Iterators

Announcements

Tuples

(Demo)

Iterators

Iterators

A container can provide an iterator that provides access to its elements in order

iter(iterable): Return an iterator over the elements of an iterable value

next(iterator): Return the next element in an iterator

```
>>> s = [3, 4, 5]
>>> t = iter(s)
>>> next(t)
3
>>> next(t)
4
>>> u = iter(s)
>>> next(u)
3
>>> next(t)
5
>>> w = u
>>> next(w)
4
```

(Demo)

Discussion Question

What will be printed?

```
a = [1, 2, 3]
b = [a, 4]
c = iter(a)
d = c
print(next(c))
print(next(d))
print(b)
```

Map Function

Map

`map(func, iterable)`: Make an iterator over the return values of calling `func` on each element of the `iterable`.

(Demo)

all and any

all(s: iterable) -> bool: Return True if bool(x) is True for all values x in s

any(s: iterable) -> bool: Return True if bool(x) is True for at least one value x in s

All values are True

```
>>> all([True, True, True])
True
```

All values evaluate to True

```
>>> all([[1], [1, 2], [1, 2, 3]])
True
```

Empty list evaluates to False

```
>>> all([[], [1], [1, 2, 3]])
False
```

At least one value evaluates to True

```
def is_leafy(t) -> bool:
    """Return true if all of t's branches
       are leaves. """
    return _____([_____________________________])
```

- Get the list of branches for the tree: branches(t)
- Get the branch at index i: branches(t)[0]
- Determine whether the tree is a leaf: is_leaf(t)

all and any

all(s: iterable) -> bool: Return True if all values in s evaluate to True
Returns as soon as a False value is reached

any(s: iterable) -> bool: Return True if at least one value in s evaluates to True
Returns as soon as a True value is reached

What's printed when evaluating:

```
x = all(map(print, range(-3, 3)))
```

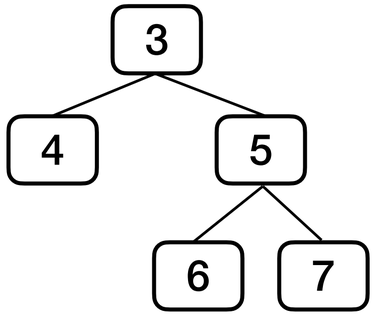
Why?

- print(-3) returns None after displaying -3
- None is a false value
- all([None, ...]) is False for any ...
- The map iterator never needs to advances beyond -3

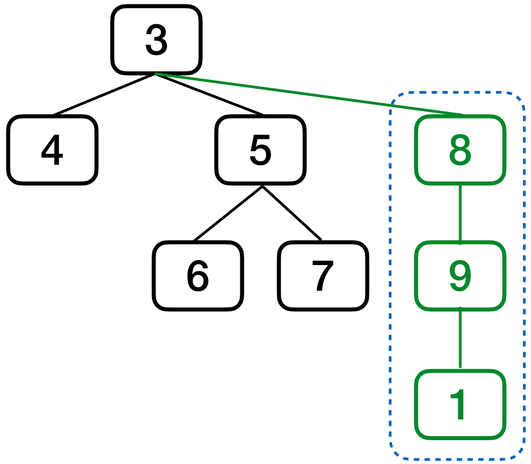
Example: Make Path

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`.

