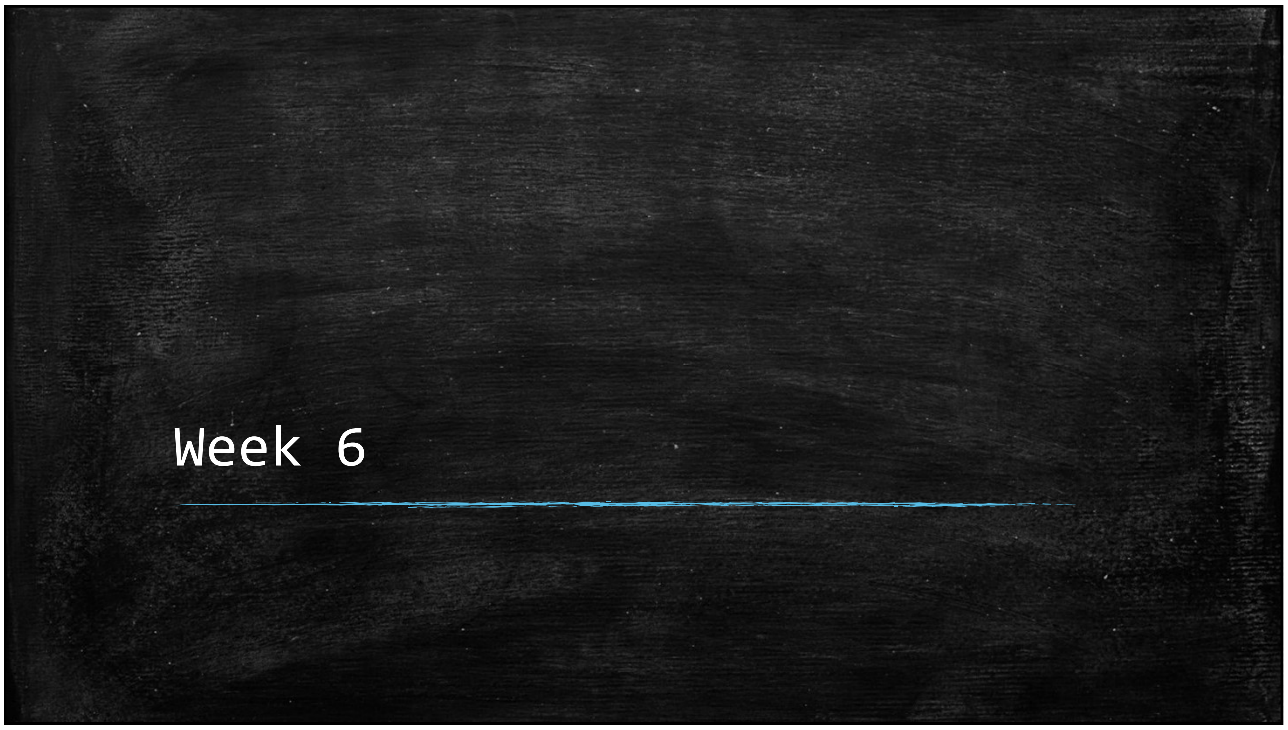
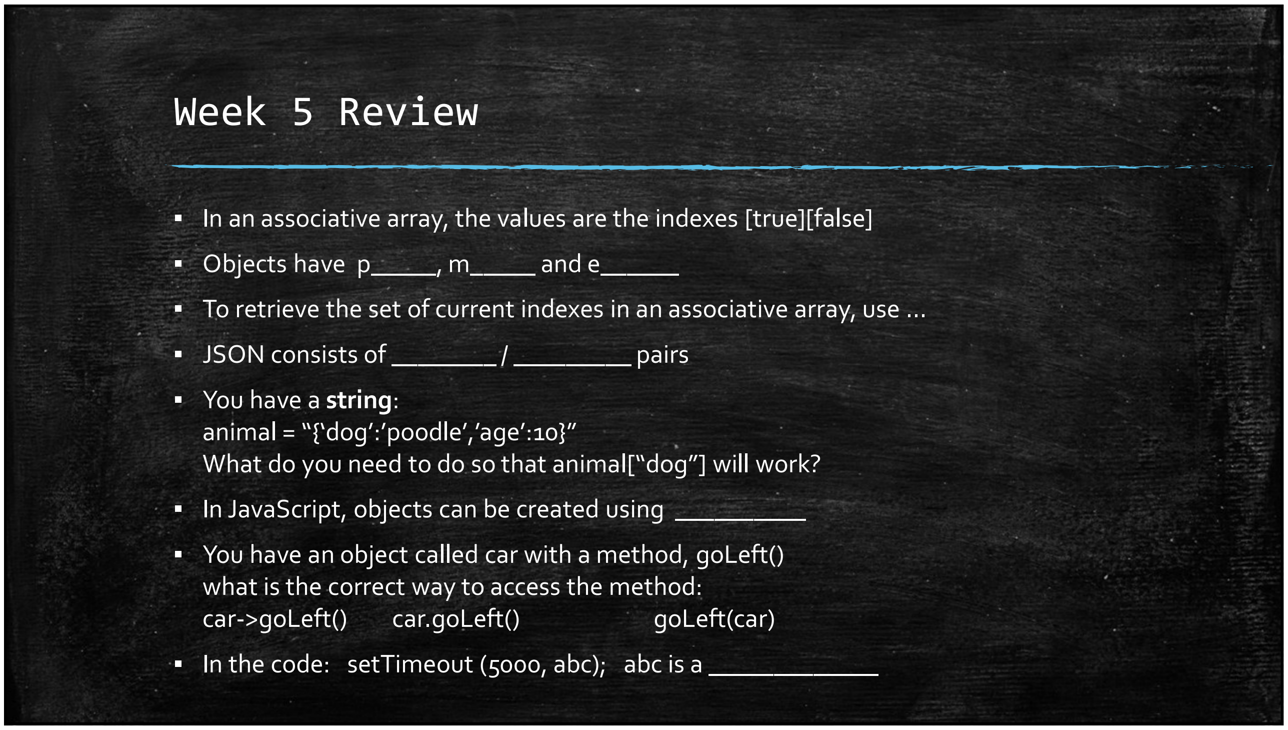


Week 6

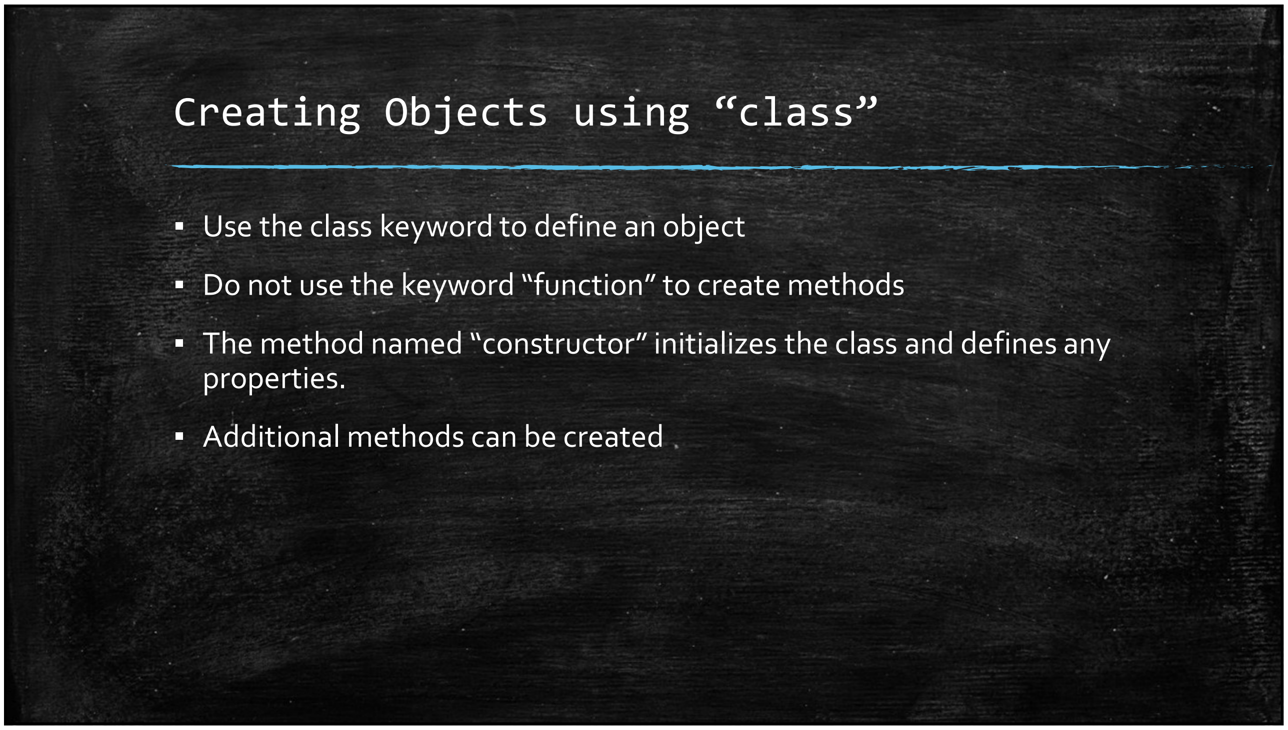
156



Week 5 Review

- In an associative array, the values are the indexes [true][false]
- Objects have p_____, m_____ and e______
- To retrieve the set of current indexes in an associative array, use …
- JSON consists of ________/ _________ pairs
- You have a **string**:
  animal = "{'dog':'poodle','age':10}"
  What do you need to do so that animal["dog"] will work?
- In JavaScript, objects can be created using __________
- You have an object called car with a method, goLeft()
  what is the correct way to access the method:
  car->goLeft()   car.goLeft()   goLeft(car)
- In the code: setTimeout (5000, abc); abc is a _____________

