1-collections

September 4, 2023

1 Python review: Basic collections of values

This notebook continues the review of Python basics. The focus here is on basic collections: tuples, dictionaries, and sets.

Exercise 0 (`minmax_test`: 1 point). Complete the function `minmax(L)`, which takes a list `L` and returns a pair---that is, 2-element Python tuple, or "2-tuple"---whose first element is the minimum value in the list and whose second element is the maximum. For instance:

```
minmax([8, 7, 2, 5, 1]) == (1, 8)
```

```
In [1]: def minmax(L):
            assert hasattr(L, "__iter__")
            ###
            ### YOUR CODE HERE
            print('Min of L = , Max of L = ',min(L),max(L))
            return(min(L),max(L))
            ###
```

```
In [2]: # `minmax_test`: Test cell

        L = [8, 7, 2, 5, 1]
        mmL = minmax(L)
        mmL_true = (1, 8)
        print("minmax({}) -> {} [True: {}]".format(L, mmL, mmL_true))
        assert type(mmL) is tuple and mmL == (1, 8)

        from random import sample
        L = sample(range(1000), 10)
        mmL = minmax(L)
        L_s = sorted(L)
        mmL_true = (L_s[0], L_s[-1])
        print("minmax({}) -> {} [True: {}]".format(L, mmL, mmL_true))
        assert mmL == mmL_true

        print("\n(Passed!)")
```

```
Min of L = , Max of L =  1 8
minmax([8, 7, 2, 5, 1]) -> (1, 8) [True: (1, 8)]
```

```
Min of L = , Max of L =  49 969
minmax([565, 770, 299, 671, 703, 459, 690, 49, 528, 969]) -> (49, 969) [True: (49, 969)]

(Passed!)
```

Exercise 1 (`remove_all_test`: 2 points). Complete the function `remove_all(L, x)` so that, given a list `L` and a target value `x`, it returns a *copy* of the list that excludes *all* occurrences of `x` but preserves the order of the remaining elements. For instance:

```
remove_all([1, 2, 3, 2, 4, 8, 2], 2) == [1, 3, 4, 8]
```

Note. Your implementation should *not* modify the list being passed into `remove_all`.

```
In [3]: def remove_all(L, x):
            assert type(L) is list and x is not None
            ###
            ### YOUR CODE HERE
            new_L = L[:]
            while (x in new_L):
                new_L.remove(x)
            return new_L
            ###
```

```
In [4]: # `remove_all_test`: Test cell
        def test_it(L, x, L_ans):
            print("Testing `remove_all({}, {})`...".format(L, x))
            print("\tTrue solution: {}".format(L_ans))
            L_copy = L.copy()
            L_rem = remove_all(L_copy, x)
            print("\tYour computed solution: {}".format(L_rem))
            assert L_copy == L, "Your code appears to modify the input list."
            assert L_rem == L_ans, "The returned list is incorrect."

        # Test 1: Example
        test_it([1, 2, 3, 2, 4, 8, 2], 2, [1, 3, 4, 8])

        # Test 2: Random list
        from random import randint
        target = randint(0, 9)
        L_input = []
        L_ans = []
        for _ in range(20):
            v = randint(0, 9)
            L_input.append(v)
            if v != target:
                L_ans.append(v)
        test_it(L_input, target, L_ans)

        print("\n(Passed!)")
```

```
Testing `remove_all([1, 2, 3, 2, 4, 8, 2], 2)`...
        True solution: [1, 3, 4, 8]
        Your computed solution: [1, 3, 4, 8]
Testing `remove_all([1, 0, 2, 1, 6, 2, 3, 1, 2, 6, 6, 1, 7, 8, 4, 3, 6, 3, 2, 1], 7)`...
        True solution: [1, 0, 2, 1, 6, 2, 3, 1, 2, 6, 6, 1, 8, 4, 3, 6, 3, 2, 1]
        Your computed solution: [1, 0, 2, 1, 6, 2, 3, 1, 2, 6, 6, 1, 8, 4, 3, 6, 3, 2, 1]

(Passed!)
```

