Tree-Structured data (JSON)

CS 106 Winter 2021
Trees

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>Class</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>General Works</td>
</tr>
<tr>
<td>AE</td>
<td>Encyclopedias</td>
</tr>
<tr>
<td>AY</td>
<td>Almanacs</td>
</tr>
<tr>
<td>B</td>
<td>Philosophy</td>
</tr>
<tr>
<td>BF</td>
<td>Psychology</td>
</tr>
<tr>
<td>BL-BX</td>
<td>Religion</td>
</tr>
<tr>
<td>C</td>
<td>History</td>
</tr>
<tr>
<td>CB</td>
<td>History of Civilization</td>
</tr>
<tr>
<td>CC</td>
<td>Archaeology</td>
</tr>
<tr>
<td>CT</td>
<td>General Biography</td>
</tr>
<tr>
<td>D</td>
<td>History</td>
</tr>
<tr>
<td>DA-DQ</td>
<td></td>
</tr>
<tr>
<td>DK</td>
<td>Russian History</td>
</tr>
<tr>
<td>DS-DT</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>U.S. History</td>
</tr>
<tr>
<td>E186</td>
<td>Colonial History</td>
</tr>
<tr>
<td>E456</td>
<td>Civil War</td>
</tr>
<tr>
<td>E740</td>
<td>Twentieth Century</td>
</tr>
<tr>
<td>F</td>
<td>History of the Americas</td>
</tr>
<tr>
<td>F1</td>
<td>State Histories</td>
</tr>
<tr>
<td>F381</td>
<td>Texas</td>
</tr>
<tr>
<td>F1001</td>
<td>Canada</td>
</tr>
<tr>
<td>F1201</td>
<td>Mexico, Latin America</td>
</tr>
<tr>
<td>G</td>
<td>Geography</td>
</tr>
<tr>
<td>N</td>
<td>Fine Arts</td>
</tr>
<tr>
<td>NA-NB</td>
<td>Architecture, Sculpture</td>
</tr>
<tr>
<td>NC-NE</td>
<td>Drawing, Painting, Prints</td>
</tr>
<tr>
<td>NK</td>
<td>Crafts</td>
</tr>
<tr>
<td>P</td>
<td>Language and Literature</td>
</tr>
<tr>
<td>PA</td>
<td>Classical Language, Literature</td>
</tr>
<tr>
<td>PC2001</td>
<td>French Language</td>
</tr>
<tr>
<td>PC4001</td>
<td>Spanish Language</td>
</tr>
<tr>
<td>PE</td>
<td>English Language</td>
</tr>
<tr>
<td>PE1128</td>
<td>English as a Second Language</td>
</tr>
<tr>
<td>PF</td>
<td>German Language</td>
</tr>
<tr>
<td>PL</td>
<td>Japanese, Korean, Chinese Languages</td>
</tr>
<tr>
<td>PN</td>
<td>Poetry, Theater, Speech, Journalism</td>
</tr>
<tr>
<td>PQ1</td>
<td>French Literature</td>
</tr>
<tr>
<td>PQ6001</td>
<td>Spanish Literature</td>
</tr>
<tr>
<td>PR</td>
<td>British Literature</td>
</tr>
<tr>
<td>PS</td>
<td>American Literature</td>
</tr>
<tr>
<td>PT</td>
<td>German Literature</td>
</tr>
<tr>
<td>PZ</td>
<td>Children's, Young Adult Literature</td>
</tr>
<tr>
<td>Q</td>
<td>Science</td>
</tr>
<tr>
<td>QA</td>
<td>Mathematics</td>
</tr>
<tr>
<td>QA76</td>
<td>Computer Science</td>
</tr>
<tr>
<td>QD</td>
<td>Astronomy</td>
</tr>
<tr>
<td>QC</td>
<td>Physics</td>
</tr>
<tr>
<td>QD</td>
<td>Chemistry</td>
</tr>
<tr>
<td>QE</td>
<td>Geology</td>
</tr>
<tr>
<td>OH</td>
<td>Natural History</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.