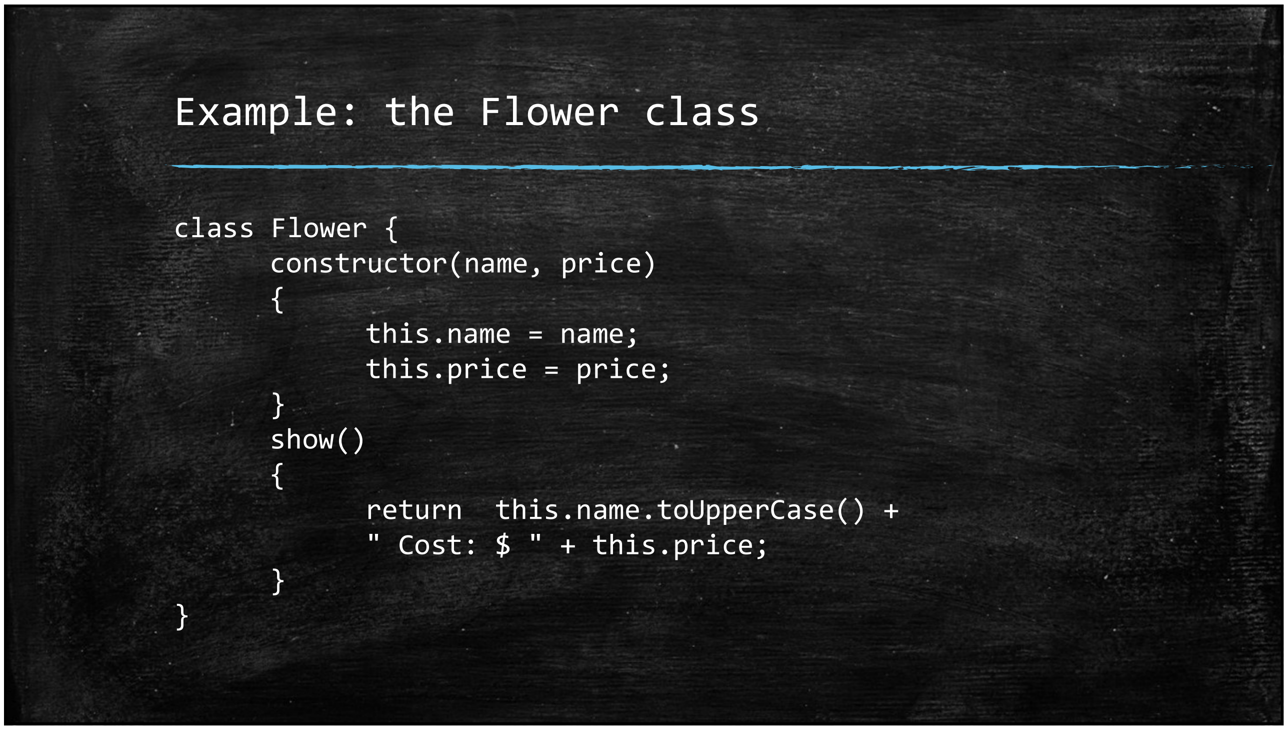
157



Creating Objects using “class”

- Use the class keyword to define an object
- Do not use the keyword “function” to create methods
- The method named “constructor” initializes the class and defines any properties.
- Additional methods can be created

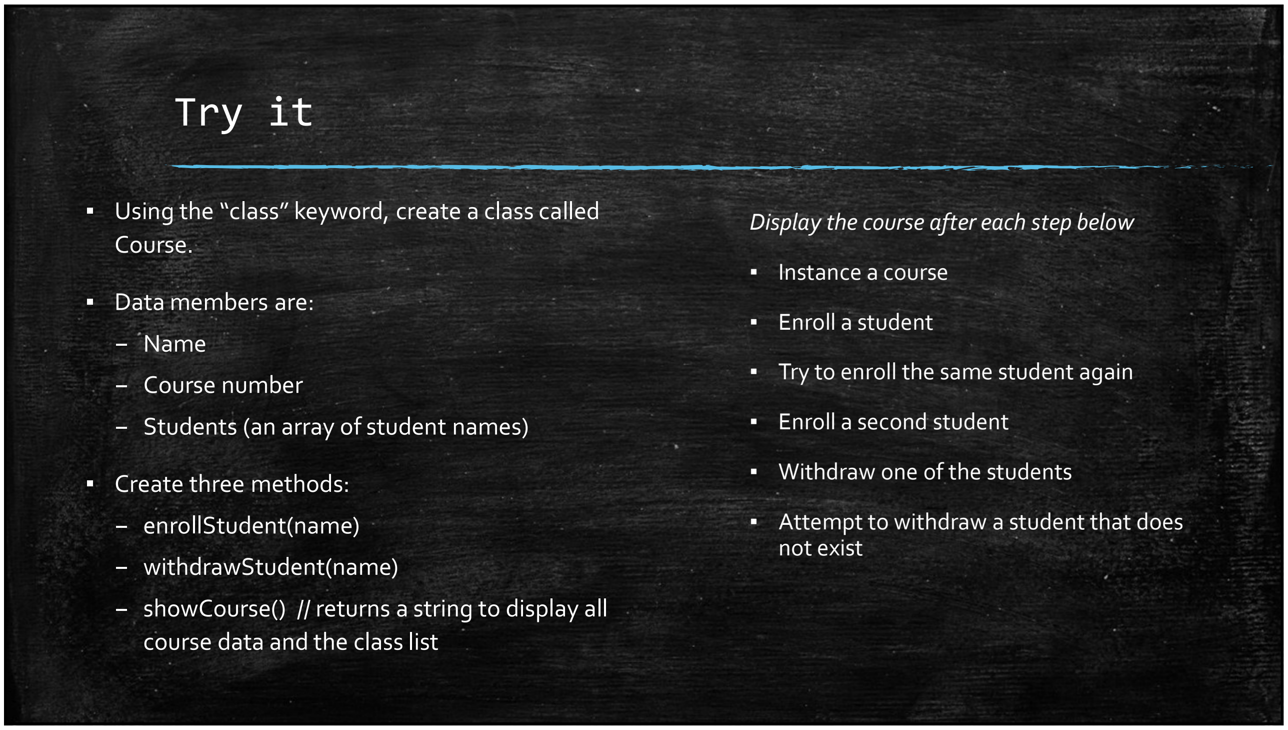
158



Example: the Flower class

```
class Flower {
      constructor(name, price)
      {
            this.name = name;
            this.price = price;
      }
      show()
      {
            return  this.name.toUpperCase() +
            " Cost: $ " + this.price;
      }
}
```