Exercise 2 (`compress_vector_test`: 2 points). Suppose you are given a vector, `x`, containing real values that are mostly zero. For instance:

```
x = [0.0, 0.87, 0.0, 0.0, 0.0, 0.32, 0.46, 0.0, 0.0, 0.10, 0.0, 0.0]
```

Complete the function, `compress_vector(x)`, so that returns a dictionary `d` with two keys, `d['inds']` and `d['vals']`, which are lists that indicate the position and value of all the *non-zero* entries of `x`. For the previous example,

```
d['inds'] = [1, 5, 6, 9]
d['vals'] = [0.87, 0.32, 0.46, 0.10]
```

Note 1. Your implementation must *not* modify the input vector `x`.

Note 2. If `x` contains only zero entries, `d['inds']` and `d['vals']` should be empty lists.

```
In [5]: def compress_vector(x):
            assert type(x) is list
            d = {'inds': [], 'vals': []}
            ###
            ### YOUR CODE HERE
            new_x = x
            print('New x = ', new_x)
            for key in new_x:
                print(key)
                data_value = new_x.index(key)
                print(data_value)
                if key != 0.0:
                    d['inds'].append(data_value)
                    d['vals'].append(key)
            ###
            return d
```

```
In [6]: # `compress_vector_test`: Test cell
        def check_compress_vector(x_orig):
            print("Testing `compress_vector(x={})`:".format(x_orig))
            x = x_orig.copy()
            nz = x.count(0.0)
```

```
    print("\t`x` has {} zero entries.".format(nz))
    d = compress_vector(x)
    print("\tx (after call): {}".format(x))
    print("\td: {}".format(d))
    assert x == x_orig, "Your implementation appears to modify the input."
    assert type(d) is dict, "Output type is not `dict` (a dictionary)."
    assert 'inds' in d and type(d['inds']) is list, "Output key, 'inds', does not have
    assert 'vals' in d and type(d['vals']) is list, "Output key, 'vals', does not have
    assert len(d['inds']) == len(d['vals']), "`d['inds']` and `d['vals']` are lists of
    for i, v in zip(d['inds'], d['vals']):
        assert x[i] == v, "x[{}] == {} instead of {}".format(i, x[i], v)
    assert nz + len(d['vals']) == len(x), "Output may be missing values."
    assert len(d.keys()) == 2, "Output may have keys other than 'inds' and 'vals'."

# Test 1: Example
x = [0.0, 0.87, 0.0, 0.0, 0.0, 0.32, 0.46, 0.0, 0.0, 0.10, 0.0, 0.0]
check_compress_vector(x)

# Test 2: Random sparse vectors
from random import random
for _ in range(3):
    print("")
    x = []
    for _ in range(20):
        if random() <= 0.8: # Make about 10% of entries zero
            v = 0.0
