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:

```python
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):

    Solution:
    return e[0]
```


def elephant_age(e):
    Solution:
    return e[1]

def elephant_can_fly(e):
    Solution:
    return e[2]

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]

    Solution: elephant[0] is a Data Abstraction Violation (DAV). We should use a selector instead. The corrected function looks like:
def elephant_roster(elephants):
    return [elephant_name(elephant) for elephant in elephants]

3. Fill out the following constructor for the given selectors.
def elephant(name, age, can_fly):

    Solution:
    return [[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?

**Solution:** No change is necessary to fix `elephant_roster` since using the `elephant` selectors “protects” the roster from constructor definition changes.

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

```python
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)
    if command == "name":
        return name
    elif command == "age":
        return age
    elif command == "can_fly":
        return can_fly
    return "Breaking abstraction barrier!"

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

```
  4
 /\   \
 5 2  1  8
/  \
 2  1
```

CS61A Summer 2018: Alex Stennet and Chris Allsman, with content by Jennie Chen and Ajay Raj, Alex Yang, Annie Tang, Brandon Fong, Catherine Han, Danelle Nachum, Elaine Park, Hyun Jae Moon, Kevin Tsang, Lindsay Yang, Michelle Cheung, Ryan Moughan, Ryan Roggenkemper, Shreya Sahoo, Surya Duggirala, Thomas Zhang
1. Construct the following tree and save it to the variable \( t \).

```
9
 / \
2  / \
4 / \
1 7 3
```

Solution:

\[
t = \text{tree}(9, [\text{tree}(2, []), \text{tree}(4, [\text{tree}(1, []), \text{tree}(7, []), \text{tree}(3, [])])])
\]

2. What do the following expressions evaluate to? If the expressions evaluates to a tree, format your answer as \text{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)
```

Solution: 9

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

Solution:

\[
\text{tree}(4, [\text{tree}(7), \text{tree}(3)])
\]

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

Solution:

\[
\text{tree}(7)
\]
3. Write the Python expression to return the integer 2 from \( t \).

**Solution:**

\[
\text{label(branches(t)[0])}
\]

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

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

**Solution:**

```python
total = label(t)
for branch in branches(t):
    total += sum_of_nodes(branch)
return total
```

Alternative solution:

```python
return label(t) +
        \sum([\text{sum_of_nodes(b)} \text{ for } b \text{ in branches(t)}])
```
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:

```
2
/  \
2   4   4
   /     \
  2   2   3
```

We would expect it to return

```
0
/  \
0   4   4
   /     \
  0   0   3
```

```python
def replace_x(t, x):
    new_label = label(t)
    if new_label == x:
        new_label = 0
    new_branches = [replace_x(b, x) for b in branches(t)]
    return tree(label, new_branches)
```
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.

```python
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

    if ________________________________:
        return __________________________

    elif label(t) == elem:
        return ____________________________

    else:
        return ____________________________
```
Solution:

```python
if n == 0:
    return True
elif is_leaf(t):
    return n == 1 and label(t) == elem
elif label(t) == elem:
    return True in [contains_n(elem, n - 1, b) for b in branches(t)]
else:
    return True in [contains_n(elem, n, b) for b in branches(t)]
```