159



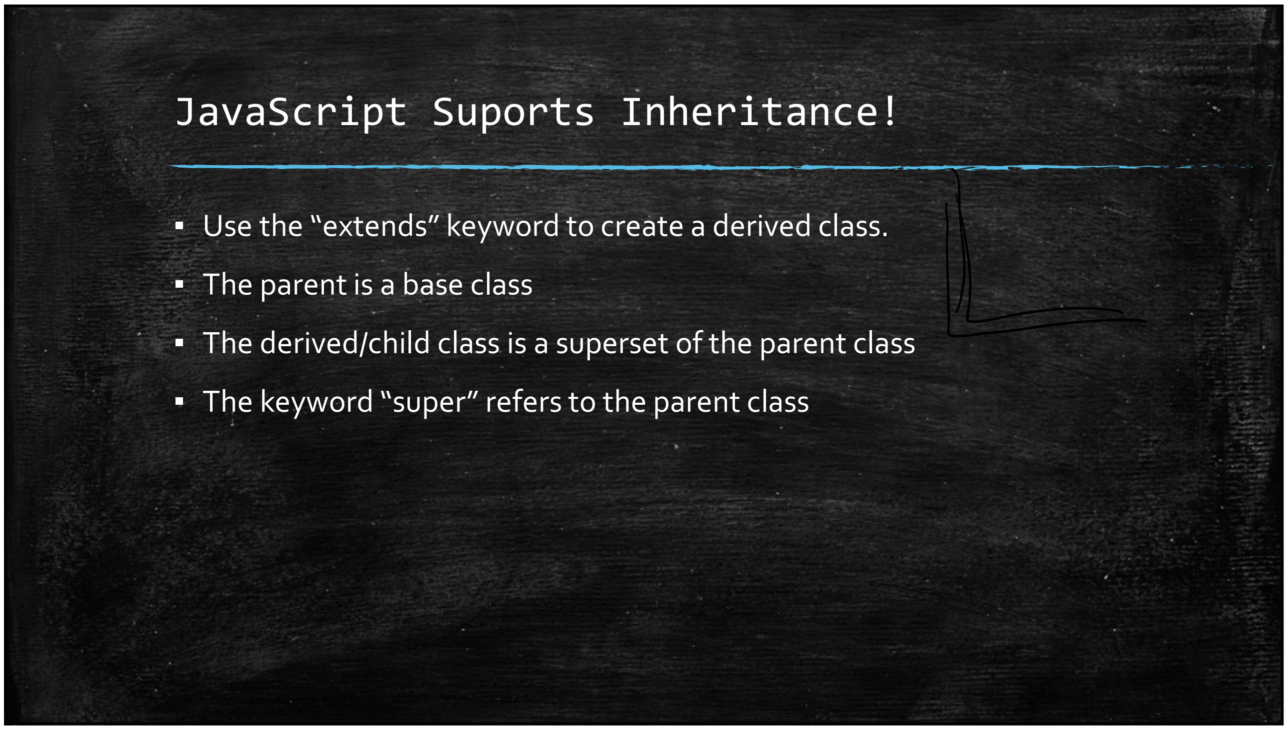
Try it

- Using the “class” keyword, create a class called Course.
- Data members are:
  - Name
  - Course number
  - Students (an array of student names)
- Create three methods:
  - enrollStudent(name)
  - withdrawStudent(name)
  - showCourse() // returns a string to display all course data and the class list

Display the course after each step below

- Instance a course
- Enroll a student
- Try to enroll the same student again
- Enroll a second student
- Withdraw one of the students
- Attempt to withdraw a student that does not exist

160



JavaScript Suports Inheritance!

- Use the “extends” keyword to create a derived class.
- The parent is a base class
- The derived/child class is a superset of the parent class
- The keyword “super” refers to the parent class

161

Extending the Flower class

```
class FlowerShopInventory extends Flower
{
	constructor(name, price, quantity=0) {
		super(name, price);
		this.quantity = quantity;
	}
	addQuantity(amt) {
		this.quantity += amt;
	}
	cost() {
		return this.quantity * this.cost;
	}
	show() {
		return super.show() + " Quantity: " + this.quantity;
	}
}   // end class FlowerShopInventory
```

162

Try It

- Create a class called LimitedCourse that extends Course. LimitedCourse restricts the number of students that can enroll.
- What updates do you need to Course to make that work?
- Test the class

163

Private Data

- You can restrict access to class elements from outside of the class
- Add a "#" at the beginning of the class name to indicate it is private
- Example: quantity is private in the FlowerShopInventory class

```
class FlowerShopInventory extends Flower
{
  #quantity;
  constructor(name, price, quantity=0)  {
       super(name, price);
       this.#quantity = quantity;
  }
```

164

Try It

- Add private data to LimitedCourse.

165

Objects and Array Notation

- An object can be accessed using object or array notation.

```
daisy = new FlowerShopInventory("daisy", 15);
daisy["quantity"] = 4      // this works
daisy.quantity = 10;       // this also works
daisy.addQuantity(3);
console.log (daisy.show())
```

166

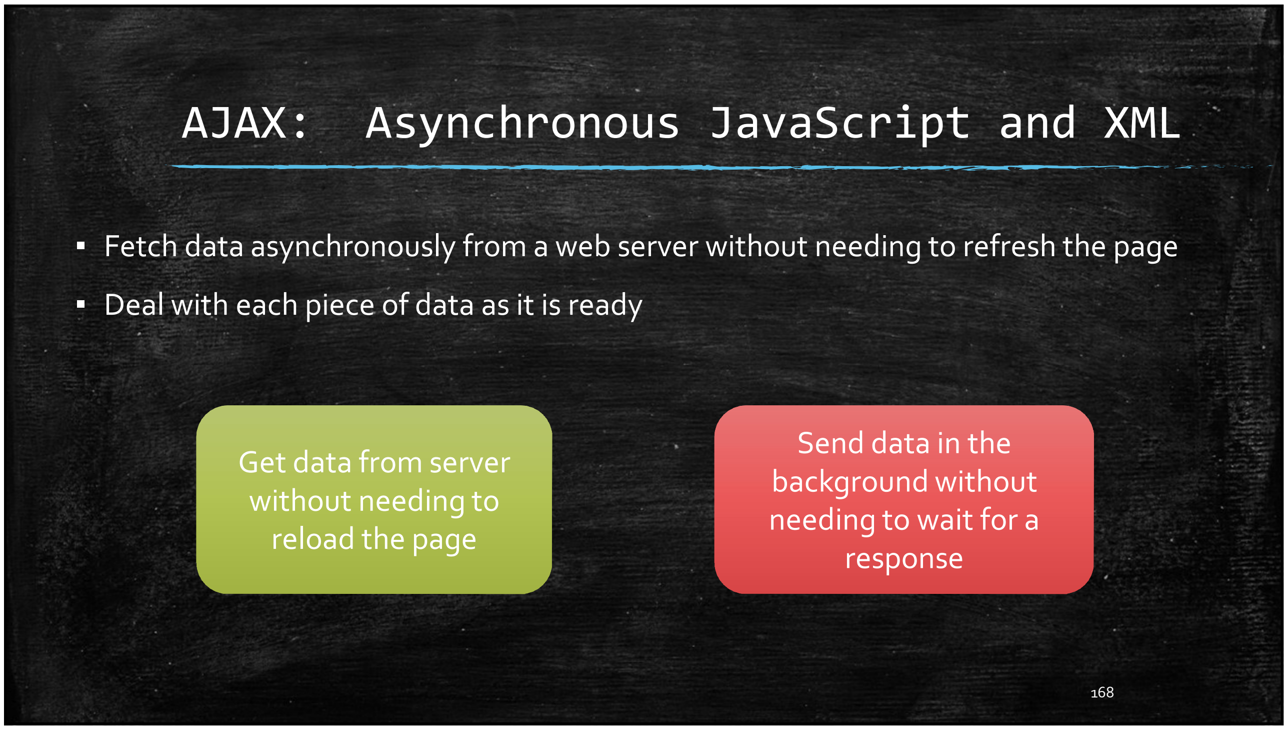
Synchronous vs Asynchronous Operations

Synchronous / Serial:
wait until something completes
before doing the next thing

Asynchronous:
start several things at once and
tend to each as it is ready

167

167



AJAX: Asynchronous JavaScript and XML

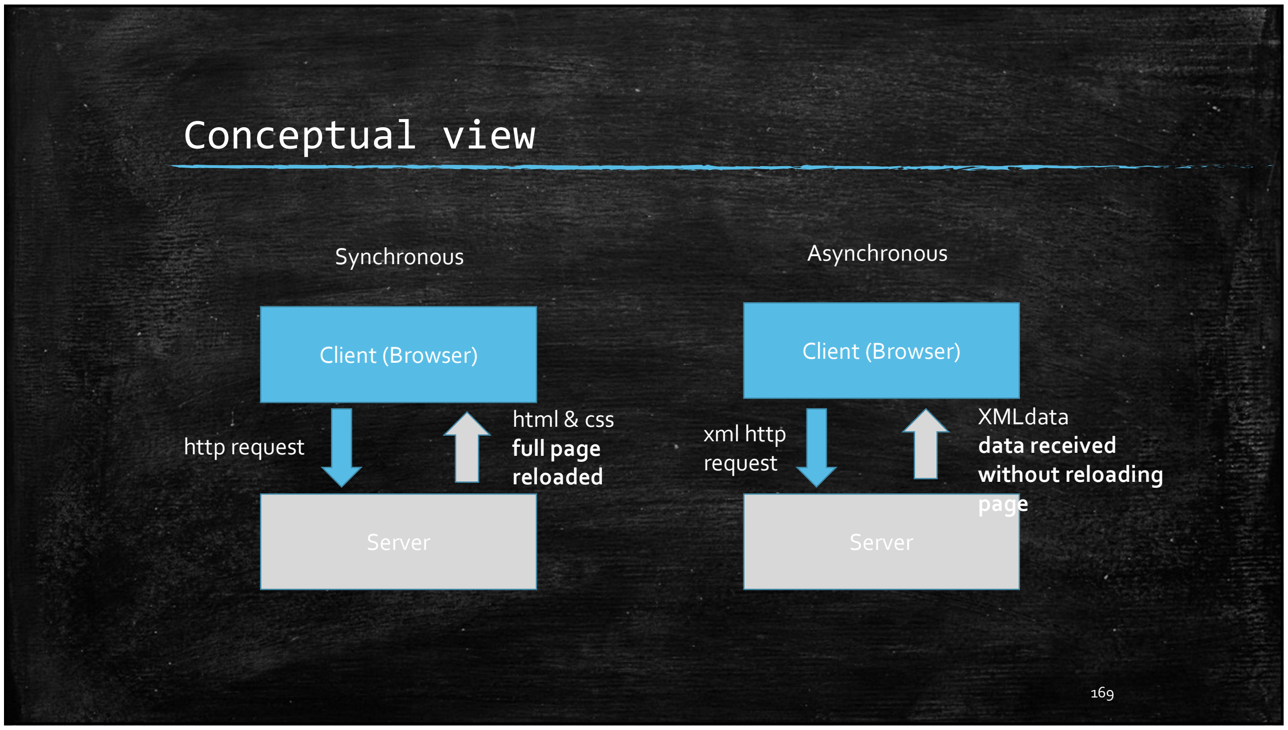
- Fetch data asynchronously from a web server without needing to refresh the page
- Deal with each piece of data as it is ready

