1. Linked Lists

empty = 'X'

def link(first, rest=empty):
    return [first, rest]

def first(s):
    return s[0]

def rest(s):
    return s[1]

1.1 What would Python display?

s = link(1, link(2, link(3)))

(a) first(s)

1

(b) rest(s)

[2, [3, 'X']]

(c) rest(first(s))

TypeError - type int

(d) first(rest(s))

2

(e) rest(rest(s))

[3, 'X']

(f) first(rest(rest(s)))

3
1.2 Define the function, `get_item`, which returns the value at index `i` in the linked list, `s`. If the index is greater than the length of the list, return None.

```python
def get_item(s, i):
    """
    >>> link1 = link(1, empty)
    >>> link21 = link(2, link1)
    >>> link421 = link(4, link21)
    >>> get(link421, 0)
    4
    >>> get(link421, 2)
    1
    >>> get(link421, 999) # returns None
    """
    if s is empty:
        return None
    elif i == 0:
        return first(s)
    return get_item(rest(s), i - 1)
```

1.3 Implement `every_other`, which returns a list containing every other element starting from the second.

```python
def every_other(s):
    """
    >>> s = link(1, link(2, link(3, link(4, link(5, empty))))))
    >>> print_link(s)
    <1 2 3 4 5>
    >>> print_link(every_other(s))
    <2 4>
    """
    if s == empty or rest(s) == empty:
        return empty
    return link(first(rest(s)), every_other(rest(rest(s))))
```
1.4 Implement `merge`, which takes in two sorted linked lists and returns a sorted linked list that contains all the elements of both.

```python
def merge(lst1, lst2):
    """
    >>> l1 = link(2, link(2, link(5, empty)))
    >>> l2 = link(1, link(5, link(6, empty)))
    >>> lst = merge(l1, l2)
    >>> print_link(lst):
    <1 2 2 5 5 6>
    """

    if lst1 == empty:
        return lst2
    elif lst2 == empty:
        return lst1
    elif first(lst1) < first(lst2):
        return link(first(lst1), merge(rest(lst1), lst2))
    else:
        return link(first(lst2), merge(lst1, rest(lst2)))
```
2 Trees

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

def root(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
```

2.1 Draw the tree that is created by the expression to the right:

```
4
  5  2  1  8
  2   1  4
```

2.2 Assign the name, `t`, to the tree to the right.

```
t = tree(9, [tree(2),
            tree(4, [tree(1)]),
            tree(4, [tree(7),
                      tree(3)]))
```

2.3 What would Python display?

(a) `root(t)`

```
9
```

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

```
[4, [7], [3]]
```

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

```
[7]
```

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

```
root(branches(t)[0])
```
2.5 Define the function `tree_sum` which takes in a tree and outputs the sum of all the values in the tree.

```python
def tree_sum(t):
    """
    >>> t = tree(...)  # Example from earlier
    >>> tree_sum(t)  # 9 + 2 + 4 + 4 + 1 + 7 + 3 = 30
    30
    """

    total = root(t)
    for branch in branches(t):
        total += tree_sum(branch)
    return total
```

Alternatively:

```python
return root(t) + sum([tree_sum(b) for b in branches(t)])
```

2.6 Define the function `factor_tree` which returns a factor tree. Recall that in a factor tree, multiplying the leaves together is the prime factorization of the root, n.

```
2
  
6
  
3
```

```
12
  
6
  
2
  
3
```

```python
def factor_tree(n):

    for i in range(2, n):
        if n % i == 0:
            return tree(n, [factor_tree(i), factor_tree(n // i)])
    return tree(n)
```

2.7 Define the function `count` which counts the number of instances of a value in the given tree.

```python
def count(t, value):

    if value != root(t):
        return sum([count(b, value) for b in branches(t)])
    else:
        return 1 + sum([count(b, value) for b in branches(t)])
```