DATA STORAGE IN PYTHON

Peter Larsson-Green

Jönköping University

Autumn 2018

WHERE DO WE STORE DATA?

In variables!

- Easy to create.
```
my_variable = 123
```
- Easy to read.
```
my_variable
```
- Easy to update.
```
my_variable = 456
```
- Very fast!
- Variables are deleted when program terminates ☹

WHERE DO WE STORE DATA?

In files!

- More complex to create.
- More complex to read.
- More complex to update.
- Slower.
- Continues to exist after the program has terminated ☺
    - Until the user manually deletes it by mistake…

HOW TO OPEN FILES

```
file_object = open("the-filename.txt", "w")
```

The mode.

The modes

- `"w"` - create the file if it does not exist,
  then use `file_object` to write strings to it.
- `"a"` - create the file if it does not exist,
  then use `file_object` to write strings to it (at the end).
- `"r"` - open the file for reading,
  then use `file_object` to read strings from it.
- `"r+"` - open the file for reading and writing,
  then use `file_object` to read and write strings to/from it.

HOW TO CLOSE FILES

```
file_object = open("the-filename.txt", "w")
# Work with the file...
file_object.close()
```

```
with open("the-filename.txt", "w") as file_object:
  # Work with the file...
```

WRITING TO AN OPENED FILE

```
with open("test-file.txt", "w") as file_object:
  file_object.write("This is the content!")
```

Must be a string.

```
with open("test-file.txt", "w") as file_object:
  file_object.write("This is the new content!")
```

<u>test-file.txt</u>

```
~~This is the content!~~
```

$\longrightarrow$

```
This is the new content!
```

EXAMPLE

```
def write_numbers_to_file(name, n):
  with open(name, "w") as file_object:
    for i in range(1, n+1):
      file_object.write(str(i)+"\n")

write_numbers_to_file("numbers.txt", 5)
```

numbers.txt

```
1
2
3
4
5
```

READING FROM AN OPENED FILE

```
with open("test-file.txt", "r") as file_object:
  all_content = file_object.read()
```

```
with open("test-file.txt", "r") as file_object:
  line1 = file_object.readline()
  line2 = file_object.readline()
```

With "\n" at the end.

"" means no more lines.

```
with open("test-file.txt", "r") as file_object:
  list_of_lines = file_object.readlines()
```

EXAMPLE

```python
def get_sum_of_numbers_in_file(name):
  with open(name, "r") as file_object:
    sum = 0
    line = file_object.readline()
    while line != "":
      sum += int(line)
      line = file_object.readline()
    return sum

fifteen = get_sum_of_numbers_in_file("numbers.txt")
```

numbers.txt

```
1
2
3
4
5
```

READING FROM AN OPENED FILE

```
with open("test-file.txt", "r") as file_object:
  for line in file_object:
    # Do something with line!
```

```
def get_sum_of_numbers_in_file(name):
  with open(name, "r") as file_object:
    sum = 0
    for line in file_object:
      sum += int(line)
    return sum
```

numbers.txt

```
1
2
3
4
5
```

STORING COMPLEX DATA

How do we store the data below in a file?

```
humans = [
  {'age': 10, 'name': "Alice"},
  {'age': 15, 'name': "Belle"},
  {'age': 20, 'name': "Chloe"}
]
```

It's your program, you decide!

STORING COMPLEX DATA

Example: one human on each line, separate values by space.

```
humans = [
  {'age': 10, 'name': "Alice"},
  {'age': 15, 'name': "Belle"},
  {'age': 20, 'name': "Chloe"}
]
```

humans.txt

```
10 Alice
15 Belle
20 Chloe
```

```
with open("humans.txt", "w") as file_object:
  for human in humans:
    file_object.write(str(human['age'])+" "+human['name']+"\n")
```

PARSING COMPLEX DATA

Example: one human on each line, separate values by space.

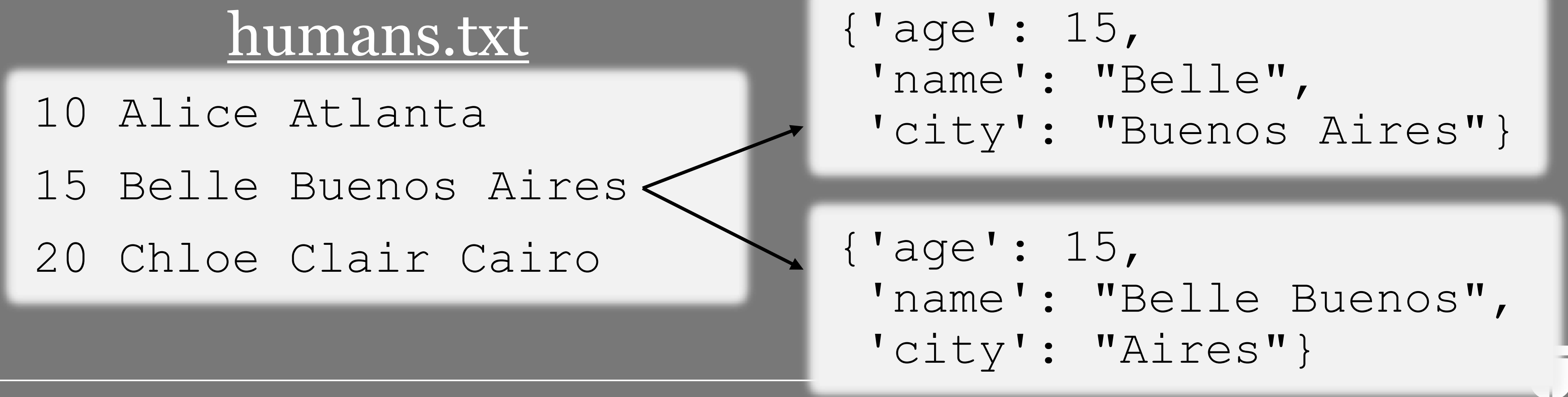
```
humans = []
with open("humans.txt", "r") as file:
  for line in file:
    values = line.split(" ")
    humans.append({
      'age': int(values[0]),
      'name': values[1].rstrip()
    })
```

