

Re-Midterm Test

15 April 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 [25 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 if the final answer is wrong.

A.

```
x = 4
y = 5
z = 8
if y%x == 0:
    print("Having")
elif z%y == 1:
    print("a")
elif 10/(z%x) > 1:
    print("Good")
else:
    print("Time?")
```

[5 marks]

B.

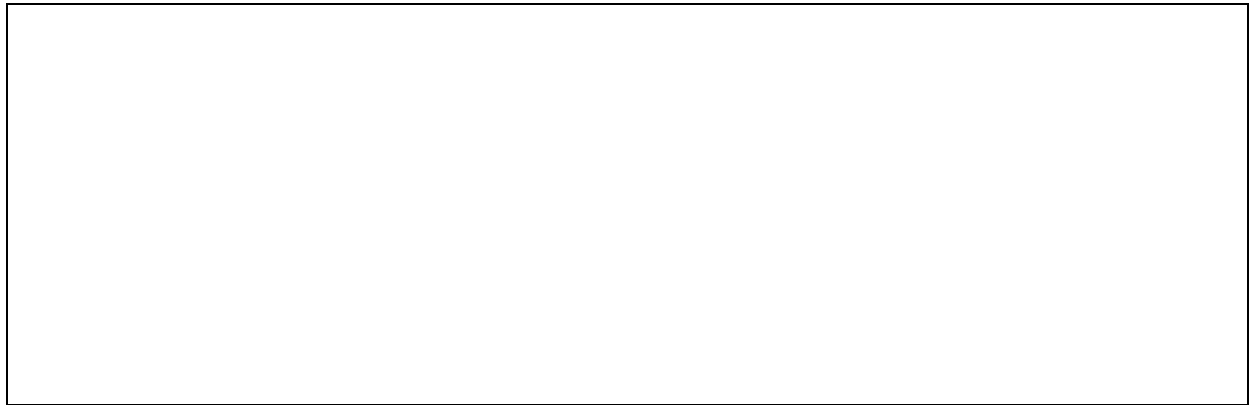
```
def x(y):
    return y**2
def y(f):
    def g(x):
        if f(x)%2 == 1:
            return f(x)+5
        else:
            return g(3*f(x)+1)
    return g(x)
def z(f):
    return lambda x: y(x)
print(z(z)(lambda x: 7))
```

[5 marks]

C.

```
result = 3
for i in range(1,9):
    if result%2 == 1:
        result = 2*result
    elif i%3 == 2:
        continue
    else:
        result += i
print(result)
```

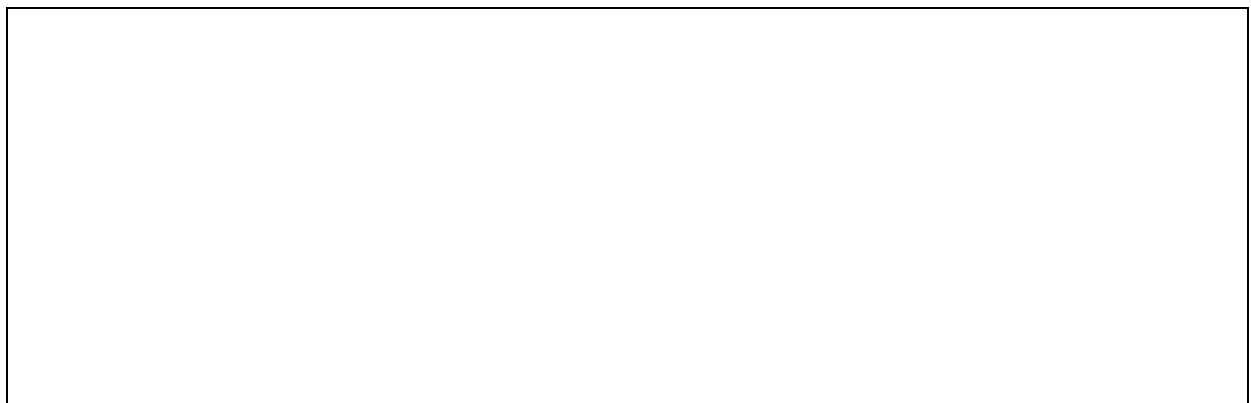
[5 marks]



D.

```
x = 9
while True:
    if x%5 == 1:
        break
    if x < 20:
        x = 2*x
    if x%2 == 0:
        x = x+1
    elif x > 20:
        x = x//2
    elif x%3 == 0:
        x = 3*x
print(x)
```

[5 marks]



E.