Get data from server without needing to reload the page

Send data in the background without needing to wait for a response

168

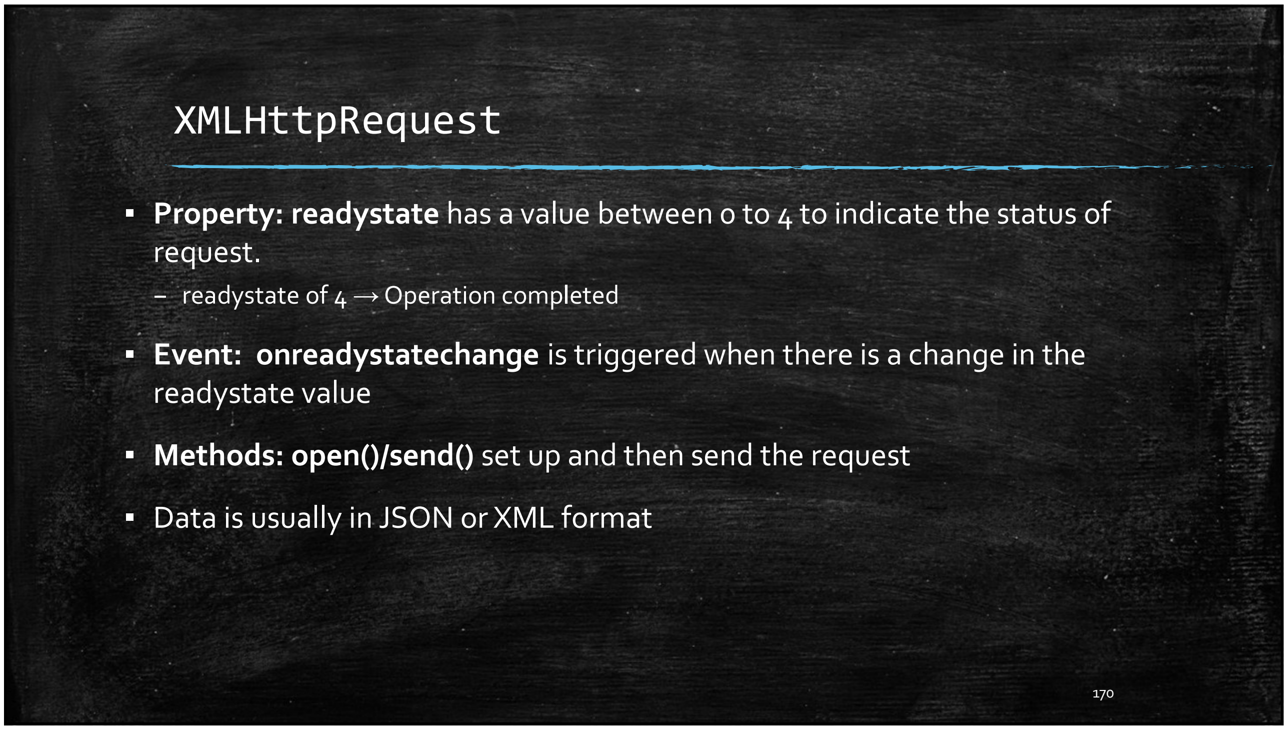
168



Conceptual view

169

169

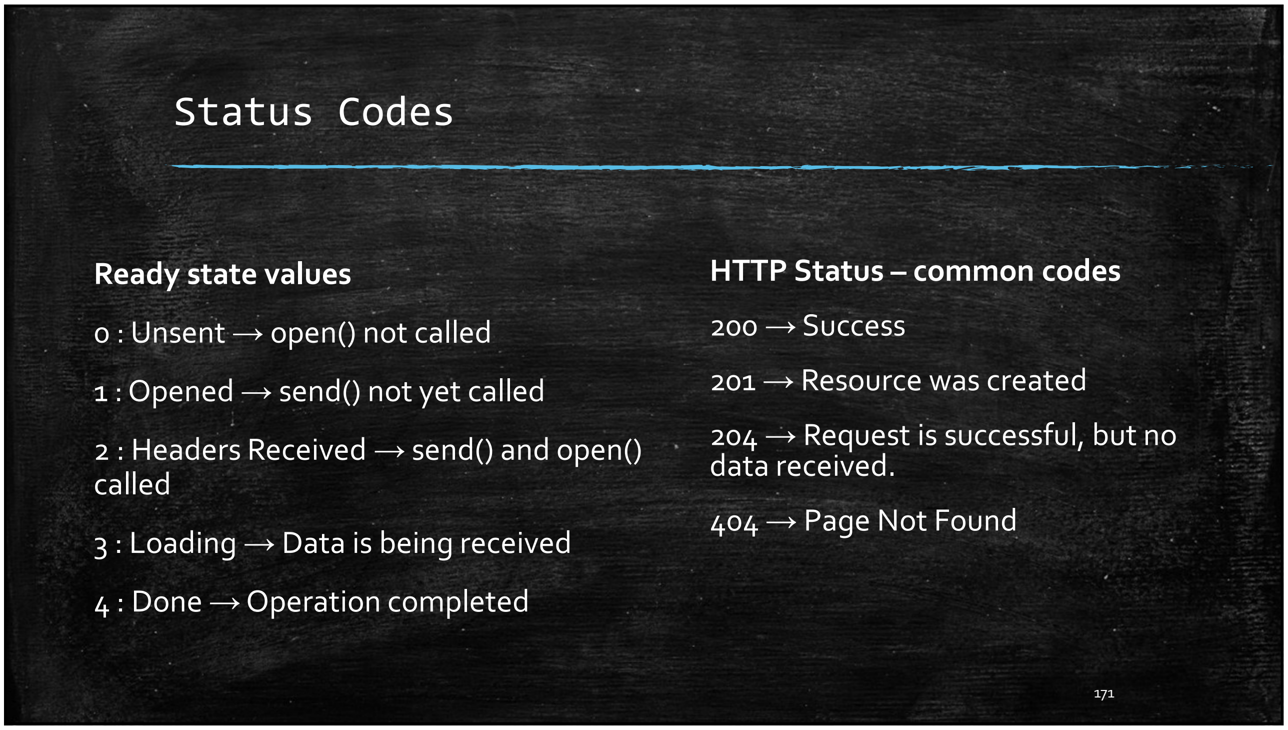


XMLHttpRequest

- **Property: readystate** has a value between 0 to 4 to indicate the status of request.
  - readystate of 4 ⟶ Operation completed
- **Event: onreadystatechange** is triggered when there is a change in the readystate value
- **Methods: open()/send()** set up and then send the request
- Data is usually in JSON or XML format

