CIS 1100

Nested Data

Python
Fall 2024
University of Pennsylvania

JSON & XML

JSON:

- JavaScript Object Notation
- It's basically just Python dictionaries that get printed out. Convenient!
- Use the `json` library to read it.

XML:

- Extensible Markup Language
- Sort of complicated tree structure of elements
- Use the `BeautifulSoup` library to read it via `BeautifulSoup(file, 'xml')`

Any Questions?

JSON

```
[
    {"name" : "CIS1100",
     "section" : 1,
     "days" : ["M", "W", "F"],
     "time" : "12:00pm",
     "instructors" : [
        {"name" : "Harry", "dept" : "CIS", "started" : 2020},
        {"name" : "Jessica", "dept" : "CIS", "started" : 2022}
     ]
    },
    {"name" : "CIS1100",
     "section" : 2,
     "days" : ["M", "W", "F"],
     "time" : "1:45pm",
     "instructors" : [
        {"name" : "Harry", "dept" : "CIS", "started" : 2020},
        {"name" : "Travis", "dept" : "CIS", "started" : 2022}
     ]
    }
]
```

(S7) How many courses are represented? If we parse this JSON, using `json.load` into a variable named `courses_json`, can you write an expression that produces that value?

JSON

```
[
    {"name" : "CIS1100",
     "section" : 1,
     "days" : ["M", "W", "F"],
     "time" : "12:00pm",
     "instructors" : [
        {"name" : "Harry", "dept" : "CIS", "started" : 2020},
        {"name" : "Jessica", "dept" : "CIS", "started" : 2022}
     ]
    },
    {"name" : "CIS1100",
     "section" : 2,
     "days" : ["M", "W", "F"],
     "time" : "1:45pm",
     "instructors" : [
        {"name" : "Harry", "dept" : "CIS", "started" : 2020},
        {"name" : "Travis", "dept" : "CIS", "started" : 2022}
     ]
    }
]
```

(S8) What time does CIS 1100 Section 1 meet? If we parse this JSON, using `json.load` into a variable named `courses_json`, can you write an expression that produces that value?

Describing the Structure

```
[
    {"name" : "CIS1100",
     "section" : 1,
     "days" : ["M", "W", "F"],
     "time" : "12:00pm",
     "instructors" : [
        {"name" : "Harry", "dept" : "CIS", "started" : 2020},
        {"name" : "Jessica", "dept" : "CIS", "started" : 2022}
     ]
    },
    {"name" : "CIS1100",
     "section" : 2,
     "days" : ["M", "W", "F"],
     "time" : "1:45pm",
     "instructors" : [
        {"name" : "Harry", "dept" : "CIS", "started" : 2020},
        {"name" : "Travis", "dept" : "CIS", "started" : 2022}
     ]
    }
]
```

(L11) What keys do the upper level dictionaries have?

What keys do the lower level dictionaries have?

Complete the Program

(C12) Finish this snippet so that it prints out a set containing every instructor's name.

- Don't assume you know how many courses there are
- Don't assume you know how many instructors each course has

```
json_file_of_courses = open("courses.json", "r")
courses_json = json.loads(json_file_of_courses) # dict representing prev. JSON
```

Some XML Terminology

- **Elements** are the entities being represented in the XML tree, e.g. an inventory or a price.
- **Tags** are the names that we give to the elements, e.g. `<inventory>` or `<price>`
- **Attributes** are properties that individual elements can have, stored in the tags
  - If the pop element is specifically a Pepsi, we could have its tag be `<pop brand="Pepsi">`.

```
<fruits>
    <berries>
        <fruit color="red">strawberry</fruit>
        <fruit color="blue">blueberry</fruit>
    </berries>
    <stonefruit>
        <fruit color="purple">plum</fruit>
        <fruit color="orange">peach</fruit>
    </stonefruit>
</fruits>
```

(S7) How many elements? What are the different tags? How many elements have attributes?

Some Tree Terminology

- The **tree** is the collection of elements being represented and the connections between them
- The **root** is the element of the tree that has no ancestors (the initial element).
- An **ancestor** is an element that contains another element.
  - A **parent** is a direct ancestor.
- A **descendant** is an element that is contained by another element.
  - A **child** is a direct descendant.

```
<fruits>
    <berries>
        <fruit color="red">strawberry</fruit>
        <fruit color="blue">blueberry</fruit>
    </berries>
    <stonefruit>
        <fruit color="purple">plum</fruit>
        <fruit color="orange">peach</fruit>
    </stonefruit>
</fruits>
```

(S8) Which element is the root?

Which elements have no children?

Parsing & Traversing XML

Parsing XML:

```
from bs4 import BeautifulSoup
file = open("your_file.xml", "r")
soup = BeautifulSoup(file, "xml")   # Second param tells BSoup how to parse: "xml".
```

This creates a `BeautifulSoup` object that we can navigate and parse through! You can think of **`soup`** as the *entire tree* structure of the xml document.

```
print(soup) # Printing soup will print the entire tree structure to the terminal.
```

However, if you want a more nicely formatted tree, you can do the following:

```
print(soup.prettify())
```

This will make sure `siblings` are printed with the same amount of indentation.

