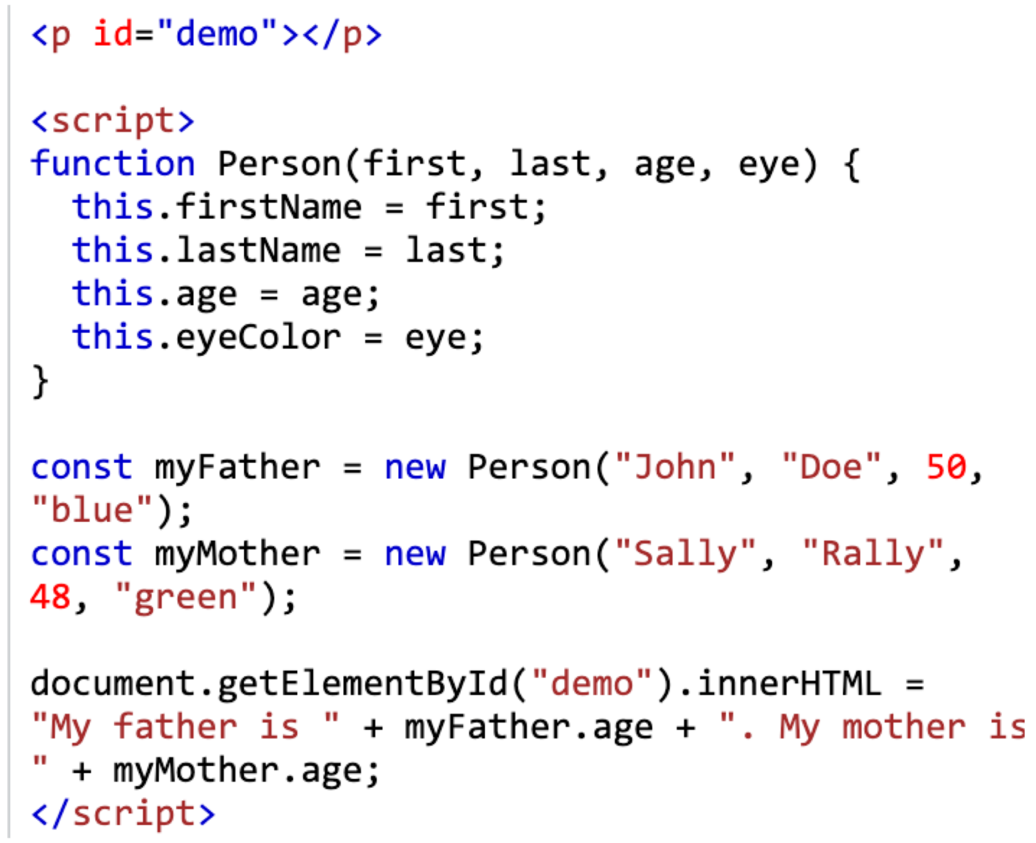
TECNOLOGIE SOFTWARE PER IL WEB

Promise, fetch, JSON
A.A. 2021/22

Prof. R. Francese

Javascript - Prototype

- All JavaScript objects inherit properties and methods from a prototype.
- **object constructor**:



```
<p id="demo"></p>

<script>
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}

const myFather = new Person("John", "Doe", 50,
"blue");
const myMother = new Person("Sally", "Rally",
48, "green");

document.getElementById("demo").innerHTML =
"My father is " + myFather.age + ". My mother is
" + myMother.age;
</script>
```

Javascript - Prototype (2)

- We can **not** add a new property to an existing object constructor:
- https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype3