170

170



Status Codes

Ready state values

0 : Unsent ⟶ open() not called

1 : Opened ⟶ send() not yet called

2 : Headers Received ⟶ send() and open() called

3 : Loading ⟶ Data is being received

4 : Done ⟶ Operation completed

HTTP Status – common codes

200 ⟶ Success

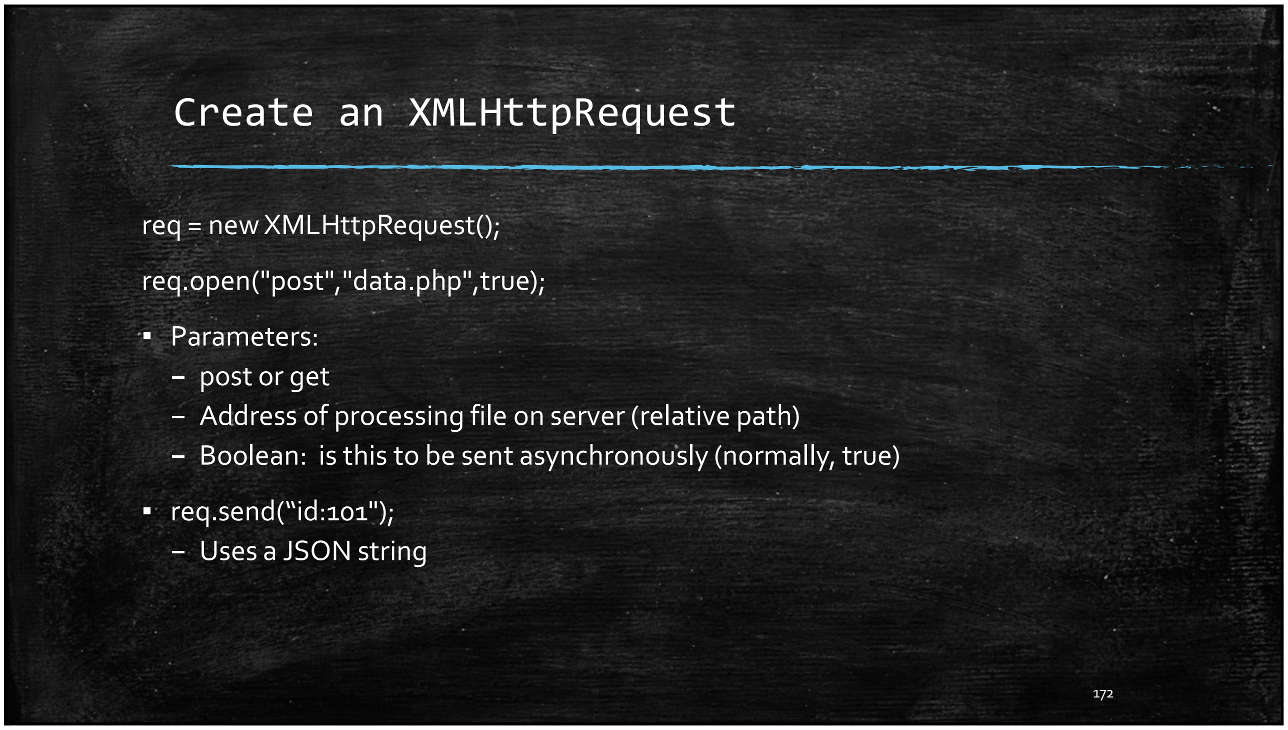
201 ⟶ Resource was created

204 ⟶ Request is successful, but no data received.

404 ⟶ Page Not Found

171

171



Create an XMLHttpRequest

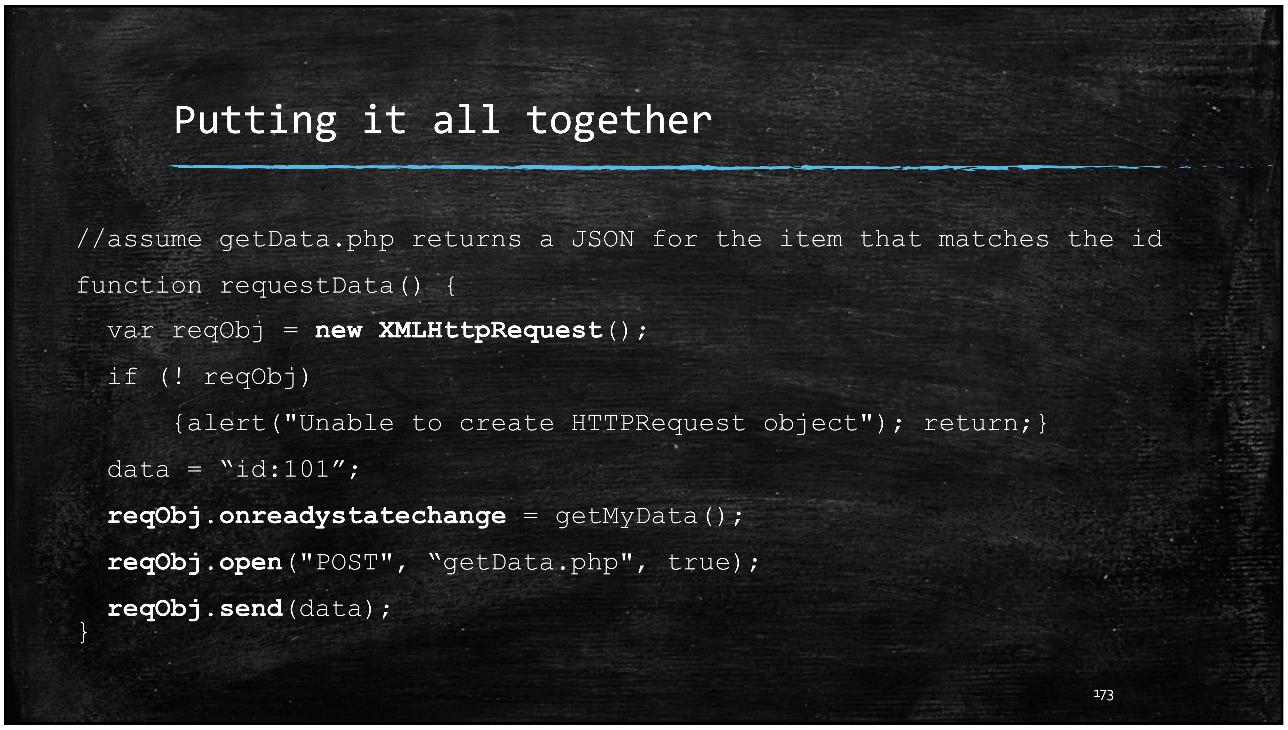
req = new XMLHttpRequest();

req.open("post","data.php",true);

- Parameters:
  - post or get
  - Address of processing file on server (relative path)
  - Boolean: is this to be sent asynchronously (normally, true)
- req.send(“id:101");
  - Uses a JSON string

172

172

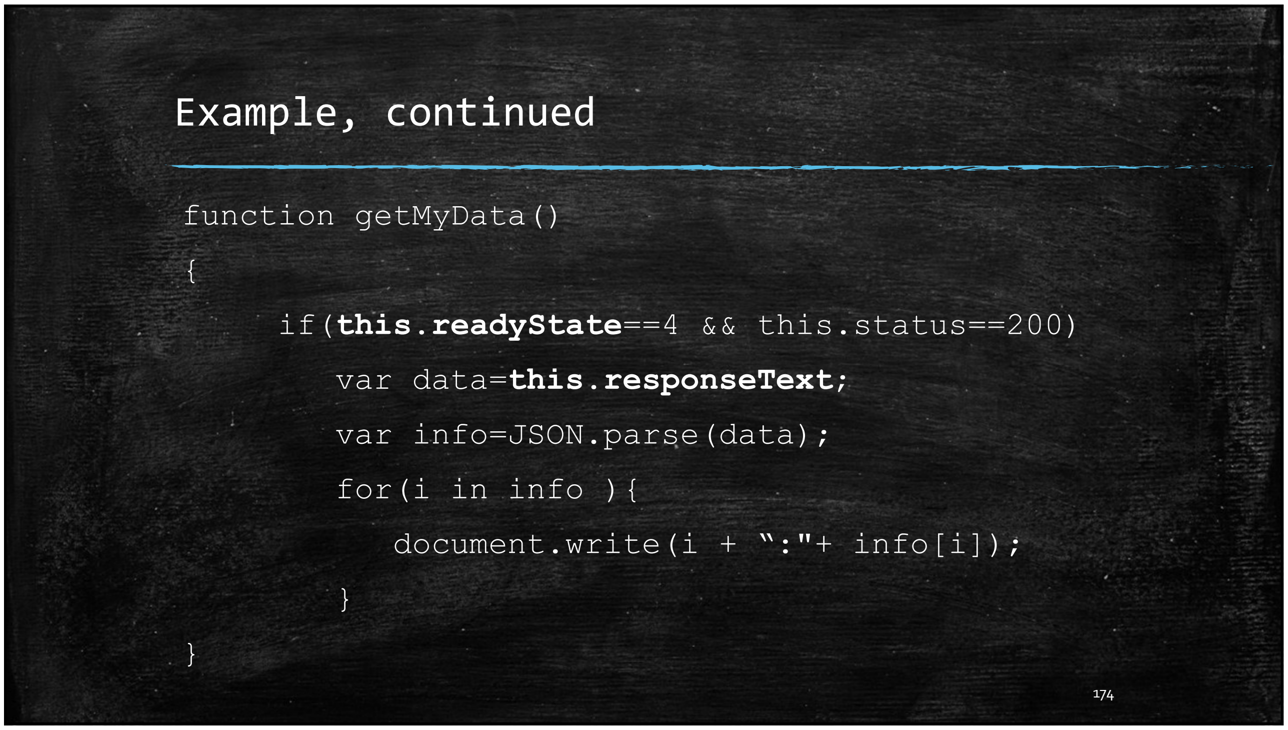


Putting it all together

```
//assume getData.php returns a JSON for the item that matches the id
function requestData() {
  var reqObj = new XMLHttpRequest();
  if (! reqObj)
      {alert("Unable to create HTTPRequest object"); return;}
  data = “id:101”;
  reqObj.onreadystatechange = getMyData();
  reqObj.open("POST", “getData.php", true);
  reqObj.send(data);
}
```

