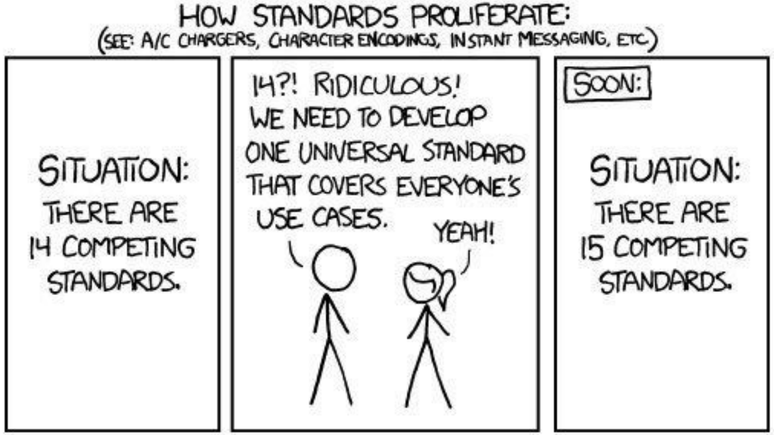
Python for Text Analysis 2018-2019

Lecture 11: Data Formats part II [block 4]
03-12-2018

Goals for today

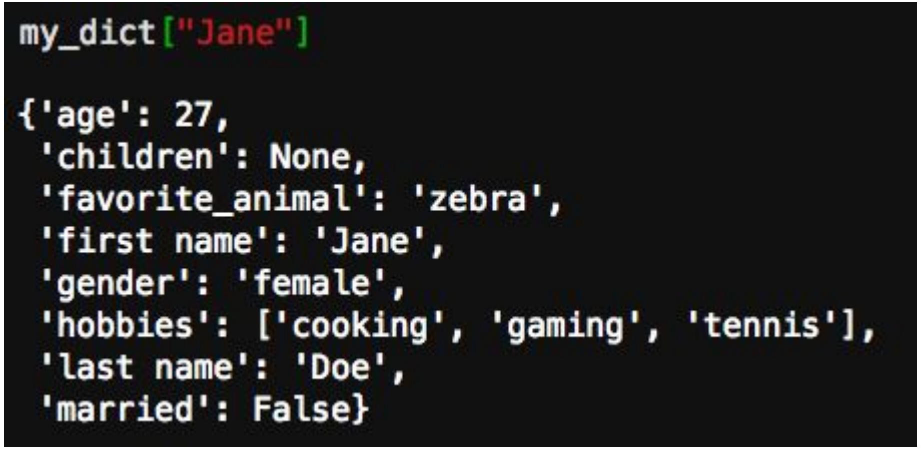
Before the break:

- Pick up where we left Thursday
  - working with **nested lists/dicts** (JSON)
  - any other questions
  - (optionally: Stranger Things exercise Ch. 17)
- Learn about XML
  - **XML elements**
  - **tags**, **attributes** and **texts** of XML elements
  - using the **lxml module** for **reading** XML
  - ~~using the **lxml module** for **writing** XML~~

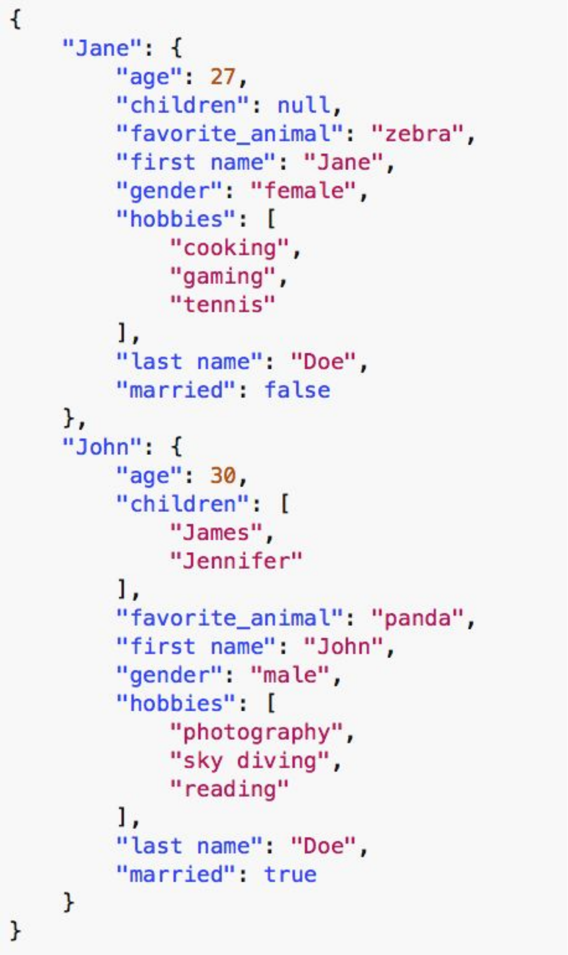

HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

After the break: work on assignment 4

Nested lists and dicts

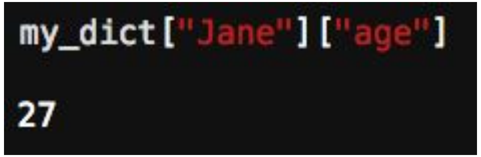


```
my_dict["Jane"]

{'age': 27,
 'children': None,
 'favorite_animal': 'zebra',
 'first name': 'Jane',
 'gender': 'female',
 'hobbies': ['cooking', 'gaming', 'tennis'],
 'last name': 'Doe',
 'married': False}
```

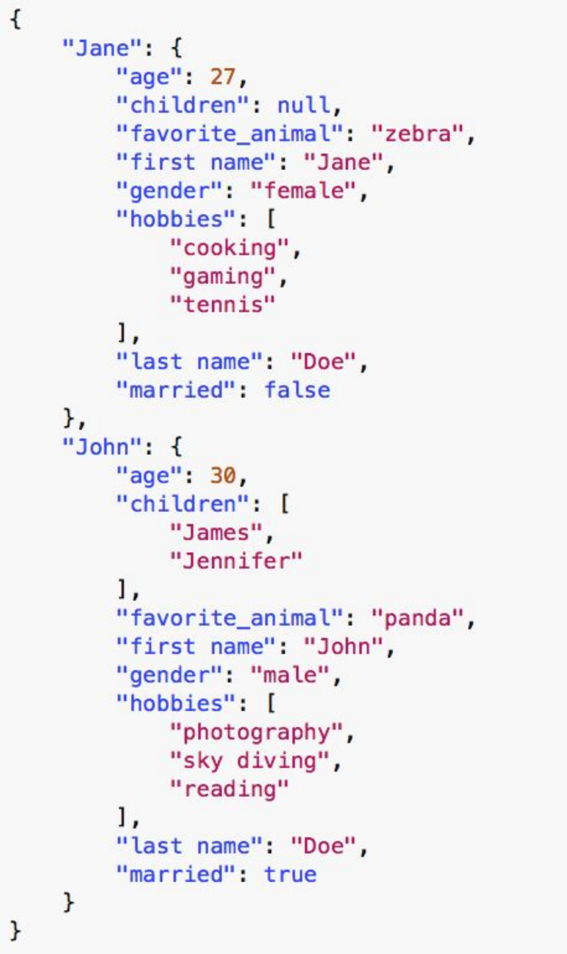


```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

Nested lists and dicts

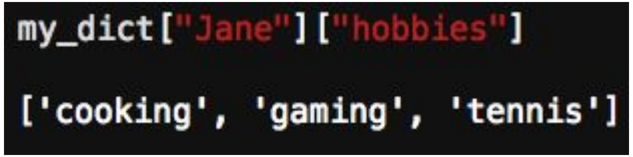