{
   "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.
{ "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": [
  
  ],
  "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");
}
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);
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”.
API
Application Program Interface

- Used to get data from a server
- For example:
  - openweathermap.org provides weather in a JSON file
  - We write a sketch to request the weather data:
    - Using loadJSON
    - Similar to loading a JSON file from within Open Processing such as “openJSON(‘dogs.json’);”
  - We then display something based on the data in the JSON file
Get yourself a free API from openweathermap.org

- Go to: [https://openweathermap.org/](https://openweathermap.org/)
- Select “API” at the top
- Click on “Please sign up and use our fast and easy-to-work weather APIs for free.”
- Follow the sign up steps and you’ll get a key which allows you to use the openweathermap.org API.
  - Your key will look something like this:
    - 01b0f58045147663b1ae918d34d88b4
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?q=Toronto&APPID=011b0f58045147663b1ea518d34d88b4');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    print("Wind speed in Toronto is: "+ weather.wind.speed);
}

https://openprocessing.org/sketch/1137956#
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?q=Toronto&APPID=011b0f58045147663b1ea518d34d88b4');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    print("Wind speed in Toronto is: "+ weather.wind.speed);
}
let weather = {};

function preload() {
    weather = loadJSON(
        'https://api.openweathermap.org/data/2.5/weather?q=Toronto&APPID=011b0f58045147663b1ea518d34d88b4');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    print("Wind speed in Toronto is: " + weather.wind.speed);
}
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?q=Toronto&APPID=011b0f58045147663b1ea518d34d88b4');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    print("Wind speed in Toronto is: " + weather.wind.speed);
}

Print out the JSON just to show that a JSON object was successfully returned by loadJSON.
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?q=Toronto&APPID=011b0f58045147663b1ea518d34d88b4');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    print("Wind speed in Toronto is: "+ weather.wind.speed);
}

Print out one value just to show something being used from the JSON object
Wind speed is here in the JSON file

Wind speed in Toronto is: 1.57
Let’s look at the API request in more detail

LoadJSON(
    'https://api.openweathermap.org/data/2.5/weather?q=Toronto&APPID=011b0f58045147663blea518d34d88b4');
}

- openweathermap.org must tell us what to specify in our API request. They specify: https://api.openweathermap.org/data/2.5/weather
- The question mark means “here comes a query”
- openweathermap.org tells us that we can use “q=“ to search for a certain city.
- openweathermap.org says that we must sign up for a free APPID and include that in our query.
- The ampersand is an “and” to string two queries together.
Sample API documentation from openweathermap.org

### API call

<table>
<thead>
<tr>
<th>Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>api.openweathermap.org/data/2.5/weather?q={city name}&amp;appid={API key}</td>
<td></td>
</tr>
<tr>
<td>api.openweathermap.org/data/2.5/weather?q={city name},{state code}&amp;appid={API key}</td>
<td></td>
</tr>
<tr>
<td>api.openweathermap.org/data/2.5/weather?q={city name},{state code},{country code}&amp;appid={API key}</td>
<td></td>
</tr>
</tbody>
</table>

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>q</td>
<td>required</td>
<td>City name, state code and country code divided by comma, use ISO 3166 country codes. You can specify the parameter not only in English. In this case, the API response should be returned in the same language as the language of requested location name if the location is in our predefined list of more than 200,000 locations.</td>
</tr>
</tbody>
</table>
Sample API documentation from openweathermap.org

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>appid</td>
<td>required</td>
<td>Your unique API key (you can always find it on your account page under the &quot;API key&quot; tab)</td>
</tr>
<tr>
<td>mode</td>
<td>optional</td>
<td>Response format. Possible values are xml and html. If you don't use the mode parameter format is JSON by default. Learn more</td>
</tr>
<tr>
<td>units</td>
<td>optional</td>
<td>Units of measurement. standard, metric and imperial units are available. If you do not use the units parameter, standard units will be applied by default. Learn more</td>
</tr>
<tr>
<td>lang</td>
<td>optional</td>
<td>You can use this parameter to get the output in your language. Learn more</td>
</tr>
</tbody>
</table>