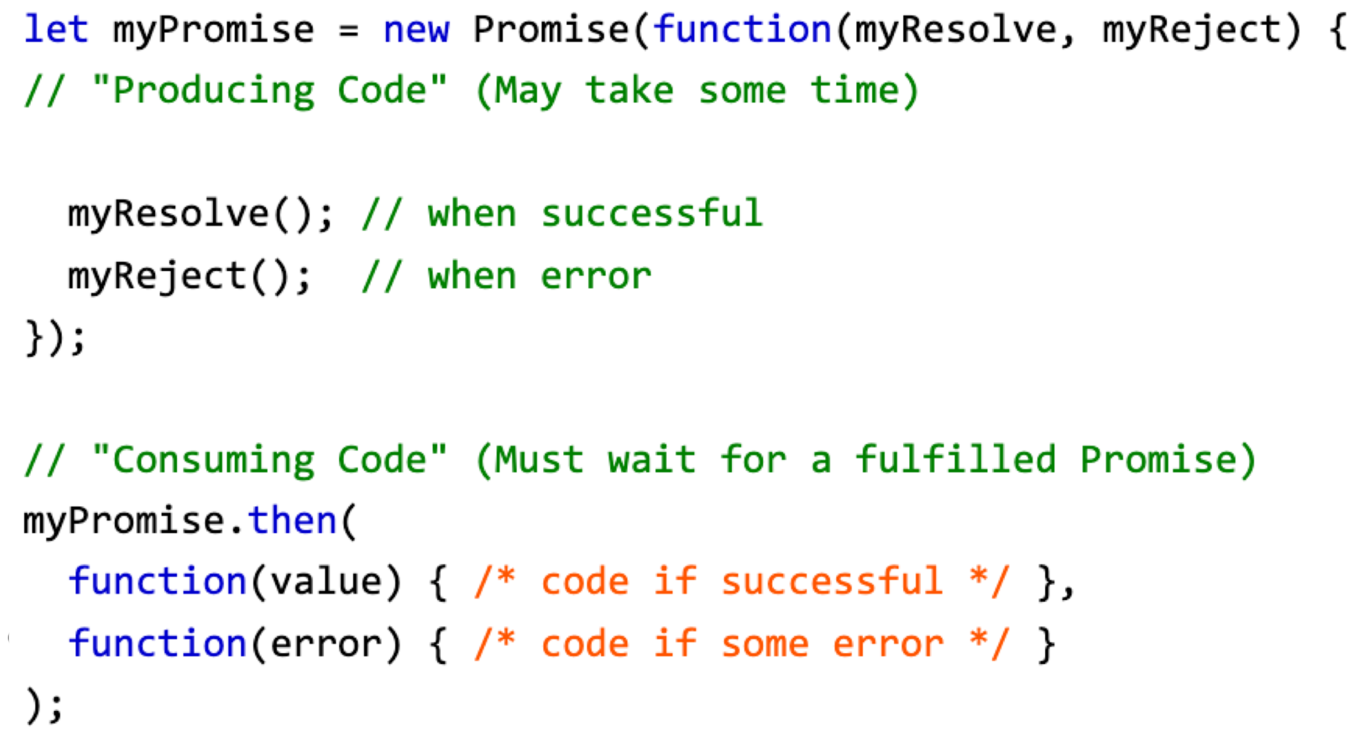
- To add a new property to a constructor, we must add it to the constructor function
- https://www.w3schools.com/js/tryit.asp?filename=tryjs_object_prototype4

Promises

- "I Promise a Result!»
- A Promise is a JavaScript object that links producing code and consuming code
- A Promise is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason.
- This lets asynchronous methods return values like synchronous methods:

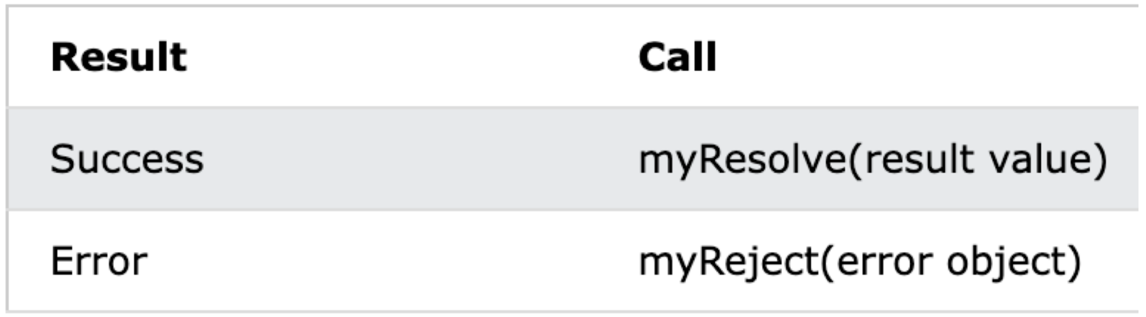
    - instead of immediately returning the final value, the asynchronous method returns a promise to supply the value at some point in the future.

