[Rosa Say on flickr]

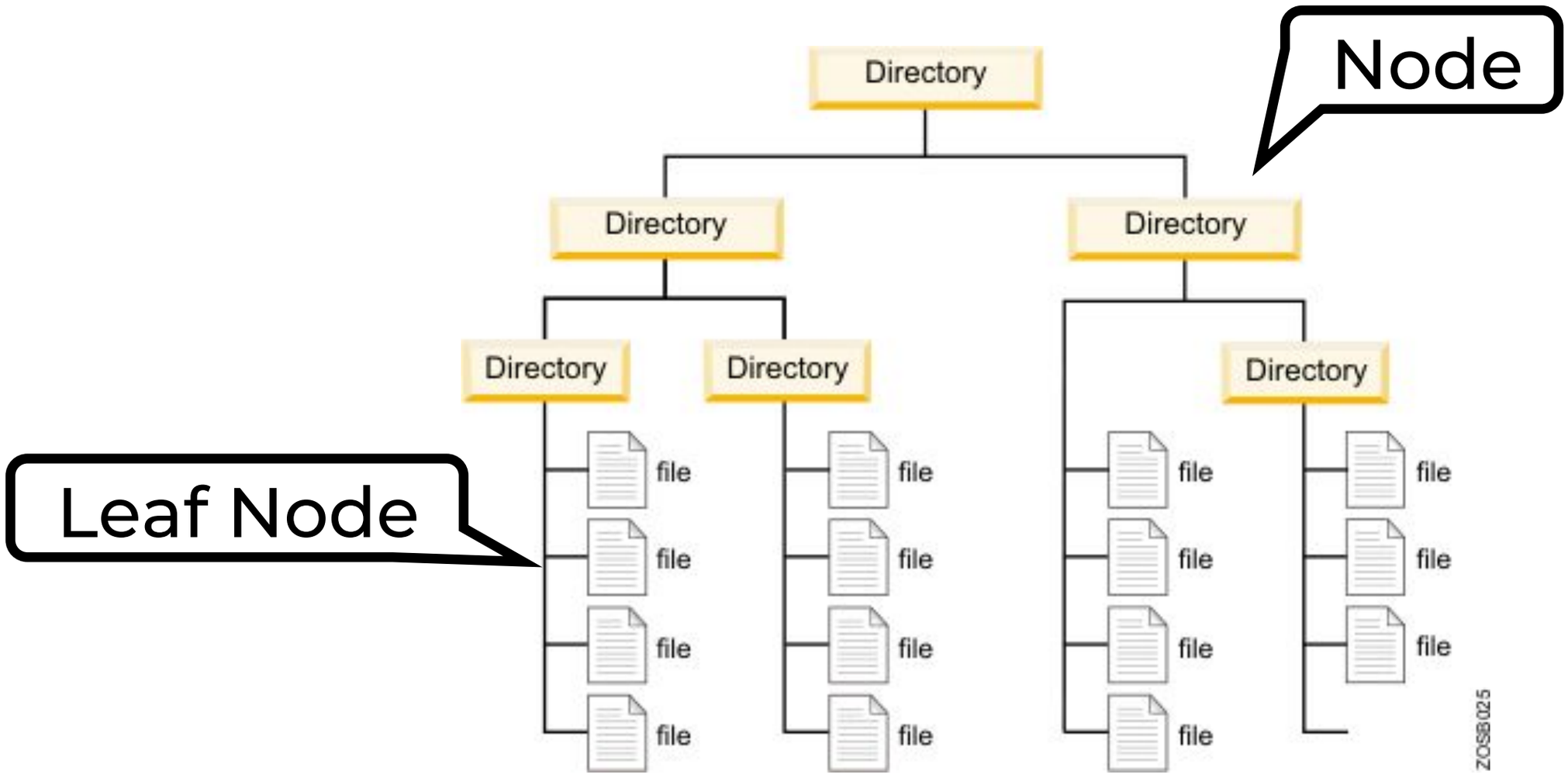
Module 12

Tree-Structured data

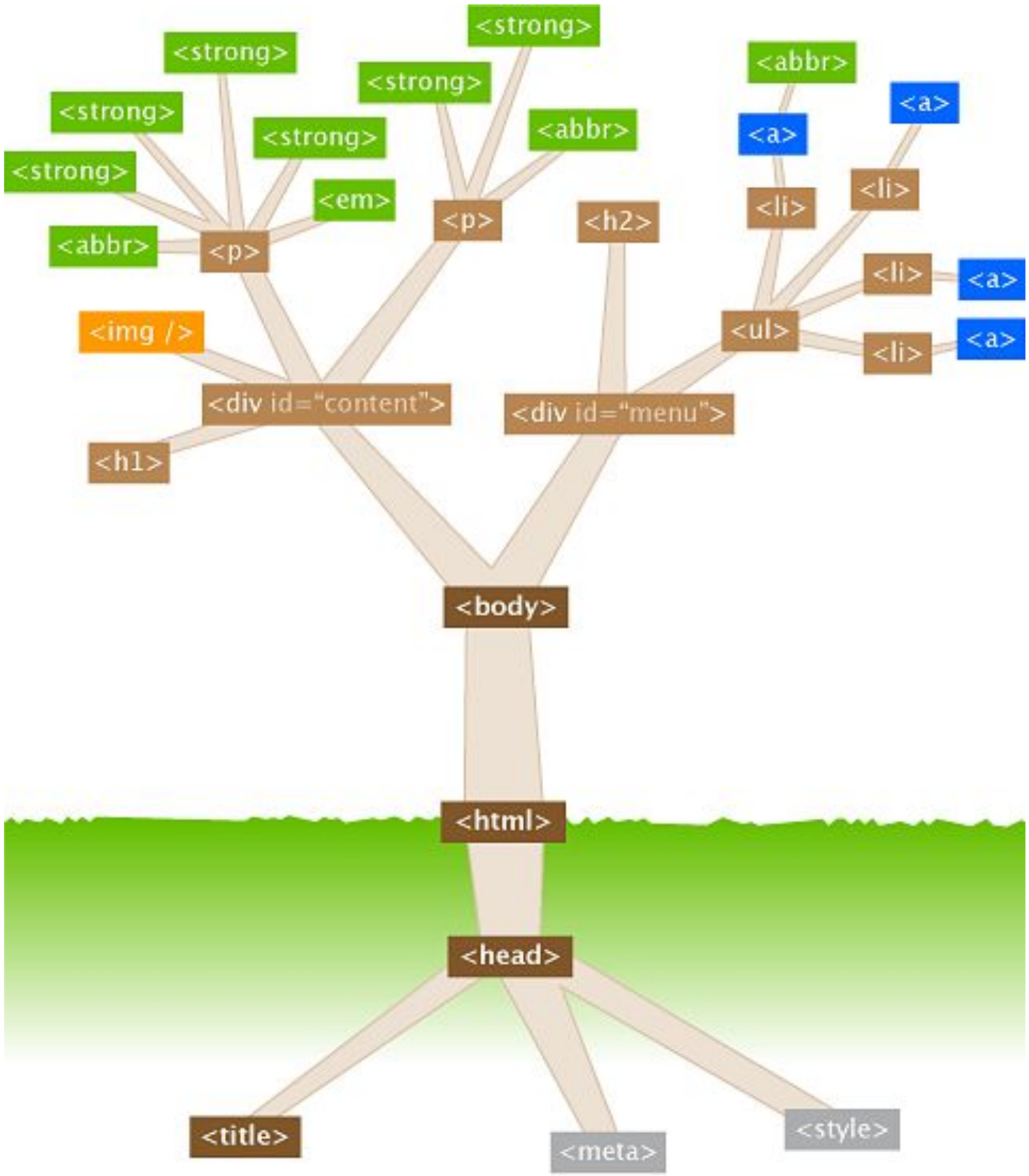
CS 106 Winter 2018

Trees

Some data is **hierarchical**: we think of each part (“node”) as “owning” or “enclosing” some sub-parts, down to some base level.




Home
Shop
Socks
Sports Socks
Business Socks
Shoes
Women's Shoes
Men's Shoes
About Us
Our History
Our People
Contact Us



watershedcreative.com/naked/html-tree.html

LIBRARY OF CONGRESS CLASSIFICATION

A GENERAL WORKS

AE	Encyclopedias
AY	Almanacs

B-BJ PHILOSOPHY

BF	Psychology
BL-BX	Religion

C HISTORY

CB	History of Civilization
CC	Archaeology
CT	General Biography

D HISTORY

DA-DQ	
DK	Russian History
DS-DT	

