For the following problems, we will use the tree data abstraction provided in lecture:

def tree(root, subtrees=[]):
    ...

def root(t):
    ...

def subtrees(t):
    ...

def is_leaf(t):
    return not subtrees(t)
1. Write the code that represents the following tree:

```
Solution:
tree(4,
    [tree(8),
     tree(2,
      [tree(3)])
    tree(6),
    tree(1)])
```

2. Draw the tree represented by the following code:

```
tree(1,
    [tree(2,
     [tree(6)]),
     tree(4,
      [tree(12),
       tree(11)])
     tree(6)])
```

```
Solution:
```
```
3. Implement `square_tree(t)`, which takes in a tree and returns a new tree with all of the original tree’s elements squared.

```python
def square_tree(t):
    """
    >>> t1 = tree(3,
                [tree(6, []),
                 tree(4,
                      [tree(2, [])]),
                 tree(5, [])])
    >>> t2 = square_tree(t1)
    >>> root(t2)
    9
    >>> [root(branch) for branch in subtrees(t2)]
    [36, 16, 25]
    >>> root(subtrees(t2)[1])
    4
    """

    Solution:
    if is_leaf(t):
        return tree(root(t) * root(t))
    else:
        return tree(root(t) * root(t),
                    [square_tree(branch) for branch in subtrees(t)])
```
2. Mutation

1. What would Python print?

   >>> bob = [1, 2, 3, 4, 5]
   >>> bob_imposter = [1, 2, 3, 4, 5]
   >>> bob == bob_imposter

   **Solution:**
   True

   >>> bob is bob_imposter

   **Solution:**
   False

   >>> bob_imposter = bob
   >>> bob is bob_imposter

   **Solution:**
   True

   >>> bob_imposter[1] = bob
   >>> bob is bob_imposter[1]

   **Solution:**
   True

   >>> bob.append(bob_imposter)
   >>> bob_imposter[1][5][1][3]

   **Solution:**
   4
2. Draw the box and pointer diagram for the following code:

```python
alice = ['a', 'b', 'c', 'd', 'e']
santa = alice

bob = alice[1:]
bob[1] = alice
bob[3] = santa
bob_imposter = [bob[1], bob]

ella = [1, 2, 3]
santa.append(ella)
ella[1] = bob_imposter
```

Solution:
3. What would Python print?

```python
print(santa[5][1][0][:3])
```

Solution:

['a', 'b', 'c']

```python
print(alice[5][2])
```

Solution:

3

```python
print(santa[5][1][1] is santa)
```

Solution:

False

```python
print(santa[5][1][1][1] is santa)
```

Solution:

True

```python
print(santa[5][1][0] is alice)
```

Solution:

True

4. Identify the error in `increment_whole(lst)` and fix it.

```python
def increment_whole(lst):
    """Increases every element in the list by 1""
    for elem in lst:
        elem += 1
```

Solution:

```python
def increment_whole(lst):
    for i in range(len(lst)):
        lst[i] += 1
```
3 Object Oriented Programming

class Skater:
    all_tricks = ["ollie", "kickflip", "360"]
    def __init__(self, name, tricks):
        self.name = name
        self.tricks = [trick for trick in tricks if trick in all_tricks]
    def do_trick(self, trick):
        if trick in self.tricks:
            print("Woah! " + self.name + " did a " + trick + "!")
        else:
            print(self.name + " wiped out!")
    def learn(self, trick):
        if not(trick in all_tricks):
            print("Chill out bruh! No one can do that yet!")
        elif trick in self.tricks:
            print("Nah, " + self.name + " already knows that")
        else:
            self.tricks.append(trick)

class ProSkater(Skater):
    def __init__(self, name):
        Skater.__init__(self, name, Skater.all_tricks)
1. What would Python print?

```python
>>> bob = Skater("Bob", ["ollie"])
>>> bob.do_trick("ollie")

Solution:
Woah! Bob did a ollie!

>>> bob.do_trick("360")

Solution:
Bob wiped out!

>>> bob.learn("360")
>>> bob.do_trick("360")

Solution:
Woah! Bob did a 360!

>>> bob.learn("900")

Solution:
Chill out bruh! No one can do that yet!

>>> tony_hawk = ProSkater("Tony Hawk")
>>> tony_hawk.learn("kickflip")

Solution:
Nah, Tony Hawk's got this already.

>>> tony_hawk.do_trick("kickflip")

Solution:
Woah! Tony Hawk did a kickflip!
```
2. Implement `invent(self, trick)` in ProSkater, which allows a ProSkater to add a brand-new trick to the list of all tricks that any Skater can learn.

```python
def invent(self, trick):
    
    Takes in argument trick and appends it to
    Skater.all_tricks if it is not already contained in that
    list. Otherwise, prints "Nah bruh, that's too easy"
    
    >>> tony_hawk = ProSkater("Tony Hawk")
    >>> bob = Skater("Bob", ["ollie")
    >>> tony_hawk.invent("ollie")
    Nah bruh, that's too easy
    >>> tony_hawk.invent("900")
    >>> tony_hawk.do_trick("900")
    Woah! Tony Hawk did a 900!
    >>> bob.learn("900")
    >>> bob.do_trick("900")
    Woah! Bob did a 900!
    
    Solution:
    if trick in Skater.all_tricks:
        print("Nah bruh, that's too easy")
    else:
        Skater.all_tricks.append(trick)
        self.learn(trick)
```