Dictionaries and files

Olatz Perez de Viñaspre Garralda
Ander Soraluze Irureta

Dictionaries

Dictionaries

(Hash in other programming languages)

They are a special kind of lists, where instead of an index we have a key.

Thus, it is composed by key:value tuples

Example of a dictionary

Animal	Sound
dog	bark
cat	meow
cock	cock-a-doodle-doo

```
sounds = {“dog”: “bark”, “cat”:“meow”, “cock”:“cock-a-doodle-doo”}
print(sounds)
```

Output:

```
{“dog”: “bark”, “cat”:“meow”, “cock”:“cock-a-doodle-doo”}
```

In this example, there are three keys: “dog”, “cat” and “cock”

About dictionaries

- There are not ordered
  - The same dictionary can have different order in different executions
- Key search are really fast
- Like it lists, the value can be anything (even a full list or dictionary)
- The key can be almost anything: anything you can not change. Not lists, but strings yes.

To access an element of a dictionary, similar to lists:

```
sounds = {“dog”: “bark”, “cat”:“meow”,
“cock”:“cock-a-doodle-doo”}
print(sounds[“dog”])
```

Output

```
bark
```

Key in the dictionary

Like in lists, we can use the **in** reserved word:

```
if “cat” in sounds:
        print(“We know the cat’s sound”)
if “mouse” in sounds:
        print(“We know the mouse’s sound”)
```

Output:

```
We know the cat’s sound
```

To print the content of a dictionary

Similar to lists, but in this case we get the keys:

```
sounds = {“dog”: “bark”, “cat”:“meow”, “cock”:“cock-a-doodle-doo”}
for animal in sounds:
        print(“The ”+animal+” does ”+sounds[animal])
```

Output:

```
The dog does bark
The cat does meow
The cock does cock-a-doodle-doo
```

Add/change dictionary’s content

```
sounds = {“dog”: “bark”, “cat”:“meow”, “cock”:“cock-a-doodle-doo”}
animal = “cat”
print(“The ”+animal+” does ”+sounds[animal])

sounds[“cat”] = “purr”
print(“The ”+animal+” does ”+sounds[animal])

sounds[“hyenas”] = “laugh”
print(”The hyenas does ”+sounds[“hyenas”])
```

Output:

```
The cat does meow
The cat does purr
The hyenas does laugh
```

To create dictionaries dynamically, like in lists, we first need to initialize them:

```
dictionary = {}
while....
```

Create dictionaries from input

Whenever we want to create dynamically them, from files or from the input() function

```
agenda = {}

line = input(“Name and number:”)
while line:
        name,number = line.split()
        agenda[name] = number
        line = input(“Name and number:”)

print(agenda)
```

Let’s try it

05-01-dictInput.py

Example of the method get

```
sounds = {“dog”: “bark”, “cat”:“meow”, “cock”:“cock-a-doodle-doo”}
animal = input('Animal: ')

while animal:
	sound = sounds.get(animal, ‘noise’)
	print(“The ”+animal+” does ”+sound)
	animal = input('Animal: ')
```

Let’s try it

05-02-dictGet.py

Summary of data structures

Seen in lectures:

- Lists
  - Ordered structures
  - Editable
- Dictionaries
  - Key/Value structures
  - Unordered
  - Good to search

Other data structures:

- Tuples
  - Lists that can not be edited
  - Useful to use as keys in Dicts
- Sets
  - No repeats
  - Unordered
  - Good to search
  - Similar to Dict’s keys

Files

Until now....

We get all the data from the user.

But, most of the data is in files...

To read a file, we need to set it’s name.

For example, “text.txt”

Directories and paths

Directory = a folder
Path = the location of the file

There are two kinds of paths:

- Absolute path
- Relative path

Absolute path vs relative path

Absolute path

- It starts from the root directory of the file system
- For example:
  - In Windows: C:/
  - In Linux/MacOS: /home/

Advantage

- Even if you change the current directory, it works

Relative path

- It starts from the current directory
- For example:
  - ../../data.txt
  - data/accounting/april.txt

Advantage

- There is no need to know the full path

In short...

- **Absolute path** is like getting directions from a known location (for example, starting from the town hall).
- **Relative path** is like the directions from the current position of the person.

Opening files

Access file = open file

Code:

```
fin = open(“example.txt”)
print(fin)
```

Output:

```
<_io.TextIOWrapper name=example.txt'
mode='r' encoding='UTF-8'>
```

Be careful!

If the file does not exist: FileNotFoundError

Reading files

We can read the full file together

requests.txt:

Tomato salad
Pumpkin puree
Roasted chicken

```
fin = open(“requests.txt”)
text = fin.read()

print(“===”)
print(text)
print(“===”)
```

Be careful!!! With the read() function we load the full content of the file in memory, and that can generate problems with big files.