```
my_dict["Jane"]["age"]

27
```

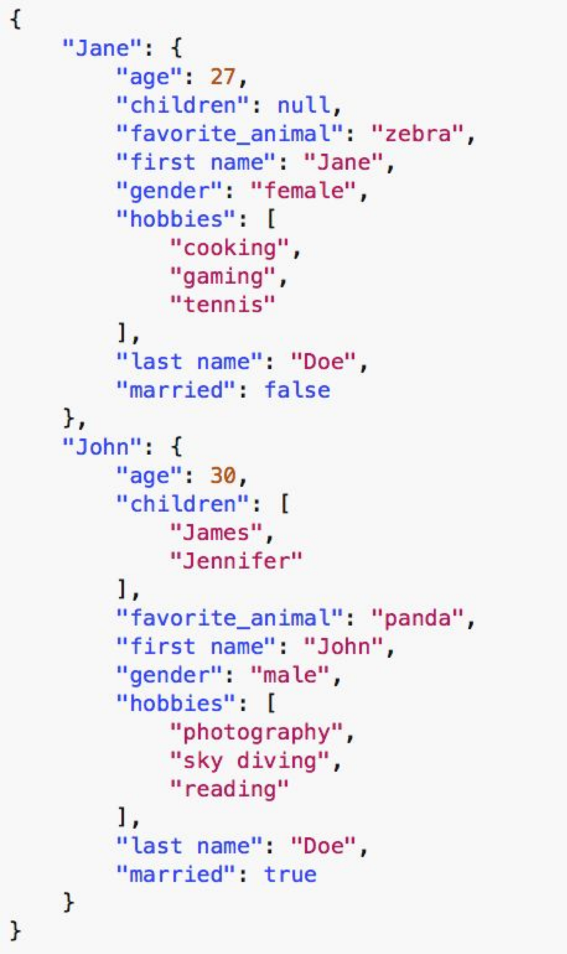


```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

Nested lists and dicts

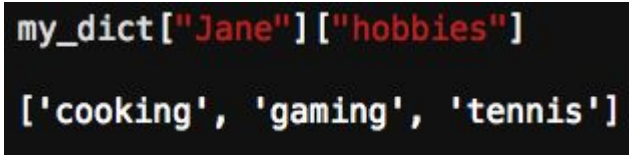


```
my_dict["Jane"]["hobbies"]

['cooking', 'gaming', 'tennis']
```

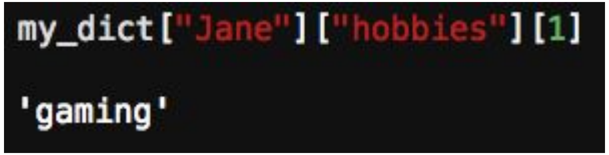


```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

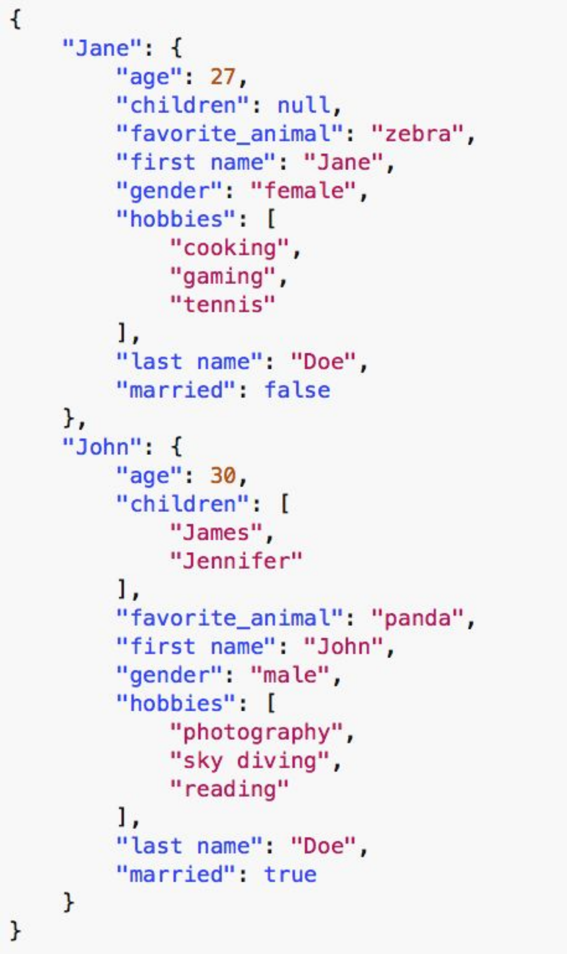
Nested lists and dicts



```
my_dict["Jane"]["hobbies"]

['cooking', 'gaming', 'tennis']
```



```
my_dict["Jane"]["hobbies"][1]