E U.S. HISTORY

E186	Colonial History
E456	Civil War
E740	Twentieth Century

F HISTORY OF THE AMERICAS

F1	State Histories
F381	Texas
F1001	Canada
F1201	Mexico. Latin America

G GEOGRAPHY

N FINE ARTS

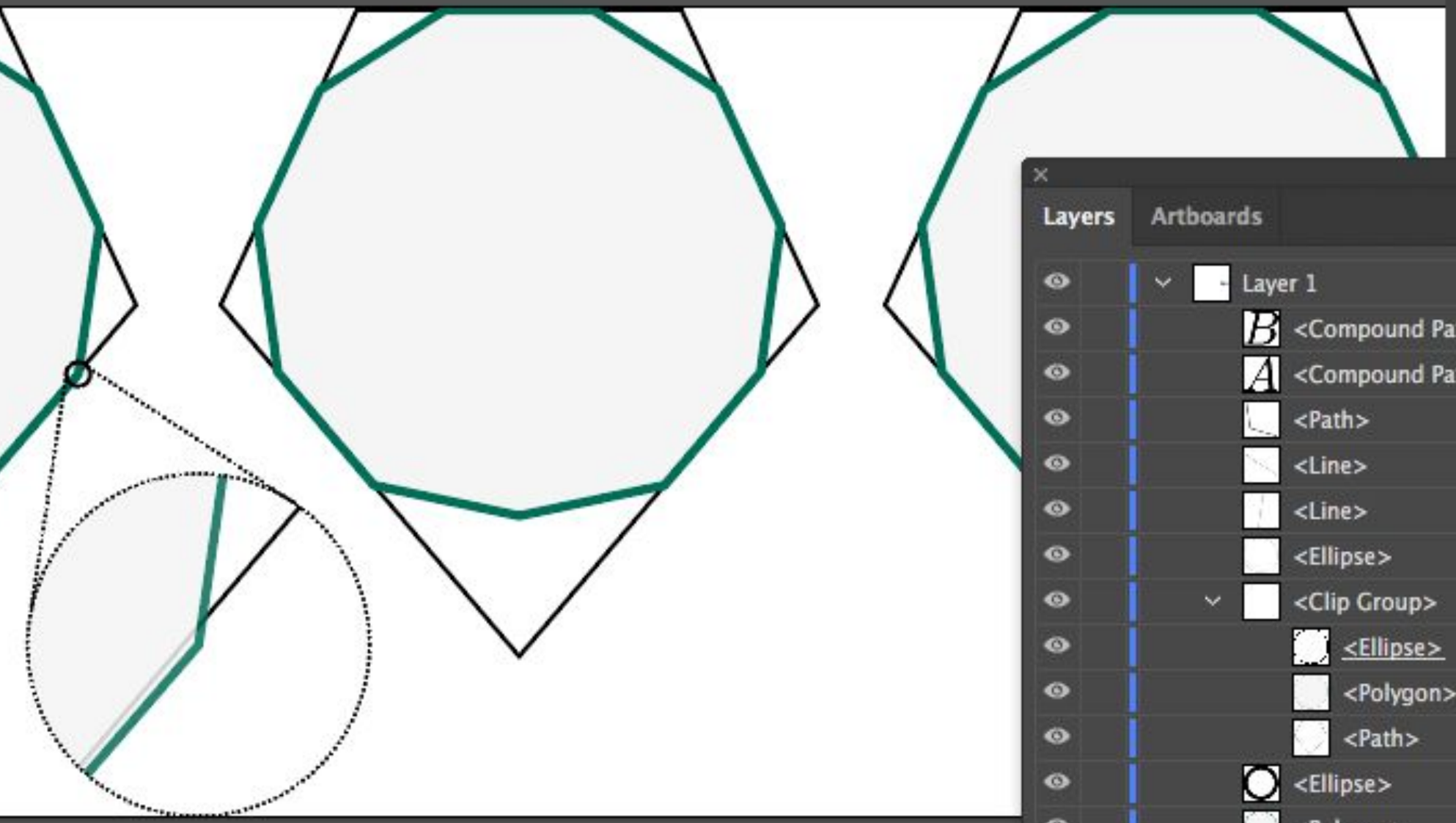
NA-NB	Architecture. Sculpture
NC-NE	Drawing, Painting, Prints
NK	Crafts

P LANGUAGE AND LITERATURE

PA	Classical Language, Literature
PC2001	French Language
PC4001	Spanish Language
PE	English Language
PE1128	English as a Second Language
PF	German Language
PL	Japanese. Korean. Chinese Languages
PN	Poetry. Theater. Speech. Journalism
PQ1	French Literature
PQ6001	Spanish Literature
PR	British Literature
PS	American Literature
PT	German Literature
PZ	Children's, Young Adult Literature