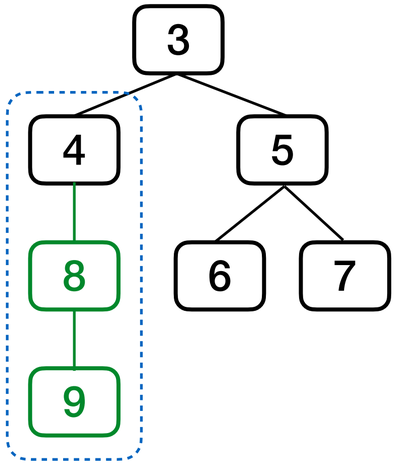
`t1`



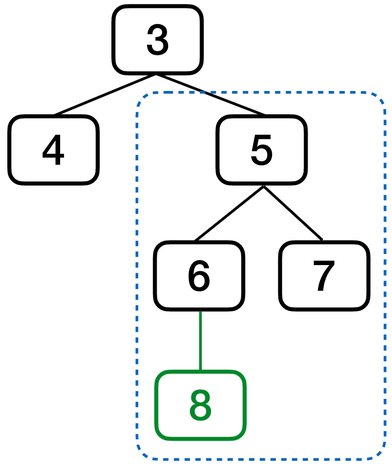
`make_path(t1, [3,8,9,1])`



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



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

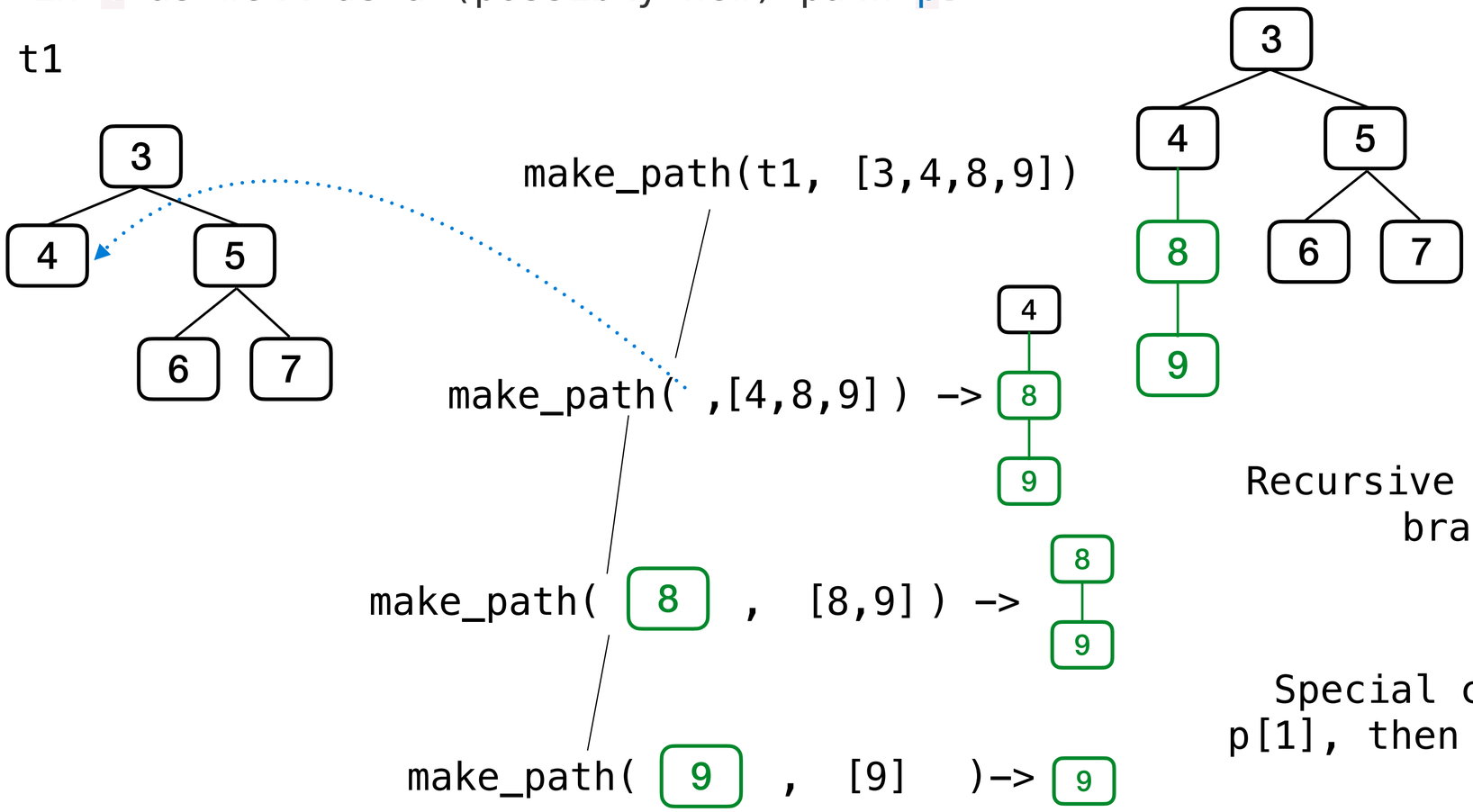


Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

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`.



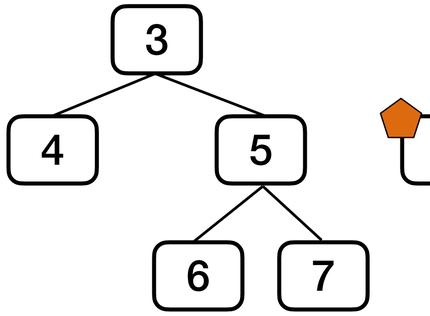
Recursive idea: `make_path(b, p[1:])` is a branch of the tree returned by `make_path(t, p)`

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

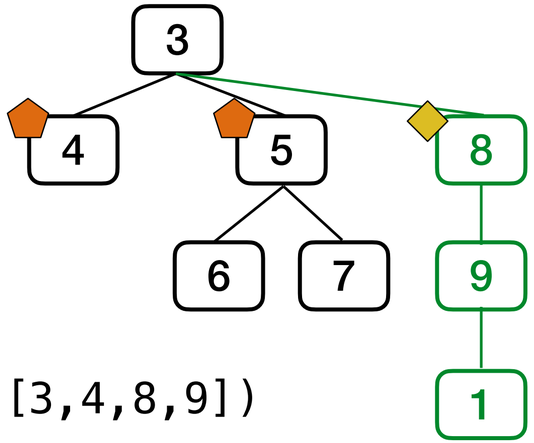
Example: Make Path

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`.

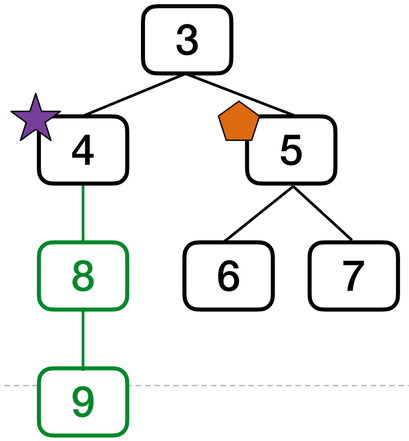
t1



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)
```

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