Warmup (L12)

- Write a function `sortByWordCount` to sort a list of strings in place by number of words (not string length)
  - Hint: `str.split()` splits a string into a list of strings using whitespace

There's much more!

- Let's use `help` to learn about
  - `list.reverse`
  - `list.index`
  - `str.join`
  - `str.split`
  - `str.find`

Dictionaries

CS114 M5

The trouble with tuples

- Here’s some info on me

```
info = (
    "Richards",
    "Gregor",
    1.76,
    "University of Waterloo",
    "Purdue University",
    2014,
    11
)
```

- … but, what means what?

Name your variables!

- Good variable naming is important to understandable code
- Tuples essentially prevent that: the values within the tuple just have indices
- If only we could group values together but still name them all!

The dictionary

- Dictionaries store various values (like tuples) but associate each value with a “key”
- The key can be anything, but let’s start with a string to demonstrate

Basic dictionary

```
info = {
    "surname": "Richards",
    "given name": "Gregor",
    "height": 1.76,
    "employer": "University of Waterloo",
    "alma mater": "Purdue University",
    "graduation year": 2014,
    "employment years": 11
}

print(
    info["given name"], info["surname"],
    "works at", info["employer"]
)
```

New syntax!

- Dictionaries are written in curly braces: `{` and `}`
- Dictionaries contain *key-value pairs*: if you use this key, you will find this value
- Key-value pair written with a colon `key: value` e.g. `"surname": "Richards"`
- The key is any Python value (confusingly), so strings can be used as names as done here

Dictionaries are mutable

- Dictionaries are mutable reference types
- Value can be changed by setting it

```
info["employment years"] = 12
print(info["employment years"]) # Now 12
```

Dictionaries are sequences

CS114 M5

Loop over a dictionary

```
info = {
    "surname": "Richards",
    "given name": "Gregor",
    "height": 1.76,
    "employer": "University of Waterloo",
    "alma mater": "Purdue University",
    "graduation year": 2014,
    "employment years": 11
}

for x in info:
    print(x) # prints "surname",
             # "given name", etc.
```

Dictionaries are sequences

- Dictionaries are actually sequences!
- When used as a sequence, they're a sequence of *keys*

```
list(info) ==
["surname", "given name", …]
```

In-lecture quiz (L12)

- https://student.cs.uwaterloo.ca/~cs114/quiz/
- Q1: What does this print?

```
print(list(
      {"c": 1, "b": 2, "a": 3}
)[0])
```

A. c

B. 1

C. a

D. 3

In-lecture quiz (L12)

- https://student.cs.uwaterloo.ca/~cs114/quiz/
- Q2: What does this print?

```
print(sorted(
      {"c": 1, "b": 2, "a": 3}
))[0])
```

A. c
B. 1
C. a
D. 3

Dictionary powers

CS114 M5

Expanding and contracting

- Dictionaries can be expanded by setting new keys

```
print(info["citizenship"]) # ERROR!
info["citizenship"] = ["USA"]
print(info["surname"]) # Still there
print(info["citizenship"]) # Now also there
```

Expanding and contracting

- With dictionaries, “**`in`**” is *key* presence

```
if "age" in info:
    print("This person is", info["age"], "years old")
```

Expanding and contracting

- Remove a key (and its value) with `.pop`

```
info.pop("employer") # Fired for tormenting
                     # Science students
print(info["employer"]) # ERROR!
```

Typing dictionaries

- The type for a dictionary is `dict`
- If you know the key and value types, and they’re consistent, `dict[key, value]`
- You can use `typing.Any` for either key or value if one is consistent but the other isn’t
- This will become clearer when we write some code, so…

Distribution

- Let’s write a function to count the number of instances of each value in a sequence
  - e.g. in [8, 6, 7, 5, 3, 0, 9, 2, 4, 6, 0, 1], we want 8 associated with 1, 6 associated with 2, etc.

Distribution

```
import typing

def distribution(
    lst: typing.Sequence
) -> dict[typing.Any, int]:
    r = {}
    for val in lst:
        if not (val in r):
            r[val] = 0
        r[val] = r[val] + 1
    return r

print(distribution([
    8, 6, 7, 5, 3, 0, 9, 2, 4, 6, 0, 1
]))
```

Before incrementing the value in the dictionary, we need to make sure there's something there

Almost a chart

- Building on distribution, let’s make a simple distribution chart by printing as many *s as there are instances of each value
  - We need one trick first:
    `"*" * 3 == "***"`

Almost a chart (first try)

```
def distributionChart(
        lst: typing.Sequence
) -> None:
        dist = distribution(lst)
        for key in dist:
                print(key, "*" * dist[key])
```

- `for` with a dictionary loops over *keys*
- This version is a bit unsatisfying, because it's printed in whatever order they first appeared in the sequence

Ordered chart

- To loop in order, we're going to have to sort the keys
- The keys are a sequence, and we know how to sort a sequence: `sorted`!

Ordered chart

```
def distributionChart(
    lst: typing.Sequence
) -> None:
    dist = distribution(lst)
    for key in sorted(dist):
        print(key, "*" * dist[key])
```

- Bonus: This isn't specific to lists! Works with any sequence, even strings!

Careful with floats!

- Remember that floats lie

```
annoying = {}
annoying[0.3-0.2] = "Hello"
annoying[0.1] = "world"
print(annoying)
```

- How do you use floats as keys? Don't!