'gaming'
```



```
{
    "Jane": {
        "age": 27,
        "children": null,
        "favorite_animal": "zebra",
        "first name": "Jane",
        "gender": "female",
        "hobbies": [
            "cooking",
            "gaming",
            "tennis"
        ],
        "last name": "Doe",
        "married": false
    },
    "John": {
        "age": 30,
        "children": [
            "James",
            "Jennifer"
        ],
        "favorite_animal": "panda",
        "first name": "John",
        "gender": "male",
        "hobbies": [
            "photography",
            "sky diving",
            "reading"
        ],
        "last name": "Doe",
        "married": true
    }
}
```

SOCRATIVE

Log in at:

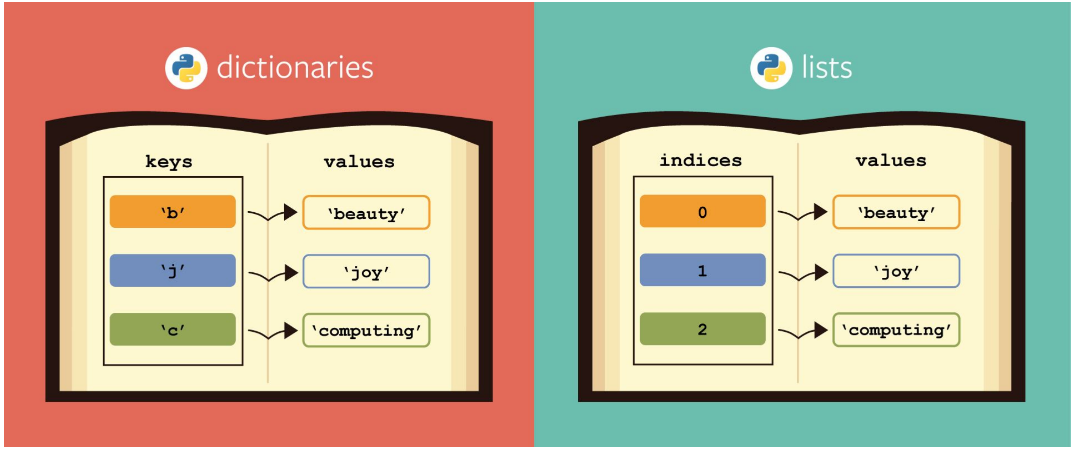
b.socrative.com/student

Room:

PYTHON1819VU

Using indices/keys versus looping

- When do you use **indices/keys** to find certain values?
- When do you **loop over** the elements in a list,
  or over the keys/values/items in a dictionary to find certain values?



dictionaries

keys	values
'b'	'beauty'
'j'	'joy'
'c'	'computing'

lists

indices	values
0	'beauty'
1	'joy'
2	'computing'

CHAPTER 18: Data Formats III(XML)

SOCRATIVE

Log in at:

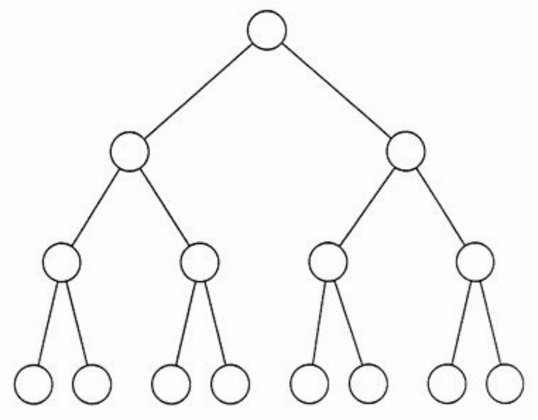
b.socrative.com/student

Room:

PYTHON1819VU

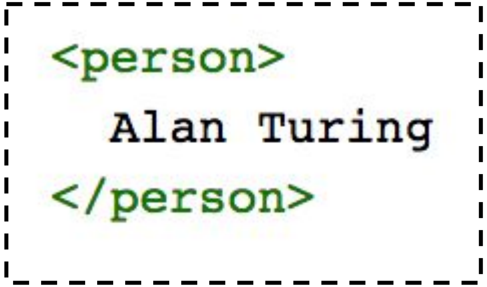
About XML

- **XML** stands for E**x**tensible **M**arkup **L**anguage
- Just as CSV and JSON, an XML file is simply a **plain text file** using certain conventions to structure information
- It is like an **ordered, labeled tree**



XML Elements

- XML consists of a collection of **XML elements** (the nodes in the tree)
- A single element is represented by:
  - an **opening tag**, such as `<person>`
  - a **closing tag**, such as `<person>`
- **Everything in between** the opening and closing tags is called the **content**
- An **empty element** (without content) is represented as: `<person/>`

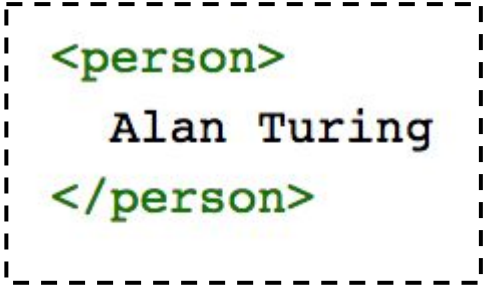


```
<person>
  Alan Turing
</person>
```

Example from: https://www.sci.unich.it/~francesc/caffe-xml/xml

XML Text

- One type of content is the **text** of an element
- In the example below, the text of `<person>` is `Alan Turing`

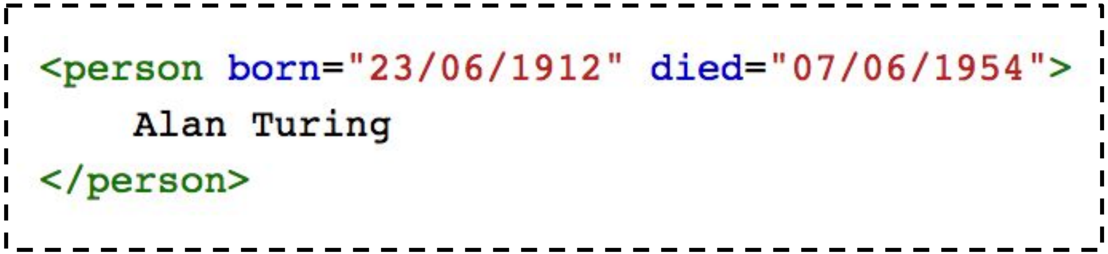


```
<person>
  Alan Turing
</person>
```

Example from: https://www.sci.unich.it/~francesc/caffe-xml/xml

XML Attributes

- XML Elements can also have **attributes**
- They are represented by the syntax `key = “value”`, where `key` is the attribute name and `value` is a string
  - For example: `born = “23/06/1912”`
- An element can have any number of attributes, but **no duplicate keys** are allowed

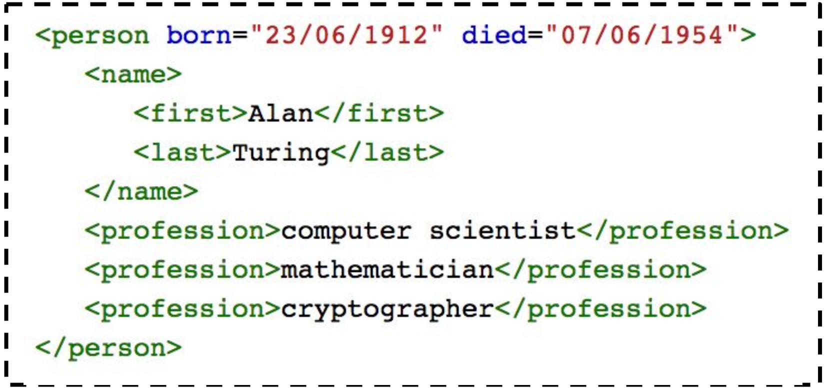


