Dictionaries

What we will cover...

1. Motivation for dictionaries
2. Creating and using dictionaries
3. KeyErrors

Why dictionary?

Lists and tuples are both collections of elements.

We saw that with destructuring and tuples, we could conveniently access pairs (or triples, etc.) of elements.

But sometimes we don't want to remember, or commit to, the order of the elements.

Dictionaries

Dictionaries are **associative** data structures.

Like the eponymous dictionary, they associate a **key** with a **value**.

This allows us to access a value via its key.

```
user = {'name': 'Foo', 'country': 'es'}

user['country'] # 'es'
user['name'] # 'Foo'
```

Creating dictionaries

To create a dictionary:

1. Start with **curly brackets** `{}`
2. Add a key (`'score'`)
3. Add a colon `:`
4. Add the value (`55`)
5. Commas separate additional key/value pairs `,`

```
{'score': 55, 'country': 'es' }
```

Creating dictionaries

keys can be strings or numbers.

values can be any data type!

keys must be unique.

Style: write on multiple lines to make more legible!

```
user = {'name': 'foo',
        'countries': ['es', 'dk', 'uk'],
        'address': {'street': 'diagonal'}}
```

Getting/Setting values

Values can be retrieved via their keys with **square brackets** `[]`, much like lists.

Values can also be set via square brackets `[]`, much like lists!

Unlike lists, however, a value can be set via square brackets `[]` even if that key did not exist.

```
user = {'name': 'Foo', 'score': 55}

user['score'] = 100

print(user['score']) # 100

user['country'] = 'es'

print(user['country]) # 'es'
```

KeyError

Dictionaries raise a `KeyError` exception when you try to access the value of a key that does not exist in the dictionary.

```
user = {'name': 'Foo', 'score': 55}

try:
    country = user['country']
except KeyError:
    country = 'No country!'

print(country)
```

Review

1. Motivation for dictionaries
2. Creating and using dictionaries
3. KeyErrors