Tree-Structured data (JSON)

CS 106 Winter 2021
Some data is **hierarchical**: we think of each part (“node”) as “owning” or “enclosing” some sub-parts, down to some base level.
<table>
<thead>
<tr>
<th></th>
<th>GENERAL WORKS</th>
<th></th>
<th>FINE ARTS</th>
<th></th>
<th>LANGUAGE AND LITERATURE</th>
<th></th>
<th>SCIENCE</th>
<th></th>
<th>GEOGRAPHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>AE Encyclopedias</td>
<td></td>
<td>NA-NB Architecture. Sculpture</td>
<td></td>
<td>PA Classical Language, Literature</td>
<td></td>
<td>QA Mathematics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AY Almanacs</td>
<td></td>
<td>NC-NE Drawing, Painting, Prints</td>
<td></td>
<td>PC2001 French Language</td>
<td></td>
<td>QB Astronomy</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B-BJ</td>
<td>BF Psychology</td>
<td></td>
<td>NK Crafts</td>
<td></td>
<td>PC4001 Spanish Language</td>
<td></td>
<td>QC Physics</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PHILOSOPHY</td>
<td>BL-BX Religion</td>
<td></td>
<td>P</td>
<td>PE English Language</td>
<td></td>
<td>QC Chemistry</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>CB History of Civilization</td>
<td></td>
<td>LF</td>
<td>PE1128 English as a Second Language</td>
<td></td>
<td>QE Geology</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HISTORICAL</td>
<td>CC Archaeology</td>
<td></td>
<td>PL</td>
<td>PF German Language</td>
<td></td>
<td>OH Natural History</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CT General Biography</td>
<td></td>
<td>PN</td>
<td>PL Japanese, Korean, Chinese Languages</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>DA-DQ</td>
<td></td>
<td>PQ</td>
<td>PN Poetry, Theater, Speech, Journalism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HISTORICAL</td>
<td>DK Russian History</td>
<td></td>
<td>PQ</td>
<td>PQ1 French Literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>DS-DT</td>
<td></td>
<td>PQ</td>
<td>PQ6001 Spanish Literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>E186 Colonial History</td>
<td></td>
<td>PR</td>
<td>PR British Literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U.S. HISTORICAL</td>
<td>E456 Civil War</td>
<td></td>
<td>PS</td>
<td>PS American Literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>E740 Twentieth Century</td>
<td></td>
<td>PT</td>
<td>PT German Literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>F1 State Histories</td>
<td></td>
<td>PZ</td>
<td>PZ Children's, Young Adult Literature</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HISTORY OF THE AMERICAS</td>
<td>F381 Texas</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1001 Canada</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>F1201 Mexico, Latin America</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>GEOGRAPHY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sometimes, a node behaves like a **set of attributes**: it has a specific slot set aside for each kind of attribute.
Attributes can have sub-attributes and so on.
Sometimes, a node holds something more like a **sequence** of children.
Sometimes, a node holds something more like a sequence of children.
There are two standard ways that tree-structured data is passed around online:

- **XML**: eXtended Markup Language
- **JSON**: JavaScript Object Notation

Both are “simple” text-based formats for more or less arbitrary data.

Both are accommodated for in the p5 library. We’ll use JSON because it’s nicer to read.
JSON objects

A **JSON Object** is a comma-separated list of key:value pairs, enclosed in curly braces. It behaves like a dictionary! It maps string keys to arbitrary values.

```json
{
    "Student ID": 123,
    "Clicker": "78%",
    "Assignments": "90%",
    "Midterm": "91%",
    "Final": "93%"
}
```
JSON objects

The values in a JSON object can be pretty much anything. ints, floats, strings, arrays, arrays of arrays, even other JSON objects!
"firstName": "John",
"lastName": "Smith",
"age": 35,
"address": {
  "streetAddress": "51 Strange Street",
  "city": "Kitchener",
  "province": "ON",
  "postalCode": "N3K 1E7"
},
"phoneNumbers": [
  {
    "type": "home",
    "number": "519 555-1234"
  },
  {
    "type": "mobile",
    "number": "226 555-4567"
  }
],
"children": ["Eunice", "Murgatroyd"],
"spouse": null
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 35,
  "address": {
    "streetAddress": "51 Strange Street",
    "city": "Kitchener",
    "province": "ON",
    "postalCode": "N3K 1E7"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "519 555-1234"
    },
    {
      "type": "mobile",
      "number": "226 555-4567"
    }
  ],
  "children": ["Eunice", "Murgatroyd"],
  "spouse": null
}
Getting JSON Objects