Q SCIENCE

QA	Mathematics
QA76	Computer Science
QB	Astronomy
QC	Physics
QD	Chemistry
QE	Geology
QH	Natural History

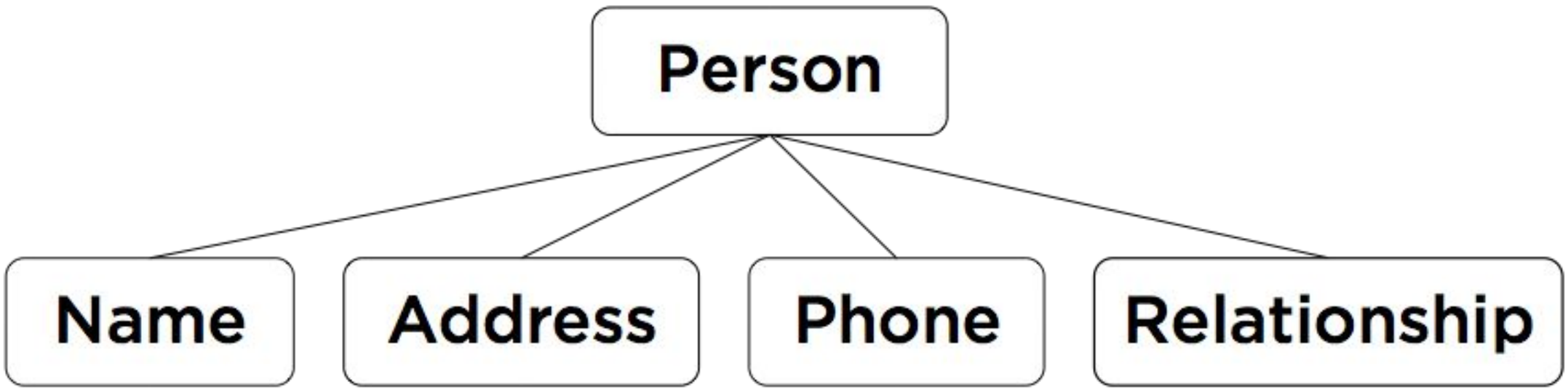


Layers | Artboards

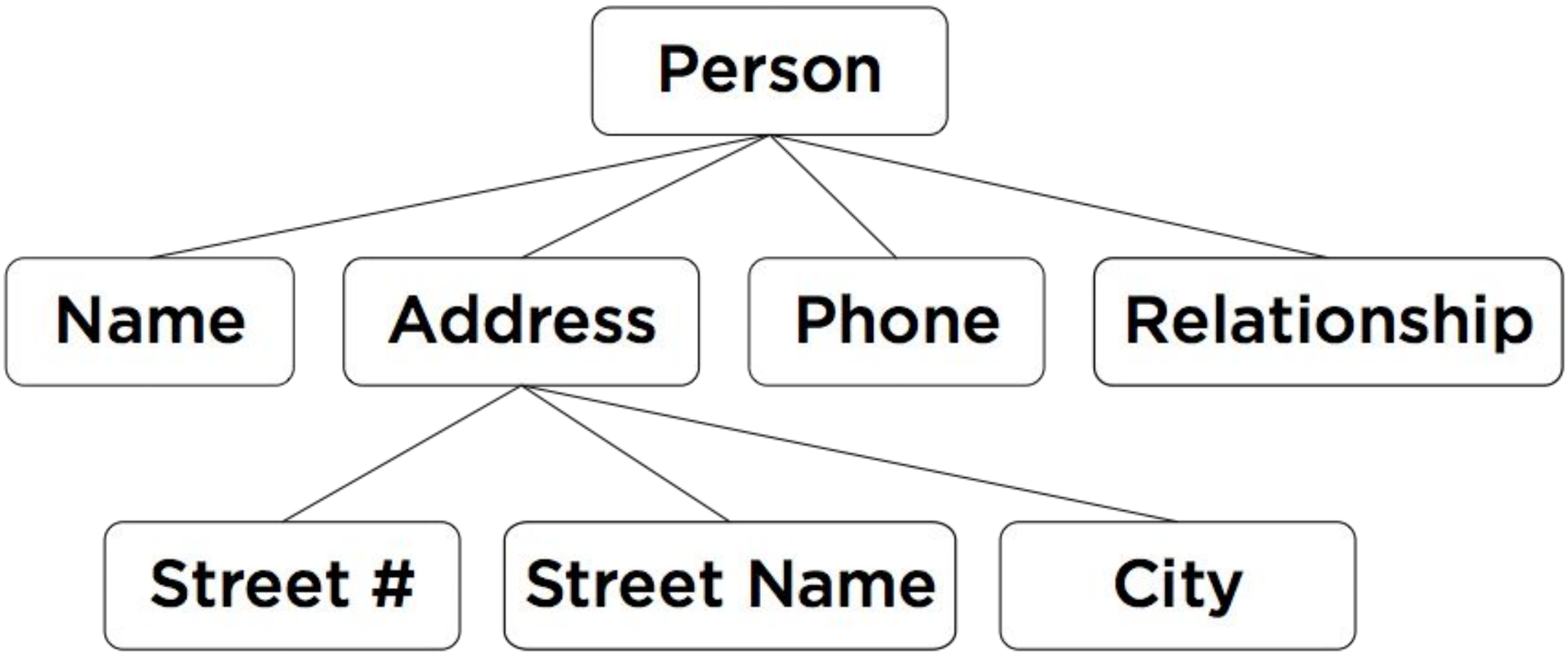
- Layer 1
  - <Compound Path>
  - <Compound Path>
  - <Path>
  - <Line>
  - <Line>
  - <Ellipse>
  - <Clip Group>
    - <Ellipse>
    - <Polygon>
    - <Path>
  - <Ellipse>
  - <Polygon>
  - <Group>
    - <Path>
    - <Path>
  - <Group>
    - <Path>
    - <Path>
  - <Path>