```
<person born="23/06/1912" died="07/06/1954">
    Alan Turing
</person>
```

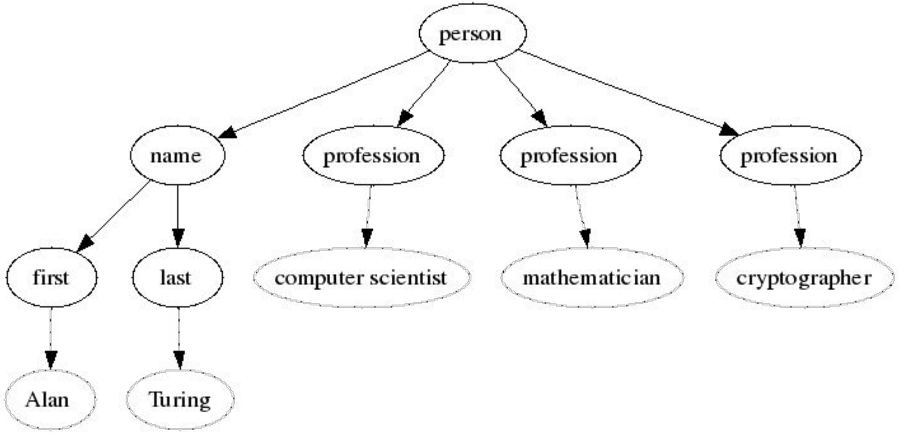
Example from: https://www.sci.unich.it/~francesc/caffe-xml/xml

XML Tree

- XML has an hierarchical **tree structure** starting from the **root element**
  - Root element in our example: `<person>`
- XML element can have **parents**, **children** and **siblings**



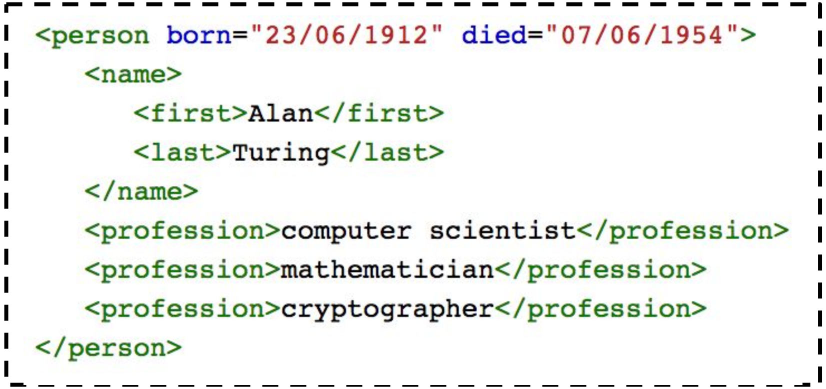
```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```



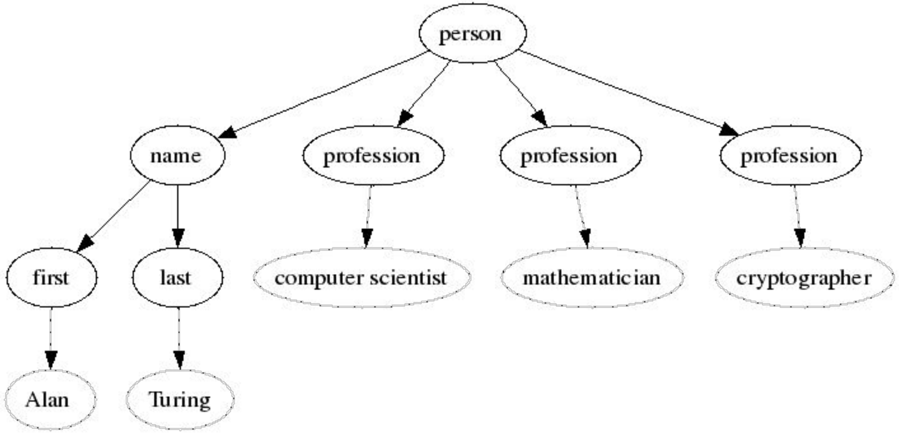
Example from: https://www.sci.unich.it/~francesc/caffe-xml/xml

XML Tree

- XML has an hierarchical **tree structure** starting from the **root element**
  - Root element in our example: `<person>`
- XML element can have **parents**, **children** and **siblings**
  - `<name>` and `<profession>` are **children** of `<person>`
  - `<name>` is the **parent** of `<first>` and `<last>`
  - `<first>` and `<last>` are **siblings**



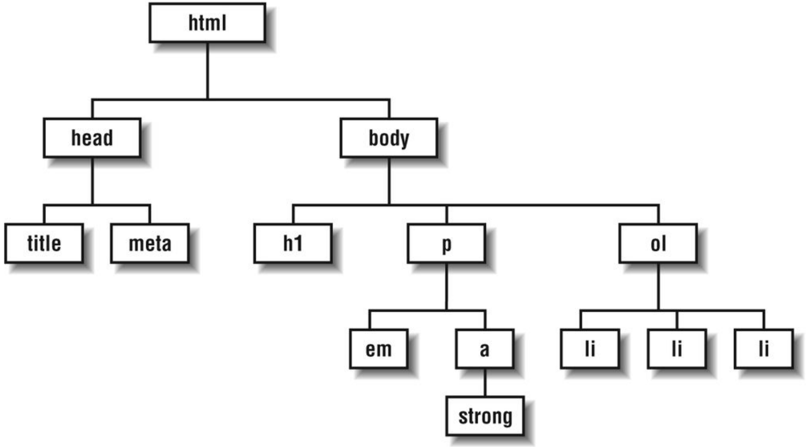
```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```



Example from: https://www.sci.unich.it/~francesc/caffe-xml/xml

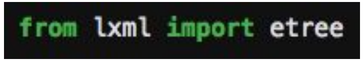
XML and HTML

- **XML** is very similar to **HTML**, with two main differences:
  - In XML you can **invent your own tags**, while the tag set of HTML is **fixed**
  - XML focuses on **carrying/describing data** (what is it; type of content), while HTML focuses on **displaying data** (how should it look like; formatting, lay-out)
- Both can be parsed in Python with the **lxml module**



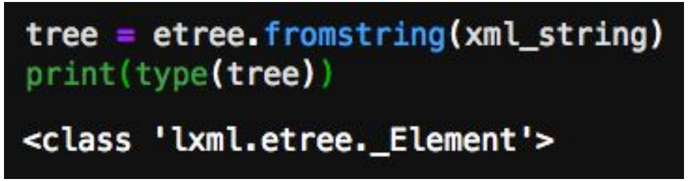
Parsing XML in Python: the lxml.etree module

- The **lxml.etree module** provides a way to work with XML in Python
- We usually import it as follows:

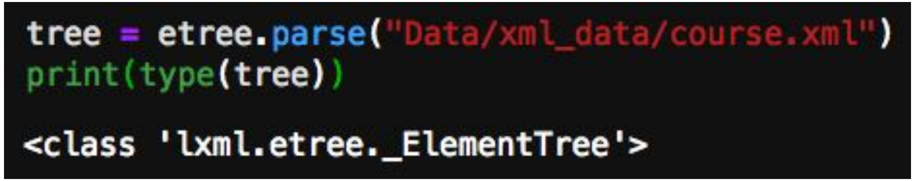


```
from lxml import etree
```

- Two methods for **reading XML**:
  - `etree.fromstring()` → load an XML formatted **string** as an `Element`
  - `etree.parse()` → load a XML formatted **file** as an `ElementTree`



```
tree = etree.fromstring(xml_string)
print(type(tree))

<class 'lxml.etree._Element'>
```



```
tree = etree.parse("Data/xml_data/course.xml")
print(type(tree))

