

# DATA ABSTRACTION AND TREES

---

CS 61A GROUP MENTORING

July 5, 2018

---

## 1 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.  
    """  
    return [elephant[0] for elephant in elephants]
```

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)  
        38  
    >>> 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")
```

---

## 2 Trees

---

**Things to remember:**

```
def tree(label, branches=[]):  
    return [label] + list(branches)
```

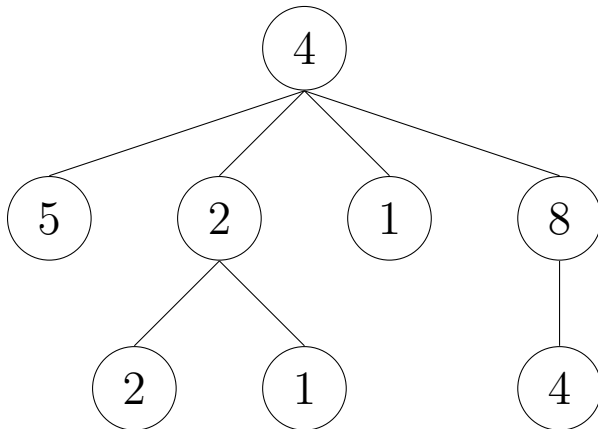
```
def label(tree):  
    return tree[0]
```

```
def branches(tree):  
    return tree[1:] #returns a list of branches
```

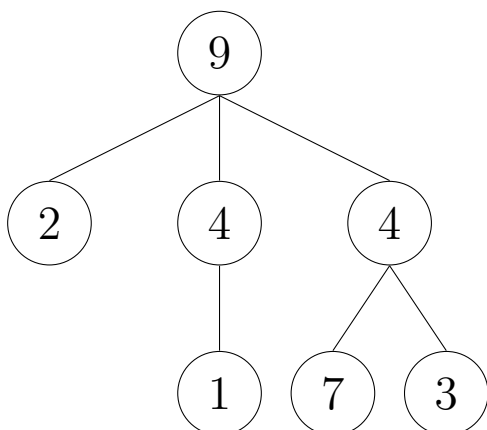
As shown above, the tree constructor takes in a label and a list of branches (which are themselves trees).

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

This creates a tree that looks like this:



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



2. What do the following expressions evaluate to? If the expressions evaluates to a tree, format your answer as `tree(..., ...)`. (Note that the python interpreter wouldn't display trees like this. We ask you to do this in order to think about trees as an ADT instead of worrying about their implementation.)

```
>>> label(t)
```

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

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

3. Write the Python expression to return the integer 2 from `t`.

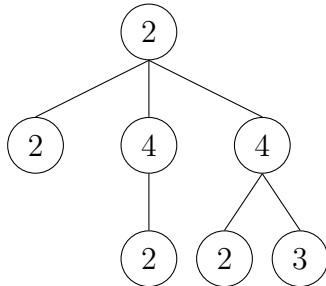
4. Write the function `sum_of_nodes` which takes in a tree and outputs the sum of all the elements in the tree.

```

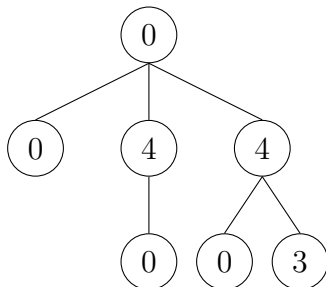
def sum_of_nodes(t):
    """
    >>> t = tree(...) # Tree from question 2.
    >>> sum_of_nodes(t) # 9 + 2 + 4 + 4 + 1 + 7 + 3 = 30
    30
    """
  
```

5. Write a function, `replace_x` that takes in a tree, `t`, and returns a new tree with all labels `x` replaced with 0.

For example, if we called `replace_x(t, 2)` on the following tree:



We would expect it to return



```
def replace_x(t, x):
```

6. Challenge: Write a function that returns true only if there exists a path from root to leaf that contains at least `n` instances of `elem` in a tree `t`.

```
def contains_n(elem, n, t):
    """
    >>> t1 = tree(1, [tree(1, [tree(2)])])
    >>> contains(1, 2, t1)
    True
    >>> contains(2, 2, t1)
    False
    >>> contains(2, 1, t1)
    True
    >>> t2 = tree(1, [tree(2), tree(1, [tree(1), tree(2)])])
    >>> contains(1, 3, t2)
    True
    >>> contains(2, 2, t2) # Not on a path
    False
    """
    if n == 0:

        return True

    elif _____:

        return _____

    elif label(t) == elem:

        return _____

    else:

        return _____
```