Question 1. [8 MARKS]

Consider these examples of a function `map_nested`:

```
# Call the abs function on each item in a list, recursively.
assert map_nested(abs, [-1, [-3, 3]]) == [1, [3, 3]]

# Call the len function on each item in a list, recursively.
assert map_nested(len, [['a'], 'abc']) == [[1], 3]
```

Complete `map_nested`:

Here are 2 solutions:

```
def map_nested(f, L):
    '''(function, list) -> list
    L contains values, some of which may be nested lists. Return a new list
    with the same structure as L, but where the values are made from calling
    f on each non-list item.'''

    res = []
    for item in L:
        if isinstance(item, list):
            res.append(map_nested(f, item))
        else:
            res.append(f(item))

    return res

def map_nested2(f, L):
    res = []
    if L:
        item = L[0]
        if isinstance(item, list):
            res.append(map_nested2(f, item))
        else:
            res.append(f(item))

        res.extend(map_nested2(f, L[1:]))

    return res

for m in [map_nested, map_nested2]:
    assert m(abs, [-1, [-3, 3]]) == [1, [3, 3]]
    assert m(len, [['a', ['ab', 'abc']], 'abc']) == [[1, [2, 3]], 3]

    assert m(len, []) == []
    assert m(abs, [[], [[[]]]]) == [[], [[[]]]]
    assert m(abs, [-1, [-3, 3, [-4]], 5, -2]) == [1, [3, 3, [4]], 5, 2]
```

Question 2. [8 MARKS]

Write a `unittest` class that thoroughly tests this function; also write the code for `NoModeError`:

```
def mode(L):
    '''(list) -> object
    Return the mode (the most frequently occurring value) in L. If there
    is more than one mode, return the value that appears closest to the
    front of the list. Raise a NoModeError if L is empty.'''
```

Test case thoughts:

- length 0: NoModeError
- only one value in list: that value
- more than one mode: first occurrence
- several values, one mode: that mode
- we should test with different kinds of data

```
class NoModeError(Exception):
    pass

class TestMode(unittest.TestCase):
    def test_mode_empty(self):
        try:
            mode([])
            self.fail()
        except NoModeError:
            pass

    def test_mode_one_int_value(self):
        self.assertEqual(mode([1]), 1)

    def test_mode_two_int_modes(self):
        self.assertEqual(mode([2, 1, 3, 1, 2]), 2)

    def test_mode_several_strs_one_mode(self):
        self.assertEqual(mode([1, 'b', 'c', 1, 'b', 'b']), 'b')
```

Question 3. [8 MARKS]

Consider this code:

```
class Point(object):
    '''A 2-D Cartesian coordinate.'''

    def __init__(self, x, y):
        '''(Point, int, int) -> NoneType
        A Point at (x, y).'''

        self.x = x
        self.y = y

def are_vertical(pts):
    '''(list of Points) -> bool
    Return True if and only if all the Points in pts have the same x
    coordinates (and thus are all on a vertical line).'''

    if len(pts) <= 1:
        # Draw the memory model diagram until you reach here, then stop.
        return True
    else:
        return pts[0].x == pts[1].x and are_vertical(pts[1:])

def use_points():
    '''Demonstrate the use of Points in order to have an exciting memory model
    diagram question on a CSC148H midterm.'''
    L = [Point(1, 2)]
    assert are_vertical(L)
    # Diagram drawn at this point in the execution.
    L.append(Point(1, -2))
    assert are_vertical(L)

if __name__ == '__main__':
    use_points()
```