Examples of API calls:

api.openweathermap.org/data/2.5/weather?q=London&appid={API key}
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?q=London,ON,CA&APPID=001b0f58045147663b1ea518d34d88b4&units=metric');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    text("Celcius temperature in London Ontario is: " + weather.main.temp, 20, 20);
}

https://openprocessing.org/sketch/1137277
Sample API documentation from openweathermap.org

<table>
<thead>
<tr>
<th>Parameters</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>lang</td>
<td>optional</td>
</tr>
</tbody>
</table>

**Examples of API calls**

```
http://api.openweathermap.org/data/2.5/weather?id=524901&lang=fr&appid={API key}
```

We support the following languages that you can use with the corresponded lang variable:

- af Afrikaans
- al Albanian
- ar Arabic
- az Azerbaijani
- bg Bulgarian
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?lang=ar&q=London,ON,CA&APPID=001b0f58045147663b1ea518d34d88b4&units=metric');
}

function setup() {
    createCanvas(400, 200);
    print(weather);
    text("Celcius temperature in London Ontario is: ", weather.main.temp, 20, 20);
}

https://openprocessing.org/sketch/1137983#
Let’s look at the API request again

```javascript
weather = loadJSON(
    'https://api.openweathermap.org/data/2.5/Weather
?lang=ar&
q=London,ON,CA&
APPID=001b0f58045147663b1ea518d34d88b4&
units=metric');
```
Let’s Practice

• Display the following:
  • Waterloo, Ontario
  • “feels_like” temperature (in Celsius)
  • noCanvas()
• Create an “H1” header
• Create a “createP” to hold the temperature
  • Set the paragraph font-size to 24px
let weather = {};

function preload() {
    weather = loadJSON('https://api.openweathermap.org/data/2.5/weather?q=Waterloo,ON,CA&APPID=001b0f58045147663b1ea518d34d88b4&units=metric');
}

function setup() {
    noCanvas();
    print(weather);
    createElement("H1", "Feels Like in Metric")
    let p = createP(weather.main.feels_like);
    p.style("font-size", "24px");
}

https://openprocessing.org/sketch/1138039
Let’s Practice a Bit More

• Visibility
  
  ```javascript
  print(weather.visibility);
  ```

• Longitude
  
  ```javascript
  print(weather.coord.lon);
  ```

• The weather description (this one is confusing)
  
  ```javascript
  print(weather.weather[0].description);
  ```

• Let’s deal with the dates. JavaScript stores dates as number of milliseconds since January 01, 1970, 00:00:00 UTC (Universal Time Coordinated).

  ```javascript
  let date = new Date(obj.dt * 1000);
  let month = date.toLocaleDateString("en-US", {month: "long"});
  let day = date.toLocaleDateString("en-US", {day: "numeric"});
  print(" " + month + " " + day);
  ```

  Date example: [https://openprocessing.org/sketch/1140627#](https://openprocessing.org/sketch/1140627#)
function setup() {
  let unix_timestamp = 1616092816;

  let date = new Date(unix_timestamp * 1000);

  let human_date_format = date.toLocaleDateString() // 3/18/2021

  // other local date formats
  //let human_date_format = date.toLocaleDateString("en-US") // 3/18/2021
  //let human_date_format = date.toLocaleDateString("en-CA") // 2021-03-18

  let month_str = date.toLocaleDateString("en-US", {month: "long"});
  let month_num = date.toLocaleDateString("en-US", {month: "numeric"});
  let day = date.toLocaleDateString("en-US", {day: "numeric"});

  print("Full date: " + human_date_format);
  print("Month string: " + month_str);
  print("Month num: " + month_num);
  print("Day: " + day);
}

https://openprocessing.org/sketch/1140641
References

- Daniel Shiffman video:
  - 10.5: Working with APIs in Javascript p5
    https://www.youtube.com/watch?v=ecT42O6l_WI