1 Layer

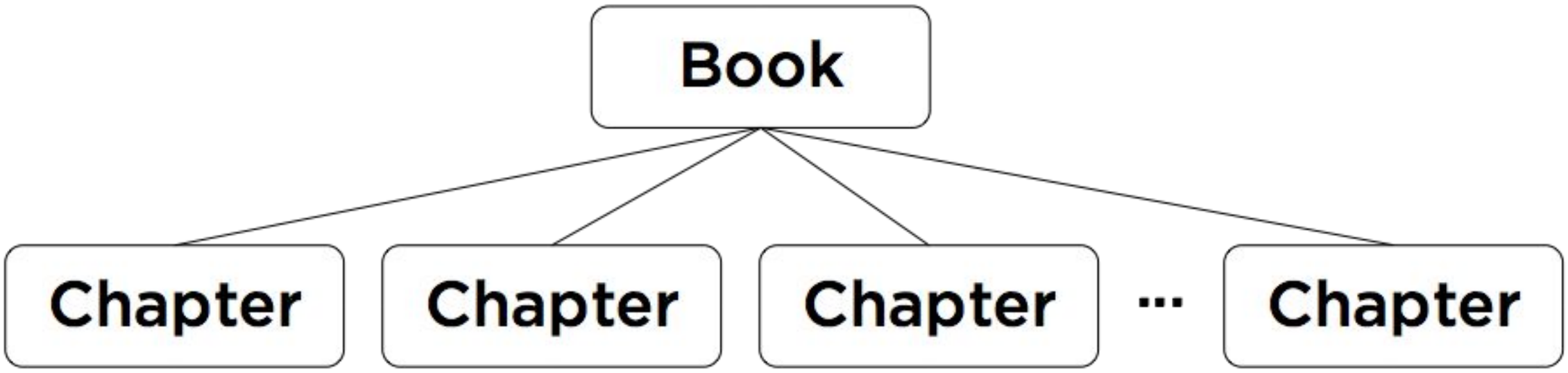
Sometimes, a node behaves like a Processing **class**: it has a specific slot set aside for each kind of child.



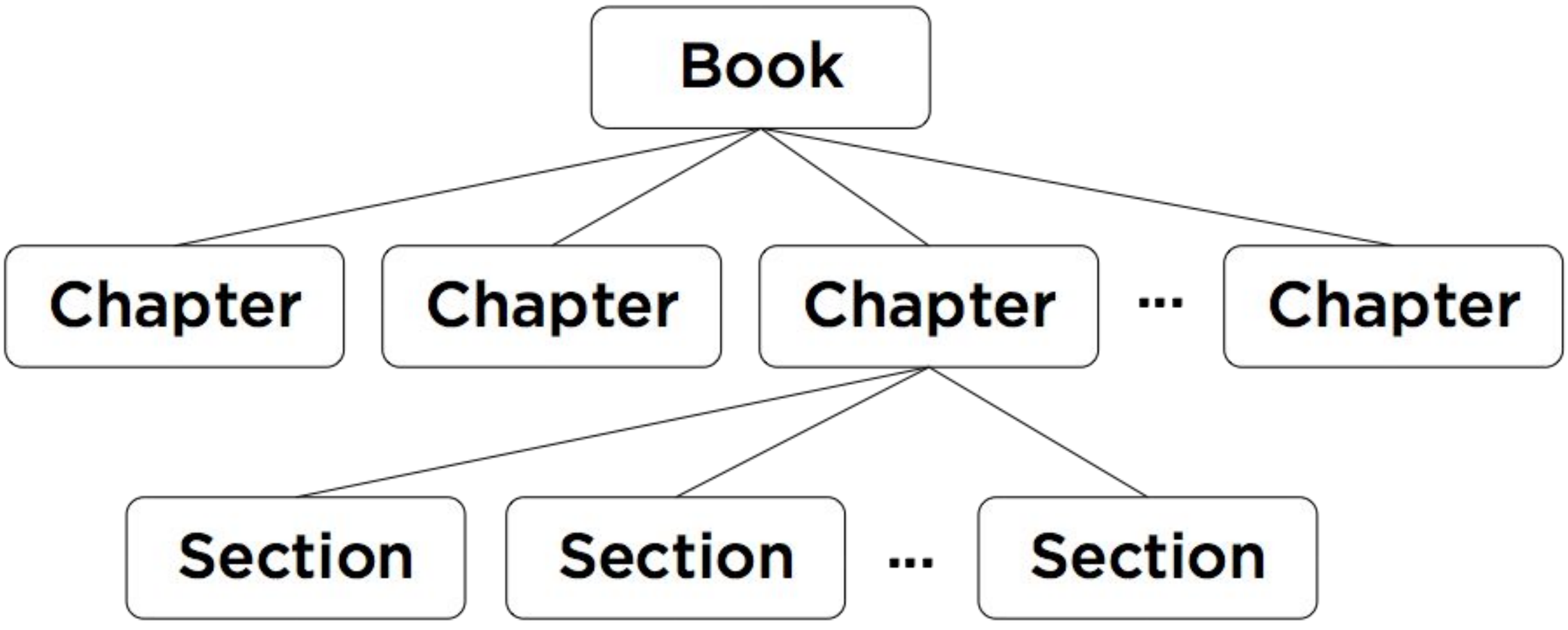
Sometimes, a node behaves like a Processing **class**: it has a specific slot set aside for each kind of child.



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 available in Processing. We’ll use JSON because it’s nicer to read.

JSON is a small subset of the syntax of Javascript, which can be used to describe a few important data types.

Primitive types:

- Integers
- Floats
- Booleans
- Strings

Compound types:

- Arrays
- Trees

JSON primitive values

Integers: `0`, `27`, `-4`…

Floats: `0.003`, `3.1415926`, `-18.77`…

Booleans: `true`, `false`

Strings: “`hello`”, “`pancakes!!`”…

`null`

JSON arrays

A **JSON Array** is a comma-separated list of values, enclosed in square brackets

```
[]
[1, 2, 3]
[1, true, “hello”]
[-3.14, “kumquat”, [true, false]]
```

Unlike Processing, the elements do not need to be of one type!

