# 1 Linked Lists in OOP

Linked lists are data abstractions that can have multiple implementations. Previously, we saw linked lists implemented using Python lists. Today, we will look at linked lists implemented using Object-Oriented Programming. Here it is:

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        assert rest is Link.empty or isinstance(rest, Link)
        self.first = first
        self.rest = rest
    def __getitem__(self, i):
        if i == 0:
            return self.first
        return self.rest[i-1]
    def __len__(self):
        return 1 + len(self.rest)
```

When we implemented linked lists using Python lists, we called `first(lnk)` and `rest(lnk)` to access the `first` and `rest` elements. This time, we can write `lnk.first` and `lnk.rest` instead. In the former, we could access the elements, but we could not modify them. In the latter, we can access and also modify the elements. In other words, linked lists implemented using OOP is mutable.

In addition to the constructor `__init__`, we have the special Python methods `__getitem__` and `__len__`. Note that any method that begins and ends with two underscores is a special Python method. Special Python methods may be invoked using built-in functions and special notation. The built-in Python element selection operator, as in `lst[i]`, invokes `lst.__getitem__(i)`. Likewise, the built-in Python function `len`, as in `len(lst)`, invokes `lst.__len__()`.

# Questions

1.1 Write a function `remove_duplicates` that takes as input a sorted linked list of integers, `lnk`, and mutates `lnk` so that all duplicates are removed.

```
def remove_duplicates(lnk):
    """
    >>> lnk = Link(1, Link(1, Link(1, Link(1, Link(5)))))
    >>> unique = remove_duplicates(lnk)
    >>> len(unique)
    2
    >>> len(lnk)
    2
    """
```

1.2 Define `reverse`, which takes in a linked list and reverses the order of the links. The function may *not* return a new list; it must mutate the original list. Return a pointer to the head of the reversed list.

```
def reverse(lnk):
    """
    >>> a = Link(1, Link(2, Link(3)))
    >>> r = reverse(a)
    >>> r.first
    3
    >>> r.rest.first
    2
    """
```

# 2 Trees in OOP

Trees are also data abstractions that can have multiple implementations. Previously, we implemented the tree abstraction using Python lists. Let's look at another implementation using objects instead. With this implementation, we can easily specify specialized tree types, such as binary trees, using inheritance.

```
class Tree:
    def __init__(self, root, branches=[]):
        for b in branches:
            assert isinstance(b, Tree)
        self.root = root
        self.branches = branches

    def is_leaf(self):
        return not self.branches
```

Notice that with this implementation we can mutate the root of a tree by reassigning `tree.root`. In the previous implementation using lists, this was not possible, because the abstraction barrier prevented us from seeing how the tree was implemented.

## Questions

2.1 Consider the following definitions and assignments and determine what Python would output for each of the calls below *if they were evaluated in order.*

```
>>> t0 = Tree(0)
>>> t0.root
```

```
>>> t0.branches
```

```
>>> t1 = Tree(0, [1, 2]) # Is this a valid tree?
```

```
>>> t2 = Tree(0, [Tree(1), Tree(2, [Tree(3)])])
>>> t2.branches[0]
```

```
>>> t2.branches[1].branches[0].root
```

2.2 Assuming that every value in `t` is a number, let's define `average(t)`, which returns the average of all the values in `t`.

```
def average(t):
    """
    Returns the average value of all the nodes in t.
    >>> t0 = Tree(0, [Tree(1), Tree(2, [Tree(3)])])
    >>> average(t0)
    1.5
    >>> t1 = Tree(8, [t0, Tree(4)])
    >>> average(t1)
    3.0
    """
```

2.3 Write a function that combines the values of two trees `t1` and `t2` together with the `combiner` function. Assume that `t1` and `t2` have identical structure. This function should return a new tree.

```
def combine_tree(t1, t2, combiner):
    """
    >>> a = Tree(1, [Tree(2, [Tree(3)])])
    >>> b = Tree(4, [Tree(5, [Tree(6)])])
    >>> combined = combine_tree(a, b, mul)
    >>> combined.root
    4
    >>> combined.branches[0].root
    10
    """
```

# 3 Binary Search Trees

A Binary Search Tree (BST) is a special kind of tree that satisfies the following properties:

- Every node of a BST has at most two branches called `left` and `right`. The branches are also BSTs.
- For every node, the left branch's value is less than or equal to its parent's value.
- For every node, the right branch's value is greater than its parent's value.

```
# Binary Search Tree (BST) Class
class BST:
    empty = ()
    def __init__(self, root, left=empty, right=empty):
        assert left is BST.empty or isinstance(left, BST)
        assert right is BST.empty or isinstance(right, BST)

        self.root = root
        self.left = left
        self.right = right

        if left is not BST.empty:
            assert left.max <= root
        if right is not BST.empty:
            assert root < right.min

    @property
    def max(self):
        if self.right is BST.empty:
            return self.root
        return self.right.max

    @property
    def min(self):
        if self.left is BST.empty:
            return self.root
        return self.left.min
```

# Questions

3.1 Define a function `insert` that takes in a `BinTree`, `bst`, and a number, `n`, and returns a new `BinTree` that is a **copy** of `bst` with a new node inserted. `insert` should place the new node as a leaf in the correct position. If `t` is the `BinTree` on the left, then calling `insert(t, 3)` will return the `BinTree` on the right.

```
        4                                   4
      /   \                               /   \
     2     5             ->              2     5
    /                                  /   \
   1                                  1     3
```

```
def insert(bst, n):
    """
    >>> bst = BinTree(4, BinTree(2, BinTree(1)), BinTree(5))
    >>> new_bst = insert(bst, 3)
    >>> new_bst.left.right.root
    3
    """
```