        else:
            v = float("{:.2f}".format(random()))
        x.append(v)
    check_compress_vector(x)

# Test 3: Empty vector
x = [0.0] * 10
check_compress_vector(x)

print("\n(Passed!)")
```

```
Testing `compress_vector(x=[0.0, 0.87, 0.0, 0.0, 0.0, 0.32, 0.46, 0.0, 0.0, 0.1, 0.0, 0.0])`:
        `x` has 8 zero entries.
New x =  [0.0, 0.87, 0.0, 0.0, 0.0, 0.32, 0.46, 0.0, 0.0, 0.1, 0.0, 0.0]
0.0
0
0.87
1
0.0
0
0.0
0
```

```
0.0
0
0.32
5
0.46
6
0.0
0
0.0
0
0.1
9
0.0
0
0.0
0
        x (after call): [0.0, 0.87, 0.0, 0.0, 0.0, 0.32, 0.46, 0.0, 0.0, 0.1, 0.0, 0.0]
        d: {'inds': [1, 5, 6, 9], 'vals': [0.87, 0.32, 0.46, 0.1]}

Testing `compress_vector(x=[0.6, 0.0, 0.0, 0.0, 0.0, 0.0, 0.73, 0.0, 0.0, 0.0, 0.77, 0.0, 0.0,
        `x` has 14 zero entries.
New x =  [0.6, 0.0, 0.0, 0.0, 0.0, 0.0, 0.73, 0.0, 0.0, 0.0, 0.77, 0.0, 0.0, 0.57, 0.0, 0.0, 0
0.6
0
0.0
1
0.0
1
0.0
1
0.0
1
0.0
1
0.73
6
0.0
1
0.0
1
0.0
1
0.77
10
0.0
1
0.0
1
```

```
0.57
13
0.0
1
0.0
1
0.0
1
0.9
17
0.49
18
0.0
1
        x (after call): [0.6, 0.0, 0.0, 0.0, 0.0, 0.0, 0.73, 0.0, 0.0, 0.0, 0.77, 0.0, 0.0, 0.5
        d: {'inds': [0, 6, 10, 13, 17, 18], 'vals': [0.6, 0.73, 0.77, 0.57, 0.9, 0.49]}

Testing `compress_vector(x=[0.0, 0.0, 0.0, 0.64, 0.0, 0.0, 0.26, 0.0, 0.9, 0.0, 0.0, 0.0, 0.58
        `x` has 15 zero entries.
New x =  [0.0, 0.0, 0.0, 0.64, 0.0, 0.0, 0.26, 0.0, 0.9, 0.0, 0.0, 0.0, 0.58, 0.14, 0.0, 0.0, 0
0.0
0
0.0
0
0.0
0
0.64
3
0.0
0
0.0
0
0.26
6
0.0
0
0.9
8
0.0
0
0.0
0
0.0
0
0.58
12
0.14
13
```

```
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
        x (after call): [0.0, 0.0, 0.0, 0.64, 0.0, 0.0, 0.26, 0.0, 0.9, 0.0, 0.0, 0.0, 0.58, 0
        d: {'inds': [3, 6, 8, 12, 13], 'vals': [0.64, 0.26, 0.9, 0.58, 0.14]}

Testing `compress_vector(x=[0.44, 0.0, 0.0, 0.0, 0.79, 0.0, 0.0, 0.0, 0.7, 0.0, 0.0, 0.54, 0.0
        `x` has 15 zero entries.
New x =  [0.44, 0.0, 0.0, 0.0, 0.79, 0.0, 0.0, 0.0, 0.7, 0.0, 0.0, 0.54, 0.0, 0.0, 0.0, 0.21, 0
0.44
0
0.0
1
0.0
1
0.0
1
0.79
4
0.0
1
0.0
1
0.0
1
0.7
8
0.0
1
0.0
1
0.54
11
0.0
1
0.0
1
0.0
1
```

```
0.21
15
0.0
1
0.0
1
0.0
1
0.0
1
        x (after call): [0.44, 0.0, 0.0, 0.0, 0.79, 0.0, 0.0, 0.0, 0.7, 0.0, 0.0, 0.54, 0.0, 0
        d: {'inds': [0, 4, 8, 11, 15], 'vals': [0.44, 0.79, 0.7, 0.54, 0.21]}
Testing `compress_vector(x=[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])`:
        `x` has 10 zero entries.
New x =  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
0.0
0
        x (after call): [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        d: {'inds': [], 'vals': []}

(Passed!)
```