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.

```
{}

{ “name”:       “Craig”,
  “extension”: 34589 }

{ “digits”: [1,2,3], “good”: true,
  “remark”: “best digits ever” }
```

```
{
  "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

```
JSONObject stuff = loadJSONObject( "filename.json" );
```

Read the contents of the file into a `JSONObject`.

Also `loadJSONArray()`, `parseJSONObject()`, `parseJSONArray()`.

Working with JSONArray

```
JSONArray arr = …

int num_entries = arr.size();

… arr.getInt( 0 ) …
… arr.getFloat( 12 ) …
… arr.getBoolean( idx ) …
… arr.getString( jdx ) …

… arr.getJSONObject( 5 ) …
… arr.getJSONArray( num_entries - 1 ) …
```

Working with JSONObject

```
JSONObject obj = …

… obj.getInt( "key" ) …
… obj.getFloat( "fieldname" ) …
… obj.getBoolean( "phone" ) …
… obj.getString( "address" ) …
… obj.getJSONObject( "whatever" ) …
… obj.getJSONArray( “etc." ) …
```

```
{
  "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
}
```


obj

```
{
  "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
}
```


```
obj.getString( "firstName" );
```

```
{
  "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
}
```


```
obj.getInt( "age" );
```

```
{
  "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
}
```


obj

```
{
  "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
}
```


```
obj.getJSONArray( "phoneNumbers" )
```

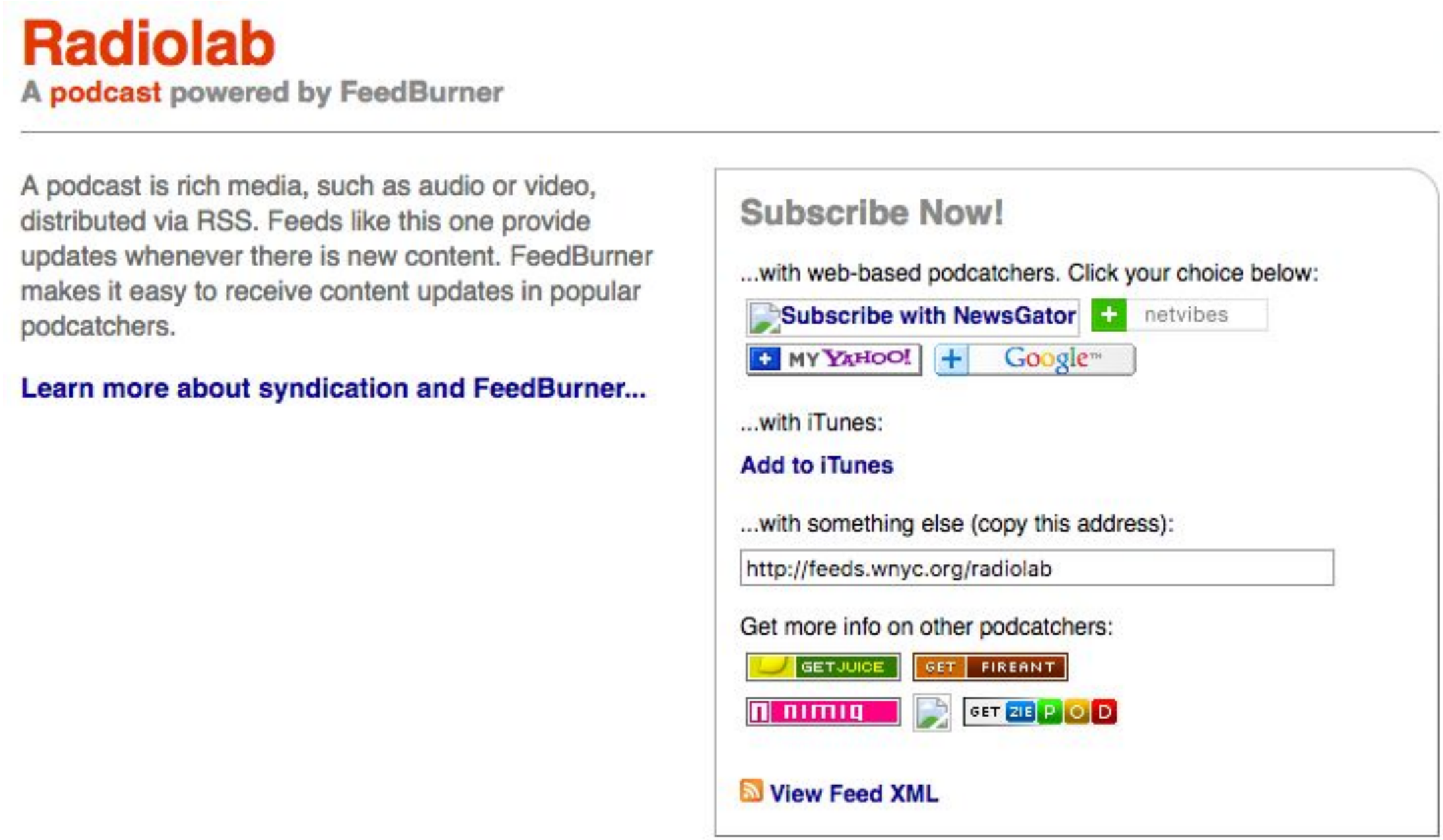
```
{
  "firstName": "John",
  "lastName": "Smith",
  "age": 35,
  "address": {
    "streetAddress": "51 Strange Street",
    "city": "Kitchener",
    "province": "ON",
    "postalCode": "N3K 1E7"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "519 555-1234"
    },
    {
      "type": "mobile",
      "number": "226 555-4567"
    }
  ],
  "children": ["Eunice", "Murgatroyd"],
  "spouse": null
}
```