<class 'lxml.etree._ElementTree'>
```

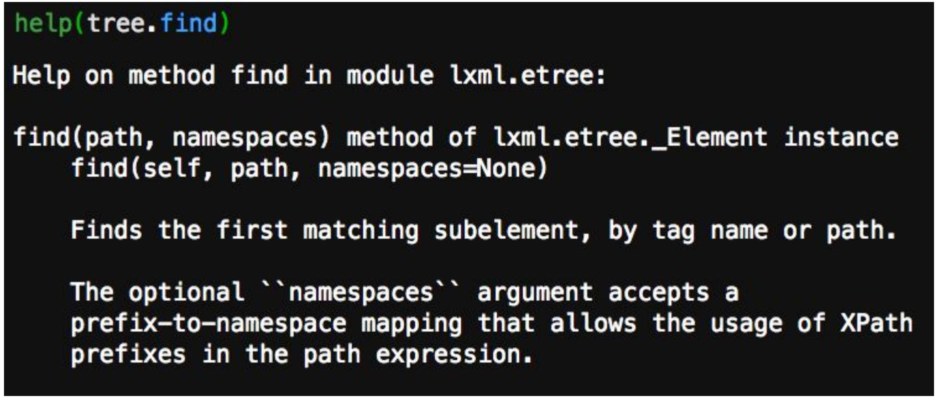
`ElementTree` and `Element` methods

- `ElementTree` and `Element` are **specific types of objects**

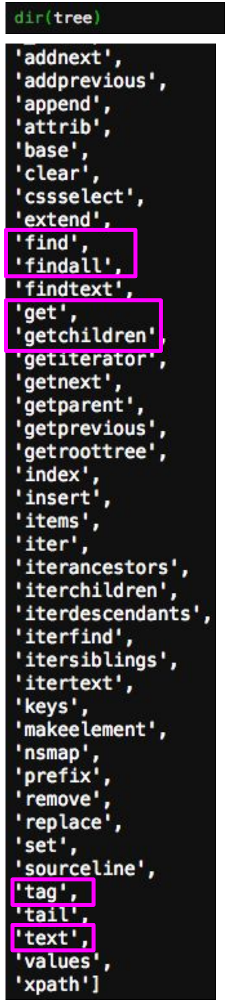
  (so no standard strings, integers, lists or dictionaries)
  - `ElementTree`: the whole XML document
  - `Element`: one XML element
- They have their own **methods** to get specific information from them
  - Remember that you can also call `dir()` and `help()` for these methods!

ElementTree and Element methods

- ElementTree and Element are **specific types of objects**
  (so no standard strings, integers, lists or dictionaries)
  - ElementTree: the whole XML document
  - Element: one XML element
- They have their own **methods** to get specific information from them
  - Remember that you can also call dir() and help() for these methods!



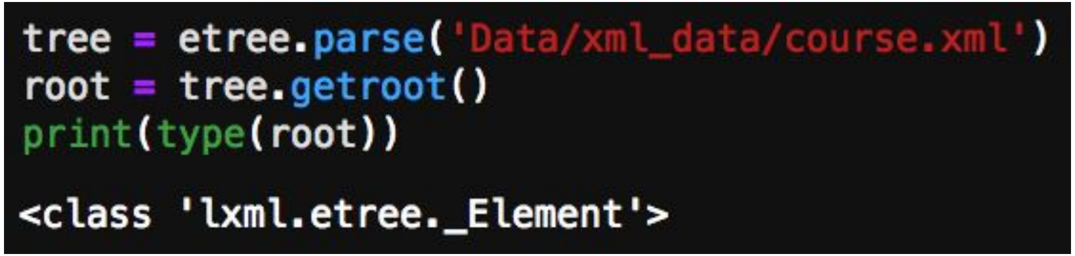
```
help(tree.find)

Help on method find in module lxml.etree:

find(path, namespaces) method of lxml.etree._Element instance
    find(self, path, namespaces=None)

    Finds the first matching subelement, by tag name or path.

    The optional ``namespaces`` argument accepts a
    prefix-to-namespace mapping that allows the usage of XPath
    prefixes in the path expression.
```



```
dir(tree)

'addnext',
'addprevious',
'append',
'attrib',
'base',
'clear',
'cssselect',
'extend',
'find',
'findall',
'findtext',
'get',
'getchildren',
'getiterator',
'getnext',
'getparent',
'getprevious',
'getroottree',
'index',
'insert',
'items',
'iter',
'iterancestors',
'iterchildren',
'iterdescendants',
'iterfind',
'itersiblings',
'itertext',
'keys',
'makeelement',
'nsmap',
'prefix',
'remove',
'replace',
'set',
'sourceline',
'tag',
'tail',
'text',
'values',
'xpath']
```

Accessing the root element

- When reading XML **from a file**, you first have to get the **root** of the tree



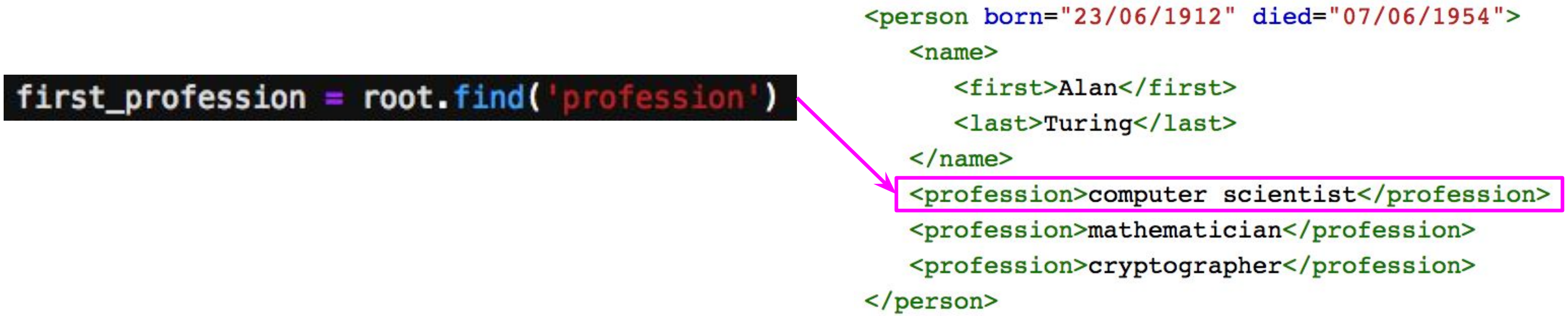
```
tree = etree.parse('Data/xml_data/course.xml')
root = tree.getroot()
print(type(root))

