

# Midterm Test

6 Mar 2019

**Time allowed:** 1 hour 30 minutes

## Instructions (please read carefully):

1. This is an **open-sheet test**. You are allowed to bring one A4 sheet of notes (written or printed on both sides).
2. The QUESTION SET comprises **FOUR (4) questions** and **TEN (10) pages**, and the ANSWER SHEET comprises of **TEN (10) pages**.
3. The time allowed for solving this test is **1 hour 30 minutes**.
4. The maximum score of this test is **75 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. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
8. Use of calculators are not allowed in the test.
9. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
10. **Marks may be deducted** for i) unrecognisable handwriting, and/or ii) excessively long code. A general guide would be not more than twice the length of our model answers.

# GOOD LUCK!

This page is intentionally left blank.

It may be used as scratch paper.

## 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, **state and explain why**. You may show your workings **outside the answer box**. Partial marks may be awarded for workings even if the final answer is wrong.

The code is replicated on the answer booklet. You may show your workings **outside the answer box** in the space beside the code. Partial marks will be awarded for workings if the final answer is wrong.

**A.** `x = 5`  
`y = 3`  
`def f(x, y):`  
 `if x > y:`  
 `return f(x//2, y*2)`  
 `if x < y:`  
 `return f(x+1, y-1)`  
 `return x+y`  
`print(f(x, y))`  
`print(f(y, x))`

[5 marks]

**D.** `i = 7`  
`for i in range(10):`  
 `if i % 5 > 3:`  
 `break`  
 `if i % 2 == 0:`  
 `i = i-1`  
 `continue`  
 `i *= 2`  
 `print(i)`  
`print(i)`

[5 marks]

**B.** `a = ()`  
`b = (1, a)`  
`a = a + (1,)`  
`c = b + a`  
`print(c)`

[5 marks]

**E.** `def foo(x):`  
 `return lambda a: a(x)(a)`  
`def bar(x):`  
 `return lambda a: a(x+1)`  
`print(foo(5)(bar)(lambda x:x))`

[5 marks]

**C.** `s = "papa"`  
`t = "mama"`  
`if s < t:`  
 `s[0] = t[0]`  
`out = (s*2)[:2] + 2*t[-2:]`  
`print(out)`

[5 marks]

## Question 2: Bacterium [18 marks]

Dr. Leeuwenhoek is doing some research on a particular strain of bacteria. He has discovered that by adjusting the formula of the nutrient broth, he can control the daily increase of the population to a constant percentage. In addition, he is also testing an antibiotic that kills a constant number of bacteria every day, depending on the dosage.

For example, suppose on day 0 we start with an initial population of 1,000 bacteria. The dosage of antibiotic kills 200 bacteria daily, while the nutrient broth is adjusted to produce 10% growth daily. The table below summarizes the population of the colony over a few days:

Day	Population	Deaths	Growth <sup>1</sup>
0	1,000	200	80
1	880	200	68
2	748	200	54
3	602	200	40
4	442	200	24
5	266	200	6
6	72	72	0
7	0	0	0

At the end of each day, the antibiotic first kills off some bacteria and those left alive will reproduce at the given rate (with the resulting population rounded down to the nearest integer). For the above example, we can see that the entire bacteria colony has been killed on day 7.

The function `days_to_kill` takes as inputs the initial population of the bacteria colony, the number of bacteria the antibiotic kills each day, and the daily growth rate of the colony. It returns the number of days it takes to completely kill the colony.

For the above example, the function call `days_to_kill(1000, 200, 0.1)` will return 7.

If the antibiotic kills too few or the growth rate is too high, the colony might never die. However, you may ignore this and assume that the parameters chosen always result in the colony eventually dying.

- A. Provide a recursive implementation of the function `days_to_kill`. [4 marks]
- B. State the order of growth in terms of time and space for the function you wrote in Part (A) **when the input growth rate is 0**. Briefly explain your answer. [2 marks]
- C. Provide a iterative implementation of the function `days_to_kill`. [4 marks]
- D. State the order of growth in terms of time and space for the function you wrote in Part (C) **when the input growth rate is 0**. Briefly explain your answer. [2 marks]

<sup>1</sup>Rounded down to the nearest integer

**E.** Now suppose Leeuwenhoek found an antibiotic that kills bacterial in intervals of every  $n$  days. That is, it allows the bacteria population to grow, and only kills on the  $n^{th}$  day.

For example, if the antibiotic kills 700 bacteria every 3 days. The table below shows the state of the population beginning with 1,000 bacteria that grows 10% daily.

