# Mutability

# Announcements

# Trees Review

# List Mutation

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B13,%2015,%2017,%2019%5D%0Afor%20i%20in%20range%28len%28s%29%29%3A%0A%20%20%20%20s%5Bi%5D%20%3D%20s%5B-%28i%2B1%29%5D&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

# List of Lists

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B1,%20%5B2,%203%5D,%204%5D%0At%20%3D%20s%5B1%5D%0At%5B1%5D%20%3D%20s%0Aprint%28s%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

# List of Lists 2

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B1,%20%5B2,%203%5D,%204%5D%0At%20%3D%20s%5B1%5D%0Au%20%3D%20%5Bt,%205%5D%0Au.append%286%29%0Au%5B0%5D.append%287%29%0At%5B1%5D%20%3D%208%0Aprint%28s%29%0Aprint%28t%29%0Aprint%28u%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

# List of Lists 2 Modified

(Demo)

https://pythontutor.com/cp/composingprograms.html#code=s%20%3D%20%5B1,%20%5B2,%203%5D,%204%5D%0At%20%3D%20s%5B1%5D%0Au%20%3D%20%5Blist%28t%29,%205%5D%0Au.append%286%29%0Au%5B0%5D.append%287%29%0At%5B1%5D%20%3D%208%0Aprint%28s%29%0Aprint%28t%29%0Aprint%28u%29&cumulative=true&curInstr=0&mode=display&origin=composingprograms.js&py=3&rawInputLstJSON=%5B%5D

# Building Lists Using Append

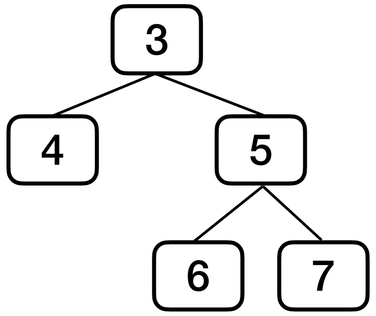
```
def sums(n, m):
    """Return lists that sum to n containing positive numbers up to m that
    have no adjacent repeats, for n > 0 and m > 0.

    >>> sums(5, 1)
    []
    >>> sums(5, 2)
    [[2, 1, 2]]
    >>> sums(5, 3)
    [[1, 3, 1], [2, 1, 2], [2, 3], [3, 2]]
    >>> sums(5, 5)
    [[1, 3, 1], [1, 4], [2, 1, 2], [2, 3], [3, 2], [4, 1], [5]]
    >>> sums(6, 3)
    [[1, 2, 1, 2], [1, 2, 3], [1, 3, 2], [2, 1, 2, 1], [2, 1, 3], [2, 3, 1], [3, 1, 2], [3, 2, 1]]
    """
    result = []
    for k in range(1, ____min(m + 1, n)____): # k is the first number of a list
        for rest in ____sums(n-k, m)____:
            if rest[0] != k:
                result.append(____[k] + rest____)  # build a list out of k and rest
    if n <= m:
        result.append([n])
    return result
```

# Building Lists of Branches

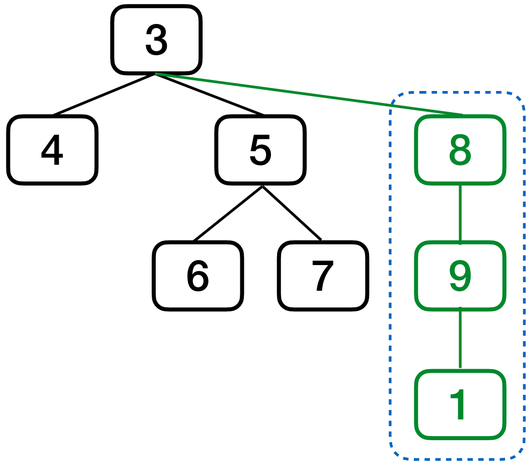
# Example: Make Path

A list describes a path if it contains labels along a path from the root of a tree. Implement `make_path`, which takes a tree `t` with unique labels and a list `p` that starts with the root label of `t`. It returns the tree `u` with the fewest nodes that contains all the paths in `t` as well as a (possibly new) path `p`.

