TREES, MUTATION, OBJECT-ORIENTED PROGRAMMING AND INHERITANCE

COMPUTER SCIENCE 61A

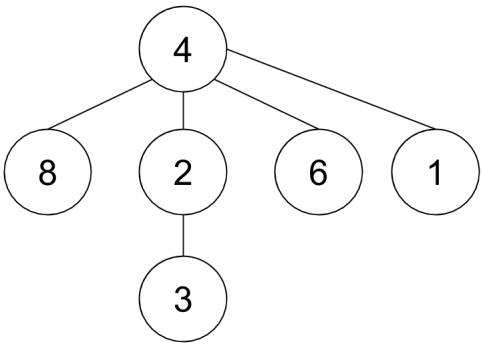
July 18 to July 24, 2015

1 Trees

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:



2. Draw the tree represented by the following code:

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

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

```
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
    """
```

2 Mutation

1. What would Python print?

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

>>> bob is bob_imposter

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

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

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

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

```
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
```

3. What would Python print?

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

print(alice[5][2])

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

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

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

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

```
def increment_whole(lst):
    """Iterates through a list and returns None"""
    for elem in lst:
        elem += 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?

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


>>> bob.do_trick("360")


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


>>> bob.learn("900")


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


>>> tony_hawk.do_trick("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.

```
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!
    """
```