Promise syntax



```
let myPromise = new Promise(function(myResolve, myReject) {
// "Producing Code" (May take some time)

  myResolve(); // when successful
  myReject();  // when error
});

// "Consuming Code" (Must wait for a fulfilled Promise)
myPromise.then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

should



Result	Call
Success	myResolve(result value)
Error	myReject(error object)

Promise states

- A Promise is in one of these states:
  - *pending*: initial state, neither fulfilled nor rejected.
  - *fulfilled*: meaning that the operation was completed successfully.
  - *rejected*: meaning that the operation failed.
- We cannot access the Promise properties **state** and **result**.
- We must use a Promise method to handle promises.

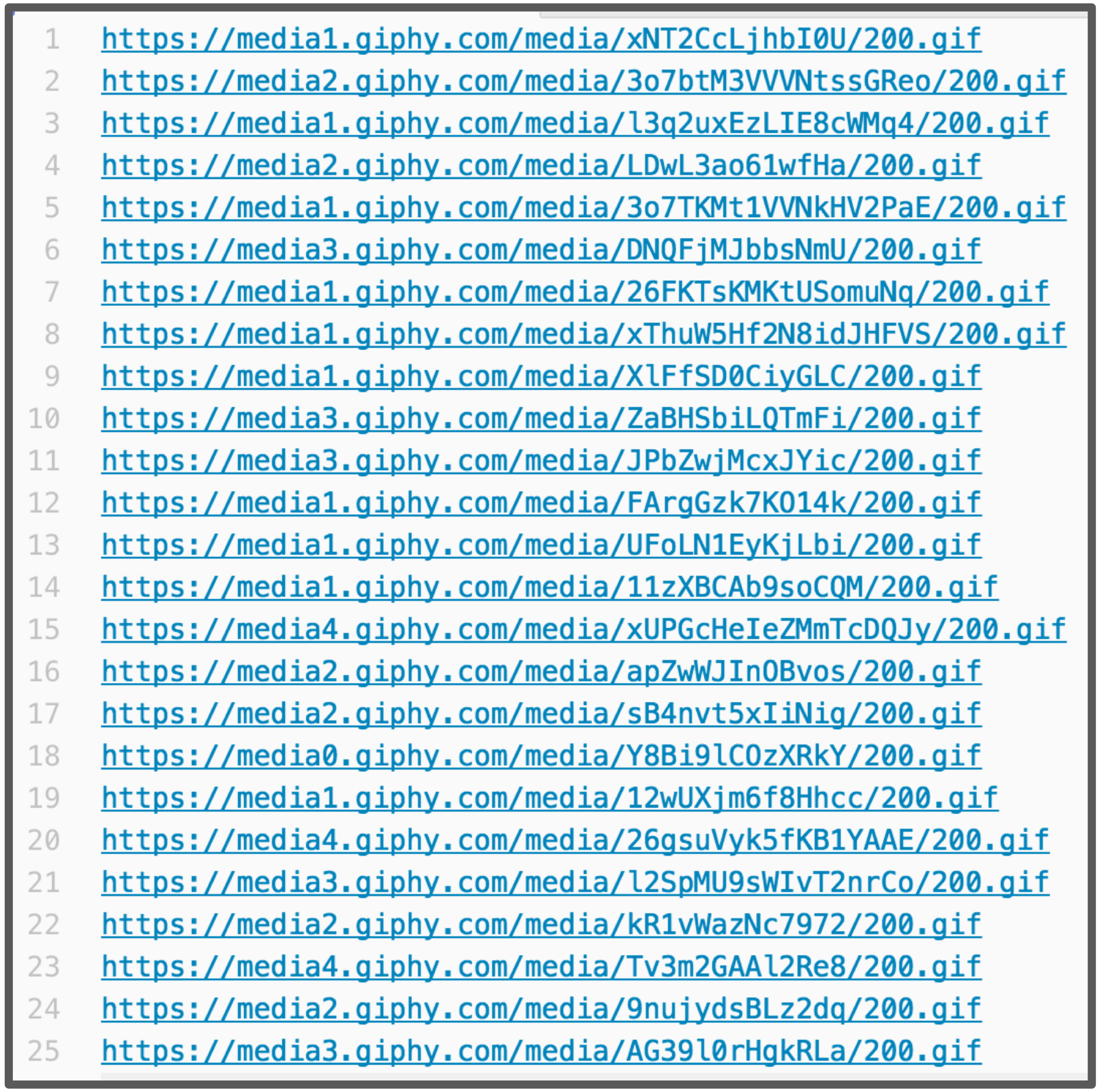
How to use promises

