

NATIONAL UNIVERSITY OF SINGAPORE

Semester 1, 2014/2015

CS1010S - PROGRAMMING METHODOLOGY

Time Allowed: 2 Hours

INSTRUCTIONS TO CANDIDATES

1. The assessment paper contains **FIVE (5) questions** and comprises **TWENTY-TWO (22) pages**.
2. Weightage of questions is given in square brackets. The maximum attainable score is 100.
3. This is a **CLOSED** book assessment, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
4. Write all your answers in the space provided in this booklet.
5. **Please write your student number below.**

STUDENT NUMBER: _____

(this portion is for the examiner's use only)

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
Total		

Question 1: Warm Up [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. If the interpreter produces an error message, or enters an infinite loop, explain why.

A.

```
x = [1,2,3]
if x is [1,2,3]:
    x = [x] + [4,5]
if len(x)<=3:
    y = [x] + [6,7]
elif len(x)<=5:
    y = [x] + [10,11]
print(y)
```

[4 marks]

```
[[1, 2, 3], 6, 7]
```

This question tests if the student understands the syntax for **if-else** clauses and also list manipulation.

-2 marks for incorrectly nested lists (i.e. [1, 2, 3, 6, 7]).

B.

```
x = (1,2,3,4)
a = x[1::2]
if not a:
    print(a)
else:
    print(a+x)
```

[4 marks]

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

This question tests if the student understands truth values and tuple slicing.

-2 marks for wrong slicing.

-2 marks for wrong if-else branch.

C.

```
a = 0
for i in range(2,7):
    if i%4==0:
        a/=2
        i+=2
    else:
        a+=i
        i+=1
print(a)
```

[4 marks]

13.5

This question tests if the student understands the `for` loop and the `%`, and `/` operators.

-1 mark for returning 13.

-2 marks for returning 8.5 (skipping `i=5`) with reasonable workout.

D.

```
x = 1
y = 5
def foo(y):
    y = 10
    def bar(z):
        return x+y+z
    return bar(y)
print(foo(x))
```

[4 marks]

21

This question tests if the student understands the passing of arguments in functions and variable scoping.

-2 marks for returning 16 with reasonable workings (where `y` is incorrectly identified as 5 instead of 10). Note that, in the return statement of `bar()`, `x` must be 1 and `z` must be 10.

E. `x=[0,1]`
`def test(x):`
 `try:`
 `for i in range(2,6):`
 `x = x.append(i)`
 `except:`
 `print("Bad things!")`
 `else:`
 `print(x)`
 `finally:`
 `print("Finished!")`
`test(x)`

[4 marks]

Bad things!
Finished!

This question tests if the student understands Exception handling syntax.

-2 marks for any incorrect responses printed out.

F. `x = [1,3,2]`
`def foo(x):`
 `x.sort()`
 `x = x + [4,5]`
 `x.extend([6,7])`
`foo(x)`
`print(x)`

[4 marks]

[1, 2, 3]

This question tests if the student understands aliasing.

Question 2: Lego Engineering++ [30 marks]

Once again, your baby brother comes running to you with his building blocks.

“Remember the staircases you showed me last time?” he asks.

“Uh huh...” you slowly nod as you search deep in your memory, vaguely recalling some `staircase` function you wrote.

“I was wondering,” he continues. “what’s the widest or shortest staircase I can build with my blocks?”

You vaguely recall writing this function for your baby brother some time ago.

“No worries,” you boast. “Watch my Python skills.”

For this question, you may assume that a function `staircases(n)` is given, that produces a list of tuples, where each tuple is a staircase, that can be built using n blocks, with at least 2 steps.

Sample Execution:

```
>>> staircases(2)
[]
```

```
>>> staircases(3)
[(1, 2)]
```

```
>>> staircases(4)
[(1, 3)]
```

```
>>> staircases(5)
[(1, 4), (2, 3)]
```

```
>>> staircases(6)
[(1, 2, 3), (1, 5), (2, 4)]
```

```
>>> staircases(7)
[(1, 2, 4), (1, 6), (2, 5), (3, 4)]
```

```
>>> staircases(8)
[(1, 2, 5), (1, 3, 4), (1, 7), (2, 6), (3, 5)]
```

```
>>> staircases(9)
[(1, 2, 6), (1, 3, 5), (1, 8), (2, 3, 4), (2, 7), (3, 6), (4, 5)]
```

A. Define the function `widest` that takes in the number of blocks n as an argument and returns the width of the widest staircase (i.e., the number of steps) that can be built with exactly n blocks. [5 marks]