<class 'lxml.etree._Element'>
```

Accessing (child) elements

- Three methods to remember:
  - `find()` → find the **first matching** child element by **tag name or path** (returns an `Element`)
  - `findall()` → find **all matching** child elements by **tag name or path** (returns a list of `Elements`)
  - `getchildren()` → find **all** child elements, **irrespective** of **tag name** (returns a list of `Elements`)

Accessing (child) elements

- Three methods to remember:
  - `find()` → find the **first matching** child element by **tag name or path** (returns an `Element`)
  - `findall()` → find **all matching** child elements by **tag name or path** (returns a list of `Elements`)
  - `getchildren()` → find **all** child elements, **irrespective** of **tag name** (returns a list of `Elements`)

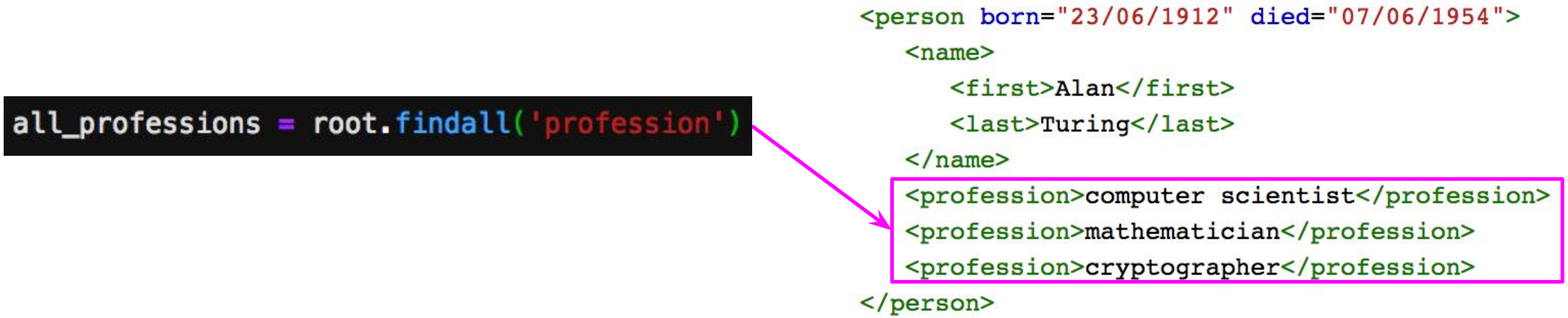


```
first_profession = root.find('profession')
```

```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Accessing (child) elements

- Three methods to remember:
  - `find()` → find the **first matching** child element by **tag name or path** (returns an `Element`)
  - `findall()` → find **all matching** child elements by **tag name or path** (returns a list of `Elements`)
  - `getchildren()` → find **all** child elements, **irrespective** of **tag name** (returns a list of `Elements`)



```
all_professions = root.findall('profession')
```

```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

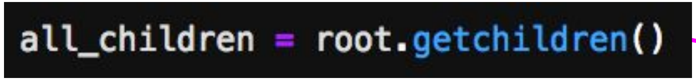
Accessing (child) elements

❖ Three methods to remember:

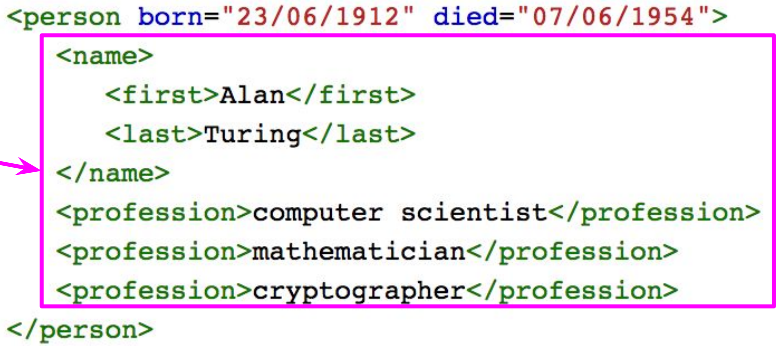
➢ `find()` → find the **first matching** child element by **tag name or path** (returns an `Element`)

➢ `findall()` → find **all matching** child elements by **tag name or path** (returns a list of `Elements`)

➢ `getchildren()` → find **all** child elements, **irrespective** of **tag name** (returns a list of `Elements`)



```
all_children = root.getchildren()
```



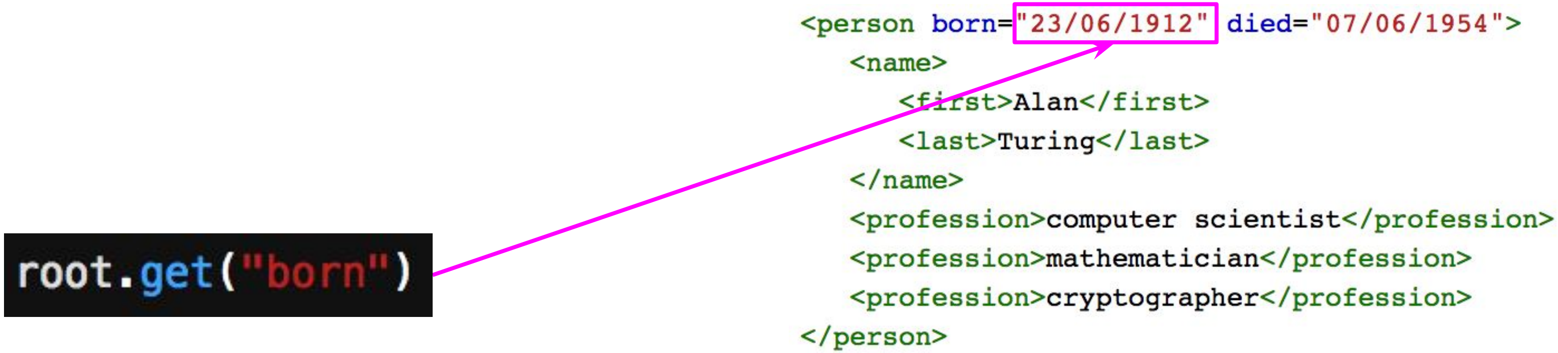
```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Accessing element information (content)

- Three methods/attributes to remember:
  - `get()` → find the **value** of a certain **attribute** of the `Element` by the attribute **key** (returns a string)
  - `text` → find the **text** of the `Element` (returns a string)
  - `tag` → find the **tag** of the `Element` (returns a string)

Accessing element information (content)

- Three methods/attributes to remember:
  - `get()` → find the **value** of a certain **attribute** of the `Element` by the attribute **key** (returns a string)
  - `text` → find the **text** of the `Element` (returns a string)
  - `tag` → find the **tag** of the `Element` (returns a string)