```
<p id="demo"></p>

<script>
function myDisplayer(some) {
  document.getElementById("demo").innerHTML =
some;
}

let myPromise = new Promise(function(myResolve,
myReject) {
  let x = 0;

// some code (try to change x to 5)

  if (x == 1) {
    myResolve("OK");
  } else {
    myReject("Error");
  }
});

myPromise.then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
</script>

</body>
```

Loading data from files

Loading data from a file

What if you had a list of URLs in a text file that you wanted to load as images in your web page?



```
https://media1.giphy.com/media/xNT2CcLjhbI0U/200.gif
https://media2.giphy.com/media/3o7btM3VVVNtssGReo/200.gif
https://media1.giphy.com/media/l3q2uxEzLIE8cWMq4/200.gif
https://media2.giphy.com/media/LDwL3ao61wfHa/200.gif
https://media1.giphy.com/media/3o7TKMt1VVNkHV2PaE/200.gif
https://media3.giphy.com/media/DNQFjMJbbsNmU/200.gif
https://media1.giphy.com/media/26FKTsKMKtUSomuNq/200.gif
https://media1.giphy.com/media/xThuW5Hf2N8idJHFVS/200.gif
https://media1.giphy.com/media/XlFfSD0CiyGLC/200.gif
https://media3.giphy.com/media/ZaBHSbiLQTmFi/200.gif
https://media3.giphy.com/media/JPbZwjMcxJYic/200.gif
https://media1.giphy.com/media/FArgGzk7KO14k/200.gif
https://media1.giphy.com/media/UFoLN1EyKjLbi/200.gif
https://media1.giphy.com/media/11zXBCAb9soCQM/200.gif
https://media4.giphy.com/media/xUPGcHeIeZMmTcDQJy/200.gif
https://media2.giphy.com/media/apZwWJInOBvos/200.gif
https://media2.giphy.com/media/sB4nvt5xIiNig/200.gif
https://media0.giphy.com/media/Y8Bi9lCOzXRkY/200.gif
https://media1.giphy.com/media/12wUXjm6f8Hhcc/200.gif
https://media4.giphy.com/media/26gsuVyk5fKB1YAAE/200.gif
https://media3.giphy.com/media/l2SpMU9sWIvT2nrCo/200.gif
https://media2.giphy.com/media/kR1vWazNc7972/200.gif
https://media4.giphy.com/media/Tv3m2GAAl2Re8/200.gif
https://media2.giphy.com/media/9nujydsBLz2dq/200.gif
https://media3.giphy.com/media/AG39l0rHgkRLa/200.gif
```

Fetch API

Fetch API

`fetch()`: Function to load resources in JavaScript

```
fetch(pathToResource)
    .then(onResponse)
    .then(onResourceReady);
```

onResponse:

- Return `response.text()` from this function to get the resource as a string in ***onResourceReady***

onResourceReady:

- Gets the resource as a parameter when it's ready

Fetch API