Sample Execution:

```
>>> widest(5)
2
```

```
>>> widest(7)
3
```

```
def widest(n): # Using high-order functions
    return max(map(lambda x:len(x), staircases(n)))

def widest(n): # Using linear search
    stairs = staircases(n)
    ret = 0
    for stair in stairs:
        if len(stair) > ret:
            ret = len(stair)
    return ret
```

B. Define the function `shortest` that takes in the number of blocks n as an argument and returns the height of the shortest staircase that can be built from exactly n blocks. [5 marks]

Sample Execution:

```
>>> shortest(6)
3
```

```
>>> shortest(7)
4
```

```
def shortest(n): # Using high-order functions
    return min(map(lambda x:x[-1], staircases(n)))

def shortest(n): # Using linear search
    stairs = staircases(n)
    ret = stairs[0][-1] # initialize with the height of first staircase
    for stair in stairs:
        if stair[-1] < ret:
            ret = stair[-1]
    return ret
```

C. “Can my staircase reach a certain height?” is the next question.

Define the function `can_reach(h, n)` which returns `True` if there exists a staircase of height h that can be built from n blocks, and `False` otherwise. [5 marks]

Sample Execution:

```
>>> can_reach(5, 7)
```

```
True
```

```
>>> can_reach(3, 7)
```

```
False
```

```
def can_reach(h, n): # Using high-order functions
    return h in map(lambda x:x[-1], staircases(n))
```

```
def can_reach(h, n): # Using linear search
    for stair in staircases(n):
        if stair[-1] == h:
            return True
    return False
```

-2 marks if mis-interpreted the question and looked for `h` in the steps. -1 mark for syntax errors.

“You know, some of these staircases are crazy,” your brother grumbles. “The gap between the steps is too large.”

D. Define a function `better_stairs(max_gap, n)` that outputs a list of staircases of exactly n blocks, but no two consecutive steps are more than `max_gap` height apart. [6 marks]

Sample Execution:

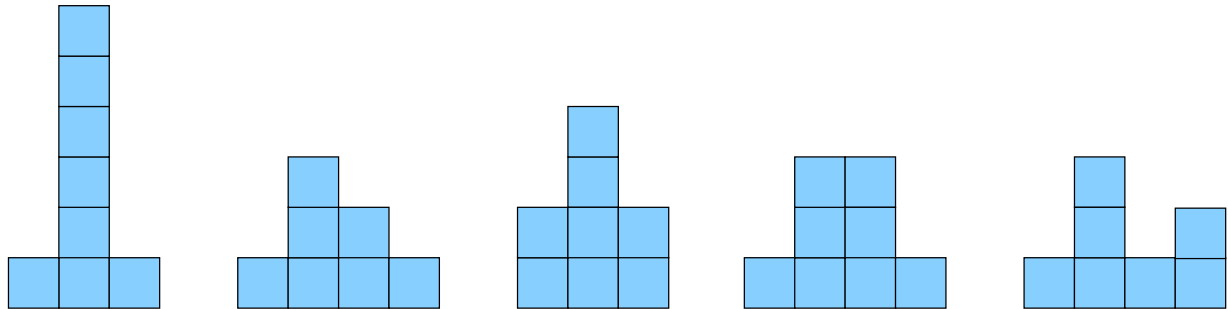
```
>>> better_stairs(2, 9)
```

```
[(1, 3, 5), (2, 3, 4), (4, 5)]
```

```
def better_stairs(max_gap, n):
    def check_gap(stair):
        for i in range(1, len(stair)):
            if stair[i-1] + max_gap < stair[i]:
                return False
        return True
    return list(filter(check_gap, staircases(n)))
```

-1 mark if trying to use `break` to break out of the nested loop.

“What about pyramids?” your brother asks, showing you some of the pyramids he built.



“Ehh... not quite,” you reply. “The first three can be considered pyramids but not the last two.”

“You see, a pyramid is a sequence of steps s , such that all steps before the highest step are in strictly increasing height, and all steps after it are strictly decreasing.”

In other words, for a sequence of steps s_0, s_1, \dots, s_n

$$s_{j-1} < s_j < s_k \text{ and } s_k > s_i > s_{i+1} \text{ for all } 0 \leq j < k < i \leq n$$

“Let me fix them for you,” you offer.