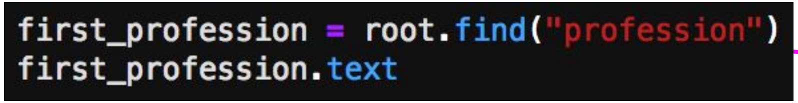


```
root.get("born")
```

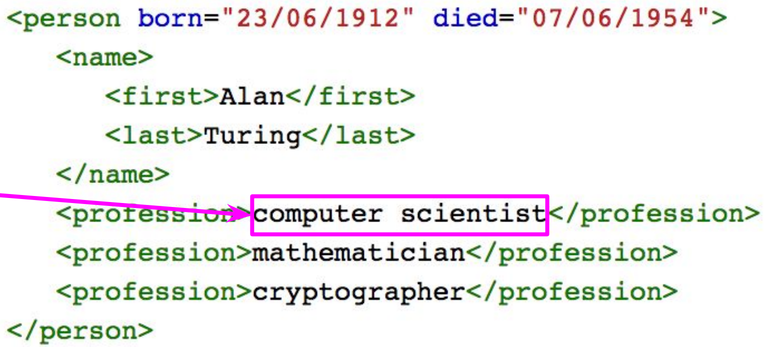
```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Accessing element information (content)

- Three methods/attributes to remember:
  - `get()` → find the **value** of a certain **attribute** of the `Element` by the attribute **key** (returns a string)
  - `text` → find the **text** of the `Element` (returns a string)
  - `tag` → find the **tag** of the `Element` (returns a string)



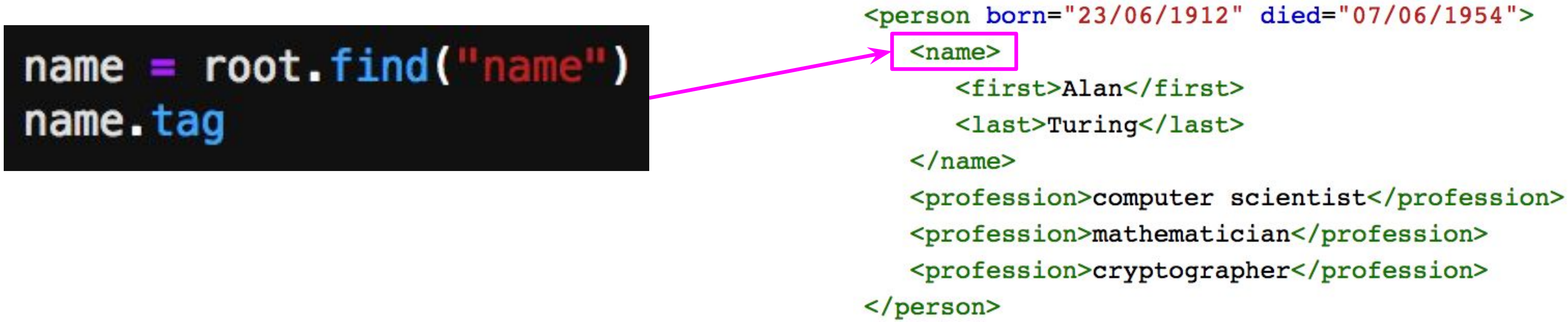
```python
first_profession = root.find("profession")
first_profession.text
```



```xml
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Accessing element information (content)

- Three methods/attributes to remember:
  - `get()` → find the **value** of a certain **attribute** of the `Element` by the attribute **key** (returns a string)
  - `text` → find the **text** of the `Element` (returns a string)
  - `tag` → find the **tag** of the `Element` (returns a string)

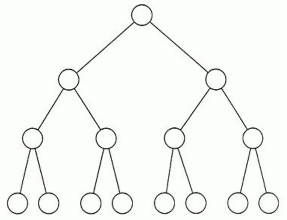


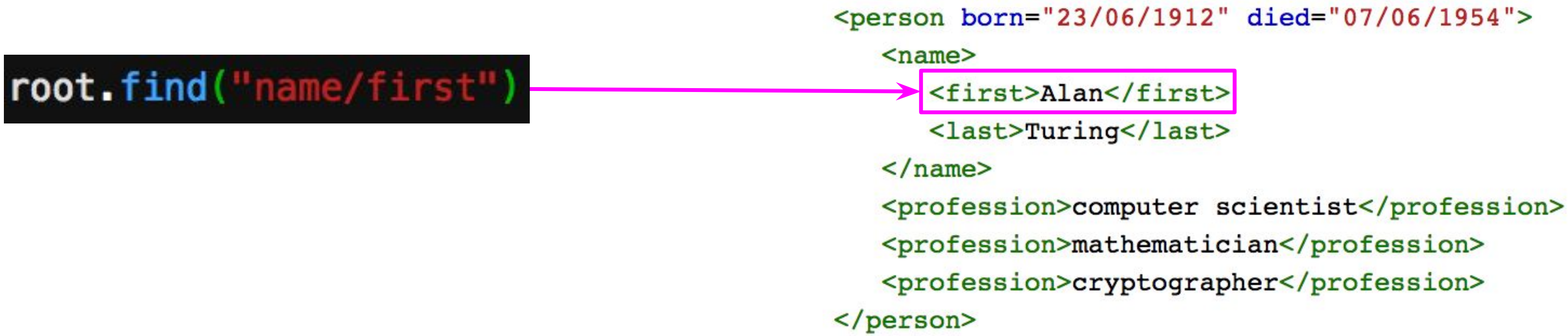
```
name = root.find("name")
name.tag
```

```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Nested XML

- XML can have multiple **nested layers**
- You can use **paths** to get elements deeper in the XML tree



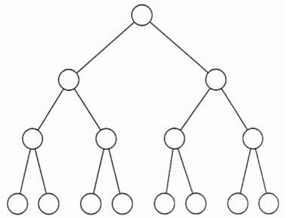


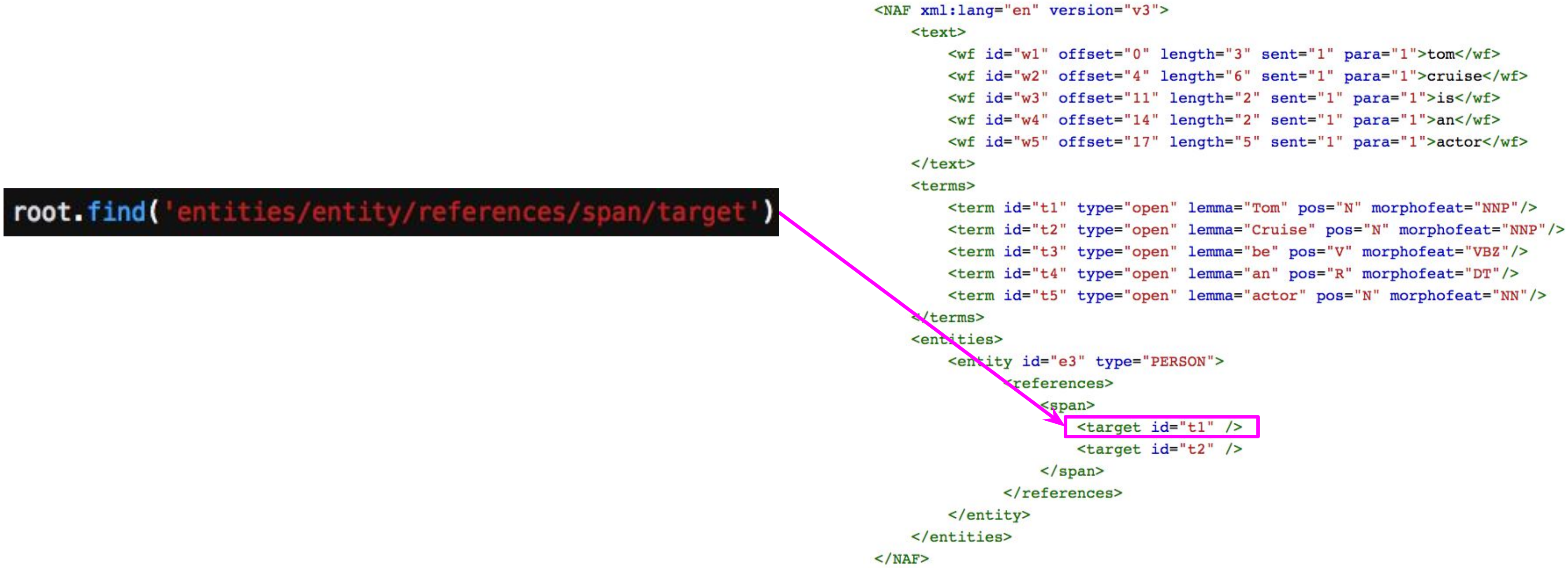
```
root.find("name/first")
```