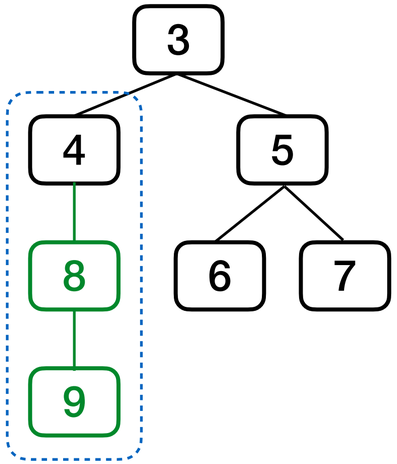
`t1`



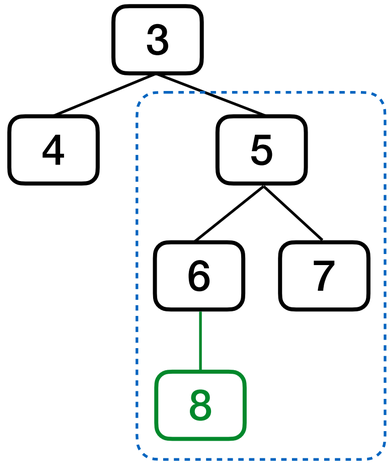

`make_path(t1, [3,8,9,1])`




`make_path(t1, [3,4,8,9])`




`make_path(t1, [3,5,6,8])`



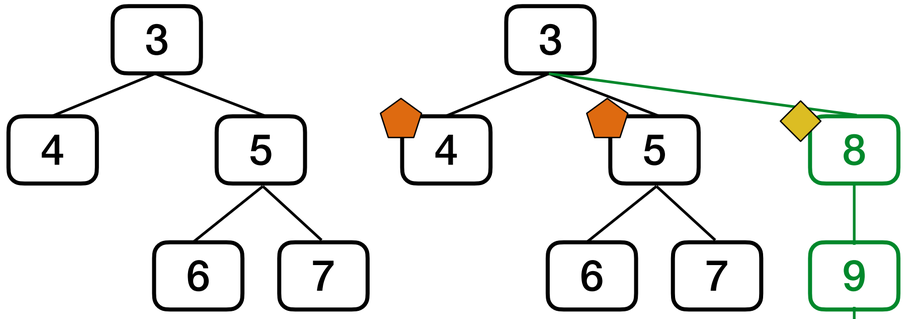

Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

Special case: if no branch starts with p[1], then a leaf labeled p[1] needs to be added

# Example: Make Path

A list describes a path if it contains labels along a path from the root of a tree. Implement `make_path`, which takes a tree `t` with unique labels and a list `p` that starts with the root label of `t`. It returns the tree `u` with the fewest nodes that contains all the paths in `t` as well as a (possibly new) path `p`.

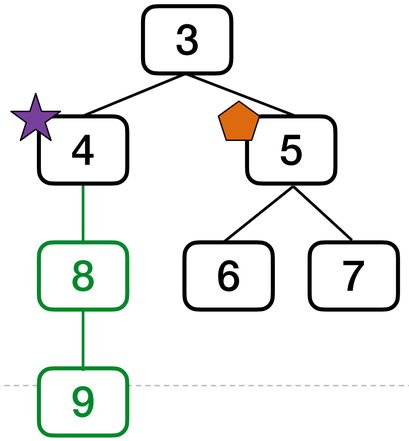
t1




make_path(t1, [3,8,9,1])




make_path(t1, [3,4,8,9])




```
def make_path(t, p):
    "Return a tree like t also containing path p."
    assert p[0] == label(t), 'Impossible'
    if len(p) == 1:
        return t
    new_branches = []
    found_p1 = False
    for b in branches(t):
        if label(b) == p[1]:
            new_branches.append(make_path(b, p[1:]))
            found_p1 = True
        else:
            new_branches.append(b)
    if not found_p1:
        new_branches.append(make_path(tree(p[1]), p[1:]))
    return tree(label(t), new_branches)
```

Break: 5 minutes

# Mutation and Identity

# Sameness and Change

- As long as we never modify objects, a compound object is just the totality of its pieces
- This view is no longer valid in the presence of change
- A compound data object has an "identity" in addition to the pieces of which it is composed
- A list is still "the same" list even if we change its contents. It has the same "identity."
- Conversely, we could have two lists that happen to have the same contents, but are different. They have different "identities."
- Being identical is a *stronger* assertion than having the same contents

```
>>> a = [10]
>>> b = a
>>> a == b
True
>>> a.append(20)
>>> a
[10, 20]
>>> b
[10, 20]
>>> a == b
True
```

```
>>> a = [10]
>>> b = [10]
>>> a == b
True
>>> b.append(20)
>>> a
[10]
>>> b
[10, 20]
>>> a == b
False
```

# Identity Operators

**Identity**

`<exp0> is <exp1>`

evaluates to True if both <exp0> and <exp1> evaluate to the same object

**Equality**

`<exp0> == <exp1>`

evaluates to True if both <exp0> and <exp1> evaluate to equal values

**Identical objects are always equal values**

(Demo)

# Mutation and Names

If multiple names refer to the same mutable object (directly or indirectly), then a change to that object is reflected in the value of all of these names.

What numbers are printed (and how many of them)?

```
s = [2, 7, [1, 8]]
t = s[2]
t.append([2])
e = s + t
t[2].append(8)
print(e)
```