

Mid-Term Test

4 March 2015

Time allowed: 1 hour 45 minutes

Matriculation No:

--	--	--	--	--	--	--	--	--

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 **FOUR (4) questions** and **SEVENTEEN (17) 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		
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. Partial marks will be awarded for workings if the final answer is wrong.

A. `x = 10` [5 marks]

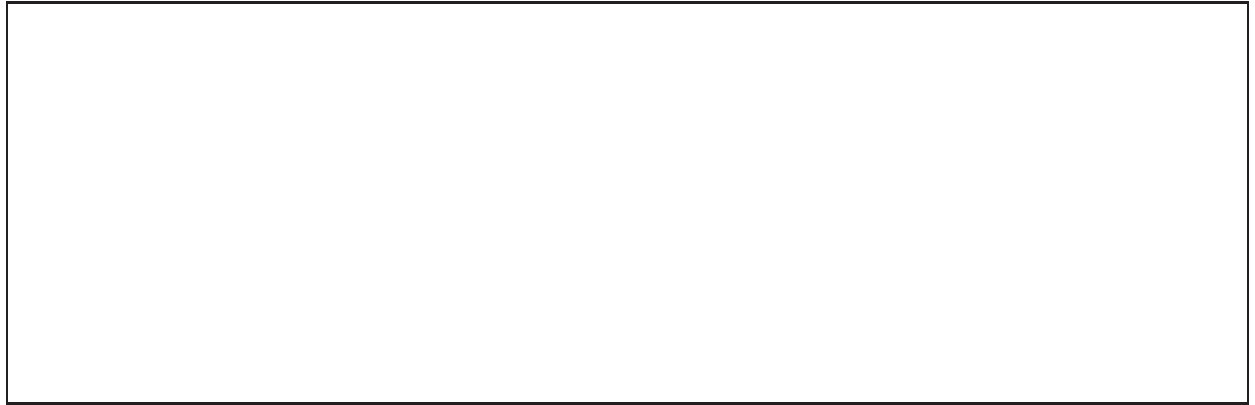
```
y = 5
def f(x):
    return x + y
def g(y):
    return x - y
print(f(g(x // y)))
```

B. `a = 12345678` [5 marks]

```
b = 1
while b < a:
    b = b * 10
    a = a // 10
print(a + b)
```

C. `x = 42` [5 marks]

```
y = 24
if y < x:
    if x > y:
        print("One")
    else:
        print("Two")
else:
    print("Three")
```



D. `def foo(x, y):` [5 marks]

```
    return (x,) + y
print(foo((1, (2, 3)), foo((4), (5))))
```



E. `def bar(x):` [5 marks]
 `return lambda n: n(x)`
 `print(bar(bar(2)(lambda x:x+1))(lambda x:x**2))`

F. `z = 1` [5 marks]
 `for i in range(-5, 5):`
 `if i % 2 == 0:`
 `continue`
 `if i == 0:`
 `break`
 `z = z * i`
 `print(z)`

Question 2: Funtorials!!! [24 marks]

Funtorials are factorials with a twist. Instead of simply multiplying all integers from 1 to n , a funtorial applies a modifier to each integer in the series.

For example, a factorial called *GCD-fun* specifies that the i th integer in the sequence is the the greatest-common-divisor of i and n , where n is the last term. In other words,

$$gcd_fun(n) = gcd(1, n) \times gcd(2, n) \times \dots \times gcd(n, n)$$

where $gcd(a, b)$ returns the greatest-common-divisor of a and b . Note that $gcd(0, x) = x$.

Suppose you are provided a function `gcd(a, b)` that returns the greatest-common-divisor of a and b .

A. Using the function `gcd`, write an **iterative** function `gcd_fun(n)` that returns the funtorial value for n . [4 marks]

```
def gcd_fun(n):
```

B. Suppose the order of growth of $gcd(a, b)$ is $O(1)$ for both time and space. What is the order of growth in terms of time and space for the function you wrote in Part (A) in terms of n . Explain your answer. [4 marks]

Time:

Space:

C. Using the function `gcd`, write an recursive function `gcd_fun(n)` that returns the factorial value for n . Explain your answer. [4 marks]

```
def gcd_fun(n):
```

D. Suppose the order of growth of `gcd(a, b)` is $O(1)$ for both time and space. What is the order of growth in terms of time and space for the function you wrote in Part (C) in terms of n . Explain your answer. [4 marks]

Time:

Space:

E. Now, suppose that the order of growth of $\text{gcd}(a, b)$ is $O(a)$ if $a \geq b$ and $O(b)$ if $b \geq a$ for both time and space.

What is the order of growth in terms of time and space of either the iterative or recursive function that you wrote in Part (A) or (C) in terms of n ? State **clearly** which version you have chosen and explain your answer. [4 marks]

Time:

Space:

F. The gcd function presented in the lecture uses Euclid's algorithm to compute the *GCD* of a and b as follows:

```
def gcd(a, b):  
    if b == 0:  
        return a  
    else:  
        return gcd(b, a % b)
```

Write an iterative version of the gcd function.