```
function onTextReady(text) {
  // do something with text
}

function onResponse(response) {
  return response.text();
}

fetch('images.txt')
    .then(onResponse)
    .then(onTextReady);
```

Completed example

```
function onTextReady(text) {
  const urls = text.split('\n');
  for (const url of urls) {
    const image = document.createElement('img');
    image.src = url;
    document.body.append(image);
  }
}

function onResponse(response) {
  return response.text();
}

fetch('images.txt')
    .then(onResponse)
    .then(onTextReady);
```

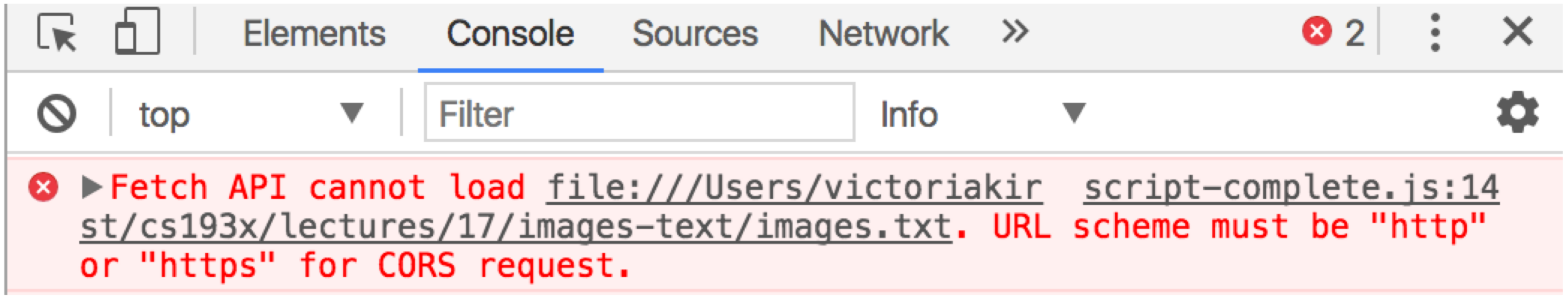
Completed example

```
function onTextReady(text) {
  const urls = text.split('\n');
  for (const url of urls) {
    const image = new Image();
    image.src = url;
    document.body.append(image);
  }
}

function onResponse(response) {
  return response.text();
}

fetch('images.txt')
    .then(onResponse)
    .then(onTextReady);
```

Images-text project

fetch() limitations

- You cannot fetch a resource that is hosted on file://
  - You must serve your resource over HTTP / HTTPS



```
Fetch API cannot load file:///Users/victoriakir  script-complete.js:14
st/cs193x/lectures/17/images-text/images.txt. URL scheme must be "http"
or "https" for CORS request.
```

Serve over HTTP

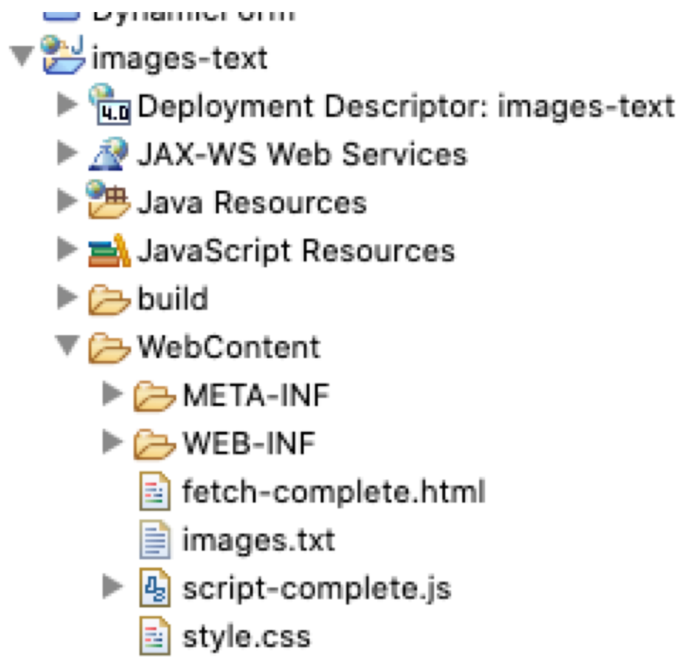
We can run a program to serve our local files over HTTP:

On Eclipse:

Create a dynamic project

Put on the webcontent folder the files

Run on a server fetch-complete.html