Repeated indices. Consider the compressed vector data structure, `d`, in the preceding exercise, which stores a list of indices (`d['inds']`) and a list of values (`d['vals']`).

Suppose we allow duplicate indices, possibly with different values. For example:

```
d['inds'] == [0,   3,   7,   3,   3,   5, 1]
d['vals'] == [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
```

In this case, the index 3 appears three times. (Also note that the indices `d['ind']` need not appear in sorted order.)

Let's adopt the convention that when there are repeated indices, the "true" value there is the *sum* of the individual values. In other words, the true vector corresponding to this example of `d` would be:

```
# ind:  0    1    2    3*    4    5    6    7
x == [1.0, 7.0, 0.0, 11.0, 0.0, 6.0, 0.0, 3.0]
```

Exercise 3 (`decompress_vector_test`: 2 points). Complete the function `decompress_vector(d)` that takes a compressed vector `d`, which is a dictionary with keys for the indices (`inds`) and values (`vals`), and returns the corresponding full vector. For any repeated index, the values should be summed.

The function should accept an *optional* parameter, `n`, that specifies the length of the full vector. You may assume this length is at least `max(d['inds'])+1`.

```
In [7]: def decompress_vector(d, n=None):
            # Checks the input
            assert type(d) is dict and 'inds' in d and 'vals' in d, "Not a dictionary or missi
            assert type(d['inds']) is list and type(d['vals']) is list, "Not a list"
            assert len(d['inds']) == len(d['vals']), "Length mismatch"

            # Determine length of the full vector
            i_max = max(d['inds']) if d['inds'] else -1
            if n is None:
                n = i_max+1
            else:
                assert n > i_max, "Bad value for full vector length"

            ###
            ### YOUR CODE HERE
            print(n)
            #Initialize value
            x = [0.0] * n
            for i,v in zip(d['inds'],d['vals']):
                x[i] += v

            return x
            ###
```

```
In [8]: # `decompress_vector_test`: Test cell
        def check_decompress_vector(d_orig, x_true):
            print("Testing `decompress_vector(d, n)`:")
            print("\tx_true: {}".format(x_true))
            print("\td: {}".format(d_orig))
            d = d_orig.copy()
            n_true = len(x_true)
            if d['inds'] and max(d['inds'])+1 == n_true:
                n = None
```

```
    else:
        n = n_true
    print("\tn: {}".format(n))
    x = decompress_vector(d, n)
    print("\t=> x[:{}]: {}".format(len(x), x))
    assert type(x) is list and len(x) == n_true, "Output vector has the wrong length."
    assert all([abs(x_i - x_true_i) < n_true*1e-15 for x_i, x_true_i in zip(x, x_true)]
    assert d == d_orig

# Test 1: Example
d = {}
d['inds'] = [0,   3,   7,   3,   3,   5, 1]
d['vals'] = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]
x_true = [1.0, 7.0, 0.0, 11.0, 0.0, 6.0, 0.0, 3.0]