Day	Population	Deaths	Growth
0	1000	0	100
1	1100	0	110
2	1210	700	51
3	561	0	56
4	617	0	61
5	678	678	0
6	0	0	0

It took 6 days for the colony to be killed.

The function `new_days_to_kill` takes in the same inputs as `days_to_kill` along with an addition input  $n$ , which is the interval of which the bacteria is killed.

For example, `new_days_to_kill(1000, 700, 0.1, 3)` will return 6.

Provide an implementation for the function `new_days_to_kill`. [4 marks]

**F.** Is your implementation in Part (E) recursive or iterative? State the order of growth in terms of time and space of your implementation **when the input growth rate is 0**. Briefly explain your answer. [2 marks]

### Question 3: Higher-Order Bacteria [10 marks]

**A. [Warning: Hard]** Consider the higher-order function `sum` which was taught in class.

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

The function `days_to_kill` in Question 2 can be defined in terms of `sum` as follows:

```
def days_to_kill(pop, kill, growth):
    <PRE>
    return sum(<T1>, <T2>, <T3>, <T4>)
```

Please provide possible implementations for the terms `T1`, `T2`, `T3`, and `T4`. You may optionally define other functions in `PRE` if needed.

Note you are to use the higher-order function and not solve it recursively or iteratively.

[4 marks]

**B.** Now it is possible that the growth of the bacteria population and the number of bacteria killed by the antibiotic is dependent on the current population size. The antibiotic might also become less effective over time.

As such, it would be better to have the inputs `kill` and `growth` of function `days_to_kill` be functions that return the number of bacteria to add or subtract from the current population.

Provide an implementation of the function `hof_days_to_kill`, which is `days_to_kill` with the modified inputs stated above. You are free to propose a reasonable signature for the inputs.

[4 marks]

**C.** Suppose we want to replicate the example in Q2E: `new_days_to_kill(1000, 700, 0.1, 3)`, where the population grows 10% daily and the antibiotics kills 700 every third day. This new `hof_days_to_kill` function defined in 3B will be used as such:

```
>>> hof_days_to_kill(1000, k, g)
```

Provide a definition of the inputs `k` and `g` that will satisfy the above requirements. [2 marks]

## Question 4: Quantum of Bacterium [22 marks]

**INSTRUCTIONS:** Please read the entire question clearly before you attempt this problem!! You are also not to use any Python data types which have not yet been taught in class.

Dr. Leeuwenhoek thinks there is something strange about the bacteria he is studying. You see, bacteria reproduce by fission, that is they clone identical copies of themselves, and each new “clone” is virtually indistinguishable from the “parent”. However, with this particular bacteria, it seems each new “clone” has some attachment with its “parent” that cannot be seen. Something like a quantum entanglement.

Since this cannot be observed in the lab, Leeuwenhoek wishes to model the bacteria in a computer. Every *bacteria* will need to contain a reference to its “*parent*”, as well as a *generation counter* which is an integer that indicates how many generations removed it is from the original bacteria. And as usual, each bacteria will need to contain a DNA, which is modeled as an abstract data type.

Since you are the new research assistant, this task naturally falls to you. Fortunately (or unfortunately) you are taking a class in Python and have just learned how to use tuples. You are to design a data structure **using tuples** to represent a bacteria, and it is to be supported by the following functions

- `new_bacteria(dna)` takes in a dna data type, and returns a new bacteria of generation 0 that contains this dna.
- `generation(bacteria)` takes in an bacteria, and returns the *generation counter* of the bacteria.
- `parent(bacteria)` takes in a bacteria, and returns the parent of the bacteria. If the bacteria has no parent, i.e. its generation is 0, then the function returns None.
- `dna(bacteria)` takes in an bacteria, and returns the *DNA* of the bacteria.

In addition to the constructor and accessor functions, the function `divide(bacteria)` simulates the reproduction of a new bacteria. The function takes a bacteria as input, and returns a new bacteria with the same DNA as the parent bacteria and a generation counter that is 1 more than its parent.

Consider the following sample run:

```
>>> b0 = new_bacteria(d) # assume d have been defined to be a particular DNA
>>> b1 = divide(b0)
>>> b2 = divide(b0)
>>> b3 = divide(b2)
```

**A.** Draw the **box-pointer diagram** of `b0`, `b1`, `b2` and `b3` at the end of the sample run above. Your diagram should be consistent with your implementation of a bacteria in part (B).

[2 marks]

**B.** Provide an implementation of the functions `new_bacteria`, `generation`, `parent`, `dna` and `divide`. [6 marks]

**[Important!]** For the remaining parts of this question, **you should not break the abstraction of a *bacteria* in your code.**

Now we can model a colony of bacteria as a tuple, i.e. a tuple of bacteria type.