```
obj.getJSONArray( "phoneNumbers" ).getJSONObject( 1 )
```

```
{
  "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
}
```


```
obj.getJSONArray( "phoneNumbers" ).getJSONObject( 1 ).getString( "number" );
```

Example: RSS Feeds



feeds.wnyc.org/radiolab

<?xml-stylesheet type="text/css" media="screen" href="http://feeds.wnyc.org/~d/style
<rss xmlns:atom="http://www.w3.org/2005/Atom" xmlns:itunes="http://www.itunes.com/dt
    <channel>
        <title>Radiolab</title>
        <link>http://www.radiolab.org/series/podcasts/</link>
        <description>Radiolab is a show about curiosity. Where sound illuminates ide
Radiolab is heard around the country on more than 500 member stations. Check your lo
Embed the Radiolab widget on your blog or website.

Radiolab is supported, in part, by the Alfred P. Sloan Foundation, enhancing public
All press inquiries may be directed to Jennifer Houlihan Roussel at (646) 829-4497.<
        <language>en-us</language>
        <lastBuildDate>Fri, 24 Mar 2017 01:00:00 -0400</lastBuildDate>
        <ttl>600</ttl>
        <itunes:explicit>no</itunes:explicit>
        <atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="self" type="app
        <feedburner:info xmlns:feedburner="http://rssnamespace.org/feedburner/ext/1.
        <atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="hub" href="http
        <media:copyright>© WNYC</media:copyright>
        <media:thumbnail url="https://media2.wnyc.org/i/1400/1400/l/80/1/Radiolab-wn
        <media:keywords>Science,Technology,Philosophy,Education,radiolab,jad,abumrad
        <media:category scheme="http://www.itunes.com/dtds/podcast-1.0.dtd">Science
        <media:category scheme="http://www.itunes.com/dtds/podcast-1.0.dtd">Society
        <media:category scheme="http://www.itunes.com/dtds/podcast-1.0.dtd">Educatio
        <itunes:author>WNYC Studios</itunes:author>
        <itunes:image href="https://media2.wnyc.org/i/1400/1400/l/80/1/Radiolab-wnyc
        <itunes:keywords>Science,Technology,Philosophy,Education,radiolab,jad,abumra

An RSS feed is an XML document. We *could* parse it directly in Processing, but we'll make life simpler by converting it to JSON first.

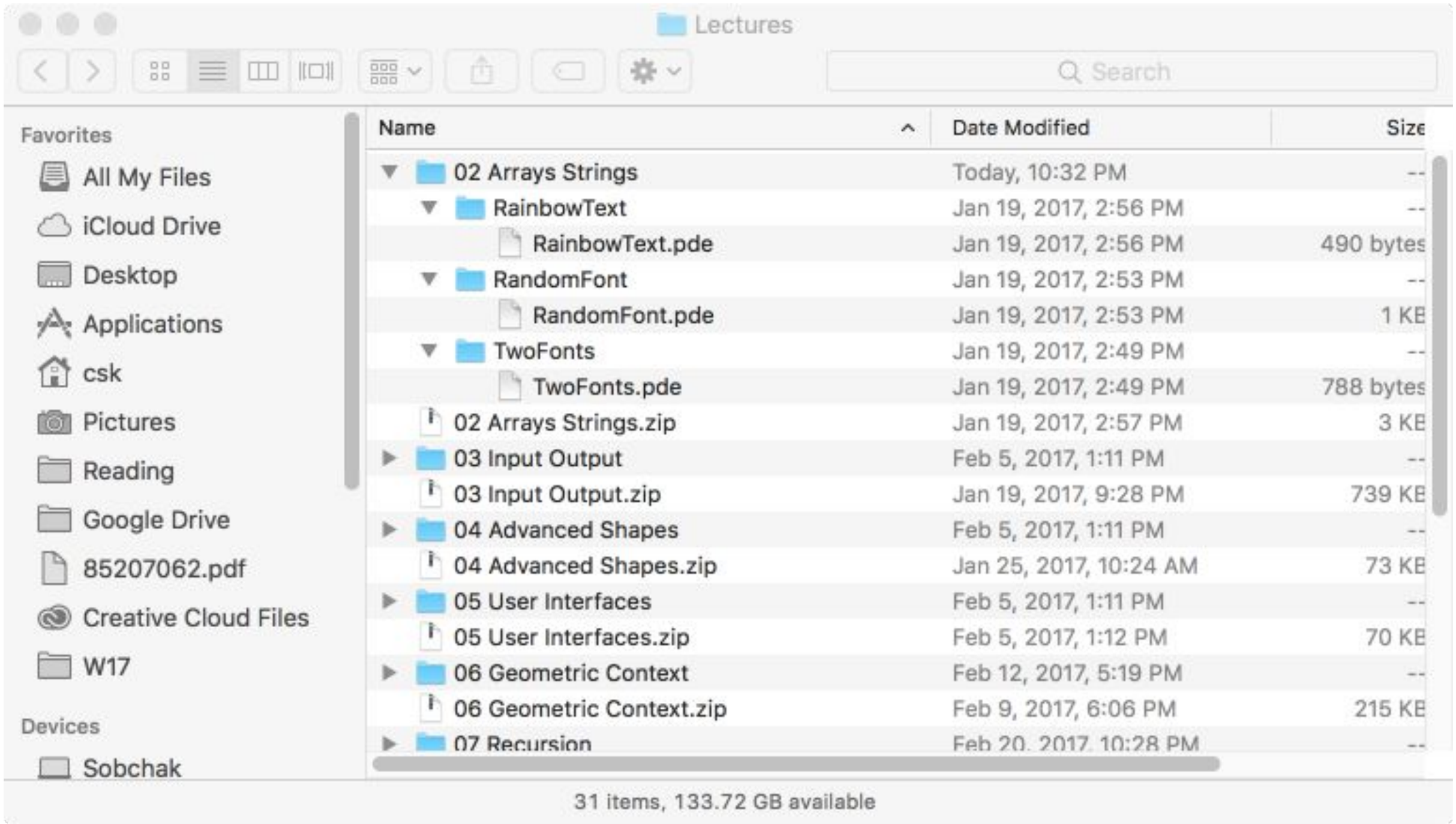
See `rss2json.com`.

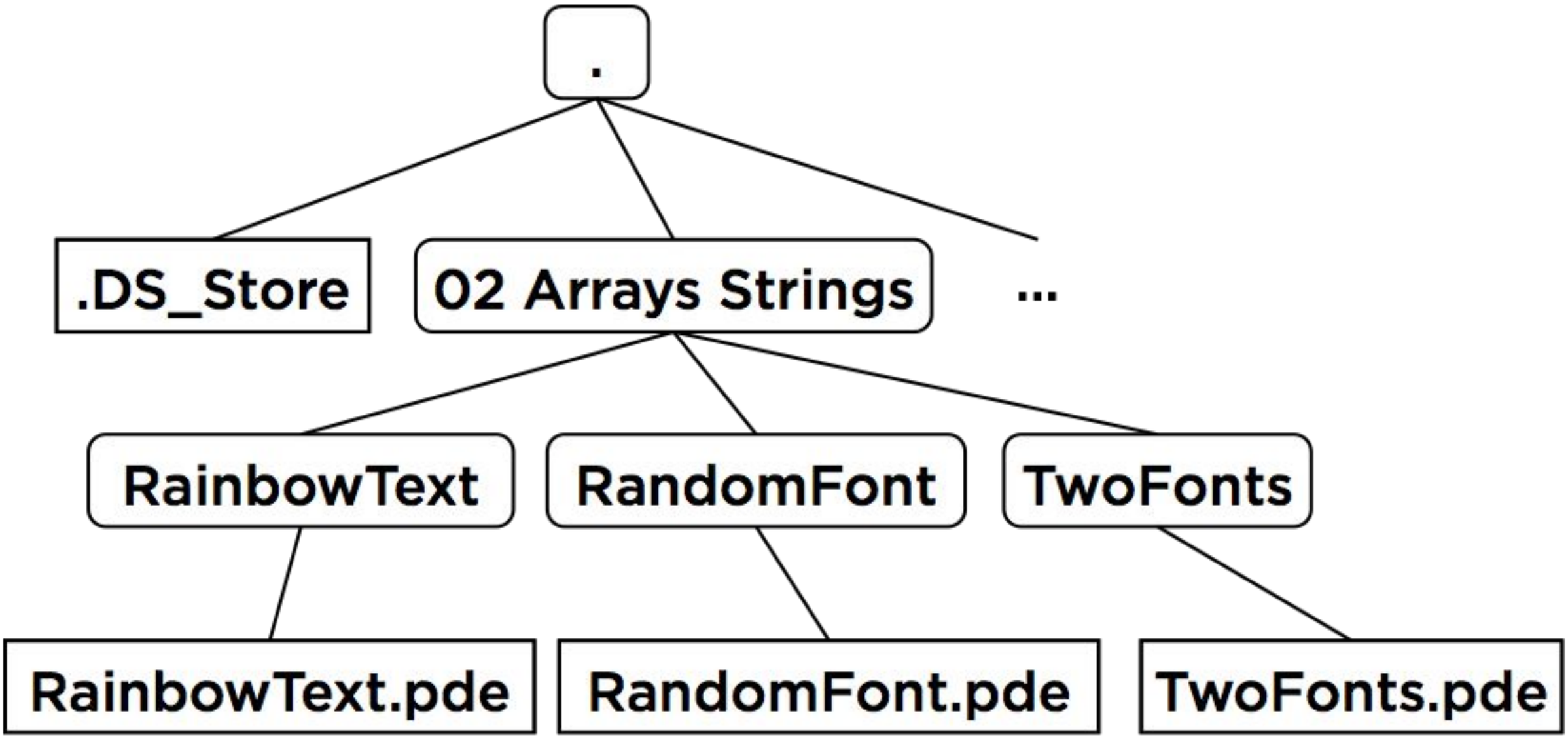
```
"image": "https://media2.wnyc.org/i/1400/1400/l/80/1/Radiolab-wnycstudios.jpg
}
"items": [
{
"title": "Shots Fired: Part 2"
"pubDate": "2017-03-24 05:00:00"
"link": "http://www.radiolab.org/story/shots-fired-part-2/"
"guid": "http://www.radiolab.org/story/shots-fired-part-2/"
"author": "WNYC Studios"
"thumbnail": "https://media2.wnyc.org/i/130/130/c/80/1/3957814193_6fd835e7c0_
"description": "We again join Ben Montgomery, reporter at the Tampa Bay Times
"content": "We again join Ben Montgomery, reporter at the Tampa Bay Times, as
"enclosure": {
"link": "https://www.podtrac.com/pts/redirect.mp3/audio.wnyc.org/rl_extras/rl
"type": "audio/mpeg"
```