check_decompress_vector(d, x_true)

# Test 2: Random vectors
def gen_cvec_reps(p_nz, n_max):
    from random import random, randrange, sample
    x_true = [0.0] * n_max
    d = {'inds': [], 'vals': []}
    for i in range(n_max):
        if random() <= p_nz: # Create non-zero
            n_rep = randrange(1, 5)
            d['inds'].extend([i] * n_rep)
            v_i = [float("{:.2f}".format(random())) for _ in range(n_rep)]
            d['vals'].extend(v_i)
            x_true[i] = sum(v_i)
    perm = sample(range(len(d['inds'])), k=len(d['inds']))
    d['inds'] = [d['inds'][k] for k in perm]
    d['vals'] = [d['vals'][k] for k in perm]
    return (d, x_true)

p_nz = 0.2 # probability of a non-zero
n_max = 10 # maximum full-vector length
for _ in range(5): # 5 trials
    print("")
    (d, x_true) = gen_cvec_reps(p_nz, n_max)
    check_decompress_vector(d, x_true)

# Test 3: Empty vector of length 5
print("")
check_decompress_vector({'inds': [], 'vals': []}, [0.0] * 5)

print("\n(Passed!)")
```

```
Testing `decompress_vector(d, n)`:
        x_true: [1.0, 7.0, 0.0, 11.0, 0.0, 6.0, 0.0, 3.0]
        d: {'inds': [0, 3, 7, 3, 3, 5, 1], 'vals': [1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0]}
        n: None
8
        => x[:8]: [1.0, 7.0, 0.0, 11.0, 0.0, 6.0, 0.0, 3.0]

Testing `decompress_vector(d, n)`:
        x_true: [0.0, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0, 2.42, 0.75, 0.0]
        d: {'inds': [7, 8, 2, 8, 7, 7, 7, 8], 'vals': [0.84, 0.03, 0.7, 0.4, 0.43, 0.99, 0.16,
        n: 10
10
        => x[:10]: [0.0, 0.0, 0.7, 0.0, 0.0, 0.0, 0.0, 2.42, 0.75, 0.0]

Testing `decompress_vector(d, n)`:
        x_true: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07, 0.0, 0.0, 0.54]
        d: {'inds': [6, 9], 'vals': [0.07, 0.54]}
        n: None
10
        => x[:10]: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.07, 0.0, 0.0, 0.54]

Testing `decompress_vector(d, n)`:
        x_true: [0.0, 0.0, 0.0, 0.0, 1.5499999999999998, 1.02, 0.0, 0.43, 0.0, 0.62]
        d: {'inds': [4, 7, 4, 5, 9, 5, 5, 7, 9, 9, 4, 9], 'vals': [0.49, 0.18, 0.45, 0.56, 0.0
        n: None
10
        => x[:10]: [0.0, 0.0, 0.0, 0.0, 1.5499999999999998, 1.02, 0.0, 0.43, 0.0, 0.62]

Testing `decompress_vector(d, n)`:
        x_true: [0.7, 0.0, 1.23, 0.0, 0.0, 0.0, 2.66, 1.07, 0.0, 0.0]
        d: {'inds': [6, 7, 6, 2, 7, 2, 2, 6, 6, 7, 0], 'vals': [0.68, 0.05, 0.33, 0.37, 0.98,
        n: 10
10
        => x[:10]: [0.7, 0.0, 1.23, 0.0, 0.0, 0.0, 2.66, 1.07, 0.0, 0.0]

Testing `decompress_vector(d, n)`:
        x_true: [0.0, 0.0, 1.2, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        d: {'inds': [2, 2, 2, 2], 'vals': [0.02, 0.62, 0.18, 0.38]}
        n: 10
10
        => x[:10]: [0.0, 0.0, 1.2000000000000002, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]

Testing `decompress_vector(d, n)`:
        x_true: [0.0, 0.0, 0.0, 0.0, 0.0]
        d: {'inds': [], 'vals': []}
        n: 5
5
        => x[:5]: [0.0, 0.0, 0.0, 0.0, 0.0]

(Passed!)
```