**C.** The function `grow` takes as inputs a colony and a probability  $p$ , where  $0 \leq p \leq 1$ . Each bacteria in the colony will divide to produce an “offspring” with probability  $p$ . The function returns a new colony containing all new offsprings along with all existing bacteria from the original colony.

Provide an implementation of the function `grow`.

Hint: the function `random()` from the `random` package returns a random float  $n$  between 0 (inclusive) and 1. [4 marks]

**D.** The function `kill` takes as inputs a colony and an integer  $n$ . It returns a new colony with  $n$  random bacteria removed from the original colony.

Provide an implementation for the function `kill`, using only the function `randint` to perform the random selection. You may assume that  $n$  will not be larger than the number of bacteria in the colony.

Hint: the function `randint(a, b)` from the `random` package returns a random integer in the range  $a$  to  $b$ , inclusive of both  $a$  and  $b$ . [4 marks]

**E.** In a certain experiment, bacteria with different DNA are added to a colony and the colony undergoes several grow and kill cycles. After a few cycles, researchers wish to know how many bacteria containing a certain DNA are left in the colony.

Implement a function `count_dna` that takes as input a DNA, and a colony, and returns the number of bacteria in the colony. that has the given DNA. [4 marks]

**F.** Testing out a possible implementation, Dr. Leeuwenhoek tries to create a colony using the bacteria created in the sample run shown earlier and got the following result:

```
>>> colony = (b0, b1, b3)
>>> parent(b3) in colony
True
>>> parent(parent(b3)) in colony
True
```

Something is wrong. The parent of  $b3$  is  $b2$ , and  $b2$  is not in the colony, yet the statement returns True. Please explain why. [2 marks]

— END OF QUESTIONS —



## 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 map(fn, seq):
    if seq == ():
        return ()
    else:
        return (fn(seq[0]),) + map(fn, seq[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 —

# Midterm Test — Answer Sheet

6 Mar 2019

**Time allowed:** 1 hour 30 minutes

**Student No:**

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

## Instructions (please read carefully):

1. Write down your **student number** on this answer sheet. **DO NOT WRITE YOUR NAME!**
2. This answer sheet comprises **TEN (10) pages**.
3. All questions must be answered in the space provided; no extra sheets will be accepted as answers. You may use the extra page at the back if you need more space for your answers.
4. You must submit only the **ANSWER SHEET** and no other documents. The question set may be used as scratch paper.
5. You are allowed to use pencils, ball-pens or fountain pens, as you like as long as it is legible (no red color, please).
6. **Marks may be deducted** for i) unrecognisable handwriting, and/or ii) excessively long code. A general guide would be not more than twice the length of our model answers.

# GOOD LUCK!

## For Examiner's Use Only

Question	Marks	Remarks
Q1	/ 25	
Q2	/ 18	
Q3	/ 10	
Q4	/ 22	
<b>Total</b>	<b>/ 75</b>	

This page is intentionally left blank.

Use it **ONLY** if you need extra space for your answers, in which case indicate the **question number clearly** as well as in the original answer box. **DO NOT** use this for your rough work.

**Question 1A**

[5 marks]

```
x = 5
y = 3
def f(x, y):
    if x > y:
        return f(x//2, y*2)
    if x < y:
        return f(x+1, y-1)
    return x+y
print(f(x, y))
print(f(y, x))
```

**Question 1B**

[5 marks]

```
a = ()
b = (1, a)
a = a + (1,)
c = b + a
print(c)
```

**Question 1C**

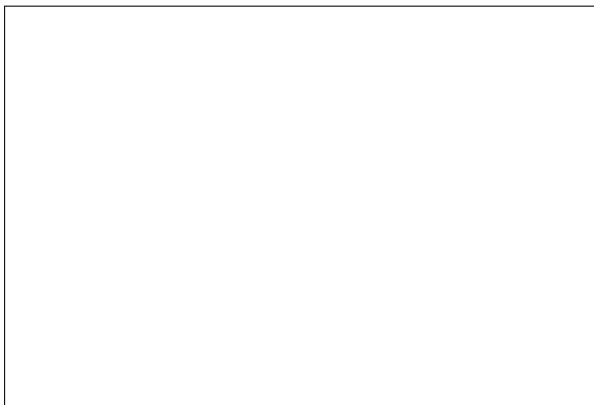
[5 marks]

```
s = "papa"
t = "mama"
if s < t:
    s[0] = t[0]
out = (s*2)[:2] + 2*t[-2:]
print(out)
```