```
JSONObject obj = …;

String first_title =
    obj.getJSONArray( "items" )
       .getJSONObject( 0 )
       .getString( "title" );
```

Example: counting files




.
.DS_Store
02 Arrays Strings
...
RainbowText
RandomFont
TwoFonts
RainbowText.pde
RandomFont.pde
TwoFonts.pde

```
"name": ".",
"children": [
  {
    "type": "file",
    "name": ".DS_Store"
  },
  {
    "type": "directory",
    "name": "02 Arrays Strings",
    "children": [
      {
        "type": "directory",
        "name": "RainbowText",
        "children": [
          {
            "type": "file",
            "name": "RainbowText.pde"
          }
        ]
      },
      {
        "type": "directory",
        "name": "RandomFont",
        "children": [
          {
            "type": "file",
```


+1!
LEVEL
UP

Going live

All `load` functions accept URLs as parameters in addition to file names!

```
loadStrings()
loadImage()
loadShape()
loadTable()
loadJSONObject()
loadJSONArray()
```

Functions like `loadStrings()` and `loadImage()` allow you to access fixed content over the internet. `loadJSONObject()` is more like *calling a function over the web*.

OPEN DATA API

Open Data API home

Register for an API key

Contact us

Welcome to Open Data API

Hello and Heads up! (September 18th, 2017)

Hello!

I wanted to let you know that effective immediately the Open Data API project at Waterloo is under a new team. We're looking forward to understanding what exists now, getting feedback from current users, and having a clear plan to communicate before moving forward. We'd like to continue to build on the great work done by those before us that made this project possible.

api.uwaterloo.ca

r the immediate future we

tem and fix only critical issues without introducing new features. Part of the motivation for

Example: classrooms

The UW API supports requests like “what courses are scheduled in a given classroom?”

```
GET /buildings/{building}/{room}/courses.{format}
```

Example: classrooms

The UW API supports requests like “what courses are scheduled in a given classroom?”

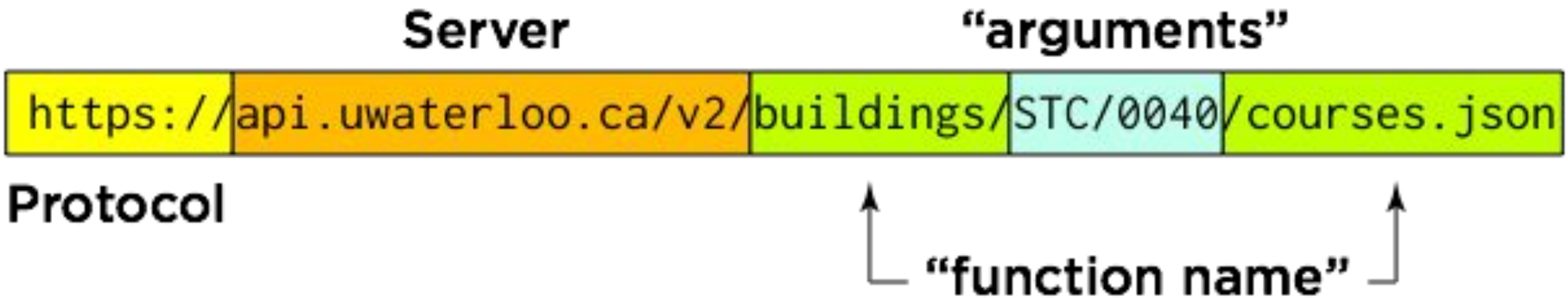
```
GET /buildings/{building}/{room}/courses.{format}
```