E. Define a function `is_pyramid` that takes in a sequence of steps and returns `True` if the steps form a pyramid, and `False` if not. You may assume that each step is an integer greater than 0 and there are at least two steps. **Note:** This definition is slightly different from that in the Practical Exam because a pyramid need not be symmetric. [5 marks]

Method 1: Keep track of ascending and descending with a flag.

```
def is_pyramid(steps):
    # first step must be ascending
    if steps[1] < steps[0]:
        return False

    up = True
    for i in range(1, len(steps)):
        if steps[i-1] == steps[i]: # steps cannot be of equal height
            return False
        if up and steps[i-1] > steps[i]:
            up = False # switch to going down
        elif not up and steps[i-1] < steps[i]:
            return False
    return not up # must end descending
```

Method 2: Find the peak and check the left and right of the peak.

```
def is_pyramid(steps):
```



```

peak = steps.index(max(steps))          # find the index of the peak

if peak == 0 or peak == len(steps)-1: # peak cannot be at ends
    return False

# check the left and right INCLUDING the peak
# If you do not include the peak in the range,
# then you have to specially check for flat peaks
for i in range(peak):                  # check all left is ascending
    if steps[i] >= steps[i+1]:
        return False
for i in range(peak, len(steps) - 1): # check all right is descending
    if steps[i] <= steps[i+1]:
        return False
return True

```

-1 mark for sorting the sides and comparing the reverse because it cannot detect same height steps. -1 mark for not handling corner cases e.g., (1,2,3,4) or (4,3,2,1) etc. -1 mark if bounds in for-loops are not correct, causing index out of range.

F. Define a function `is_symmetric` that takes in a pyramid and returns `True` if the pyramid is symmetrical, and `False` if not. You may assume the input is a proper pyramid, i.e., that `is_pyramid` would return `True`. [4 marks]

Method 1:

```

def is_symmetric(s):
    return s == s[::-1]

```

Note: `s.reverse()` only works if `s` is a list and it mutates the list while returning `None`.

Method 2:

```

def is_symmetric(s):
    for i in range(len(s)//2):
        if s[i] != s[-i-1]:
            return False
    return True

```

Method 3:

```

def is_symmetric(s):
    if len(s) <= 1:
        return True
    if s[0] != s[-1]:
        return False
    return is_symmetric(s[1:-1])

```

-1 mark for using `reverse()` wrongly.

Question 3: Espionage Deja Vu [18 marks]

You are working for a spy agency and you have been tasked with building a multi-level encryption system for the agency. Your job is to create a new class `SecretMessage` which is used to store a secret message that supports multiple levels of encryption. The initial `SecretMessage` object is initialized with an input message (`String`) as an argument.

A `SecretMessage` object supports for the following methods:

1. `read()`: returns the message stored in the `SecretMessage`, unless the message is encrypted, which would return the string `"*ENCRYPTED*"`.
2. `encrypt(p_1, p_2, \dots, p_k)`: which can take in an arbitrary number of passwords and encrypts the secret message using the passwords in order p_1 , followed by p_2 and so forth.
3. `decrypt(p_k, p_{k-1}, \dots, p_1)`: which can take in an arbitrary number of passwords and attempts to decrypts the secret message using the passwords in order p_k , followed by p_{k-1} and so forth. Note that to successfully decrypt and recover an encrypted message, `decrypt` must be applied in strict reverse order. If the wrong password is used in the decryption, the message will be messed up and any subsequent attempts to `read` it will return the string `"*GARBLED*"`. Attempts to decrypt a non-encrypted message will have no effect.
4. `copy()`: returns a copy of the secret message, that retains all encryption properties of the original message.

The following is a sample trace showing how `SecretMessage` works:

```
>>> m1 = SecretMessage("Congrats on surviving CS1010S!")
>>> m1
<__main__.SecretMessage object at 0x7fc600584110>

>>> m1.read()
Congrats on surviving CS1010S!

>>> m1.encrypt(12345)
<__main__.SecretMessage object at 0x7fc600584110>

>>> m1.read()
*ENCRYPTED*

>>> m2 = m1.copy()
>>> m1.decrypt(54321)
<__main__.SecretMessage object at 0x7fc600584110>

>>> m1.read()
*GARBLED*
```

```
>>> m2.read()
*ENCRYPTED*

>>> m2.decrypt(12345)
<__main__.SecretMessage object at 0x7fc600583322>

>>> m2.read()
Congrats on surviving CS1010S!

>>> m2.encrypt(54321)
<__main__.SecretMessage object at 0x7fc600583322>

>> m2.read()
*ENCRYPTED*

>>> m2.encrypt(12345)
<__main__.SecretMessage object at 0x7fc600583322>

>>> m2.read()
*ENCRYPTED*

>>> m3 = m2.copy()
>>> m2.decrypt(12345)
<__main__.SecretMessage object at 0x7fc600583322>

>>> m2.read()
*ENCRYPTED*

>>> m2.decrypt(54321)
<__main__.SecretMessage object at 0x7fc600583322>

>>> m2.read()
Congrats on surviving CS1010S!

>>> m2.decrypt(11111)
<__main__.SecretMessage object at 0x7fc600583322>

>>> m2.read()
Congrats on surviving CS1010S!

>>> m3.decrypt(54321)
<__main__.SecretMessage object at 0x7fc600584178>

>>> m3.read()
*GARBLED*
```

