Duration: **70 minutes**
Aids Allowed: ***None***

Student Number: |_|_|_|_|_|_|_|_|_|_|_|_|

UTORid: _____

Last (Family) Name(s): _____

First (Given) Name(s): _____

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

Please put an x beside your tutorial section

☐	TUT0001	Philip (Jianho) Yang	MO 13-15
☐	TUT0002	Eric Ren	MO 15-17
☐	TUT0005	Nick (Ruo Fan) Li	TU 10-12
☐	TUT0006	Faisal Usmani	WE 12-14
☐	TUT0007	Lev Karatun	WE 13-15
☐	TUT0008	Yang Song	WE 14-16
☐	TUT0009	Nick Olson-Harris	WE 15-17
☐	TUT0010	Yamn Chalich	TU 11-13
☐	TUT0011	Harmen Kahlon	MO 10-12
☐	TUT0012	Kenneth Ma	TU 16-18
☐	TUT0013	Denning Campbell	TH 15-17
☐	TUT0014	Judy Duong	FR 11-13

This term test consists of 3 questions on 12 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete, fill in the identification section above, and write your name on the back of the last page.*

Answer each question directly on the test paper, in the space provided, and use one of the "blank" pages for rough work. If you need more space for one of your solutions, use a "blank" page and *indicate clearly the part of your work that should be marked.*

MARKING GUIDE

1: _____/10

2: _____/10

3: _____/10

TOTAL: _____/30

Good Luck!

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Question 1. [10 marks]

```
class B:
    def __init__(self: 'B', name: str) -> None:
        self._name = name

    def __str__(self: 'B') -> str:
        return ("I'm a B named " + self._name)

    def mB(self: 'B', n: int) -> None:
        print("mB", self._name, str(n))

class C(B):
    def __init__(self: 'C', name: str) -> None:
        B.__init__(self, name)
        self._name_len = len(name)

    def __str__(self: 'C') -> str:
        return ("I'm a C named " +
                  self._name + "-" +
                  str(self._name_len))

    def mC(self: 'C', n: int) -> None:
        print("mC", end = " ")
        B.mB(self, n+1)

class D(C):
    def __init__(self: 'D', name: str) -> None:
        C.__init__(self, "Super " + name)
        self._junk = self._name_len + 3

    def mC(self: 'D', n: int) -> None:
        print(str(self._junk), end = " ")
        C.mC(self, n+1)

if __name__ == "__main__":
    b = B("Bob")
    c = C("Carole")
    d = D("Dan")

    print("1:", b)
    print("2:", c)
    print("3:"d)
    print("4:")
    b.mB(12)
    print("5:")
    c.mB(13)
    print("6:")
    d.mB(14)
    print("7:")
    b.mC(15)
    print("8:")
    c.mC(16)
    print("9:")
    d.mC(17)
```

Write the output of the code in the box below. If a line of code would cause Python to crash, write `CRASH`, and then continue tracing as though that line had been commented out of the main block.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Question 2. [10 marks]