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

[5 marks]

Question 2: Counting fractions [25 marks]

We can compute the greatest common divisor of two integers a and b using Euclid's algorithm as follows:

$$\begin{aligned} \gcd(a, 0) &= a \\ \gcd(a, b) &= \gcd(b, a \bmod b) \end{aligned}$$

A. [Warm Up] Implement the function `gcd` that takes two integers a and b , and returns the greatest common divisor for the 2 numbers. [3 marks]

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

[Reduced Proper Fraction] Consider the fraction, $\frac{n}{d}$, where n and d are positive integers. If $n < d$ and $GCD(n, d) = 1$, it is called a reduced proper fraction.

If we list the set of reduced proper fractions for $n \leq 8$ in ascending order of size, we get:

$1/8, 1/7, 1/6, 1/5, 1/4, 2/7, 1/3, 3/8, 2/5, 3/7, 1/2, 4/7, 3/5, 5/8, 2/3, 5/7, 3/4, 4/5, 5/6, 6/7, 7/8$

It can be seen that there are 21 elements in this set.

B. Implement the function `count_fraction` that takes an integer n and returns the number of reduced proper fractions for n . [5 marks]

Example execution:

```
>>> count_fraction(1)
0
```

```
>>> count_fraction(2)
1
```

```
>>> count_fraction(8)
21
```

```
def count_fraction(n):
```

C. Assuming that the order of growth (in time) for `gcd` is $O(\log n)$, what is the order of growth in terms of time and space for the function you wrote in Part (B) in terms of n . Explain your answer. [4 marks]

Time:

Space:

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

```
def count_fraction(n):
```

E. Assuming that the order of growth (in time) for `gcd` is $O(\log n)$, what is the order of growth in terms of time and space for the function you wrote in Part (D) in terms of n . Explain your answer. [4 marks]

Time:

Space:

F. Implement the function `fraction_set(n)` that returns the set of reduced fractions for n as a tuple of tuple pairs. The order of the pairs does not matter. Any order is acceptable. [4 marks]

Example execution:

```
>>>fraction_set(1)
()

>>> fraction_set(2)
((1, 2),)

>>> fraction_set(3)
((1, 3), (2, 3), (1, 2))

>>>fraction_set(8)
((1, 8), (3, 8), (5, 8), (7, 8), (1, 7), (2, 7), (3, 7), (4, 7), (5, 7),
(6, 7), (1, 6), (5, 6), (1, 5), (2, 5), (3, 5), (4, 5), (1, 4), (3, 4),
(1, 3), (2, 3), (1, 2))
```

```
def fraction_set(n):
```


Question 3: Higher Order Functions [24 marks]

Consider the following higher order function `budge`:

```
def budge(tup, combiner, base, pred):
    if pred(tup):
        return base(tup)
    else:
        return combiner(tup[0], budge(tup[1:], combiner, base, pred))
```

where `tup` is a tuple.

A. We note that we can implement `length` which returns the length of a tuple `tup` as follows:

```
def length(tup):
    return budge(tup, <t1>, <t2>, <t3>)
```

Example execution:

```
>>> length((1, 2, 3, 4))
4
```

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

```
>>> length((1, (2, 1, 3), (3, 4), 5, 6))
5
```

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

[6 marks]

<T1>:
[2 marks]

--

<T2>:
[2 marks]

--

<T3>:
[2 marks]

--

B. We note that we can implement `odd` that returns a tuple with only the odd numbers in a tuple `tup` as follows:

```
def odd(tup):  
    return budge(tup, <t4>, <t5>, <t6>)
```

Example execution:

```
>>> odd((1, 2, 3, 4, 5, 6, 7, 8, 9, 10))  
(1, 3, 5, 7, 9)
```

```
>>> odd((34, 36, 38))  
()
```

```
>>> odd((5, 77, 91, 93))  
(5, 77, 91, 93)
```

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

[6 marks]

<T4>:
[2 marks]

--

<T5>:
[2 marks]

--

<T6>:
[2 marks]

