

Mutability

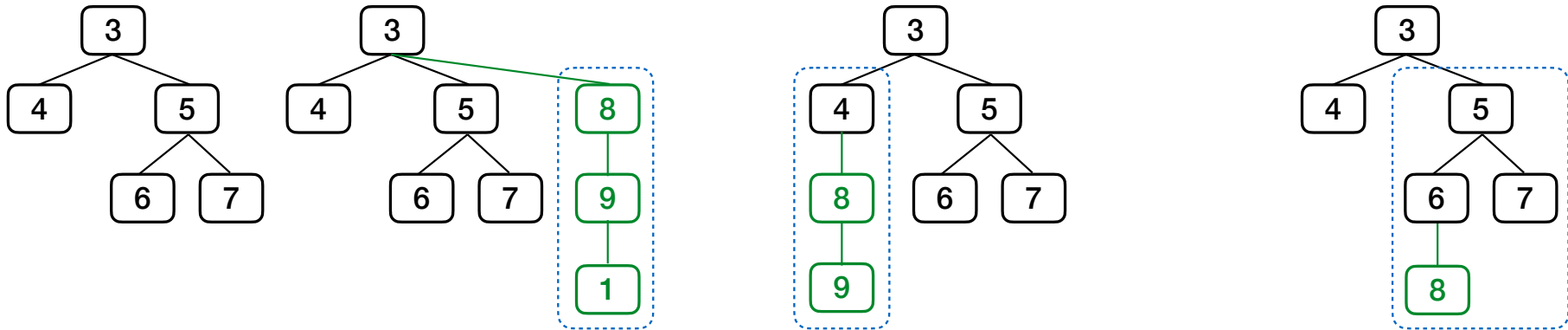
Announcements

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])`



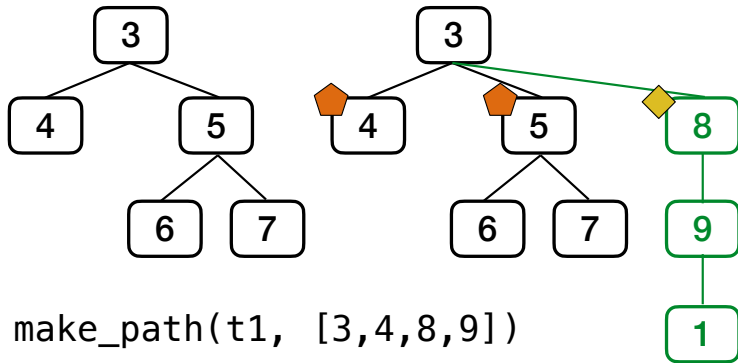
Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

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

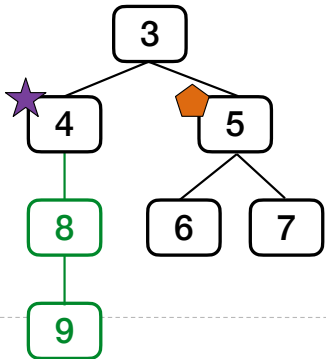
Example: Make Path

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)
```

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- This view is no longer valid in the presence of change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
- Conversely, we could have two lists that happen to have the same contents, but are different

```
>>> 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)
```