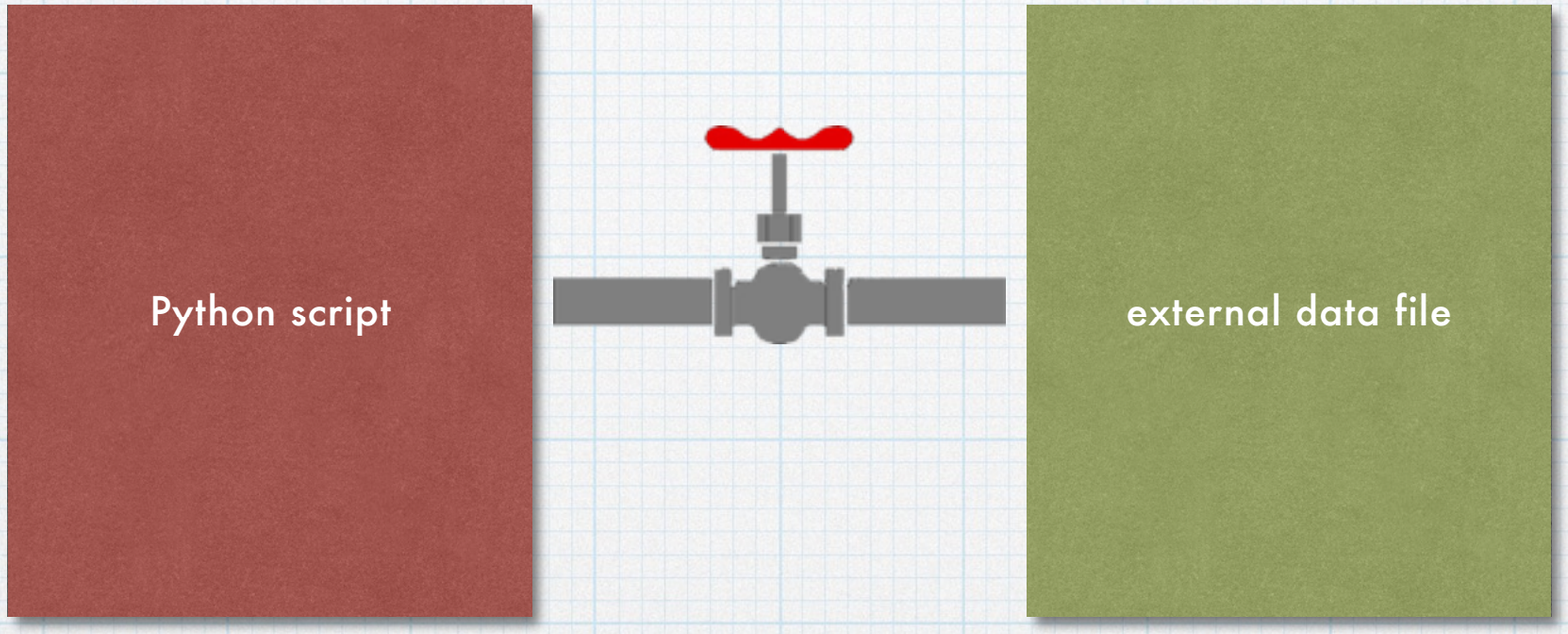
D3 Data Loading

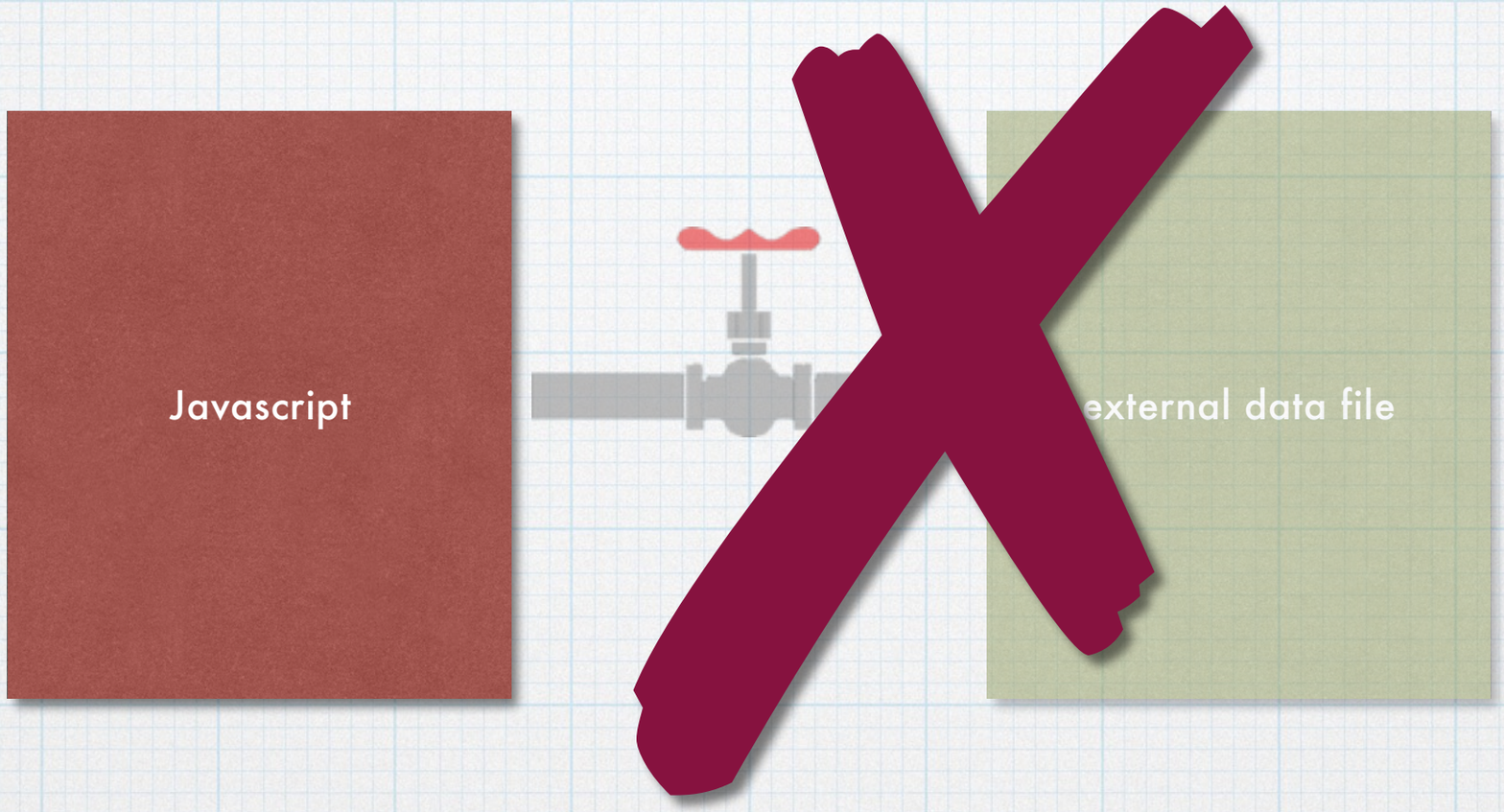
Now with Promises

Python and Files

In Python and other scripting languages, we often read data from—and write data to—external data files. We open a