NATIONAL UNIVERSITY OF SINGAPORE

Semester 2, 2015/2016

Solutions for CS1010S — PROGRAMMING METHODOLOGY

Time Allowed: 2 Hours

INSTRUCTIONS TO STUDENTS

1. Please write your Student Number only. Do not write your name.
2. The assessment paper contains **FIVE (5) questions** and comprises **TWENTY-TWO (22) pages** including this cover page.
3. Weightage of questions is given in square brackets. The maximum attainable score is 100.
4. This is a **CLOSED** book assessment, but you are allowed to bring **TWO** double-sided A4 sheets of notes for this exam.
5. Write all your answers in the space provided in this booklet.
6. You are allowed to write with pencils, as long as it is legible.
7. **Please write your student number below.**

   STUDENT NO: _____

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

Question	Marks	Remark
Q1		
Q2		
Q3		
Q4		
Q5		
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 and **write the exact output in the answer box**. If the interpreter produces an error message, or enters an infinite loop, explain why. You may show your workings **outside the answer box** in the space beside the code. Partial marks may be awarded for workings if the final answer is wrong.

A. [5 marks]

```
a = 20
b = 4
c = 5
def x(a):
    return a + b
def y(a, b):
    return a(b) - c
print(y(x, a))
```

```
19
```

This question tests the understanding of scoping. One way to remove confusion is to change the names of the input parameters.

B. [5 marks]

```
s = "cs1010s c001"  # last character is digit one
d = {}
for i in range(len(s)):
    d[s[i]] = i
print(d)
```

```
{'': 7, '1': 11, 's': 6, '0': 10, 'c': 8}
```

This question tests the understanding of Python dictionaries.

C. [5 marks]

```
x, y = 2, 1
def g(x):
    return lambda x: lambda y: x-y
print(g(x)(3)(4))
```

```
-1
```

This question tests the understanding of lambda functions.

D. [5 marks]

```
lst1 = [1, 2, 3, 4]
lst2 = [5, 6, 7, 8]
for i in lst1:
    lst2.append(i)
    lst1.remove(i)
print(lst1)
print(lst2)
```

```
[2, 4]
[5, 6, 7, 8, 1, 3]
```

This question tests the understanding the problem with removing while iterating a list.

E. [5 marks]

```
def do(x):
    try:
        return x[0] + x[1]
    except IndexError:
        print("Bad")
    except TypeError:
        print("Good")
    finally:
        return x[0] + x
print(do([[1], 2]))
```

```
Good
[1, [1], 2]
```

This question tests the understanding of exception handling and list addition.

F. [5 marks]

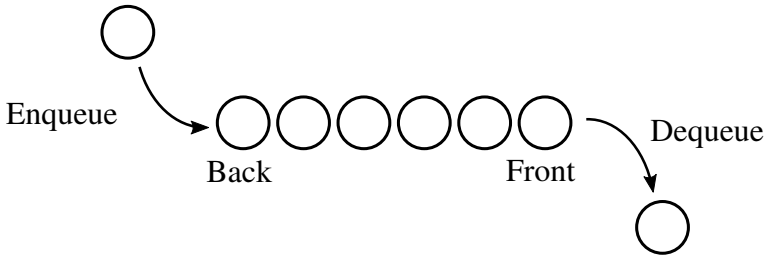
```
a = [['a', 'b'], ['c'], 'd']
b = a[:-1]
a[1], b[0][1] = b[0], a[2]
print(a)
print(b)
```

```
[['a', 'd'], ['a', 'd'], 'd']
[['a', 'd'], ['c']]
```

This question tests the understanding of shallow copy of lists.

Question 2: Priority Queues [24 marks]

Queues are found in everyday situations and we have learned how a List can be used to model a queue. Recall that a queue is a First-in-First-Out (FIFO) data structure, where items are removed in the same order as they are added.



Queues are supported by the following functions:

- `make_queue()`: returns an empty queue.
- `enqueue(q, item)`: adds item to the back of queue q.
- `dequeue(q)`: removes the item at the front of the queue q, and returns it. If the queue is empty, `None` is returned.
- `size(q)`: returns the number of items in the queue q.