173

173

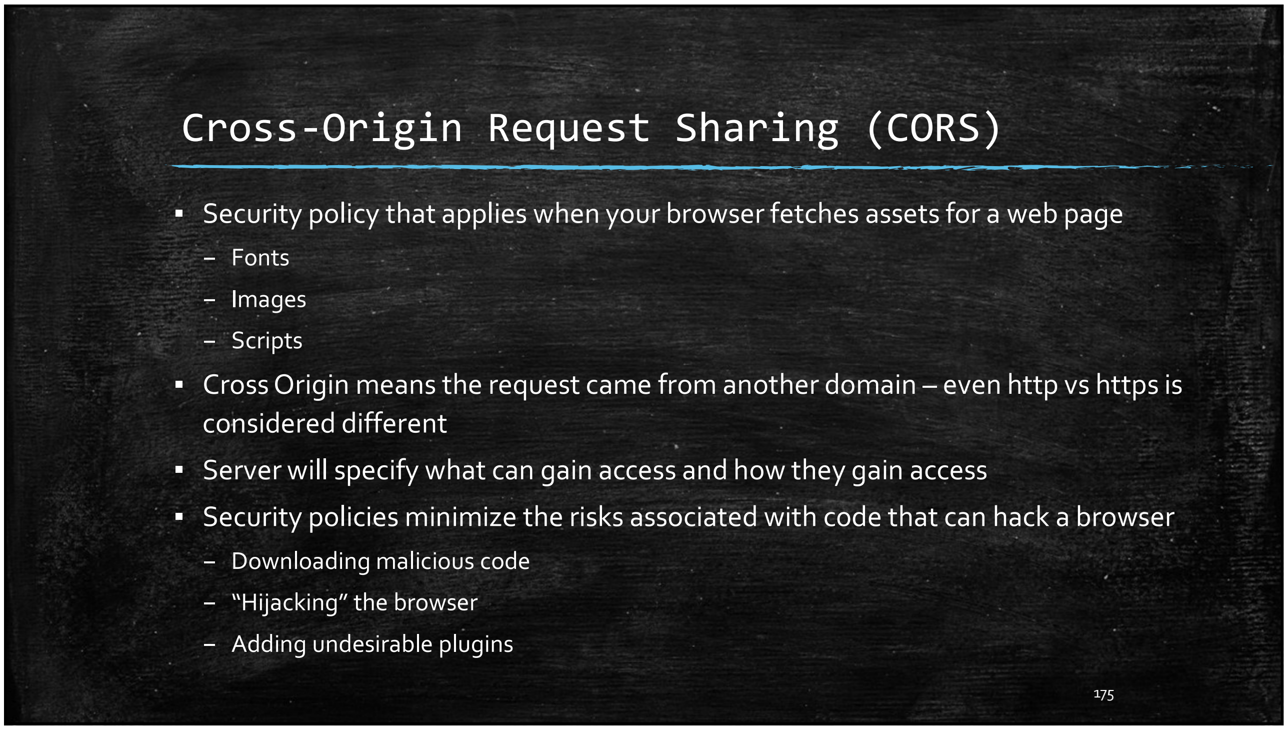


Example, continued

```
function getMyData()
{
      if(this.readyState==4 && this.status==200)
         var data=this.responseText;
         var info=JSON.parse(data);
         for(i in info ){
            document.write(i + ":"+ info[i]);
         }
}
```

174

174

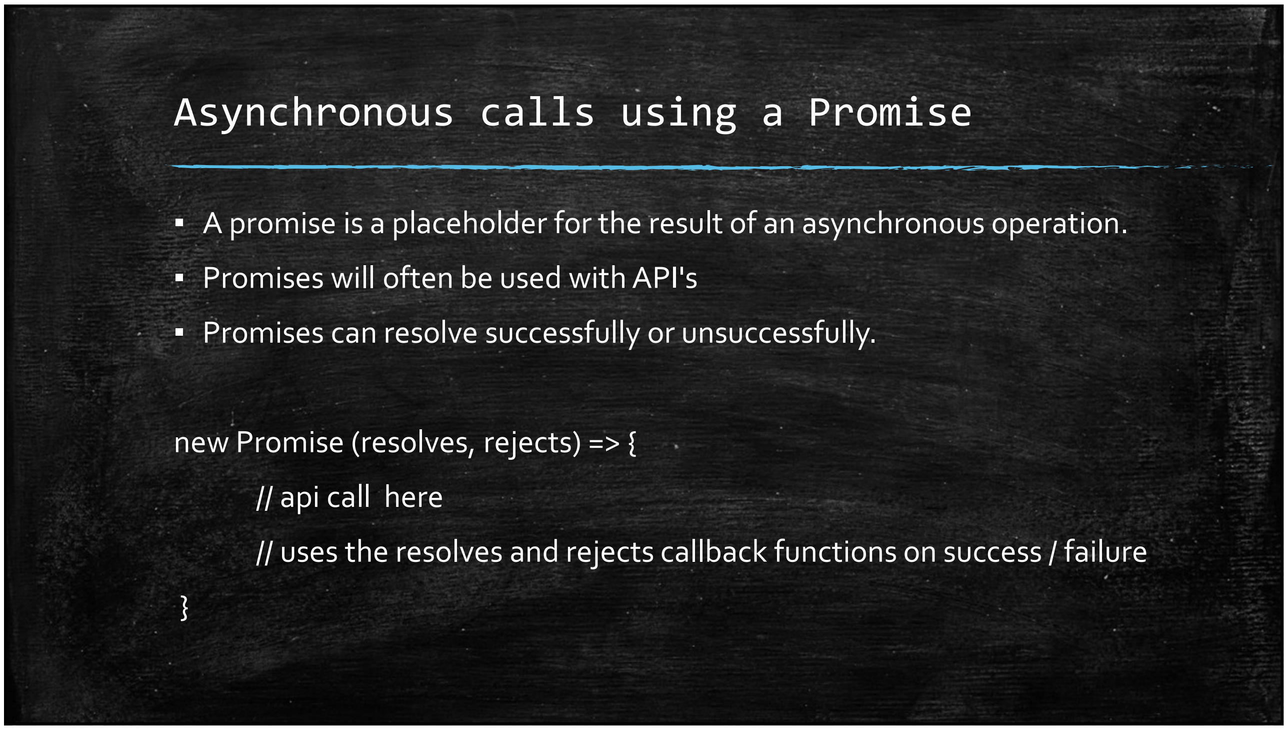


Cross-Origin Request Sharing (CORS)

- Security policy that applies when your browser fetches assets for a web page
  - Fonts
  - Images
  - Scripts
- Cross Origin means the request came from another domain – even http vs https is considered different
- Server will specify what can gain access and how they gain access
- Security policies minimize the risks associated with code that can hack a browser
  - Downloading malicious code
  - "Hijacking" the browser
  - Adding undesirable plugins