metaphoric pipeline for that information transfer:



JAVASCRIPT AND FILES

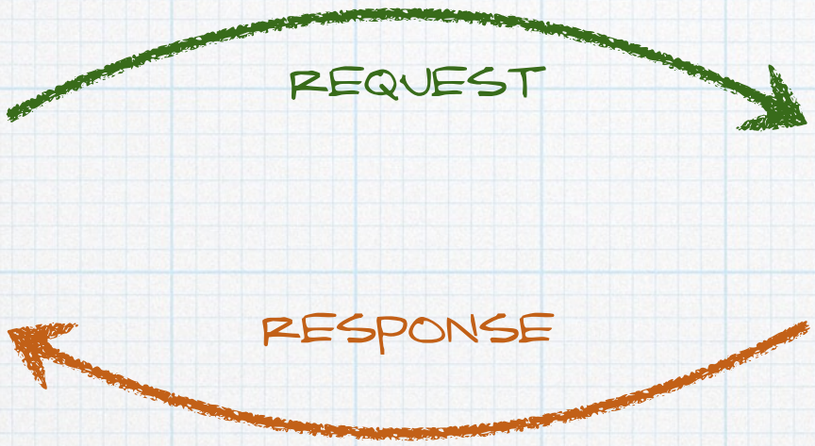


However, Javascript cannot do this. "Standard JavaScript provides no such functionality," writes Marijn Haverbeke in *Eloquent Javascript*.