```
<person born="23/06/1912" died="07/06/1954">
   <name>
      <first>Alan</first>
      <last>Turing</last>
   </name>
   <profession>computer scientist</profession>
   <profession>mathematician</profession>
   <profession>cryptographer</profession>
</person>
```

Nested XML

- XML can have multiple **nested layers**
- You can use **paths** to get elements deeper in the XML tree



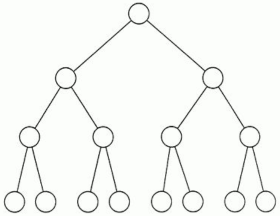


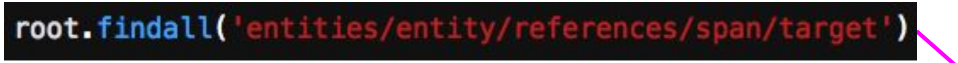
```
root.find('entities/entity/references/span/target')
```

```
<NAF xml:lang="en" version="v3">
    <text>
        <wf id="w1" offset="0" length="3" sent="1" para="1">tom</wf>
        <wf id="w2" offset="4" length="6" sent="1" para="1">cruise</wf>
        <wf id="w3" offset="11" length="2" sent="1" para="1">is</wf>
        <wf id="w4" offset="14" length="2" sent="1" para="1">an</wf>
        <wf id="w5" offset="17" length="5" sent="1" para="1">actor</wf>
    </text>
    <terms>
        <term id="t1" type="open" lemma="Tom" pos="N" morphofeat="NNP"/>
        <term id="t2" type="open" lemma="Cruise" pos="N" morphofeat="NNP"/>
        <term id="t3" type="open" lemma="be" pos="V" morphofeat="VBZ"/>
        <term id="t4" type="open" lemma="an" pos="R" morphofeat="DT"/>
        <term id="t5" type="open" lemma="actor" pos="N" morphofeat="NN"/>
    </terms>
    <entities>
        <entity id="e3" type="PERSON">
            <references>
                <span>
                    <target id="t1" />
                    <target id="t2" />
                </span>
            </references>
        </entity>
    </entities>
</NAF>
```

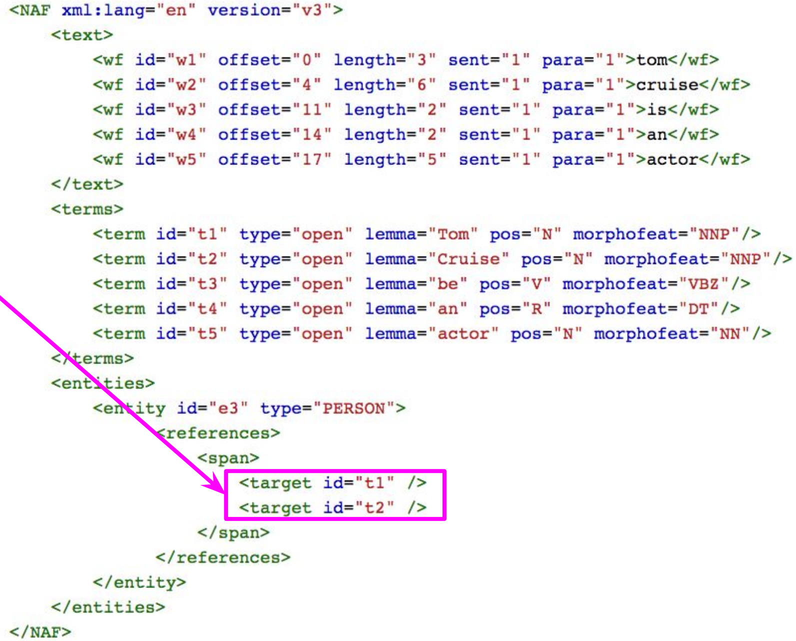
Nested XML

- XML can have multiple **nested layers**
- You can use **paths** to get elements deeper in the XML tree





```
root.findall('entities/entity/references/span/target')
```



```
<NAF xml:lang="en" version="v3">
    <text>
        <wf id="w1" offset="0" length="3" sent="1" para="1">tom</wf>
        <wf id="w2" offset="4" length="6" sent="1" para="1">cruise</wf>
        <wf id="w3" offset="11" length="2" sent="1" para="1">is</wf>
        <wf id="w4" offset="14" length="2" sent="1" para="1">an</wf>
        <wf id="w5" offset="17" length="5" sent="1" para="1">actor</wf>
    </text>
    <terms>
        <term id="t1" type="open" lemma="Tom" pos="N" morphofeat="NNP"/>
        <term id="t2" type="open" lemma="Cruise" pos="N" morphofeat="NNP"/>
        <term id="t3" type="open" lemma="be" pos="V" morphofeat="VBZ"/>
        <term id="t4" type="open" lemma="an" pos="R" morphofeat="DT"/>
        <term id="t5" type="open" lemma="actor" pos="N" morphofeat="NN"/>
    </terms>
    <entities>
        <entity id="e3" type="PERSON">
            <references>
                <span>
                    <target id="t1" />
                    <target id="t2" />
                </span>
            </references>
        </entity>
    </entities>
</NAF>
```

SOCRATIVE

Log in at:

b.socrative.com/student

Room:

PYTHON1819VU

Practicing a bit more

Let’s look at some code :-)

Chapter 18, Exercise 2 (FrameNet)

This week

- On **Thursday**:
  - Revising some difficult concepts
    - Functions
    - For-loops
    - **Your requests?**
  - More time to work on Assignment 4 and to ask your questions (or start preparing for the exam)
- Deadline Assignment 4: **Friday 7 December at 23:59**
- **Reminder:** when sharing your code snippets, please use our e-mail addresses:
  - cm.vanson@gmail.com / c.m.van.son@vu.nl
  - pia.sommerauer@vu.nl
  - (e-mailing both of us has the best chance of getting a quick reply)