A. [Warm-up] Aida implemented her queue using a List. Give a possible implementation of the each of the queue functions of Adia's representation. [4 marks]

```
def make_queue():
    return []




def enqueue(q, item):
    q.append(item)
```

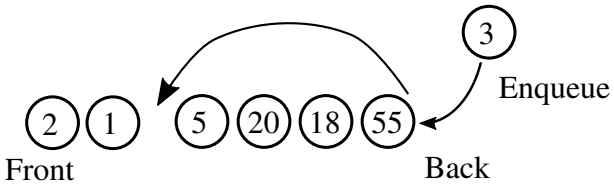
```
def dequeue(q):
    if q:
        return q.pop(0)
    else:
        return None




def size(q):
    return len(q)
```

In a certain medical centre, the queue to see the doctor is not strictly first-in-first-out. For example, priority is given to children under the age of 4. That means if a child under the age of 4 joins the queue, he will be allowed to move forward until he reaches the front of the queue, or he ends up just behind another child under the age of 4.

For example, suppose the numbers in the following figure denotes the age of the people in the queue. If a child of age 3 joins the queue, he will be inserted in the 3rd position.



Note that even though "3" was allowed to jump queue, the queue remains stable. i.e., if two persons have the same priority, they must be served in the order which they joined the queue.

One way to determine the priority between two persons is to define a *priority function*. A priority function e.g. `fn(p1, p2)` takes as inputs two persons, p1 and p2, and returns `True` if **p1 has priority over p2** and `False` otherwise.

B. Provide an implementation for a priority function `below_4` such that only children under the age of 4 will have priority over persons age 4 and above. Assume that the function `age(person)` takes as input a person object and returns the age of the person. [4 marks]

```
def below_4(p1, p2):
    return age(p1) < 4 and age(p2) >= 4
```

C. To enable priority in the queue, write a new enqueue function `priority_enqueue(q, fn, p)` which takes as inputs a queue `q`, a priority function `fn` and a person `p`, and enqueues the person to the queue according to `fn`, which was defined earlier.

Provide an implementation of `priority_enqueue`. [6 marks]

Hint: To determine the position where the person joins the priority queue, one can start at the back of the queue and keep moving the person forward if he has priority over the other person in the queue.

Hint II: the function `list.insert(pos, item)` inserts `item` into position `pos` of the list.

```
def priority_enqueue(q, fn, p):
    for i in range(len(q)-1, -1, -1):
        if not fn(p, q[i]):
            q.insert(i+1, p)
            return
    q.insert(0, person)
```

D. In addition to children under the age of 4 having priority, the medical centre also decided to give priority to senior citizens, i.e., people over the age of 60. However, children under the age of 4 remains a higher priority than senior citizens.

Give an example of how the function `priority_enqueue` will be called to enqueue person `p` into queue `q` using this new priority scheme? [6 marks]

```
priority_enqueue(q,
                 lambda p1, p2: (age(p1) < 4 and age(p2) >= 4) or \
                                (age(p1) > 60 and 4 <= age(p2) <= 60),
                 p)
```

E. Aida thinks that since there is a priority function that can determine the order at which to place the people, she can simply append a new person to her underlying List representation and call the Python sort function like so:

```
def priority_enqueue(q, fn, p):
    q.append(p)
    q.sort(key=fn)
```

What is the error in Aida's thinking? [2 marks]

The function to be used as a key in the python sort function takes in one input while the priority function takes in two inputs.

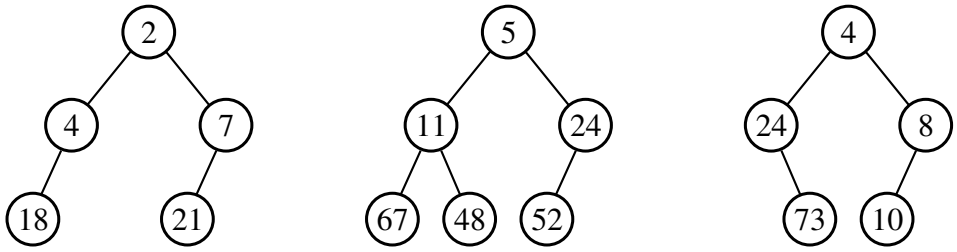
F. Aida's friend, Bernice, thinks that she can fix the error by using `key=lambda x: age(x)` in the sort function. Do you think this is correct? Explain briefly. [2 marks]

No. This will sort the queue according to the age of the people. Though we do end up with those under 4 being in the front of the queue. This messes up the original order of the queue.