A. Describe how you will keep track of the state of a message and give a possible implementation of the `__init__` method for the `SecretMessage` object. [4 marks]

We will use the variable `msg` to store the text message and another list `passwords` to use to keep track of the passwords that are used to encrypt the message. `passwords[0]` is the earliest password and subsequent passwords are appended to `passwords`.

```
def __init__(self, text):
    self.msg = text
    self.passwords = []
```

This question serves to test that students have learnt their lesson on Q4 of the midterm exam. This question is however not the same because it tests if students fully understand how to implement mutable data structures.

2 marks for a reasonable description and 2 marks for correct code.

In this Question 3, unless specified otherwise, 1 mark will be deducted for each wrong code.

B. Please provide a possible implementation for the `read` method. [3 marks]

```
def read(self):
    if self.passwords == []:
        return self.msg
    else:
        return "*ENCRYPTED*"
```

-1 mark for not returning the original message properly.

-1 mark for not returning `*ENCRYPTED*` properly.

-1 mark for not returning `*GARBLED*` properly.

-1 mark for using `print` instead of `return`.

C. Please provide a possible implementation for the `encrypt` method that will encrypt the message according to Part (A) above and return the encrypted message. [4 marks]

```
def encrypt(self,*passwords):
    self.passwords.extend(passwords)
    return self
```

-1 mark for not taking in an arbitrary number of passwords.

-1 mark for misusing `append` and `extend`.

D. Please provide a possible implementation for the `decrypt` method that will decrypt the message that was encrypted by the `encrypt` function in Part (C) above and return the decrypted message. [4 marks]

```
def decrypt(self,*passwords):
    for password in passwords:
        if self.passwords != []:
            if password == self.passwords[-1]:
                self.passwords.pop()
            else:
                self.msg = "*GARBLED*"
                self.passwords = []
    return self
```

-1 mark if `"*GARBLED*"` is not handled properly.

-1 mark for not taking in an arbitrary number of passwords.

-1 mark for not checking an empty list of passwords or for causing out-of-range errors.

-1 mark for misusing `remove()` instead of `pop()`.

E. Please provide a possible implementation for the `copy` method. [3 marks]

```
def copy(self):  
    msg = SecretMessage(self.msg)  
    msg.passwords.extend(self.passwords)  
    return msg
```

-2 marks if the entire object is aliased (e.g. by returning `self`).

-1 mark for any aliasing issue (e.g. the list `passwords` can be aliased).

Depending on your implementation, you can also use the built-in `deepcopy()` with a proper import.

```
from copy import *
```

```
...
```

```
def copy(self):  
    return deepcopy(self)
```

Question 4: To Infinity and Beyond!! [24 marks]

So far, we have been dealing with finite objects in CS1010S. For example, when you work with tuples, lists or dictionaries, you can count the number of elements. But there are objects with an infinite number of elements. For example, consider the following series of natural numbers:

$$1, 2, 3, 4, 5, 6, 7, 8, \dots$$

It turns out that it is possible to manipulate what are effectively infinite series by generating the required objects *on-the-fly*, since even if we had an infinite number of objects, we couldn't possibly be accessing them all at once. We can define *series* objects that support only one method `get` that will just produce the next term in a series. For example, the following `NaturalNumbers` class will generate all the natural numbers (i.e. positive integers) with a `get` method:

```
class NaturalNumbers:
```

```
    def __init__(self):
        self.count = 0

    def get(self):
        self.count += 1
        return self.count
```

For convenience, we can define the following function `get_terms` that will return a list of the first n elements for a series:

```
def get_terms(series, n):
    result = []
    for i in range(n):
        result.append(series.get())
    return result
```

The following is an example of how we can obtain a bunch of positive integers:

```
>>> pos_ints = NaturalNumbers()
>>> get_terms(pos_ints, 10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

>>> get_terms(pos_ints, 10)
[11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
```

A. [Warm Up] Define a new `Series` class that will take as an argument a function f and return the following series using a `get` method like `NaturalNumbers`: [4 marks]

$$f(1), f(2), f(3), f(4), f(5), f(6), f(7), f(8), \dots$$

Sample Execution:

```
>>> even_nums = Series(lambda x: 2*x)
>>> get_terms(even_nums, 5)
[2, 4, 6, 8, 10]
```

```
>>> get_terms(even_nums, 5)
[12, 14, 16, 18, 20]
```

```
class Series:

    def __init__(self, f):
        self.count = 0
        self.f = f

    def get(self):
        self.count += 1
        return self.f(self.count)
```

-1 mark if student forgets `self` for `self.f`.