That's by design. When we visit a webpage, we assume the page is safe—that it will not delete files from our hard drive or copy files from our hard drive. Webpages should be black-boxed and independent. Javascript has been handicapped for our safety.

HTTP REQUESTS

USER / CLIENT



SERVER

Our browsers use a language called **HyperText Transport Protocol** (HTTP) to ask for webpages and other resources (images, files, media, etc.). That's called a **request** and they are written in plain text, believe it or not. The **response** includes the resources we've asked for.

Browsers obviously have the capacity to generate and to send requests. But how?

LOADING DATA SOLUTION

We can use Python to start a small, tiny, "baby" web server on your computer. Once we do that, we need to enter a URL into the address bar of our browser (rather than using the File > Open technique) because D3 needs to use the HTTP protocol to fetch the data.

a) In Terminal (Mac) or Anaconda Shell (Windows), change directories to the location of your D3 code.

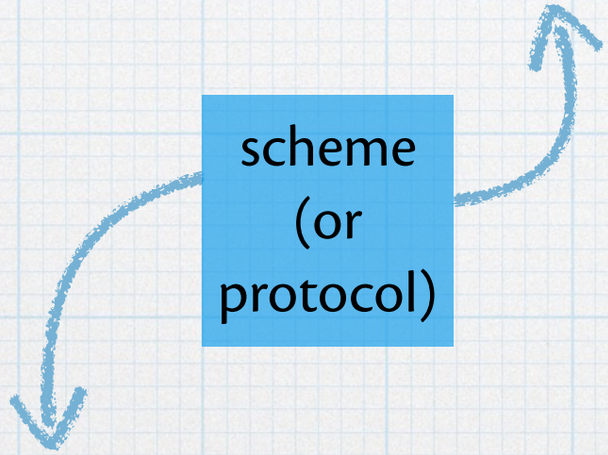
b) **`python -m http.server 8888`**

c) In browser, go to **`http://localhost:8888/`***`filename.html`*

PARTS OF THE URL

We can use Python to start a small, tiny, "baby" web server on your computer. Once we do that, we need to enter a URL into the address bar of our browser (rather than using the File > Open technique) because D3 needs to use the HTTP protocol to fetch the data.

```
python -m http.server 8888
```



http://localhost:8888/*filename.html*

PARTS OF THE URL

We can use Python to start a small, tiny, "baby" web server on your computer. Once we do that, we need to enter a URL into the address bar of our browser (rather than using the File > Open technique) because D3 needs to use the HTTP protocol to fetch the data.

```
python -m http.server 8888
```

`http://`**`localhost`**`:8888/`*`filename.html`*

PARTS OF THE URL

We can use Python to start a small, tiny, "baby" web server on your computer. Once we do that, we need to enter a URL into the address bar of our browser (rather than using the File > Open technique) because D3 needs to use the HTTP protocol to fetch the data.

```
python -m http.server 8888
```

port (default 80)

```
http://localhost:8888/filename.html
```

NOW WE NEED TO LOAD DATA

D3 has written some data loading functions that help us. The plumbing under the hood is quite complex and we're not going to go there.* Depending what our data looks like, we can call one of many different functions to get our data:

```
d3.csv( <filename>, <row processing fn> )    for comma-separated values

d3.tsv( <filename>, <row processing fn> )    for tab-separated values

d3.json( <filename>, <row processing fn> )   for JSON data

d3.xml( <filename>, <row processing fn> )    for XML data
```

* If you're curious, D3 uses an asynchronous technique called *Fetch* to accomplish this. But there are some consequences to this behaviour that we need to deal with (starting in the next slide).

JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

```
d3.csv( <file>, <row conversion> )
  .then( <callback function> )
  .catch( <callback function> );
```

JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

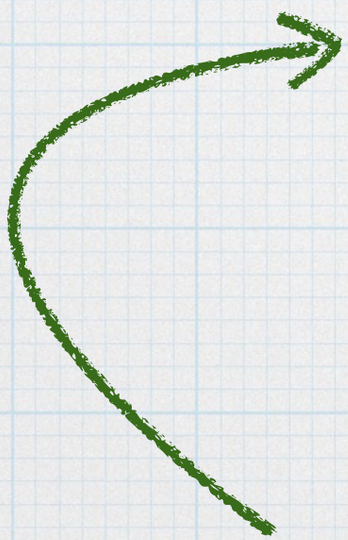
```
d3.csv( <file>, <row conversion> )
.then( <callback function> )
.catch( <callback function> );
```

a file to load

JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

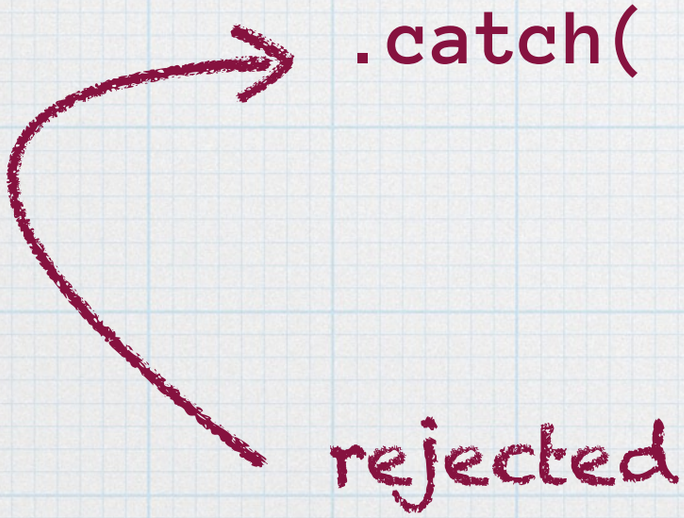
```
d3.csv( <file>, <row conversion> )
    .then( <callback function> )
    .catch( <callback function> );
```



JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

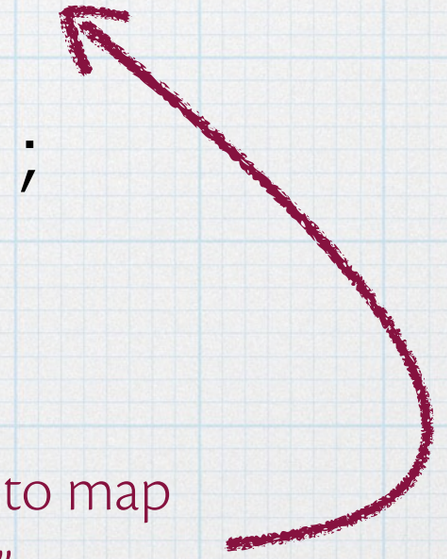
```
d3.csv( <file>, <row conversion> )
    .then( <callback function> )
    .catch( <callback function> );
```



JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

```
d3.csv( <file>, <row conversion> )
  .then( <callback function> )
  .catch( <callback function> );
```



"An optional row conversion function may be specified to map and filter row objects to a more-specific representation."

D3 API documentation

JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

```
d3.csv( <file>, <row conversion> )
    .then( <callback function> )
    .catch( <callback function> );
```

You can think of this like an if/else conditional. If this file loads THEN do this function, ELSE do the other function.

JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

```
d3.csv( <file>, <row conversion> )
  .then( <callback function> )
  .catch( <callback function> );
```

If you know how Exceptions work, this is really more like a try/catch block (or, in Python, try/except).

JAVASCRIPT PROMISES

D3 (starting with version 5) uses a Javascript function called **`fetch()`**, which loads data via HTTP "behind the scenes" (that is, asynchronously). A function like **`d3.csv()`** now creates a "***promise***," an obligation that will eventually either be ***fulfilled*** (if the data loads properly, for example) or ***rejected*** (if the CSV file doesn't exist, for example). While in progress, the promise is ***pending***.

```
d3.csv('data/parents_1550.csv', clean_rows)
  .then( function(d) {} )
  .catch( function(e) {} );
```

D3 Data Loading

Now with Promises