Exercise 4 (`find_common_inds_test`: 1 point). Suppose you are given two compressed vectors, `d1` and `d2`, each represented as described above and possibly with repeated indices. Complete the function `find_common_inds(d1, d2)` so that it returns a list of the indices they have in common.

For instance, suppose:

```
d1 == {'inds': [9, 9, 1, 9, 8, 1], 'vals': [0.28, 0.84, 0.71, 0.03, 0.04, 0.75]}
d2 == {'inds': [0, 9, 9, 1, 3, 3, 9], 'vals': [0.26, 0.06, 0.46, 0.58, 0.42, 0.21, 0.53, 0
```

Then:

```
find_common_inds(d1, d2) == [1, 9]
```

Note 1. The returned list must not have duplicate indices, even if the inputs do. In the example, the index 9 is repeated in both `d1` and `d2`, but the output includes just one 9.

Note 2. In the returned list, the order of indices does not matter. For instance, the example shows `[1, 9]` but `[9, 1]` would also be valid.

```
In [9]: def find_common_inds(d1, d2):
            assert type(d1) is dict and 'inds' in d1 and 'vals' in d1
            assert type(d2) is dict and 'inds' in d2 and 'vals' in d2
            ###
            ### YOUR CODE HERE
            set1 = set(d1['inds'])
            set2 = set(d2['inds'])
            return list(set1 & set2)
            ###
```

```
In [10]: # `find_common_inds_test`: Test cell
         def check_find_common_inds(d1, d2, ans):
             print("Testing `check_find_common_inds(d1, d2, ans)`:")
             print("\td1: {}".format(d1))
             print("\td2: {}".format(d2))
             print("\texpected ans: {}".format(ans))
             common = find_common_inds(d1, d2)
             print("\tcomputed common: {}".format(common))
             assert type(common) is list
             assert sorted(common) == sorted(ans), "Answers do not match."

         # Test 1: Example
         d1 = {'inds': [9, 9, 1, 9, 8, 1], 'vals': [0.28, 0.84, 0.71, 0.03, 0.04, 0.75]}
         d2 = {'inds': [0, 9, 9, 1, 3, 3, 9], 'vals': [0.26, 0.06, 0.46, 0.58, 0.42, 0.21, 0.5
         ans = [1, 9]
         check_find_common_inds(d1, d2, ans)

         # Test 2: Random tests
         from random import random, randrange, sample, shuffle
```

```python
p_common = 0.2
for _ in range(5):
    print("")
    n_min = 10
    x = sample(range(2*n_min), 2*n_min)
    i1, i2 = x[:n_min], x[n_min:]
    inds1, inds2 = [], []
    ans = []
    for k, i in enumerate(i1):
        if random() <= p_common:
            i2[k] = i
            ans.append(i)
        inds1.extend([i] * randrange(1, 4))
        inds2.extend([i2[k]] * randrange(1, 4))
    shuffle(inds1)
    d1 = {'inds': inds1, 'vals': [float("{:.1f}".format(random())) for _ in range(len
    shuffle(inds2)
    d2 = {'inds': inds2, 'vals': [float("{:.1f}".format(random())) for _ in range(len
    check_find_common_inds(d1, d2, ans)

print("\n(Passed!))")
```

```
Testing `check_find_common_inds(d1, d2, ans)`:
        d1: {'inds': [9, 9, 1, 9, 8, 1], 'vals': [0.28, 0.84, 0.71, 0.03, 0.04, 0.75]}
        d2: {'inds': [0, 9, 9, 1, 3, 3, 9], 'vals': [0.26, 0.06, 0.46, 0.58, 0.42, 0.21, 0.53]
        expected ans: [1, 9]
        computed common: [9, 1]

Testing `check_find_common_inds(d1, d2, ans)`:
        d1: {'inds': [1, 3, 15, 7, 10, 15, 3, 12, 10, 8, 17, 7, 8, 4, 6, 7, 6, 6, 1, 1], 'vals
        d2: {'inds': [10, 14, 13, 4, 13, 16, 2, 18, 2, 5, 9, 13, 10, 4, 18, 14, 0, 16, 18], 'va
        expected ans: [4, 10]
        computed common: [10, 4]

Testing `check_find_common_inds(d1, d2, ans)`:
        d1: {'inds': [0, 4, 13, 18, 17, 18, 4, 0, 9, 18, 3, 4, 0, 13, 3, 3, 1, 6, 9, 14], 'vals
        d2: {'inds': [5, 18, 11, 18, 7, 7, 15, 9, 6, 10, 16, 7, 10, 5, 17], 'vals': [0.9, 0.5,
        expected ans: [17, 9, 18, 6]
        computed common: [9, 18, 6, 17]

Testing `check_find_common_inds(d1, d2, ans)`:
        d1: {'inds': [10, 7, 8, 1, 4, 7, 14, 9, 17, 2, 9, 10, 17, 15, 4, 15, 9, 4, 17, 10], 'va
        d2: {'inds': [13, 3, 19, 11, 13, 16, 19, 18, 3, 5, 11, 5, 16, 5, 11, 0, 19, 6, 0, 12],
        expected ans: []
        computed common: []

Testing `check_find_common_inds(d1, d2, ans)`:
        d1: {'inds': [9, 14, 18, 5, 1, 18, 14, 19, 14, 3, 17, 9, 1, 1, 3, 15, 15, 7, 18, 17],
        d2: {'inds': [8, 8, 10, 11, 14, 6, 14, 16, 16, 14, 9, 9, 2, 9, 11, 2, 10, 7, 4, 8, 4,
        expected ans: [9, 14, 7]
        computed common: [9, 14, 7]

Testing `check_find_common_inds(d1, d2, ans)`:
        d1: {'inds': [2, 10, 5, 11, 2, 13, 3, 14, 5, 13, 7, 5, 6, 2, 7, 3, 14, 7, 14, 8, 8, 13
        d2: {'inds': [19, 3, 10, 15, 16, 17, 10, 9, 4, 15, 18, 19, 0, 15, 3, 0, 10, 18, 0, 18,
        expected ans: [10, 3]
        computed common: [10, 3]

(Passed!))
```

Fin! You've reached the end of this part. Don't forget to restart and run all cells again to make sure it's all working when run in sequence; and make sure your work passes the submission process. Good luck!