Reading files with loops

```
fin = open(“requests.txt”)
for line in fin:
    print(line.strip())
```

More ways to open files

```
fin = open(“example.txt”)

for line in fin:
        print(line.strip())

fin.close()
```

```
for line in open(“example.txt”):
        print(line.strip())
```

```
with open(“example.txt”) as fin:
        for line in fin:
                print(line.strip())
```

Let’s try it!

05-03-files.py

Writing in files

Very similar to reading

When opening, writing mode

```
fout = open(“output.txt”, “w”)
```

Writing:

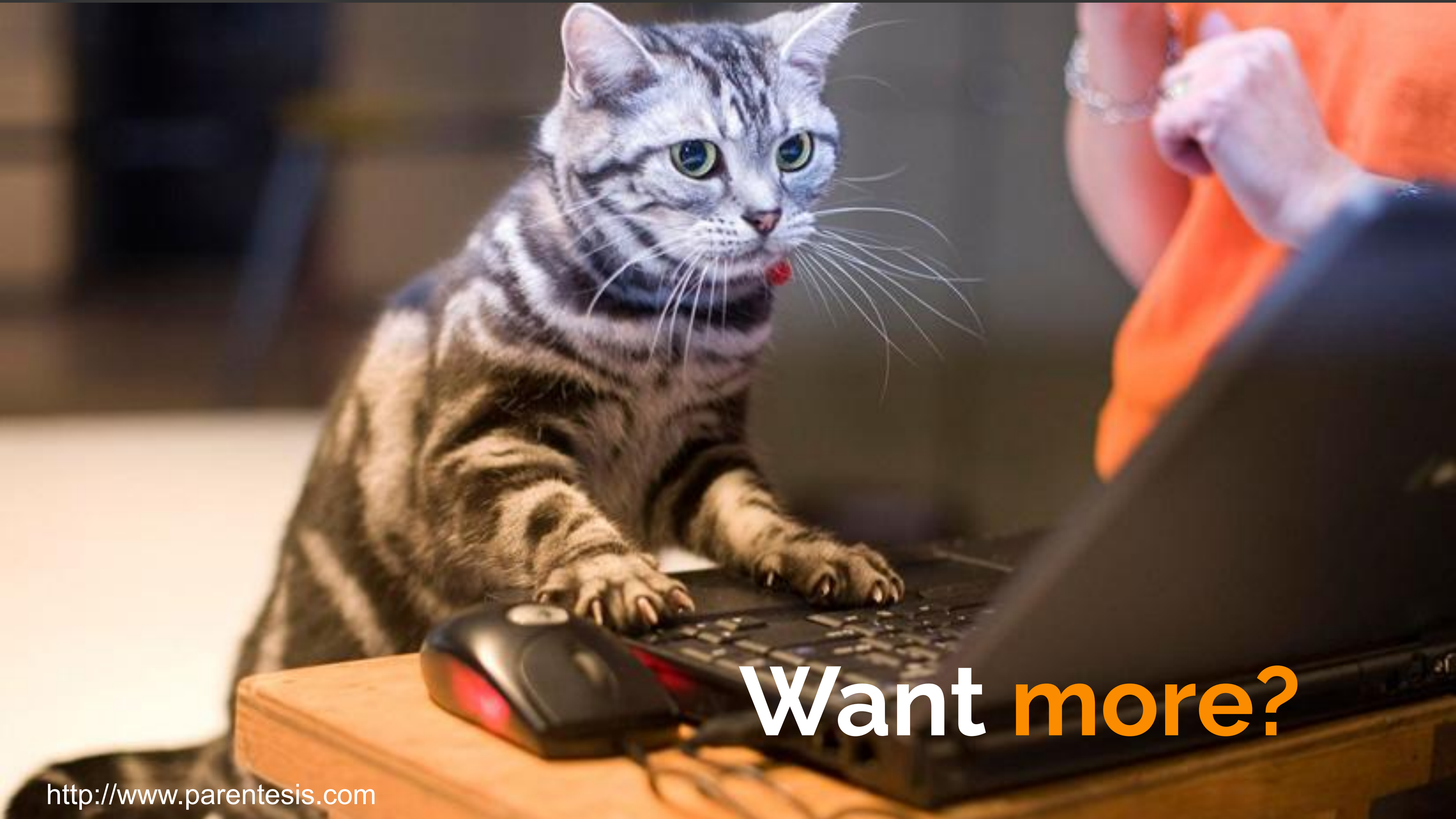
```
fout.write(“Hello, world”)
fout.write(“Writing in files”)
```

The file has to be closed:

```
fout.close()
```

Let’s try it!

05-04-writingFiles.py



Want more?

http://www.parentesis.com