--

C. We note that we can implement the reverse of a tuple as follows:

```
def reverse(tup):  
    return budge(tup, <t7>, <t8>, <t9>)
```

Example execution:

```
>>> reverse((1, 2, 3, 4))  
(4, 3, 2, 1)
```

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

```
>>> reverse((1, (2, 1, 3), (4, 5), 6))  
(6, (4, 5), (2, 1, 3), 1)
```

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

[6 marks]

<T7>:
[2 marks]

--

<T8>:
[2 marks]

--

<T9>:
[2 marks]

--

D. We note that we can implement the deep reverse of a tuple as follows:

```
def deep_reverse(tup):  
    if type(tup) != tuple:  
        return tup  
    return budge(tup, <t10>, <t11>, <t12>)
```

Example execution:

```
>>> deep_reverse((1,2,3,4))  
(4, 3, 2, 1)  
  
>>> deep_reverse((1, (2,1,3), (1,3), 4))  
(4, (3, 1), (3, 1, 2), 1)  
  
>>> deep_reverse(((4,3),))  
((3, 4),)
```

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

[6 marks]

<T10>:
[2 marks]

<T11>:
[2 marks]

<T12>:
[2 marks]

Question 4: Piggybank LOL! [23 marks]

Your younger sibling wants to learn how to save money. You have been asked to implement a virtual piggy bank with the following functions:

1. `make_piggy` takes in no parameters and returns a new empty piggybank.
2. `add_coin(piggy, x)` returns a new piggy bank with a coin of denomination x added to it.
3. `remove_coin(piggy, x)` returns a new piggy bank with a coin of denomination x taken out from it. If the piggy bank does not contain any coin of denomination x , this just returns a piggy bank equivalent to the original piggy bank.
4. `count(piggy, x)` returns the number of coins of denomination x in the piggybank.
5. `get_coins(piggy)` returns a tuple of the denominations of the coins in the piggy bank.

Example execution:

```
>>> p = make_piggy()
>>> p2 = add_coin(p, 10)
>>> p3 = add_coin(p2, 5)
>>> p4 = add_coin(p3, 10)

>>> get_coins(p4)
(5, 10)

>>> count(p4, 10)
2

>>> p5 = add_coin(p4, 10)
>>> p6 = add_coin(p5, 20)
>>> p7 = add_coin(p6, 50)
>>> get_coins(p7)
(5, 10, 20, 50)

>>> count(p7, 10)
3

>>> count(p7, 5)
1

>>> count(p7, 50)
1
```

A. Decide on an implementation for the piggy bank object and implement `make_piggy`. Describe how the state is stored in your implementation when the piggy is not empty. [4 marks]

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

```
def make_piggy():
```

B. Implement the function `add_coin(piggy, x)`.

[4 marks]

```
def add_coin(piggy, x):
```

C. Implement the function `remove_coin(piggy, x)`.

[4 marks]

```
def remove_coin(piggy, x):
```

D. Implement the function `count(piggy, x)` that returns the number of coins of denomination x in the piggy bank.

[3 marks]

```
def count(piggy, x):
```

E. Implement the function `get_coins(piggy)` that returns a tuple of the denominations of coins that are in the piggy bank. The order of the coin denominations in the tuple is not important. [3 marks]

```
def get_coins(piggy):
```


F. [Count Change Piggy Style] Implement the function `count_change(a, piggy)` that takes an amount of money a and returns the number of ways to make the change based on the coins in the piggy. [5 marks]

Example execution:

Given p7 from above

```
>>> count_change(100, p7)
1
```

```
>>> count_change(99, p7)
0
```

```
>>> count_change(80, p7)
2
```

```
>>> count_change(30, p7)
2
```

```
>>> count_change(5, p7)
1
```

```
def count_change(a, piggy):
```

Question 5: What did you learn at the Midterm or since the Midterm [3 marks]

Tell us what you have learnt at the Midterm and what you have done or will do for the rest of the module in order to do well for this modules.

Appendix

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

```
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):
    if kinds_of_coins == 1:
        return 1
    elif kinds_of_coins == 2:
        return 5
    elif kinds_of_coins == 3:
        return 10
    elif kinds_of_coins == 4:
        return 20
    elif kinds_of_coins == 5:
        return 50

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