Dictionaries

Part 2

March 30, 2022

Administrative notes

If you missed Monday's lecture, make sure you know what the new schedule looks like

- In particular, midterm #2 is now April 20, not April 13

Project 1 is out

- It's due on April 11 before midnight
- We talked about it on Monday; we'll talk some more today

Dictionaries

- A mapping between a set of keys and a set of values
  - Each key must be unique, and must be an immutable data type
  - Each key is mapped to a specific value, which can be a list or dictionary in addition to a string, int, float or boolean (okay, a value can be any type)
- We talked about creating a dictionary
  - An empty dictionary
  - A dictionary that isn't empty
- We talked about adding a new key/value pair to a dictionary
  - Just use an assignment statement
    - If the key's not in the dictionary, this assignment adds the key and the associated value
    - If the key is in the dictionary, the assignment statement changes the value associated with that key

Monday, redux

- We talked about accessing a key/value pair in a dictionary
  - Remember, this is dangerous - if you try to access a key that isn't there your program will crash
  - You can use the "get" method to provide a safe way to access an element in a dictionary
    - If the key you're looking for isn't there, the program doesn't crash
    - The statement just returns whatever value you specify in the statement - or None, if you don't specify anything
- We also talked about the .keys() and .values() methods
  - .keys() returns an object containing the set of all keys currently defined in the dictionary
    - It is NOT a list, even though if you print it out it sort-of, kind-of, looks like it might be one
  - .values() returns an object containing the set of all values currently defined in the dictionary
    - Same as for .keys(); it's not a list even if you think it looks like one
    -

Searching a dictionary

- You can search for keys in a dictionary; you cannot easily search for values
- It’s kind of like searching a paper dictionary for a definition if you don’t know the corresponding word - you have to look at each key to see if the definition is the one you want
- Our old friend “in” is useful
    - You can check to see if something is in the dictionary’s keys or in the dictionary’s values

Now for some new material

- We’ll quickly go through how to delete an element from a dictionary
- Then move on to some examples of how and why you might want to use a dictionary in a Python program

Removing an element from a dictionary

```
del dict[key]
```

In the dogs example:

```
  del dogs['cinder']
```

Removes the entire key/value pair associated with ‘cinder’

But again, if that key doesn’t exist in the dictionary, you’ll error out. Your program may crash.

```
   del dogs['lady']
```

```
if 'lady' in dogs.keys():
    del dogs['lady']  # gets rid of key and value pair
```

A more resilient way to delete - the pop method

```
dogs.pop('lady')
```

Still causes an error if ‘lady’ is not a key in the dictionary

But you can specify a default value to return if the key isn’t there

```
dogs.pop('lady', None)
```

OR

```
dogs.pop('lady', -1)
```

etc.

Why would you use a dictionary?

When you want to keep all information about an object together, and there are different types of information

Suppose you want to collect information about last year’s Nobel Prize winners?