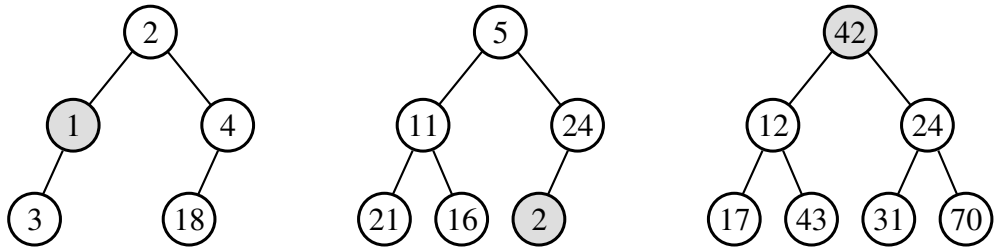
Question 3: Heaps [26 marks]

For the purpose of this question, a heap is a binary tree that satisfies this property: the value of a node is smaller or equal to the values of all its children. (More specifically this is a *minimum binary heap*. It is slightly different from the binary search tree encountered in your assignment.)

Some examples of valid heaps:



Here are some examples of invalid heaps, with the offending node highlighted:



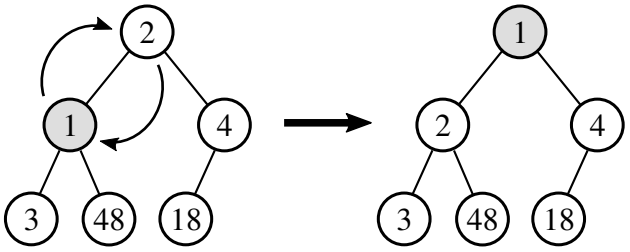
A node in the heap is implemented as a `Node` object with the following properties:

- `Node.value` contains the value of the node.
- `Node.left` contains the reference to the left child of the node. If `Node` has no left child, the value will be `None`.
- `Node.right` contains the reference to the right child of the node. If `Node` has no right child, the value will be `None`.
- `Node.parent` contains the reference to the parent of the node. If `Node` has no parent, i.e. it is the root node, the value will be `None`.

A. Write a function `valid_heap` which takes as input the root node of a heap, and returns `True` if the heap is valid (according to the property stated), and `False` otherwise. [6 marks]

```
def valid_heap(node):
    if node == None:
        return True
    if node.left and node.left.value < node.value:
        return False
    if node.right and node.right.value < node.value:
        return False
    return valid_heap(node.left) and valid_heap(node.right)
```

B. It is useful to be able to swap nodes in a heap. For example, we can swap the nodes in an invalid heap to make it valid like this:

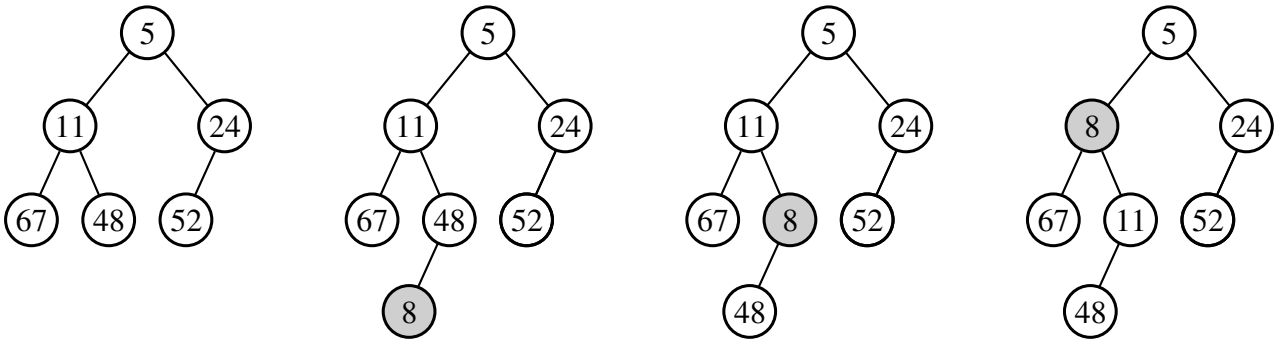


Write a function `swap(n1, n2)` that takes as inputs two nodes, n1 and n2 and swap them.
Hint: The structure of the heap is preserved after swapping. [4 marks]

```
def swap(n1, n2):
    n1.value, n2.value = n2.value, n1.value
```

C. To add a new value into a heap, we first create a new node and attach it as a new left to the bottom of the heap. Next, to make it a valid heap, we compare the value of the new node with its parent. If the new node is smaller than the parent, we swap them. This "bubbling up" continues until the new node finds a place which makes the heap valid.

Example:



Original heap | 8 added to the bottom | Swap with parent | Swap again. Done!

