for i in range(10,5,-1):
    result = result**2 % i
    if result%2 == 1:
        result = 2*result
print(result)
```

[4 marks]

**D.**

```
i, count = 3,0
while i != 1:
    if i%2==1:
        i = 3*i+1
    else:
        i = i//2
    if count > i:
        break
    count += 1
print(count)
```

[4 marks]



E.

```
def twice(f):  
    return lambda x: f(f(x))  
print(twice(twice)(twice(lambda x: x+3))(2))  
print(twice(twice)(twice)(lambda x: x+3)(2))
```

[4 marks]

F.

```
def foo(x):  
    if x < 4:  
        return x  
    else:  
        return bar(x//2)  
def bar(y):  
    if y%2:  
        return foo(3*y+1)  
    else:  
        return foo(y//2)  
print(foo(10))
```

[4 marks]

Question 2: Continued Fractions [27 marks]

The following is mathematical structural called continued fraction, which is a fraction of the following form:

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_n}}}} \quad (1)$$

A. Suppose you are given a tuple of coefficients (a_0, a_1, \dots, a_n) . Implement the function `evaluate_cfraction` that takes as input the tuple of coefficients and returns the value of the continued fraction. [4 marks]

Example execution:

```
>>> evaluate_cfraction((1,))
1

>>> evaluate_cfraction((5,))
5

>>> evaluate_cfraction((3, 3))
3.3333333333333335

>>> evaluate_cfraction((4, 1, 5))
4.833333333333333
```

```
def evaluate_cfraction(coefficients):
```

B. What is the order of growth in terms of time and space for the function you wrote in Part (A) in terms of n , the number of coefficients. Explain your answer. [4 marks]

Time:

Space:

C. Is the function `evaluate_cfraction` that you wrote in Part (A) recursive or iterative? If it is recursive, implement the iterative version; if it is iterative, implement the recursive version. [5 marks]

```
def evaluate_cfraction(coefficients):
```

D. What is the order of growth in terms of time and space for the function you wrote in Part (C) in terms of n , the number of coefficients. Explain your answer. [4 marks]

Time:

Space:

E. It is great that you can evaluate a continued fraction. Now suppose you are given a rational number $\frac{a}{b}$, where a and b are integers and $b \neq 0$, implement the function `compute_coefficients` that takes in two integers a and b and returns a tuple of the coefficients for the continued fraction of the form:

$$a_0 + \frac{1}{a_1 + \frac{1}{a_2 + \frac{1}{\dots + \frac{1}{a_n}}}} \quad (2)$$

where a_i is an integer for all i .

[5 marks]

Example execution:

```
>>> compute_coefficients(1,1)
(1,)

>>> compute_coefficients(10,2)
(5,)

>>> compute_coefficients(10,3)
(3,3)

>>> compute_coefficients(145,30)
(4,1,5)
```

```
def compute_coefficients(a,b):
```

F. Is the function `compute_coefficients` that you wrote in Part (E) recursive or iterative? If it is recursive, implement the iterative version; if it is iterative, implement the recursive version. [5 marks]

```
def compute_coefficients(a,b):
```


Question 3: Higher Order Functions [23 marks]

Consider now the following sequence of terms:

$$\begin{aligned}
 T_0 &= a \\
 T_1 &= a + \frac{b}{a} \\
 T_2 &= a + \frac{b}{a + \frac{b}{a}} \\
 &\vdots \\
 T_n &= a + \frac{b}{a + \frac{b}{a + \frac{b}{\ddots + \frac{b}{a}}}}
 \end{aligned}$$

A. [Warm-up] Implement the function `compute_nterms(a,b,n)` that computes the n th term in the series given a and b . [3 marks]

```
def compute_nterms(a,b,n):
```

B. We note that we can define `compute_nterms(a,b,n)` in terms of `fold` (see Appendix) as follows:

```
def compute_nterms(a,b,n):
    return fold(<T1>, <T2>, <T3>)
```

Please provide possible implementations for $T1$, $T2$, and $T3$.

[5 marks]

<T1>:
[2 marks]

<T2>:
[2 marks]

<T3>:
[1 marks]

C. Given the following higher order function `foldl`:

```
def foldl(op, f, n):  
    if n == 0 :  
        return f(0)  
    else :  
        return op(foldl(op, f, n-1), f(n))
```

We note that we can define `compute_nterms(a, b, n)` as follows:

```
def compute_nterms(a, b, n):  
    return foldl(<T4>, <T5>, <T6>)
```

Please provide possible implementations for T4, T5, and T6.

[5 marks]

<T4>:
[2 marks]

--

<T5>:
[2 marks]

--

<T6>:
[1 marks]

--

D. Given the following higher order function `tail`:

```
def tail(f, a, n):
    if n == 0:
        return a
    else:
        return tail(f, f(n, a), n-1)
```

We note that we can define `compute_nterms(a,b,n)` as follows:

```
def compute_nterms(a,b,n):
    return tail(<T7>, <T8>, <T9>)
```

Please provide possible implementations for T7, T8, and T9.

[5 marks]

<T7>:
[2 marks]

--

<T8>:
[2 marks]

--

<T9>:
[1 marks]

--

E. Given the following higher order function `head`:

```
def head(f, a, n):  
    result = 0  
    for i in range(n):  
        result = f(result, i, n)  
    return result
```

We note that we can define `compute_nterms(a, b, n)` as follows:

```
def compute_nterms(a, b, n):  
    return head(<T10>, <T11>, <T12>)
```

Please provide possible implementations for T10, T11, and T12.

[5 marks]

<T10>:
[2 marks]

--

<T11>:
[2 marks]

--

<T12>:
[1 marks]

--

Question 4: Pokemon LOL! [23 marks]

Pokemon are the pocket monsters that populate the world of Pokemon Go! In this problem, you will implement Pokemons. The basic problem is very simple, but in order to fully satisfy the requirements for the problem, you are advised to read through all the subquestions before you start. Your answer for Part(A) will affect your answer to subsequent sub-problems. If you do the wrong thing in Part(A), you might end up not being able to solve later sub-problems.

You are required to implement the following functions in this problem.

1. `make_pokemon` takes in the species of the pokemon, type and combat points and returns a new pokemon with these characteristics.
2. `get_name` returns the species of a pokemon.
3. `get_type` returns the type of a pokemon, i.e. "Water", "Poison", etc.
4. `get_cp` returns the number of combat points for a pokemon.
5. `is_pokemon(obj)` returns `True` if `obj` is a pokemon.
6. `fight(p1, p2)` returns the pokemon, either `p1` or `p2` with the higher number of combat points, or `None` if neither is higher.
7. `evolve` will return a new evolved pokemon. More details given in Part(D) below.
8. `is_same` will compare two pokemons and return `True` if they are the same pokemon. If we compare a pokemon to its evolved form, `is_same` will return `True`.

Example execution:

```
>>> bulbasaur = make_pokemon("Bulbasaur", "Poison", 20)
>>> rattata = make_pokemon("Rattata", "Normal", 10)
>>> squirtle = make_pokemon("Squirtle", "Water", 34)

>>> get_name(bulbasaur)
Bulbasaur

>>> get_type(squirtle)
Water

>>> get_cp(rattata)
10

>>> winner = fight(squirtle, bulbasaur)
>>> get_name(winner)
Squirtle

>>> winner = fight(bulbasaur, squirtle)
>>> get_name(winner)
Squirtle

>>> winner = fight(bulbasaur, bulbasaur)
>>> winner
None
```

A. Decide on an implementation for the `pokemon` object and implement `make_pokemon`, `get_name`, `get_type` and `get_cp`. Describe how the state is stored in your implementation and explain. [5 marks]

Note: You are limited to using tuples for this question, i.e. you cannot use lists and other Python data structures.

B. Implement the function `is_pokemon(p)` that returns `True` if `p` is a `pokemon`, or `False` otherwise. [4 marks]

```
def is_pokemon(object):
```

C. Implement the function `fight(p1, p2)` that returns the pokemon that has the higher number of combat points or `None` if `p1` and `p2` have the same number of points. [4 marks]

```
def fight(p1,p2):
```

Pokemons can evolve to become stronger. The function `evolve` takes a pokemon, a list of possible evolutions and an increase in combat points and returns new evolved pokemon if it is possible. The list of possible evolutions is a tuple of tuples of pokemon species.

Example execution:

```
>>> bulbasaur = make_pokemon("Bulbasaur", "Poison", 20)
>>> rattata = make_pokemon("Rattata", "Normal", 10)
>>> squirtle = make_pokemon("Squirtle", "Water", 34)

>>> evolution_list = (("Bulbasaur", "Ivysaur", "Venusaur"), ("Spearow", "Fearow"), \
("Squirtle", "Wartortle", "Blastoise"), ("Rattata", "Raticate"))

>>> b2 = evolve(bulbasaur, evolution_list, 40)
>>> get_name(b2)
Ivysaur

>>> winner = fight(b2, squirtle)
>>> get_name(winner)
Ivysaur
>>> get_cp(winner)
60
```

```
>>> b3 = evolve(b2, evolution_list, 30)
>>> get_name(b3)
Venusaur
>>> get_cp(b3)
90

>>> b4 = evolve(b3, evolution_list, 40) # cannot evolve!
>>> get_name(b4)
Venusaur
>>> get_cp(b4)
90

>>> b = make_pokemon("Bulbasaur", "Poison", 20)
>>> is_same(bulbasaur, b)
False
>>> is_same(bulbasaur, b2)
True
>>> is_same(bulbasaur, b3)
True
>>> is_same(b2, b3)
True
```

D. Implement the function `evolve`

[6 marks]

```
def evolve(pokemon, elist, increase):
```


E. Implement the function `is_same`.

[4 marks]

```
def is_same(p1,p2):
```

Question 5: Tell us a Story! [3 marks]

You have been taking CS1010X for about 3 months now. Tell us what you think about the module and also share with us what you think might be an interesting story related to the class.

Appendix

The following are some functions that were introduced in class. For your reference, they are reproduced here.

```
def sum(term, a, next, b):
    if (a > b):
        return 0
    else:
        return term(a) + sum(term, next(a), next, b)

def product(term, a, next, b):
    if a > b:
        return 1
    else:
        return term(a) * product(term, next(a), next, b)

def fold(op, f, n):
    if n==0:
        return f(0)
    else:
        return op(f(n), fold(op, f, n-1))

def enumerate_interval(low, high):
    return tuple(range(low,high+1))

def filter(pred, seq):
    if seq == ():
        return ()
    elif pred(seq[0]):
        return (seq[0],) + filter(pred, seq[1:])
    else:
        return filter(pred, seq[1:])

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

Scratch Paper

— END OF PAPER —