175

175

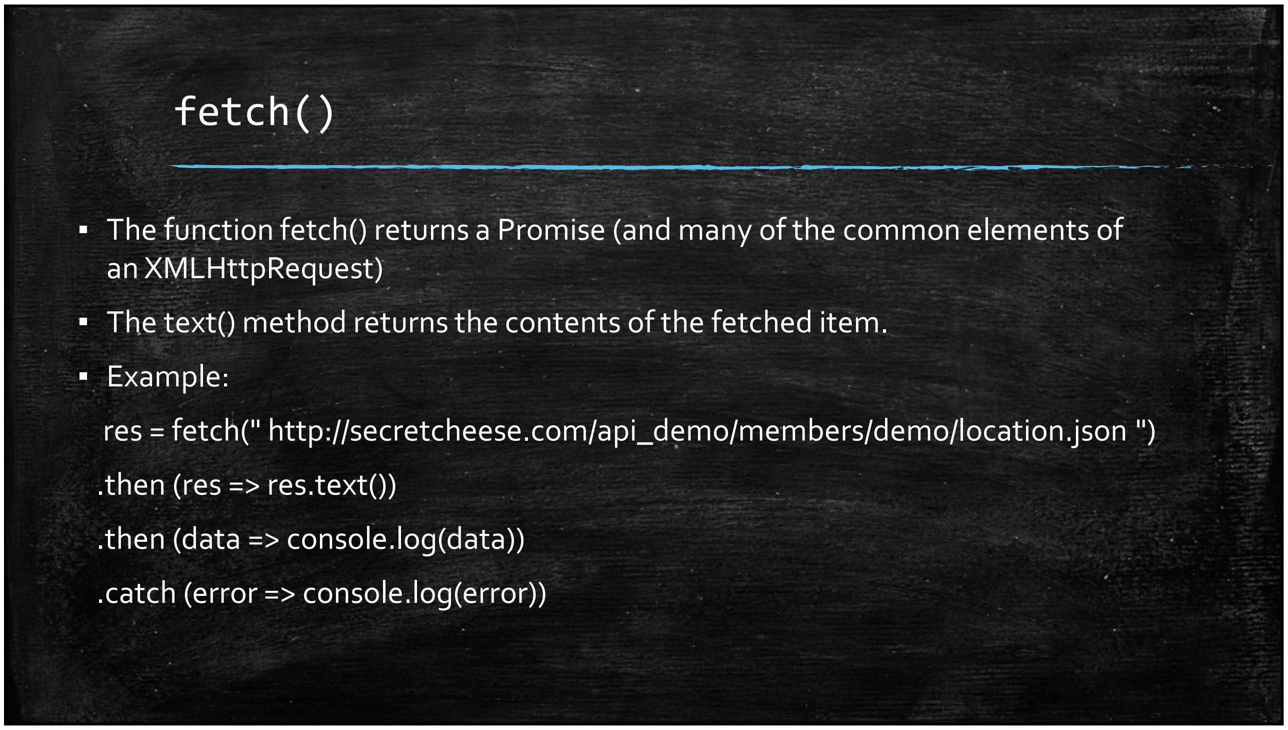


Asynchronous calls using a Promise

- A promise is a placeholder for the result of an asynchronous operation.
- Promises will often be used with API's
- Promises can resolve successfully or unsuccessfully.

```
new Promise (resolves, rejects) => {
        // api call  here
        // uses the resolves and rejects callback functions on success / failure
}
```

176



fetch()

- The function fetch() returns a Promise (and many of the common elements of an XMLHttpRequest)
- The text() method returns the contents of the fetched item.
- Example:

```
 res = fetch(" http://secretcheese.com/api_demo/members/demo/location.json ")
.then (res => res.text())
.then (data => console.log(data))
.catch (error => console.log(error))
```

177

Working with an API

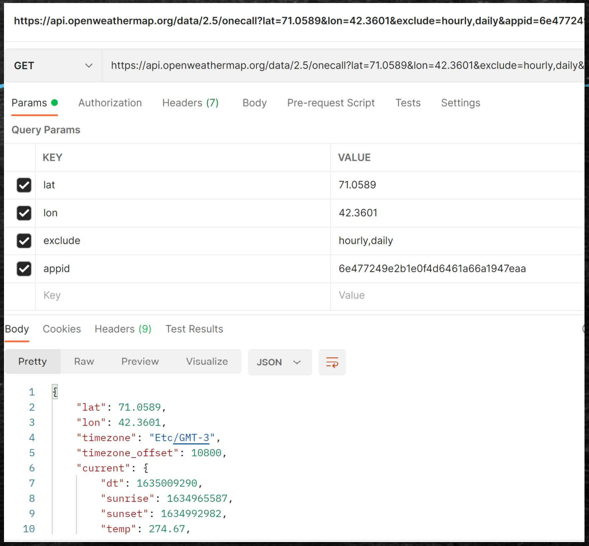
- An API or Application Programming Interface can gain access to specialized functionality that lives on a web server
- The API allows an organization to provide access to their data without compromising their data
- Often an API can be accessed with AJAX
- Example: zippopotam.us
- You may need an API key to access the data

178

178

postman.com

- Postman is a web application that helps to test an API call – with NO coding needed.
- A postman collection is a json object that details the correct parameters for an API call.



https://api.openweathermap.org/data/2.5/onecall?lat=71.0589&lon=42.3601&exclude=hourly,daily&appid=6e47724

GET https://api.openweathermap.org/data/2.5/onecall?lat=71.0589&lon=42.3601&exclude=hourly,daily&

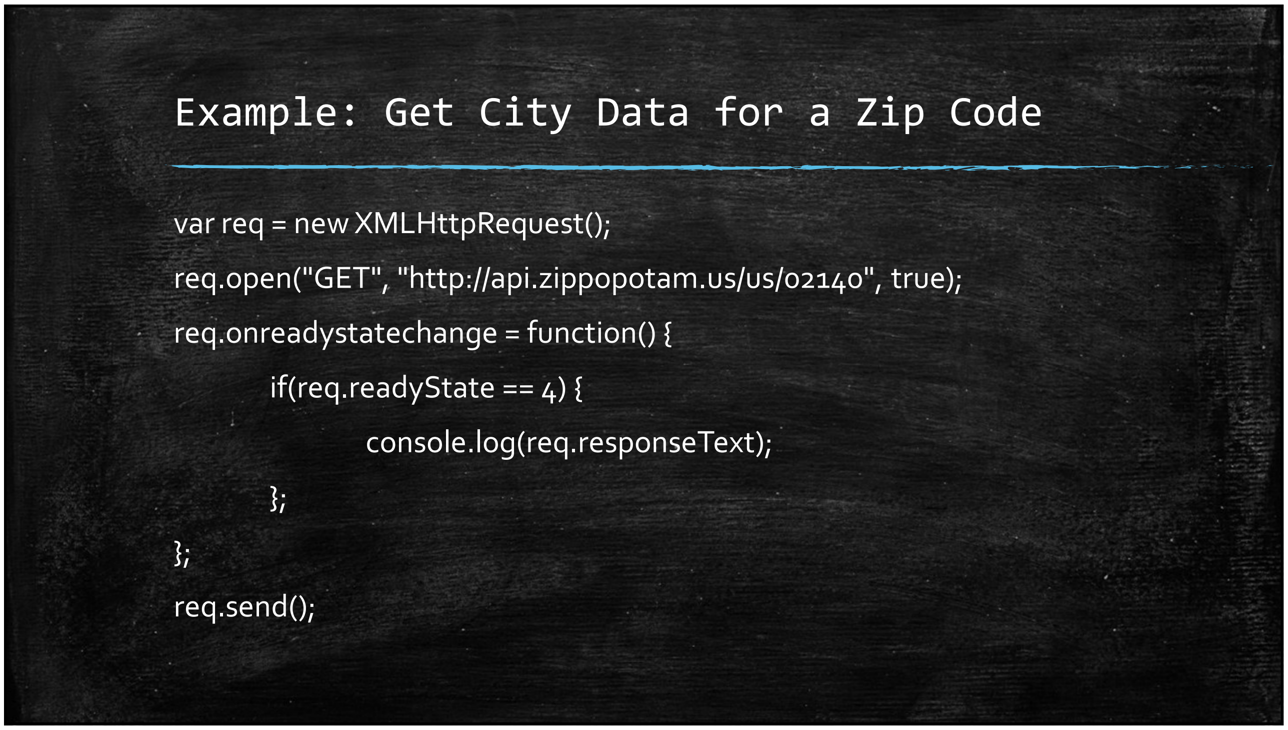
Query Params

	KEY	VALUE
✓	lat	71.0589
✓	lon	42.3601
✓	exclude	hourly,daily
✓	appid	6e477249e2b1e0f4d6461a66a1947eaa
	Key	Value

```
{
    "lat": 71.0589,
    "lon": 42.3601,
    "timezone": "Etc/GMT-3",
    "timezone_offset": 10800,
    "current": {
        "dt": 1635009290,
        "sunrise": 1634965587,
        "sunset": 1634992982,
        "temp": 274.67,
```