**Question 1D**

[5 marks]

```
i = 7
for i in range(10):
    if i % 5 > 3:
        break
    if i % 2 == 0:
        i = i-1
        continue
    i *= 2
    print(i)
print(i)
```

**Question 1E**

[5 marks]

```
def foo(x):
    return lambda a: a(x)(a)
def bar(x):
    return lambda a: a(x+1)
print(foo(5)(bar)(lambda x:x))
```



**Question 2A** Provide a recursive implementation

[4 marks]

```
def days_to_kill(pop, kill, growth):
```

**Question 2B**

[2 marks]

Time:

Space:

**Question 2C** Provide an iterative implementation

[4 marks]

```
def days_to_kill(pop, kill, growth):
```

**Question 2D**

[2 marks]

Time:

Space:

**Question 2E**

[4 marks]

```
def new_days_to_kill(pop, kill, growth, interval):
```

**Question 2F**

[2 marks]

My implementation is **RECURSIVE / ITERATIVE** (circle/delete accordingly)

Time:

Space:



**Question 3A**

[4 marks]

\*optional  
<PRE>:

&lt;T1&gt;:

&lt;T2&gt;:

&lt;T3&gt;:

&lt;T4&gt;:

**Question 3B**

[4 marks]

```
def hof_days_to_kill(pop, kill, growth):
```

**Question 3C** Provide a definition of the inputs `k` and `g` [2 marks]

**Question 4A** Draw the pointer diagram of `b0`, `b1`, `b2` and `b3` [2 marks]

[6 marks]

```
from random import random
def grow(colony, p):
```

[4 marks]

```
from random import random
def grow(colony, p):
```

**Question 4D**

[4 marks]

```
from random import randint
def kill(colony, num):
```

**Question 4E**

[4 marks]

```
def count_dna(dna, colony):
```

**Question 4F** Explain why the output returns True

[2 marks]

— END OF ANSWER SHEET —

**Question 1A**

[5 marks]

```
x = 5
y = 3
def f(x, y):
    if x > y:
        return f(x//2, y*2)
    if x < y:
        return f(x+1, y-1)
    return x+y
print(f(x, y))
print(f(y, x))
```

```
8
8
```

**Question 1B**

[5 marks]

```
a = ()
b = (1, a)
a = a + (1,)
c = b + a
print(c)
```

```
(1, (), 1)
```

 Tests understanding of tuple addition.**Question 1C**

[5 marks]

```
s = "papa"
t = "mama"
if s < t:
    s[0] = t[0]
out = (s*2)[:2] + 2*t[-2:]
print(out)
```

```
'pamama'
```

**Question 1D**

[5 marks]

```
i = 7
for i in range(10):
    if i % 5 > 3:
        break
    if i % 2 == 0:
        i = i-1
        continue
    i *= 2
    print(i)
print(i)
```

2  
6  
4

Tests the understanding of **while** loops, and **break** and **continue**

**Question 1E**

[5 marks]

```
def foo(x):
    return lambda a: a(x)(a)
def bar(x):
    return lambda a: a(x+1)
print(foo(5)(bar)(lambda x:x))
```

7

**Question 2A** Provide a recursive implementation

[4 marks]

```
def days_to_kill(pop, kill, growth):  
    if pop <= 0:  
        return 0  
    else:  
        return 1 + days_to_kill(int((pop - kill)*(1+growth)), kill, growth)
```

**Question 2B**

[2 marks]

Time:  $O(n)$ , where  $n$  the return value of the function, or  $n = \frac{pop}{kill}$ . Both *pop* and *kill* are inputs so the order of growth depends on both. There is a total of  $n$  recursive calls.

Space:  $O(n)$ , where  $n$  the return value of the function, or  $n = \frac{pop}{kill}$ . There is a total of  $n$  recursive calls each creating a new environment on the stack, which takes up space.

**Question 2C** Provide an iterative implementation

[4 marks]

```
def days_to_kill(pop, kill, growth):  
    days = 0  
    while pop > 0:  
        pop = int((pop - kill)*(1+growth))  
        days += 1  
    return days
```

**Question 2D**

[2 marks]

Time:  $O(n)$ , where  $n$  the return value of the function, or  $n = \frac{pop}{kill}$ . The loop will iterate  $n$  times as  $kill$  is repeatedly subtracted from  $pop$  until it is  $pop \leq 0$ .

Space:  $O(1)$ , no extra memory is needed because the variables are overwritten with the new values.