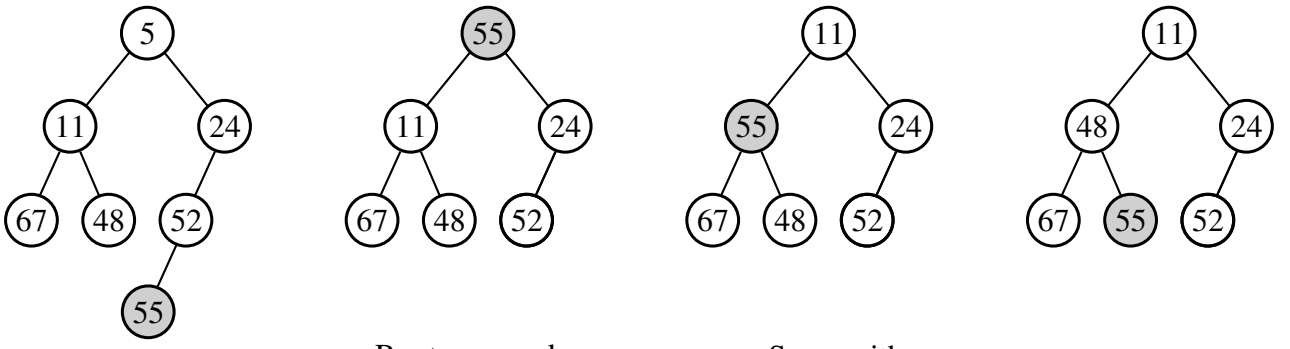
Implement the function `bubble_up(node)` which takes as input, a newly added node to a valid heap, and performs the swapping needed to make the heap valid. Assume that the new node has been attached to the bottom of the heap. [6 marks]

```
def bubble_up(node):
    while node.parent and node.parent.value > node.value:
        swap(node, node.parent)
        node = node.parent
```

D. Because of the property of the heap, the root will always be the smallest value found in the heap. When removing the root, we perform the following strategy:

First, a bottom node is moved to replace the root that was extracted. Next this node is compared with its children, and swapped with the smallest child (this ensures that after swapping the new parent will be smaller than both children.) This "bubble down" process continues until the heap is valid.

Example:



Original heap | Root removed, replaced with 55 | Swap with smallest child | Swap again. Done!

Suppose there is a function `min_node(n1, n2, n3)` which takes in three nodes and return the node with the smallest value. Suppose also `min_node` will ignore inputs with the value `None`. Implement the function `bubble_down(node)`, which takes as input the new swapped root of a heap after the old root is extracted, and perform the "bubble down" process to make the heap valid. [6 marks]

```
def bubble_down(node):
    new_parent = min_node(node, node.left, node.right)
    if not new_parent is node:
        swap(node, new_parent)
        bubble_down(new_parent)
```

E. Removing from the root will always give us the smallest value of the heap. In some ways it functions like a priority queue, where the highest priority item (i.e., smallest value) is always extracted. If we are careful to ensure the heap remains a balanced tree[1] when inserting new nodes and extracting the root, in what ways would using a heap be more efficient than using a List to maintain a priority queue as done in Question 2? [2 marks]

When using a List, an enqueue or dequeue operation will have time complexity of $O(n)$, where n is the length of the queue. When using a heap, the time complexity of both enqueue and dequeue operation be $O(\log n)$. Thus, in terms of time complexity, using a heap is faster than a List.

F. Cherry thinks that she can implement the priority queue for the medical center described in Question 2D using a heap. She intends to use the priority function to determine when to bubble up or bubble down the nodes.

Briefly explain why Cherry might be making a mistake. [2 marks]

The priority queue for the medical centre requires the queue to be stable. That is people of the same priority should remain in the same order. Naively using our heap implementation does not guarantee this property.

[1] A tree is balanced when the height of its left and right subtrees differ by at most 1, and both subtrees are also balanced.

