Dictionaries and Files

CS101
Fall 2017
List Review

How would we find the maximum element in a list L?

find_max(L)
Packing

Functions can return **multiple values** at the same time!

How?

**Packing** – Returning an N item tuple (where N is the number of values you want to return)

```python
find_min_max(L)
```
Unpacking

If I make a call to a function that returns a multiple-value tuple, how can I access the values?

Unpacking – Assigning variables to the individual values returned in a tuple.

unpack_min_max(L)
Dictionaries

When you hear dictionary, what do you think of?
Dictionaries

From now on, when you hear the word dictionary, think of this:

A MAPPING of **KEYS** to **VALUES**
Dictionaries

Another way to think of it is unordered \((\text{key}, \text{value})\) pairs where every \text{key} MUST be unique AND have exactly 1 associated \text{value}. 
How do we make one?

Method 1:

Using the built-in `dict()` constructor

\[
d = \text{dict}()
\]

OR

\[
d = \text{dict([((key1, value1), (key2, value2), ...)])}
\]
How do we make one?

Method 2:

Using squiggly bracket `{}` short-hand

d = {}

OR

d = {key1:value1, key2:value2, ...}
What can we put in them?

**Keys** must be an immutable type:
- Integers
- Strings
- Tuples

**Values** can be anything!
Can I put a dictionary inside a dictionary?!

Yup.

I bet you thought I was going to make a dictionaryception joke
How do we access them?

By indexing a KEY, which returns the VALUE associated with it.

\[
d = \{\text{“cat”: 6, “dog”: 4, “horse”: 2}\}
\]

\[
d[\text{“dog”}] \text{ gives back 4}
\]
\[
d[\text{“horse”}] \text{ gives back 2}
\]
How do we access them?

Dictionaries have NO ORDER.

Therefore, we cannot slice or index them by position numbers.

We ONLY know that each unique key maps to a specific value.
Assign a **value** to a **key**

```javascript
let d = {"cat": 6, "dog": 4, "horse": 2}

d["fish"] = 20

d["cat"] = d["cat"] + 1

d is now {"cat": 7, "dog": 4, "horse": 2, "fish": 20}
```
Files

Python has the ability to open text files on your computer using the built-in `open()` command.

`open()` expects a string as an argument, specifically a `filename`.

It returns a File object, which we can iterate through (line-by-line).