let obj = loadJSON( "JohnSmith.json" );
Read the contents of the file into a JSONObject.
```json
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 35,
    "address": {
        "streetAddress": "51 Strange Street",
        "city": "Kitchener",
        "province": "ON",
        "postalCode": "N3K 1E7"
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "519 555-1234"
        },
        {
            "type": "mobile",
            "number": "226 555-4567"
        }
    ],
    "children": ["Eunice", "Murgatroyd"],
    "spouse": null
}
```
```json
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 35,
    "address": {
        "streetAddress": "51 Strange Street",
        "city": "Kitchener",
        "province": "ON",
        "postalCode": "N3K 1E7"
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "519 555-1234"
        },
        {
            "type": "mobile",
            "number": "226 555-4567"
        }
    ],
    "children": ["Eunice", "Murgatroyd"],
    "spouse": null
}
```
{"firstName": "John",
"lastName": "Smith",
"age": 35,
"address": {
    "streetAddress": "51 Strange Street",
    "city": "Kitchener",
    "province": "ON",
    "postalCode": "N3K 1E7"
},
"phoneNumbers": [
    {
        "type": "home",
        "number": "519 555-1234"
    },
    {
        "type": "mobile",
        "number": "226 555-4567"
    }
],
"children": ["Eunice", "Murgatroyd"],
"spouse": null}
{  
  "firstName": "John",
  "lastName": "Smith",
  "age": 35,
  "address": {  
    "streetAddress": "51 Strange Street",
    "city": "Kitchener",
    "province": "ON",
    "postalCode": "N3K 1E7"
  },
  "phoneNumbers": [  
    {
      "type": "home",
      "number": "519 555-1234"
    },
    {
      "type": "mobile",
      "number": "226 555-4567"
    }
  ],
  "children": ["Eunice", "Murgatroyd"],
  "spouse": null
}
{  
  "firstName": "John",
  "lastName": "Smith",
  "age": 35,
  "address": {
    "streetAddress": "51 Strange Street",
    "city": "Kitchener",
    "province": "ON",
    "postalCode": "N3K 1E7"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "519 555-1234"
    },
    {
      "type": "mobile",
      "number": "226 555-4567"
    }
  ],
  "children": ["Eunice", "Murgatroyd"],
  "spouse": null
}
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 35,
  "address": {
    "streetAddress": "51 Strange Street",
    "city": "Kitchener",
    "province": "ON",
    "postalCode": "N3K 1E7"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "519 555-1234"
    },
    {
      "type": "mobile",
      "number": "226 555-4567"
    }
  ],
  "children": ["Eunice", "Murgatroyd"],
  "spouse": null
}
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 35,
    "address": {
        "streetAddress": "51 Strange Street",
        "city": "Kitchener",
        "province": "ON",
        "postalCode": "N3K 1E7"
    },
    "phoneNumbers": [ {
        "type": "home",
        "number": "519 555-1234"
    }, {
        "type": "mobile",
        "number": "226 555-4567"
    } ],
    "children": [ "Eunice", "Murgatroyd" ],
    "spouse": null
}
let obj = {};

function preload() {
    obj = loadJSON("JohnSmith.json");
}

https://openprocessing.org/sketch/1130707
function setup() {
    noCanvas();
    createElement("H1", "Practice with a JSON Object");
    createP();
    let fName = obj.firstName;
    createP(fName);
    let lName = obj.lastName;
    createP(lName);
    let fullName = fName + " " + lName;
    createP(fullName);
let age = obj.age;

createP(age);
// "address" is an object within the object "obj".
// An object within an object.

let addr = obj.address;
createP(addr);
print(addr);

let addrStreet = obj.address.streetAddress;
createP(addrStreet);

let addrCity = obj.address.city;
createP(addrCity);

let addrProvince = obj.address.province;
createP(addrProvince);

let addrPostal = obj.address.postalCode;
createP(addrPostal);
// "phoneNumbers" is an array of objects within the object "obj".
// Each of the "PhoneNumbers" has a "type" and a "number".

let pNumbers = obj.phoneNumbers;
createP(pNumbers);

let phone1Type = obj.phoneNumbers[0].type;
createP(phone1Type);
let phone1Number = obj.phoneNumbers[0].number;
createP(phone1Number);

let phone2Type = obj.phoneNumbers[1].type;
createP(phone2Type);
let phone2Number = obj.phoneNumbers[1].number;
createP(phone2Number);

for (i = 0; i < pNumbers.length; i++) {
    createP(pNumbers[i].type);
    createP(pNumbers[i].number);
}
// "kids" is an array of strings within the object "obj".
let kids = obj.children;
for (i = 0; i < kids.length; i++) {
    createP(kids[i]);
}

let partner = obj.spouse;
createP(partner);

createP("The End")
References

• Daniel Shiffman videos:
  • 10.2: What is JSON? Part I - p5.js Tutorial
    • https://www.youtube.com/watch?v=NFkzw6oFtQ
  • 10.3: What is JSON? Part II - p5.js Tutorial
    • https://www.youtube.com/watch?v=118sDpLOClw
• Excellent introduction to JSON objects
• Remember: He uses “var” rather than “let”.