RECURSION AND HIGHER ORDER FUNCTIONS

COMPUTER SCIENCE MENTORS 61A

February 15 to February 19, 2016

1 Lists

1. Draw box-and-pointer diagrams for the following.

```
>>> a = [1, 2, 3]
>>> a

>>> a[2]

>>> b = a
>>> a = a + [4, 5]
>>> a

>>> b

>>> c = a
>>> a = [4, 5]
>>> a

>>> c
```

2. Write a function that takes in a list `nums` and returns a new list with only the primes from `nums`. Assume that `is_prime(n)` is defined. You may use a `while` loop, a `for` loop, or a list comprehension.

```
def all_primes(num):
```

2 Data Abstraction

1. The following is an **Abstract Data Type (ADT)** for elephants. Each elephant keeps track of its name, age, and whether or not it can fly. Given our provided constructor, fill out the selectors:

```
def elephant(name, age, can_fly):
    """
    Takes in a string name, an int age, and a boolean can_fly.
    Constructs an elephant with these attributes.
    >>> dumbo = elephant("Dumbo", 10, True)
    >>> elephant_name(dumbo)
        "Dumbo"
    >>> elephant_age(dumbo)
        10
    >>> elephant_can_fly(dumbo)
        True
    """
    return [name, age, can_fly]
def elephant_name(e):




def elephant_age(e):




def elephant_can_fly(e):
```

2. This function returns the correct result, but there's something wrong about its implementation. How do we fix it?

```
def elephant_roster(elephants):
    """
    Takes in a list of elephants and returns a list of their
       names.
    """
    result = []
    for elephant in elephants:
        result = result + [elephant[0]]
    return result
```

3. Fill out the following constructor for the given selectors.

```
def elephant(name, age, can_fly):
```

```
def elephant_name(e):
    return e[0][0]
def elephant_age(e):
    return e[0][1]
def elephant_can_fly(e):
    return e[1]
```

4. How can we write the fixed `elephant_roster` function for the constructors and selectors in the previous question?

5. **(Optional)** Fill out the following constructor for the given selectors.

```
def elephant(name, age, can_fly):
    """
    >>> chris = elephant("Chris Martin", 38, False)
    >>> elephant_name(chris)
        "Chris Martin"
    >>> elephant_age(chris)
        37
    >>> elephant_can_fly(chris)
        False
    """
    def select(command)
```

```
    return select
def elephant_name(e):
    return e("name")
def elephant_age(e):
    return e("age")
def elephant_can_fly(e):
    return e("can_fly")
```

3 Trees

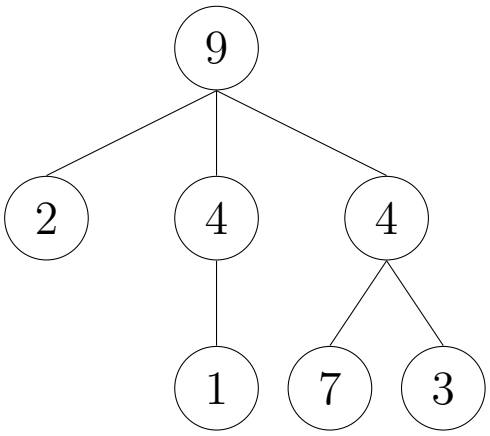
Things to remember

```
def tree(root, branches=[]): # ALWAYS OUTPUTS A TREE
    for branch in branches:
        assert is_tree(branch), 'branches must be trees'
    return [root] + list(branches)

def root(t): # ALWAYS OUTPUTS A NUMBER
    return t[0]

def branches(t): # ALWAYS OUTPUTS A LIST
    return t[1:]
```

1. Draw the tree that is created by the following statement:

```
tree(4,
     [tree(5, []),
      tree(2,
         [tree(2, []),
          tree(1, [])]),
      tree(1, []),
      tree(8,
         [tree(4, [])])])
```

2. Construct the following tree and save it to the variable `t`.



3. What would this output?

```
>>> root(t)

>>> branches(t)[2]

>>> branches(branches(t)[2])[0]
```

4. Write the Python expression to get the integer 2 from t.

5. Write the function `sum_of_nodes` which takes in a tree and outputs the sum of all the elements in the tree.

```
def sum_of_nodes(t):
    """
    >>> t = Tree(...) # Tree from question 2.
    >>> sum_of_nodes(t) # 9 + 2 + 4 + 4 + 1 + 7 + 3 = 30
    30
    """
```