Question 4: The Force Awakens [16 marks]

Having learned Python, Rey decides to model the force users in the Star Wars universe using classes as follows:

```
class Jedi:
    def __init__(self, name):
        self.name = name
        self.powers = ['jump', 'heal', 'mind trick', 'push']

    def do(self, action):
        if action in self.powers:
            print(self.name + " performs Force " + action)
        else:
            print(self.name + " does not know Force " + action)

class Sith:
    def __init__(self, name):
        self.name = "Darth " + name
        self.powers = ['jump', 'lightning', 'choke', 'push']

    def do(self, action):
        if action in self.powers:
            print(self.name + " performs Force " + action)
        else:
            print(self.name + " does not know Force " + action)
```

A. What will be the output when the following statements are executed in the shell?

```
>>> maul = Sith("Maul")
>>> maul.do("lightning")

>>> yoda = Jedi("Yoda")
>>> yoda.do("choke")
```

[2 marks]

```
"Darth Maul performs Force lightning"
"Yoda does not know Force choke"
```

Rey then decided that since some Jedis have in the past, turned to become Siths, she adds a new sub-class `JediTurnSith` as follows:

```
class JediTurnSith(Jedi, Sith):
    def __init__(self, name):
        super().__init__(name)
```

Rey then tests her code with:

```
>>> vader = JediTurnSith("Vader")
>>> vader.do("jump")
Vader performs Force jump

>>> vader.do("heal")
Vader performs Force heal

>>> vader.do("choke")
Vader does not know Force choke
```

"Something is wrong!" Rey exclaims. She wonders why it does not print Darth Vader and why Vader does not know Force choke.

B. Explain briefly why this is so. [2 marks]

This is because `JediTurnSith` subclass `Jedi` before `Sith`, so the Jedi's `__init__` function is called which does not add "Darth" in front of the name.

Also, only powers from the Jedi's were added to `self.power` since `Sith`'s `__init__` function was never called.

What Rey wanted was for a `JediTurnSith` to not only have "Darth" prepended to the name, but also have both Jedi and Sith powers.

A friend shows you an example how `super().__init__` will repeatedly call `__init__` throughout the class hierarchy:

```
class A:
    def __init__(self):
        print("A")

class B(A):
    def __init__(self):
        print("B")
        super().__init__() # this calls C's __init__

class C(A):
    def __init__(self):
        print("C")
        super().__init__() # this calls A's __init__

class D(B, C):
    def __init__(self):
        print("D")
        super().__init__() # this calls B's __init__

>>> d = D()
D
B
C
A
```

This gives you an idea! We can create a new super class called `ForceUser` which both `Jedi` and `Sith` can subclass from:

```
class ForceUser:
    def __init__(self, name):
        self.name = name
        self.powers = []  # creates empty list of powers

    def do(self, action):
        if action in self.powers:
            print(self.name + " performs Force " + action)
        else:
            print(self.name + " does not know Force " + action)
```

C. Complete the implementations for `Jedi`, `Sith` and `JediTurnSith` using `ForceUser` as the base class, that will allow `vader` to posses the powers of both Jedi and Sith.

Example:

```
>>> vader = JediTurnSith("Vader")
>>> vader.do("lightning")
Darth Vader performs Force lightning
>>> vader.do("heal")
Darth Vader performs Force heal

>>> isinstance(vader, Jedi)
True
>>> isinstance(vader, Sith)
True
```

[8 marks]

```
class Jedi(ForceUser):
    def __init__(self, name):
        super().__init__(name)
        self.powers.extend(['jump', 'heal', 'mind trick', 'push'])

class Sith(ForceUser):
    def __init__(self, name):
        super().__init__("Darth " + name)
        self.powers.extend(['jump', 'lightning', 'choke', 'push'])

class JediTurnSith(Sith, Jedi):
    def __init__(self, name):
        super().__init__(name)
```

Now it turns out that Siths have an alias which they use in place of their name so others can not tell that they are Siths. We could allow the `Sith` class to take in an alias as an extra input like so:

```
>>> emperor = Sith("Sidious", "Palpatine")
>>> emperor.name
'Darth Sidious'
>>> emperor.alias
'Palpatine'

>>> vader = JediTurnSith("Vader", "Anakin")
>>> vader.name
'Darth Vader'
>>> vader.alias
'Anakin'
```

State what modifications you will need to do to your classes to support this change. You do not need to rewrite all the code, just provide snippets of the modifications. [4 marks]

The inputs the `__init__` function of the following classes have to be changed, and a new field must be added in `Sith`:

```
class Sith(ForceUser):
    def __init__(self, name, alias):
        super().__init__("Darth " + name)  # this remains the same
        ...
        self.alias = alias  # new field must be added

class JediTurnSith(Sith, Jedi):
    def __init__(self, name, alias):
        super().__init__(name, alias)
```

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) tell us an interesting story about your experience with CS1010S this semester. [4 marks]

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

— E N D O F P A P E R —

Scratch Paper

Scratch Paper

Scratch Paper

– H A P P Y H O L I D A Y S ! –