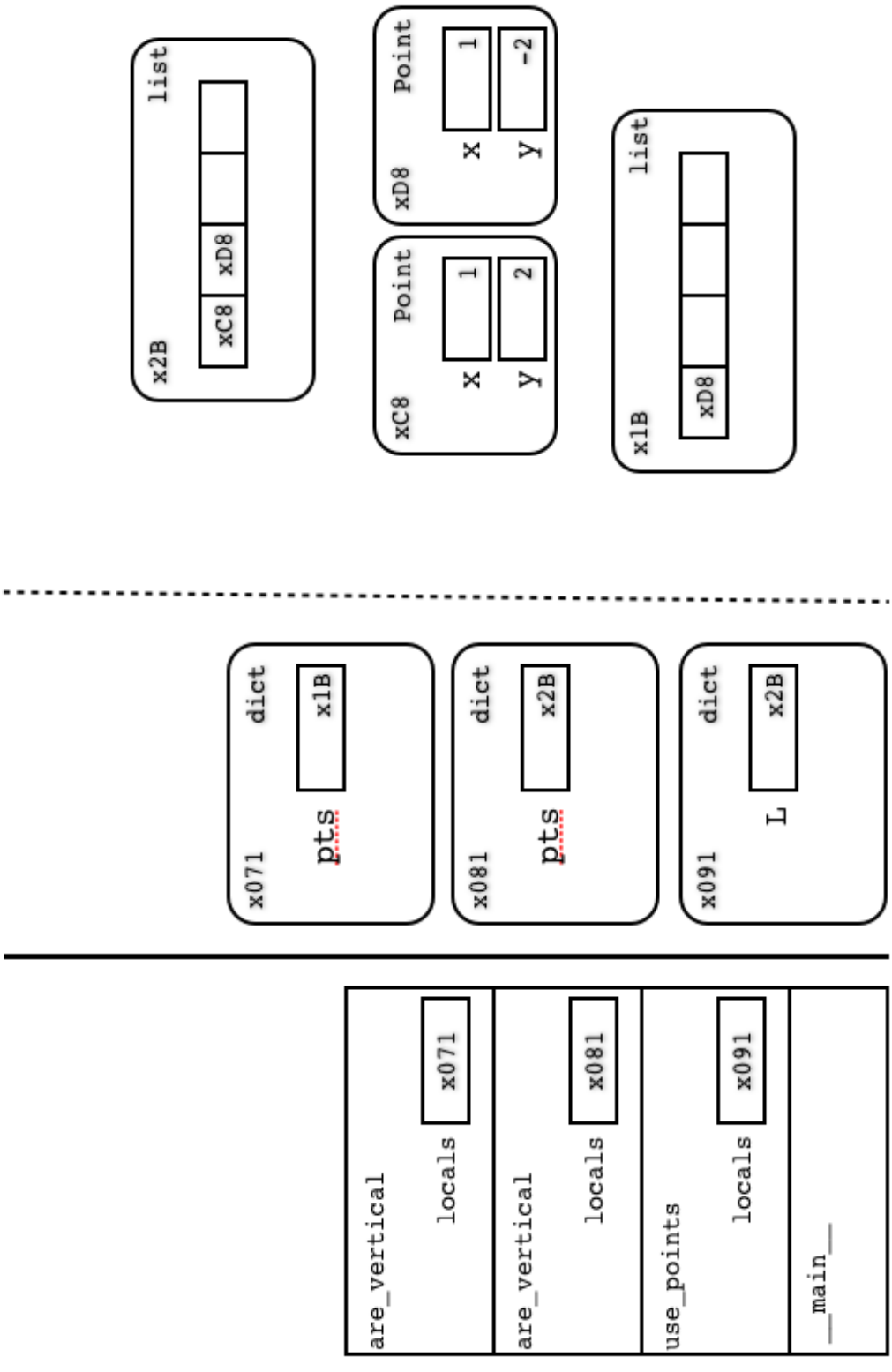
On the opposite page is a memory model diagram in the style we have used in lecture. The diagram represents the contents of memory after `use_points` has been called and this line has been reached:

```
***1:  Diagram drawn here.***
```

Continue tracing the code until this line has been reached:

```
***2:  Stop tracing when you reach here.***.
```

Create any boxes that you think are needed. (There are no global variables, so we have not shown the global namespace.)


list
x2B
xC8
xD8
Point
xD8
x
1
y
-2
Point
xC8
x
1
y
2
list
x1B
xD8
dict
x071
pts
x1B
dict
x081
pts
x2B
dict
x091
L
x2B
are_vertical
locals
x071
are_vertical
locals
x081
use_points
locals
x091
__main__