```
▼ images-text
  ▶ Deployment Descriptor: images-text
  ▶ JAX-WS Web Services
  ▶ Java Resources
  ▶ JavaScript Resources
  ▶ build
  ▼ WebContent
    ▶ META-INF
    ▶ WEB-INF
      fetch-complete.html
      images.txt
    ▶ script-complete.js
      style.css
```

JSON

JavaScript Object Notation

JSON: Stands for **J**ava**S**cript **O**bject **N**otation

- Created by Douglas Crockford
- Defines a way of **serializing** JavaScript objects
    - **to serialize**: to turn an object into a string that can be deserialized
    - **to deserialize**: to turn a serialized string into an object

JSON.stringify()

We can use the `JSON.stringify()` function to seralize a JavaScript object:

```
const bear = {
  name: 'Ice Bear',
  hobbies: ['knitting', 'cooking', 'dancing']
};

const serializedBear =
JSON.stringify(bear);
console.log(serializedBear);
```

JSON.parse()

We can use the `JSON.parse()` function to deseralize a JavaScript object:

```
const bearString = '{"name":"Ice Bear","hobbies":["knitting","cooking","dancing"]}';

const bear = JSON.parse(bearString);
console.log(bear);
```

Fetch API and JSON

The Fetch API also has built-in support for JSON:

```
function onJsonReady(json) {
  console.log(json);
}

function onResponse(response) {
  return response.json();
}

fetch('images.json')
    .then(onResponse)
    .then(onJsonReady);
```

Return `response.json()` instead of `response.text()` and Fetch will essentially call `JSON.parse()` on the response string.

Why JSON?

Let's say we had a file that contained a list of albums.

Each album has:

- Title
- Year
- URL to album image

We want to display each album in chronological order.

Text file?

We could create a text file formatted consistently in some format that we make up ourselves, e.g.:

```
The Emancipation Of Mimi
2005
https://i.scdn.co/image/dca82bd9c1ccae90b09972027a408068f7a4d700

Daydream
1995
https://i.scdn.co/image/0638f0ddf70003cb94b43aa5e4004d85da94f99c

E=MC²
2008
https://i.scdn.co/image/bca35d49f6033324d2518656531c9a89135c0ea3

Mariah Carey
1990
https://i.scdn.co/image/82f13700dfa78fa877a8cdecd725ad552c9a9653
```

Text file processing

We would have to write all this custom file processing code:

- Must convert numbers from strings
- If you ever add another attribute to the album, we'd have to change our array indices

```
function onTextReady(text) {
  const lines = text.split('\n\n');
  const albums = [];
  for (let i = 0; i < lines.length; i++) {
    const infoText = lines[i];
    const infoStrings = infoText.split('\n');
    const name = infoStrings[0];
    const year = infoStrings[1];
    const url = infoStrings[2];
    albums.push({
      name: name,
      year: parseInt(year),
      url: url
    });
  }
  ...
}
```

Album project

JSON file

It'd be much more convenient to store the file in JSON format:

```
{
  "albums": [
    {
      "name": "The Emancipation Of Mimi",
      "year": 2005,
      "url":
"https://i.scdn.co/image/dca82bd9c1ccae90b09972027a408068f7a4d700
"
    },
    {
      "name": "Daydream",
      "year": 1995,
      "url":
"https://i.scdn.co/image/0638f0ddf70003cb94b43aa5e4004d85da94f99c
"
    },
    {
```

JSON processing

Since we're using JSON, we don't have to manually convert the response strings to a JavaScript object:

- JavaScript has built-in support to convert a JSON string into a JavaScript object.

```
function onJsonReady(json) {
  const albums = json.albums;
  ...
}
```

Album project