B. Suppose you are given the following definition of the Fibonacci function `fib`:

```
def fib(n):
    if n <= 1:
        return n
    else:
        return fib(n-1) + fib(n-2)
```

Using `fib` and your definition of `Series` in Part (A), create an infinite series of Fibonacci numbers `fibs`, i.e. [4 marks]

```
>>> get_terms(fibs, 10)
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

```
fibs = Series(lambda x: fib(x-1))
```

-1 mark for `Series(lambda x: fib(x))` or `Series(fib)`.
-2 marks for `Series(fib(n))`.

C. What is the order of growth in both time and space for the expression `get_terms(fibs,n)` in terms of n for the definition of `fibs` from Part (B)? Explain. [4 marks]

First, we know that `fib` has $O(\Phi^n)$ order of growth in time and $O(n)$ order of growth in space. Total time taken for `get_terms(fibs,n)` is approximately:

$$\Phi + \Phi^2 + \dots + \Phi^n = \Phi \left(\frac{\Phi^n - 1}{\Phi - 1} \right) = O(\Phi^n)$$

$O(2^n)$ is an acceptable answer. -1 mark for $O(n2^n)$.

Similarly, the order of growth in space is dominated by `fib(n)` and is thus $O(n)$.

D. Define a new series object `Fibs` that will produce a series of Fibonacci numbers in $O(n)$ time. For example, in the following code:

```
>>> faster_fibs = Fibs()
>>> get_terms(faster_fibs,5)
[0, 1, 1, 2, 3]
```

```
>>> get_terms(faster_fibs,5)
[5, 8, 13, 21, 34]
```

`get_terms(faster_fibs,n)` will run in $O(n)$ time. [4 marks]

```
class Fibs:

    def __init__(self):
        self.a = 0
        self.b = 1

    def get(self):
        result = self.a
        self.a, self.b = self.b, self.a + self.b
        return result
```

Some students implemented memoization and were awarded full credit. 2 points were awarded to students who implemented iterative `fib` in some way but not entirely correctly.

E. Like lists and tuples, we can also manipulate infinite series. One natural operation is a *filter* operation. Define a new series object `FilterSeries` that takes in a filter function f and another series s and returns a new series which contains terms from s that satisfy the filter f . [4 marks]

Sample Execution:

```
primes = FilterSeries(is_prime, NaturalNumbers())
>>> get_terms(primes, 10)
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29]
```

```
class FilterSeries:

    def __init__(self, f, s):
        self.f = f
        self.s = s

    def get(self):
        term = self.s.get()
        while not self.f(term):
            term = self.s.get()
        return term
```

-2 marks were taken off for:

```
def get(self):
    while True:
        if self.f(self.s.get()):
            return self.s.get()
```

because this would have caused one term to be lost.

F. Sometimes we might want to work with a limited series of numbers instead of an infinite series. Define a new series object `LimitedSeries` that takes in a series `s` and a number `n` and returns a new series which contains only the first `n` terms from `s`. After the first `n` terms are exhausted, `None` will be returned by the `get` function. [4 marks]

Sample Execution:

```
>>> first5 = LimitedSeries(NaturalNumbers(),5)
>>> get_terms(first5,8)
[1, 2, 3, 4, 5, None, None, None]
```

```
class LimitedSeries:

    def __init__(self,s,n):
        self.s = s
        self.n = n
        self.count = 0

    def get(self):
        self.count += 1
        if self.count > self.n:
            return None
        else:
            return self.s.get()
```

Question 5: 42 and the Meaning of Life [4 marks]

Either: (a) explain how you think some of what you have learnt in CS1010S will be helpful for you for the rest of your life and/or studies at NUS; or (b) write a short snippet of code that you think will convince the examiner that you have learnt Python well and hence deserve a good grade for CS1010S; or (c) tell us an interesting story about your experience with CS1010S this Semester.

The student will be awarded points as long as he/she is coherent and doesn't say something obviously wrong

Appendix

The following are some functions that were introduced in class:

```
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:]))

def is_prime(x):
    if x == 1:
        return False
    else:
        for i in range(2, int(math.sqrt(x) + 1)):
            if x%i == 0:
                return False
        return True
```

— E N D O F P A P E R —

Scratch Paper

– H A P P Y H O L I D A Y S ! –