179

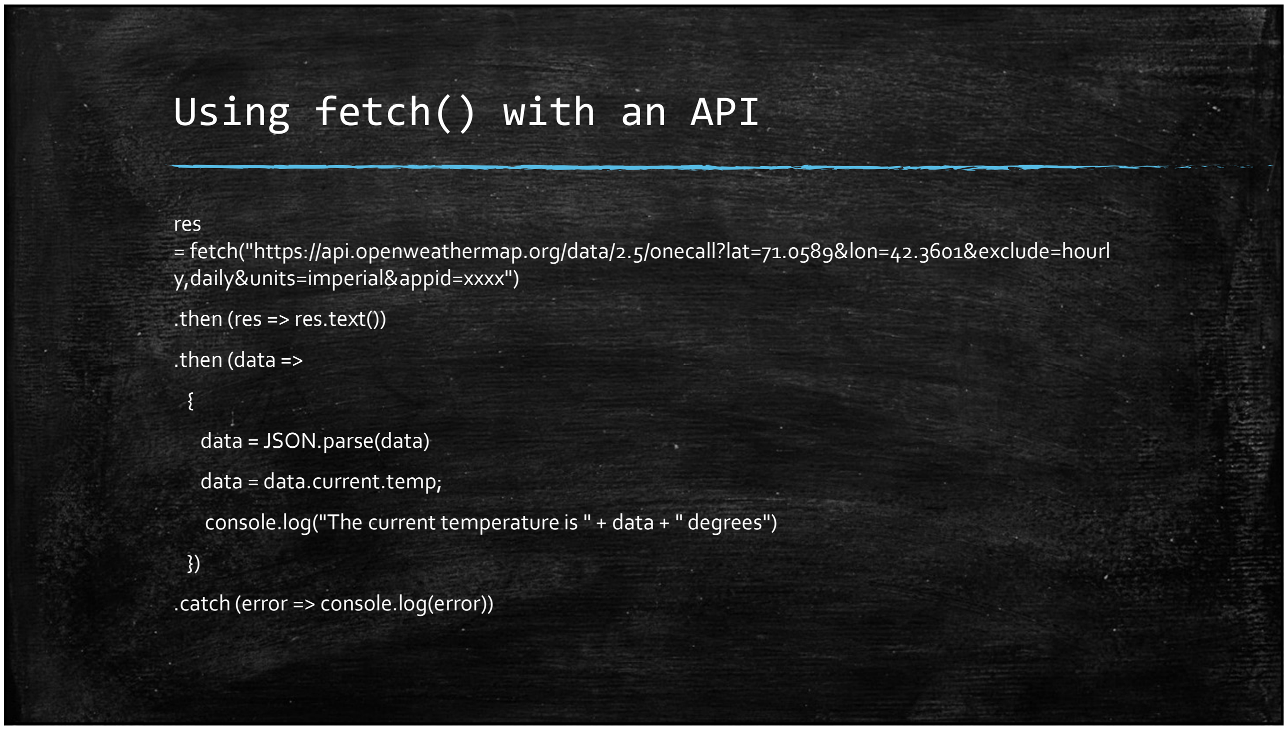
179



Example: Get City Data for a Zip Code

```
var req = new XMLHttpRequest();
req.open("GET", "http://api.zippopotam.us/us/02140", true);
req.onreadystatechange = function() {
        if(req.readyState == 4) {
                console.log(req.responseText);
        };
};
req.send();
```

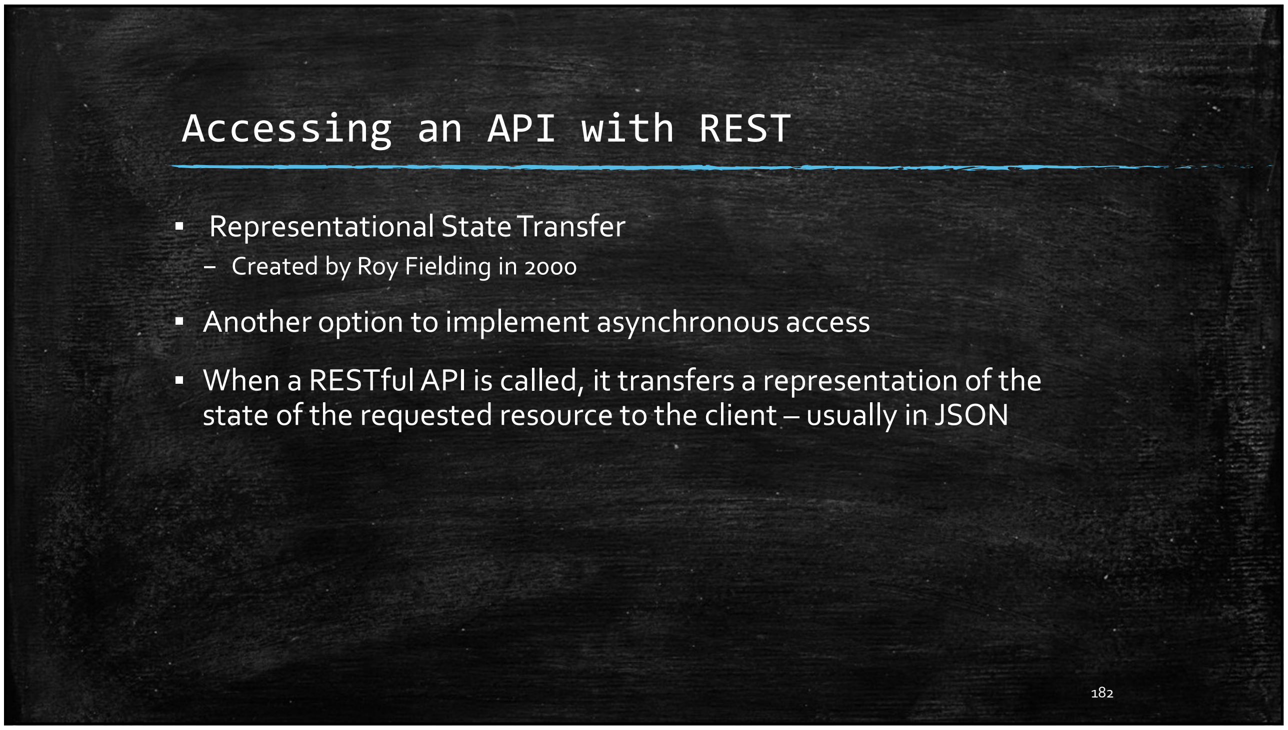
180



Using fetch() with an API

```
res
= fetch("https://api.openweathermap.org/data/2.5/onecall?lat=71.0589&lon=42.3601&exclude=hourl
y,daily&units=imperial&appid=xxxx")
.then (res => res.text())
.then (data =>
  {
    data = JSON.parse(data)
    data = data.current.temp;
     console.log("The current temperature is " + data + " degrees")
  })
.catch (error => console.log(error))
```

181

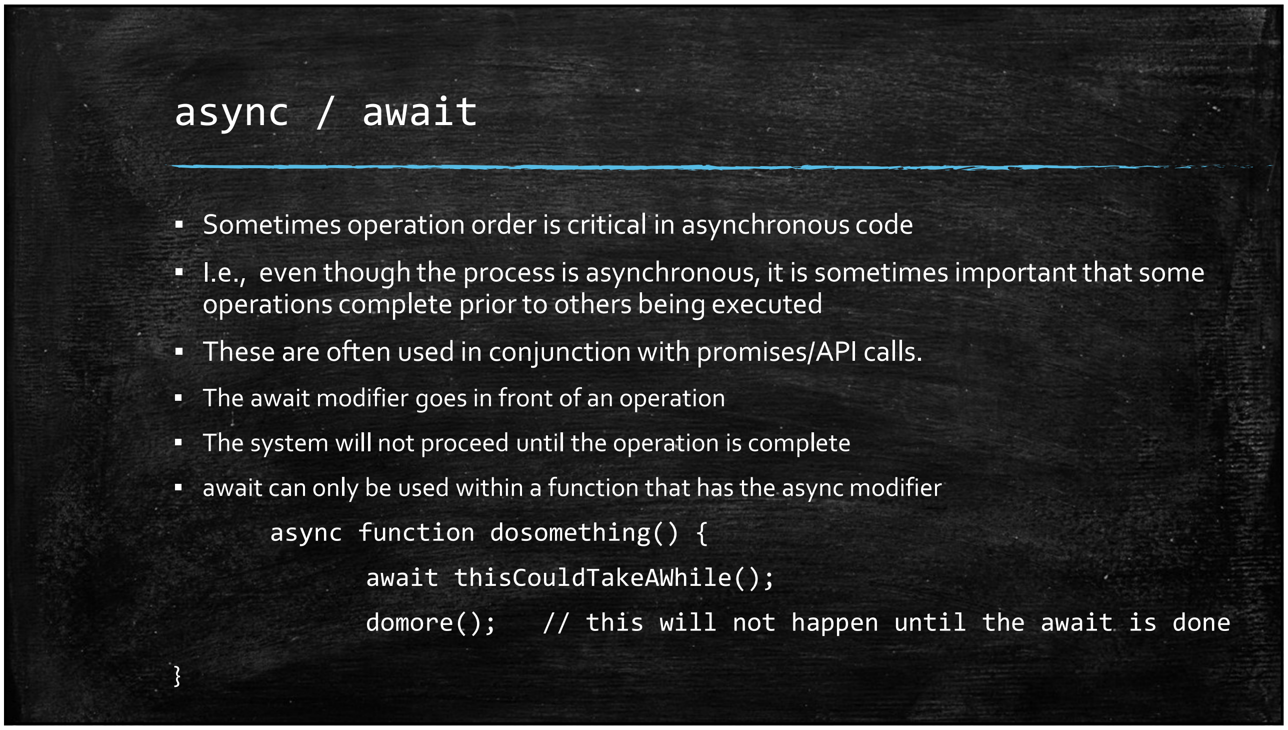


Accessing an API with REST

- Representational State Transfer
  - Created by Roy Fielding in 2000
- Another option to implement asynchronous access
- When a RESTful API is called, it transfers a representation of the state of the requested resource to the client – usually in JSON

182

182



async / await

- Sometimes operation order is critical in asynchronous code
- I.e., even though the process is asynchronous, it is sometimes important that some operations complete prior to others being executed
- These are often used in conjunction with promises/API calls.
- The await modifier goes in front of an operation
- The system will not proceed until the operation is complete
- await can only be used within a function that has the async modifier

```
      async function dosomething() {
            await thisCouldTakeAWhile();
            domore();   // this will not happen until the await is done
}
```

183