[4 marks]

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

Question 3: Higher-Order Funtorials [22 marks]

`make_funtorial` is a function that takes as input a funtorial modifier and returns a particular funtorial. For example, `gcd_fun` can be defined as such:

```
gcd_fun = make_funtorial(gcd)
```

Now with `make_funtorial`, it will be easy to create new funtorials. `factor_funtorial` is a funtorial that only multiplies the factors of the last term n . In other words,

$$factor_fun(n) = f_1 \times f_2 \times \dots \times f_i \text{ where } f_1 \dots f_i \text{ are all the factors of } n$$

Some examples are:

$$factor_fun(6) = 1 \times 2 \times 3 \times 6$$

$$factor_fun(11) = 1 \times 11$$

$$factor_fun(15) = 1 \times 3 \times 5 \times 15$$

$$factor_fun(24) = 1 \times 2 \times 3 \times 4 \times 6 \times 8 \times 12 \times 24$$

`factor_fun` can be defined as:

```
factor_fun = make_funtorial(factor)
```

where $factor_fun(n) = factor(1,n) \times factor(2,n) \times \dots \times factor(n,n)$

A. Please provide a possible implementation for `factor`, and state any assumptions that you need. [4 marks]

B. It turns out we can express `make_functorial` in terms of `product` (see Appendix) as follows:

```
def make_functorial(fun):
    def helper(n):
        <PRE>
        return product(<T1>, <T2>, <T3>, <T4>)
    return helper
```

Please provide possible implementations for the terms T1, T2, T3 and T4. You may also optionally define other helper functions in <PRE> if needed. [6 marks]

*optional
<PRE>:

<T1>:

<T2>:

<T3>:

<T4>:

C. We can also express `make_funtorial` in terms of `fold` (see Appendix) as follows:

```
def make_funtorial(fun):
    def helper(n):
        <PRE>
        return fold(<T5>, <T6>, <T7>)
    return helper
```

Please provide possible implementations for the terms `T5`, `T6`, and `T7`. You may also optionally define other helper functions in `<PRE>` if needed. [6 marks]

*optional
<PRE>:

<T5>:

<T6>:

<T7>:

D. While a regular functorial applies a single function f to every term in the series, a double functorial applies a function f to the odd terms and a function g to the even terms in the series. In other words,

$$f(1,n) \times g(2,n) \times f(3,n) \times g(4,n) \times \dots \times \begin{cases} f(n,n), & \text{if } n \text{ is odd} \\ g(n,n), & \text{if } n \text{ is even} \end{cases}$$

The function `make_double_functorial(f, g)` takes in the two functions f and g and produces a functorial of f, g . For example:

```
gcd_factor_fun = make_double_functorial(gcd, factor)
gcd_factor_fun(6) = gcd(1,6) × factor(2,6) × gcd(3,6) × factor(4,6) × gcd(5,6) × factor(6,6)
                  = 1 × 2 × 3 × 1 × 1 × 6
```

Provide an implementation of `make_double_functorial` **using either the product or fold function**. [6 marks]

```
def make_double_functorial(f, g):
```

Question 4: Fly Genetics [24 marks]

Warning: Please read the question description clearly before you attempt this problem!!

The common fruit flies (*Drosophila melanogaster*) is widely used for biological research in the study of genetics. You have been tasked by the Genome Institute of Singapore (GIS) to simulate the genetic interactions on a computer.

The GIS is interested in modeling the sex-linked traits, in particular the gene responsible for the eye-colour of the flies. A fruit fly has two sex-genes which determines its gender. For our purposes, it is sufficient to model three different types of genes: “X”, “w” and “Y”.

The function `make_fly` takes in as inputs, two genes and returns a fruit fly. The functions `gene1` and `gene2` takes in a fruit fly as the input and returns the first and second gene respectively.

Example:

```
>>> male_fly = make_fly('w', 'Y')
>>> gene1(male_fly)
'w'

>>> gene2(male_fly)
'Y'

>>> female_fly = make_fly('X', 'w')
>>> gene1(female_fly)
'X'

>>> gene2(female_fly)
'w'
```

A. Explain how you will use a tuple to represent the state of a fly. [2 marks]

B. Provide an implementation for the functions `make_fly`, `gene1` and `gene2`. [6 marks]

```
def make_fly(gene1, gene2):
```

```
def gene1(fly):
```

```
def gene2(fly):
```

C. The gender and eye-colour of a fruit fly are governed by the following rules:

Gender: A fruit fly is male if one of its genes is “Y”, and female if it has no “Y” gene. It is not possible for a fly to have two “Y” genes.

Eye-colour: The colour of the eyes of a fruit fly is white if it does not have any “X” gene, i.e., a male fly with white eyes will have a pair of “wY” genes and a female fly with white eyes will have a pair of “ww” genes. Otherwise, the eyes will be red.

The function `is_male` takes in a fruit fly as its input and returns `True` if the fly is a male and `False` if it is a female.

Similarly, the function `eye_colour` takes in a fruit fly as its input and returns either the colour of the fly’s eyes as a string “red” or “white”.

Example:

```
>>> is_male(male_fly)
True
```

```
>>> is_male(female_fly)
False
```

```
>>> eye_colour(male_fly)
'white'
```

```
>>> eye_color(female_fly)
'red'
```

D. Provide an implementation of the functions `is_male` and `eye_colour`, [4 marks]

```
def is_male(fly):

def eye_colour(fly):
```

E. In order to observe genetics traits, there is a need to breed the fruit flies. The function `breed` takes in as inputs two fruit flies, one male and one female, and returns a new fruit fly. The new fruit fly inherits (copies) exactly one gene from each parent. Since each parent has two genes, there is a 50% chance for any one to be copied.

Example:

```
>>> offspring = breed(male_fly, female_fly)
>>> print(gene1(offspring) + gene2(offspring))
wY

>>> breed(offspring, male_fly)
None
```

Provide an implementation of `breed` that returns a new fly if the inputs are a male and female fly. `breed` returns `None` if the inputs are both male or both female. Hint: the `random()` function returns a random float between 0 and 1. [6 marks]

```
def breed(fly1, fly2):
```

F. Oftentimes, scientists will need to breed several offsprings from two flies. The offsprings from a set of parents are said to belong to the same generation.

Write a function `breed_generation(fly1, fly2, n)` which takes in a male and female fly, and returns a tuple of n offsprings. [6 marks]

```
def breed_generation(fly1, fly2, n):
```

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 —