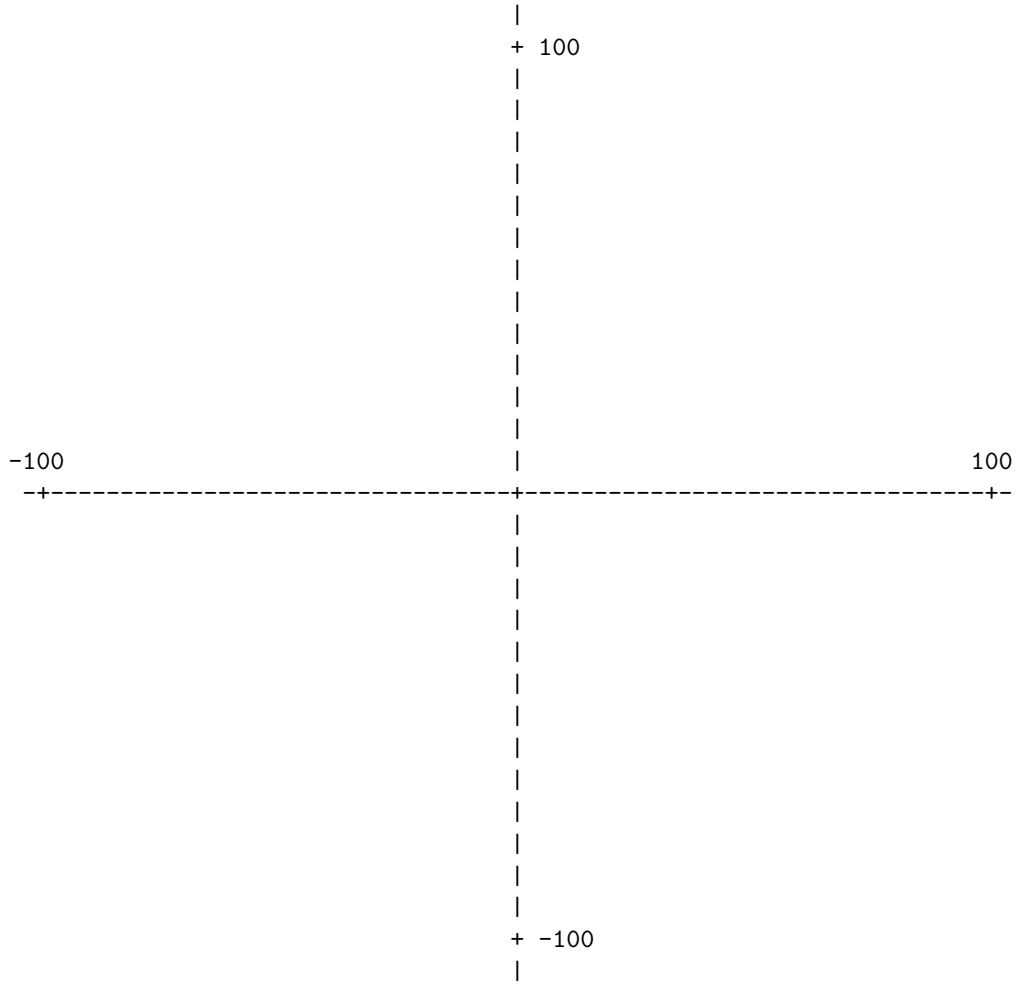
Suppose the function call
`draw_rectangle(x1, y1, x2, y2)`
draws a rectangle on an x-y plane with vertices
`(x1,y1), (x2,y1), (x2,y2), (x1,y2).`
i.e., it draws a rectangle which sits flat on the line y=y1, with its bottom left corner at (x1,y1), and its top right corner at (x2,y2).
Now consider the following recursive function.

```
def funky_rect(x1: int, y1: int, x2: int, y2: int, level: int) -> None:
    if level <= 0:
        return

    xmid = (x1 + x2) // 2
    ymid = (y1 + y2) // 2

    draw_rectangle(x1, y1, x2, y2)
    funky_rect(xmid + 5, y1 + 5, x2 - 5, ymid - 5, level - 1)
    funky_rect(x1 + 5, ymid + 5, xmid - 5, y2 - 5, level - 1)
```

a) On the x-y plane below, draw what following function call would draw.
`funky_rect(-100, -100, 100, 100, 3)`



Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

b) On your diagram above clearly indicate which rectangle that was drawn first, and which was drawn last.
c) Rewrite `funky_rect` in the space below so that the call

```
funky_rect(-100, -100, 100, 100, 3)
```

will still produce the same diagram, except the rectangle that was drawn first will now be drawn last, and the rectangle that was drawn last will now be drawn first.

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Question 3. [10 marks]

Consider the following class for a node used in a (singly) linked list of items, each with a non-negative integer called "importance". We let users of `ImportanceNode` muck around as much as they want with the data stored in a node.

```
class ImportanceNode:
    def __init__(self: 'ImportanceNode',
                 importance: int,
                 data: object) -> None:
    '''
    Create a new ImportanceNode with importance importance.
    REQ: importance >= 0.
    '''
         self._importance = importance
         self.next = None
         self.data = data

    def get_importance(self: 'ImportanceNode') -> int:
        return self._importance
```

At times we want to move the item with the highest importance to the front of the list. If there are multiple items with highest importance, then the last one of these in the list is moved.
Some time ago, Nick started writing the following function to perform the node movement as described above. However, he never finished, nor did he write any internal comments. Here's what he wrote.

```
def move_high(il_head: 'ImportanceNode') -> 'ImportanceNode':
    '''
    From the linked list whose head is il_head, move the node with highest
    importance to the front.  If there are multiple nodes with highest
    importance, then the last such node in the list is moved.
    The list is unchanged if it is empty.
    Return the (head of) updated list.
    '''

    prev = None
    curr = il_head
    high = -1
    while curr != None:
        curr_imp = curr.get_importance()
        if curr_imp > high:
            high = curr_imp
            high_node = curr
            high_prev = prev
        curr = curr.next
        prev = curr

    if high_prev == None:
        return il_head

    high_node.next = il_head
    return high_node
```

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

There are 4 bugs in Nick's code. For each bug, describe the bug, and a small change (at most one line) that could be made to fix the bug. To help you get started, we provided the answer for one of the bugs.

```
Bug #0:
 On a list with multiple nodes of equal highest importance, high_node
 (node to move) is incorrectly identified.
Fix: Change the if condition in loop to
    if curr_imp >= high:
```

On this page, please write nothing except your name.

Last (Family) Name(s): _____

First (Given) Name(s): _____

Total Marks = 30