Section 4.5.2, Miller 3rd ed, Mode (the most frequently occurring value)

Examples of different modes:

Mode of [1, 2, 4, 2, 9, 2, 7] → 2
Mode of ['a', 'b', 'c', 'a', 'b'] → 'a' and 'b'
Mode of [1, 2, 3, 4, 5, 6] → 1, 2, 3, 4, 5, 6
Mode of ['Ben', 'Ben', 'Ben', 'Ben', 'Ben'] → 'Ben'

Algorithm to compute mode

1. Go through list – count how many times each item appears
   [4, 1, 2, 4, 2, 9, 9, 4, 2, 7]

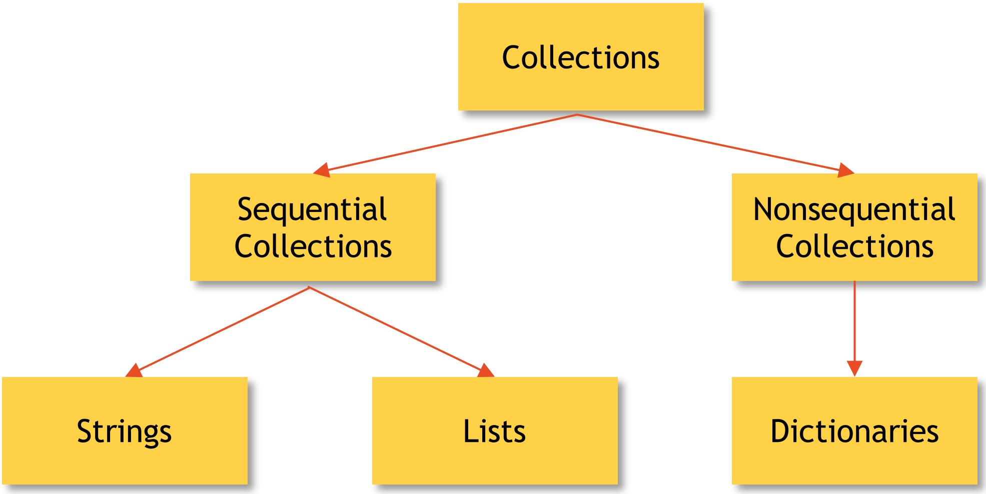
Item	Count
4	| | |
1	|
2	| | |
9	| |
7	|

2. Find max count
   Max count: 3

3. Go through counts and find every item with that count
   4 and 2 both have count 3
   Mode is 4 and 2

Dictionary

Implement algorithm with a Python dictionary



1. Dictionary contains key:value pairs

    Pairs are written inside { }

2. Simple example - ages

    Dictionary literal

    ```
    ages = {'sean':27, 'rebekah':33, 'connor':20}
    ```

    → ages: {'sean': 27, 'rebekah': 33, 'connor': 20}

    give a key - get the value

    `ages['rebekah']` → 33

    Change value with assignment

    ```
    ages['connor'] = 21
    ```

    → ages: {'sean': 27, 'rebekah': 33, 'connor': 21}

    ```
    ages['sean'] += 1
    ```

    → ages: {'sean': 28, 'rebekah': 33, 'connor': 21}

    Add a new key:value pair the same way

    ```
    ages['grace'] = 25
    ```

    → ages: {'sean': 28, 'rebekah': 33, 'connor': 21, 'grace': 25}

    Not guaranteed to be stored in any particular order

    (It's a non-sequential collection)

    len gives the number of key:value pairs

    `len(ages)` → 4

3. Dictionary Methods

    ```
    ages = {'sean':27, 'rebekah':33, 'connor':20, 'grace':25}
    ```

    keys()

    `ages.keys()` → dict_keys(['sean', 'rebekah', 'connor', 'grace'])

    Similar to:

    `list(ages)` → ['sean', 'rebekah', 'connor', 'grace']

    values()

    `ages.values()` → dict_values([27, 33, 20, 25])

ages.items()

```
ages.items()
```

→ Returns: dict_items([('sean', 27), ('rebekah', 33), ('connor', 20), ('grace', 25)])

Iterating over key:value pairs

```
for key in ages:
    print(f'{key} is {ages[key]} years old')
```

```
   sean is 27 years old
   rebekah is 33 years old
   connor is 20 years old
   grace is 25 years old
```

This is more "Pythonic" - Uses Multiple Assignment

```
for key, value in ages.items():
    print(f'{key} is {value} years old')
```

```
   sean is 27 years old
   rebekah is 33 years old
   connor is 20 years old
   grace is 25 years old
```

get(key) returns value for key, None if key not found

`ages.get('rebekah')` → 33

`ages.get('marie')` → None

Similar to:

`ages['rebekah']` → 33

`ages['marie']` → KeyError

get(key, alternate) returns value for key, "alternate" if key not found

`ages.get('sean', 'not found')` → 27

`ages.get('marie', 'not found')` → 'not found'

4. Dictionary Containment

key in dictionary / key not in dictionary

`'sean' in ages` → True

`'sean' not in ages` → False

`'marie' in ages` → False

`'marie' not in ages` → True

5. Removing key:value Pair from Dictionary

`ages` →{'sean': 27, 'rebekah': 33, 'connor': 20, 'grace': 25}

`del ages['rebekah']`

`ages` →{'sean': 27, 'connor': 20, 'grace': 25}

6. Ways to create a dictionary

An Empty Dictionary

`ages {}` → {}
`ages['sean'] = 27` → {'sean': 27}

Dictionary Literal - what we've seen

`ages = {'sean': 27, 'rebekah': 33, 'connor': 20, 'grace': 25}`

→ ages: {'sean': 27, 'rebekah': 33, 'connor': 20, 'grace': 25}

Using initializer with (key, value) pairs

`ages = dict([('sean', 27), ('molly', 31), ('pat', 29), ('grace', 25)])`

→ ages: {'sean': 27, 'molly': 31, 'pat': 29, 'grace': 25}

Using zip function

```
keys = ['pat', 'grace', 'molly']
values = [29, 25, 31]
```

`zip(keys, values)` → Returns: <zip object>

`list(zip(keys, values))` → [('pat', 29), ('grace', 25), ('molly', 31)]

`ages = dict(zip(keys, values))` → ages: {'pat': 29, 'grace': 25, 'molly': 31}

7. More on zip

First list is shorter

`list(zip([1, 2, 3], ['a', 'b', 'c', 'd']))`

→ [(1, 'a'), (2, 'b'), (3, 'c')]

Second list is shorter

`list(zip([1, 2, 3, 4], ['a', 'b', 'c']))`

→ [(1, 'a'), (2, 'b'), (3, 'c')]

More than one list

`list(zip([1, 2, 3], ['a', 'b', 'c'], [7, 8, 9]))`

→ [(1, 'a', 7), (2, 'b', 8), (3, 'c', 9)]