CPS 109 - Lab 7

Agenda

- Dictionaries
- File I/O

Dictionaries

Most of you have probably seen dictionaries at this point, but let’s look at them in depth.

What is a dictionary? It’s an UNORDERED collection of key/value pairs.

Dictionaries

But what the heck is a key/value pair?

Let’s think about how a physical dictionary works. First, you look up a word and then you read its associated definition. The key is the word, the value is the definition!

Dictionaries

Here is the structure of a dictionary:

```
example_dict = {"kitty":"cat", "doggy":"dog", "birdy":"bird"}
```

That’s fine and dandy, but what if we want to start with an empty dictionary? Let’s look at an example...

Dictionaries

What the heck does unordered mean? And why is Cassandra emphasizing it?

Up until now, you've dealt with ordered collections: tuples, lists and sets. You can access data at an index. You can't do this with dictionaries.

Directories

I’ve noticed some of you still aren’t familiar with how and where files are stored on your computers.

How? Why??

Directories

File Directory: Where your files are located, a specific directory refers to a specific folder.

Path: Refers to the internal address of a given file or folder. You generally see these at the top of the file explorer. Example: C:\Users\Courier\Documents\Grad School\Athens Trip

File I/O

What is file I/O? Well, your programs don’t exist in a vacuum. You need to be able to communicate with lots of other files on your machine!

File I/O

How do you deal with files? First, you have to open it!

```
our_file = open("a_file.txt", "w")
```

Your first argument is the file you want to open, your second is your option for dealing with the file.

File I/O

You can do this in several ways:

```
our_file = open("a_file.txt", "w") # w means you "write to it"

our_file = open("a_file.txt", "r") # r means you "read from it"

our_file = open("a_file.txt", "a") # a means you "append to the list"
```

File I/O

Now what do we do with our file? We can do lots of things! We can write to it, we can read from it, etc.

File I/O

This is a simple example of opening a file to read, then reading from it.

```
our_file = open("a_file.txt", "r")

st = our_file.read(250)
print("The file reads: ", st)

our_file.close()
```

“I open at the close”

(Harry Potter rulez)

So you might be wondering, “what’s with this close nonsense? Why should I be forced to close my files? Won’t they just close when I’m done running the program?”

While the Python you’re running will be nice and close your files when you’re done, this is not a guarantee!

“Close encounters with the second slide”

Consider if you are running a Python program in the background, or multiple programs simultaneously. It takes resources to keep the files open. See below for best practice on how to open/close files.

It is good practice to use the `with` keyword when dealing with file objects. The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point. Using `with` is also much shorter than writing equivalent `try-finally` blocks:

```
>>> with open('workfile') as f:
...     read_data = f.read()

>>> # We can check that the file has been automatically closed.
>>> f.closed
True
```

If you’re not using the `with` keyword, then you should call `f.close()` to close the file and immediately free up any system resources used by it.