```
https://api.uwaterloo.ca/v2/buildings/STC/0040/courses.json
```

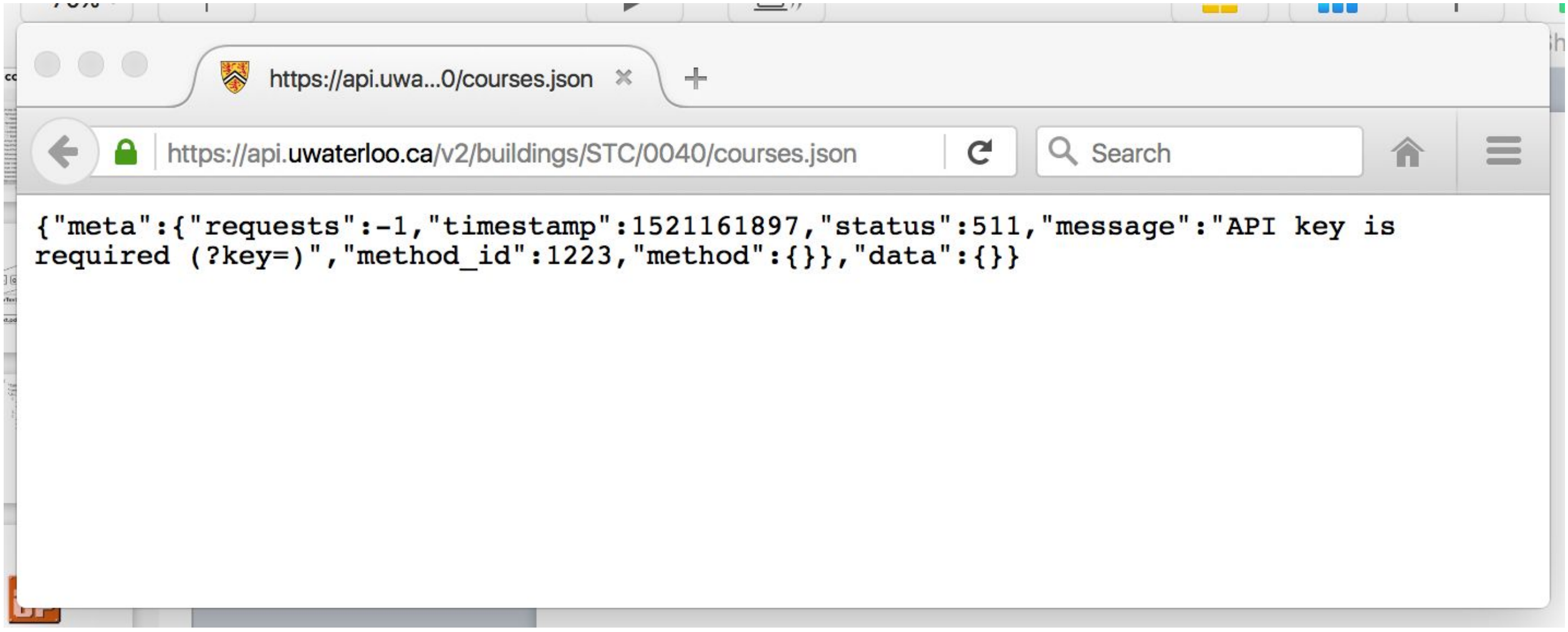
Example: classrooms

The UW API supports requests like “what courses are scheduled in a given classroom?”

```
GET /buildings/{building}/{room}/courses.{format}
```



Most online APIs require you to register for a key.

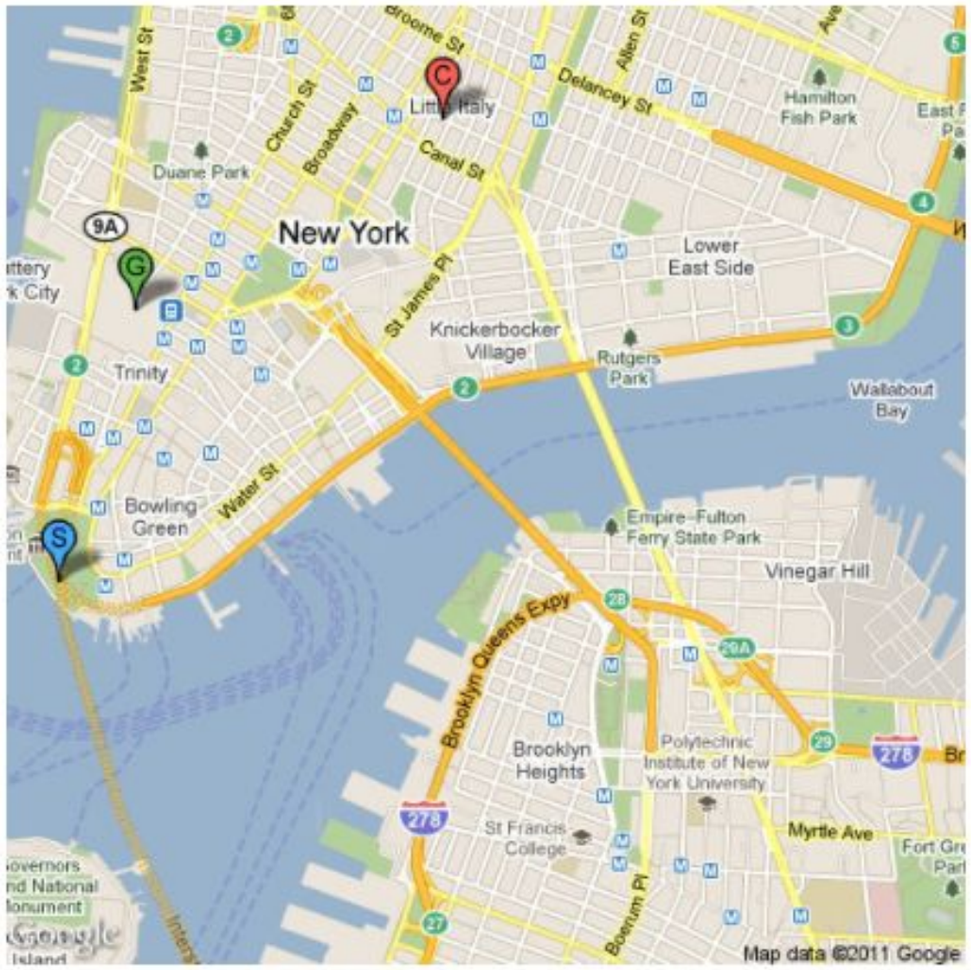


The Google APIs

Google offers dozens of APIs for web designers and developers.

Some are specifically related to popular Google products, like Gmail and Analytics, while others are more specialized and aren't part of public programs.

All are free to use, of course. You can view all of Google's APIs and code tools on their site directory.



- Feed API – The Google Feed API lets you download any public feed (including RSS, Media RSS, and Atom) and then combine them into mashups. It simplifies the mashup process by using JavaScript rather than more complex server-side coding.
- Places API – Google Places is a large directory of local businesses and attractions all around the world. The Places API lets you access that information and display it on your website, as well as display check-ins by users.
- Geocoding API – The Geocoding API lets you convert any address into geographic