**Question 2E**

[4 marks]

```
def new_days_to_kill(pop, kill, growth, interval): # recursive
    if pop <= 0:
        return 0
    if i % interval == interval-1:
        pop -= kill
    return 1 + new_days_to_kill(int(pop*(1+growth)), kill, growth, interval)

def new_days_to_kill(pop, kill, growth, interval): # iterative
    i = 0
    while pop > 0:
        if i % interval == interval-1:
            pop -= kill
        pop = int(pop*(1+growth))
        i += 1
    return i
```

**Question 2F**

[2 marks]

Exact answer will depend on the actual code written. If code is incomplete, or has glaring errors, e.g. infinite loop, obviously different algorithm, this part will be awarded zero marks.

Time: will be similar as part B and D,  $O(n)$  where  $n = \frac{interval \times pop}{kill}$ .

Space:

$O(n)$  for recursive and  $O(1)$  for iterative. Explanation is the same as part B/D.



**Question 3A**

[4 marks]

\*optional  
<PRE>:

<T1>: **lambda** x: 1

<T2>: -pop

<T3>: **lambda** x: **int**((x+kill)\*(1+growth))

<T4>: -1

**Question 3B**

[4 marks]

```
def hof_days_to_kill(pop, kill, growth):  
    days = 0  
    while pop > 0:  
        pop -= kill(pop, days)  
        pop += growth(pop)  
        days += 1  
    return days
```

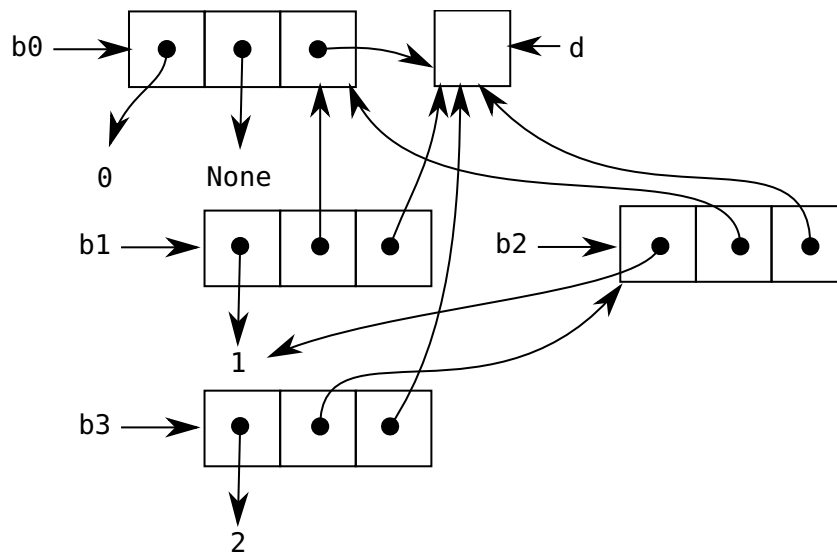
**Question 3C** Provide a definition of the inputs `k` and `g` [2 marks]

```
g = lambda p: int(0.1 * p)
```

```
def k(pop, day):
    if day % 3 == 2:
        return 700
    else:
        return 0
```

**Question 4A** Draw the pointer diagram of `b0`, `b1`, `b2` and `b3` [2 marks]

One possible implementation is this:



Other representations are also valid as long as it works with the implemented functions. Note that DNA is a “black-box” and we do not know anything about its implementation.

-1 mark if the two parent of `b1` and `b2` do not point to the same tuple. -1 mark if student assumes some implementation of dna

**Question 4B**

[6 marks]

```
def new_bacteria(dna):  
    return return (0, None, dna)  
  
def generation(bacteria):  
    return bacteria[0]  
  
def parent(bacteria):  
    return bacteria[1]  
  
def dna(bacteria):  
    return bacteria[2]  
  
def divide(bacteria):  
    return (generation(bacteria)+1, bacteria, dna(bacteria))
```

**Question 4C**

[4 marks]

```
from random import random  
def grow(colony, p):  
    for bacteria in colony:  
        if random() < p:  
            colony += (divide(bacteria),)  
    return colony
```

**Question 4D**

[4 marks]

```
from random import randint
def kill(colony, num):
    for i in range(num):
        k = randint(0, len(colony)-1)
        colony = colony[:k] + colony[k+1:]
    return colony
```

**Question 4E**

[4 marks]

```
def count_dna(dna, colony):
    return len(tuple(filter(lambda x: x == dna, map(dna, colony))))
```

**Question 4F** Explain why the output returns True

[2 marks]

This is because b1 and b2 have the same contents, and being represented as tuples, they are equivalent. And since the in operator uses the equivalence test, it returns True.