humans.txt

```
10 Alice
15 Belle
20 Chloe
```

STORING COMPLEX DATA

```
humans = [
  {'age': 10, 'name': "Alice", 'city': "Atlanta"},
  {'age': 15, 'name': "Belle", 'city': "Buenos Aires"},
  {'age': 20, 'name': "Chloe Clair", 'city': "Cairo"}
]
```



humans.txt

```
10 Alice Atlanta
15 Belle Buenos Aires
20 Chloe Clair Cairo
```

```
{'age': 15,
 'name': "Belle",
 'city': "Buenos Aires"}
```

```
{'age': 15,
 'name': "Belle Buenos",
 'city': "Aires"}
```

STORING COMPLEX DATA

Well known data formats has evolved.

Advantages:

- "Everybody" already know these formats.
- Others have already written code for generating/parsing them.

CSV: COMMA SEPARATED VALUES

humans.csv

```
10,Alice,Atlanta
15,Belle,Buenos Aires
20,Chloe Clair,Cairo
```

CSV IN PYTHON

```
import csv
humans = [
  {'age': 10, 'name': "Alice", 'city': "Atlanta"},
  {'age': 15, 'name': "Belle", 'city': "Buenos Aires"},
  {'age': 20, 'name': "Chloe Clair", 'city': "Cairo"}
]
with open('humans.csv', 'w', newline="\n") as csv_file:
  writer = csv.writer(csv_file, delimiter=',', quotechar='"')
  for h in humans:
    writer.writerow([h['age'], h['name'], h['city']])
```

CSV IN PYTHON

```
import csv
humans = []
with open('humans.csv', 'r') as csv_file:
  reader = csv.reader(csv_file, delimiter=',', quotechar='"')
  for row in reader:
    humans.append({
      'age': int(row[0]),
      'name': row[1],
      'city': row[2]
    })
```

XML: EXTENSIBLE MARKUP LANGUAGE

```
<humans>
  <human>
    <age>10</age>
    <name>Alice</name>
    <city>Atlanta</city>
  </human>
  <human>
    <age>15</age>
    ...
</humans>
```

XML IN PYTHON

```
import xml.etree.ElementTree as ET
humans = [{'age': 10, 'name': "Alice", 'city': "Atlanta"},
          {'age': 15, 'name': "Belle", 'city': "Buenos Aires"},
          {'age': 20, 'name': "Chloe Clair", 'city': "Cairo"}]
humans_element = ET.Element('humans')
for h in humans:
  human_element = ET.SubElement(humans_element, 'human')
  age_element = ET.SubElement(human_element, 'age')
  age_element.text = str(h['age'])
  name_element = ET.SubElement(human_element, 'name')
  name_element.text = h['name']
  city_element = ET.SubElement(human_element, 'city')
  city_element.text = h['city']
```

```
<humans>
 <human>
  <age>
   10
  </age>
  <name>
   Alice
  </name>
  <city>
   Atlanta
  </city>
 </human>
 ...
</humans>
```

XML IN PYTHON

```
xml_string = ET.tostring(humans_element,
                                encoding="unicode")

with open('humans.xml', 'w') as xml_file:

  xml_file.write(xml_string)
```

```
<humans>
 <human>
  <age>
   10
  </age>
  <name>
   Alice
  </name>
  <city>
   Atlanta
  </city>
 </human>
 ...
</humans>
```

XML IN PYTHON

```
import xml.etree.ElementTree as ET
humans = []
with open('humans.xml', 'r') as xml_file:
  xml_string = xml_file.read()
  humans_element = ET.fromstring(xml_string)
  for human_element in humans_element:
    humans.append({
      'age': int(human_element.find("age").text),
      'name': human_element.find("name").text,
      'city': human_element.find("city").text
    })
```

```
<humans>
 <human>
  <age>
   10
  </age>
  <name>
   Alice
  </name>
  <city>
   Atlanta
  </city>
 </human>
 ...
</humans>
```

JSON: JAVASCRIPT OBJECT NOTATION

Numbers in JSON: `41 3.14`

Strings in JSON: `"Hello" "Hi"`

Booleans in JSON: `true false`

Arrays in JSON: `[12, "Hi", false]`

Objects in JSON: `{"a": 1, "b": true}`

```
[
 {"age": 10, "name": "Alice", "city": "Atlanta"},
 {"age": 15, "name": "Belle", "city": "Buenos Aires"},
 {"age": 20, "name": "Chloe Clair", "city": "Chicago"}
]
```

JSON IN PYTHON

```
import json
humans = [
  {'age': 10, 'name': "Alice", 'city': "Atlanta"},
  {'age': 15, 'name': "Belle", 'city': "Buenos Aires"},
  {'age': 20, 'name': "Chloe Clair", 'city': "Cairo"}
]
json_string = json.dumps(humans)
with open('humans.json', 'w') as json_file:
  json_file.write(json_string)
```

JSON IN PYTHON

```
import json
humans = []
with open('humans.json', 'r') as json_file:
  json_string = json_file.read()
  humans = json.loads(json_string)
```

MORE FILE OPERATIONS

```
import os

os.remove("the-filename.txt")
os.rename("current-filename.txt", "new-filename.txt")

import os.path

exists = os.path.isfile("the-filename.txt")
```