Using .prettify()

```
print(soup)
```

```
<fruits>
<berries>
<fruit color="red">strawberry</fruit>
<fruit color="blue">blueberry</fruit>
</berries>
<stonefruit>
<fruit color="purple">plum</fruit>
<fruit color="orange">peach</fruit>
</stonefruit>
</fruits>
```

```
print(soup.prettify())
```

```
<fruits>
    <berries>
        <fruit color="red">strawberry</fruit>
        <fruit color="blue">blueberry</fruit>
    </berries>
    <stonefruit>
        <fruit color="purple">plum</fruit>
        <fruit color="orange">peach</fruit>
    </stonefruit>
</fruits>
```

Parsing & Traversing XML

Accessing the children of an element:

```
soup = BeautifulSoup(file, "xml")
# Returns a list containing all the children a given element
soup.find_all(recursive = False)

# Returns a singular object, the first child of the element
soup.find(recursive = False)
```

Equivalently, you can use `.contents` which returns a list of all the children of an elem.

```
soup = BeautifulSoup(file, "xml")
# This example grabs the first element in the list `.contents` returns.
root = soup.contents[0]
```

Lecture Activitiy: `recursive = false`

```
from bs4 import BeautifulSoup

xml_file = open("fruits.xml", "r")

soup = BeautifulSoup(xml_file, "xml")
root = soup.contents[0]
element_found = soup.find("stonefruit")
print(element_found)
```

```
<!-- "fruits.xml" -->
<fruits>
    <berries>
        <fruit color="red">strawberry</fruit>
        <fruit color="blue">blueberry</fruit>
    </berries>
    <stonefruit>
        <fruit color="purple">plum</fruit>
        <fruit color="orange">peach</fruit>
    </stonefruit>
</fruits>
```

(L11) Instead of using `soup.find("stonefruit")` to find a `stonefruit` element, we'll use `soup.find("stonefruit", recursive = False)`.

What will be printed? Why?

Lecture Activity, Tags, Names, Strings

Let's go ahead and take a look at this small `xml` document that contains just one element and one tag.

```
<fruit color="blue">blueberry</fruit>
```

```
small_file = open("small_fruit.xml", "r")
soup = BeautifulSoup(small_file, "xml")
blueberry = soup.fruit
print(f"{blueberry.name}, {blueberry.attrs}, {blueberry.string}")
```

L13: Talk to your neighbor and discuss: what are the `.name`, `.attrs`, `.string` of this element? What should be printed?

Lecture Activity: Tags, Names, Strings?

```python
from bs4 import BeautifulSoup

xml_file = "fruits.xml"
xml_handler = open(xml_file, "r")

soup = BeautifulSoup(xml_handler, "xml")

for child in soup.find_all(recursive = False):
    print(child.name, child.attrs, child.string)
```

```xml
<!-- "fruits.xml" -->
<fruits>
    <berries>
        <fruit color="red">strawberry</fruit>
        <fruit color="blue">blueberry</fruit>
    </berries>
    <stonefruit>
        <fruit color="purple">plum</fruit>
        <fruit color="orange">peach</fruit>
    </stonefruit>
</fruits>
```

(S9) Take a look at the code on the left hand side. *What will be printed?*

Grab all the fruit

(M1) Which of these lines return all the fruit elements of a given file, `fruits.xml`, that has the same structure as before. Fill in all that are correct.

```
from bs4 import BeautifulSoup
file = open("fruits.xml")
soup = BeautifulSoup(file, "xml")
...
```

- A) print(soup.find_all("fruits"))
- B) print(soup.find_all("fruit", recursive = False))
- C) print(soup.find("fruit"))
- D) print(root.find_all("fruit"))

Finish the Snippet

(C12) Finish the snippet to print out **just the names** of all the fruits inside of a given file `fruits.xml` that has the same structure as before. Don't assume that the file has the same number of fruits & categories. You can assume that the structure is the same.

```
from bs4 import BeautifulSoup
file = open("fruits.xml")
soup = BeautifulSoup(file, "xml")


...
```

Return of the Books

(L15) Describe the structure of the Library XML (Do you remember where this is from?) What is the root element? What elements do the root element hold?

```
<library>
    <book>
        <title>
            Trust
        </title>
        <author>
            Hernan Diaz
        </author>
        <year>
            2022.0
        </year>
        <pages>
            402.0
        </pages>
        <rating>
            3.82
        </rating>
    </book>
    <!-- More books here -->
</library>
```

Finish the Snippet

(C14) Finish the snippet to print out the total number of pages harry has read based on all the books in his Library, `books.xml`. that has the same structure as seen in the previous slide. Don't assume that there are always the same number of books. You can assume that the structure is the same.

```
from bs4 import BeautifulSoup
file = open("books.xml")
soup = BeautifulSoup(file, "xml")


...
```