JavaScript Notes

What Is JavaScript Used For?

- ✔ DOM (Document Object Model) Manipulation
- ✔ Event Handling
- ✔ Asynchronous Requests
- ✔ Animations & Effects
- ✔ Data Manipulation (Sorting, filtering, etc)
- ✔ Storing Data (Cookies, LocalStorage, etc)
- ✔ Single Page Applications (SPA)
- ✔ Creating APIs & Web Services (Node.js, Deno)

What Is JavaScript?

JavaScript is one of the core technologies of the web, alongside HTML & CSS

It is a high-level, interpreted programming language that can be used on the client side as well as the server-side with Node.js

Why Learn JavaScript?

POPULARITY

One of the most widely used languages. Many job opportunities

VERSATILITY

Used for many different things, such as dynamic web pages, APIs, mobile and even desktop applications

RELATIVELY EASY TO LEARN

Compared to other lower level languages, JavaScript is pretty easy to learn and a great language to start with

COMMUNITY

Great community, support, resources and tools

Primitive Data Types

- **String** - Sequence of characters. Must be in quotes or backticks
- **Number** - Integers as well as floating-point numbers
- **Boolean** - Logical entity / true or false
- **Null** - Intentional absence of any object value
- **Undefined** - A Variable that has not yet been defined / assigned
- **Symbol** - Built-in object whose constructor returns a unique symbol
- **BigInt** - Numbers that are greater than the “Number” type can handle

Languages	Frameworks	Bundlers + Transpilers	HTTP
HTML			
CSS	Bootstrap	SASS	
JavaScript	React/Angular/Vue	Webpack	Typescript
			axios fetch

You'll notice that it's really just

Naming Conventions

camelCase	`cartQuantity`
PascalCase	`CartQuantity`
kebab-case	`cart-quantity`
snake_case	`cart_quantity`

Event Listeners

onclick	= click
onkeydown	= key press
onscroll	= scrolling
onmouseenter	= hovering over
onmouseleave	= stop hovering over

... and many more

null vs undefined

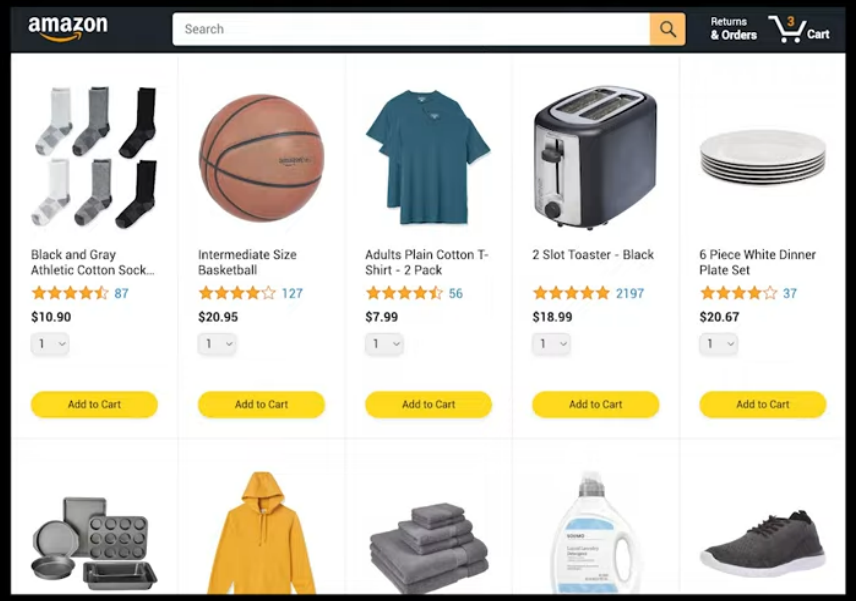
null = intentionally want something to be empty

```
function func(parameter = 'default') {
  console.log(parameter);
}

func();              => 'default'
func(undefined);     => 'default'

func(null);          => null
```

- Backend

1. Backend and HTTP
2. XMLHTTPRequest and fetch()
3. Asynchronous code
   Callbacks, promises, async await
4. How to test asynchronous code
5. Error handling
6. Use the backend in our project
7. URL parameters

Amazon Project

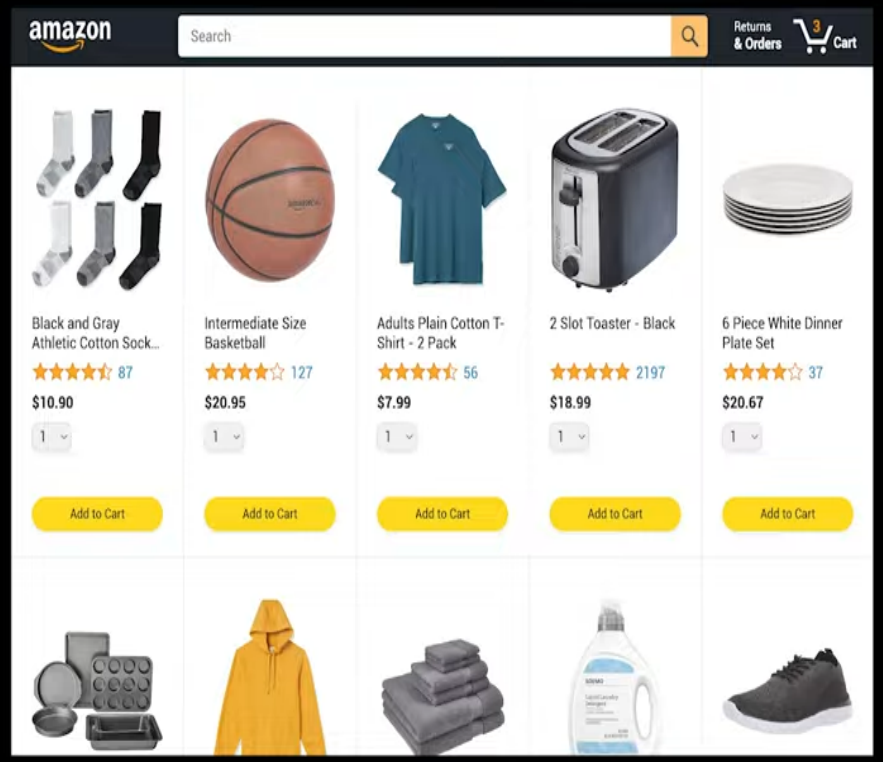


How does Amazon know which products we ordered?

Owned by Amazon

there has to be a second computer somewhere that's owned by Amazon,

Amazon Project

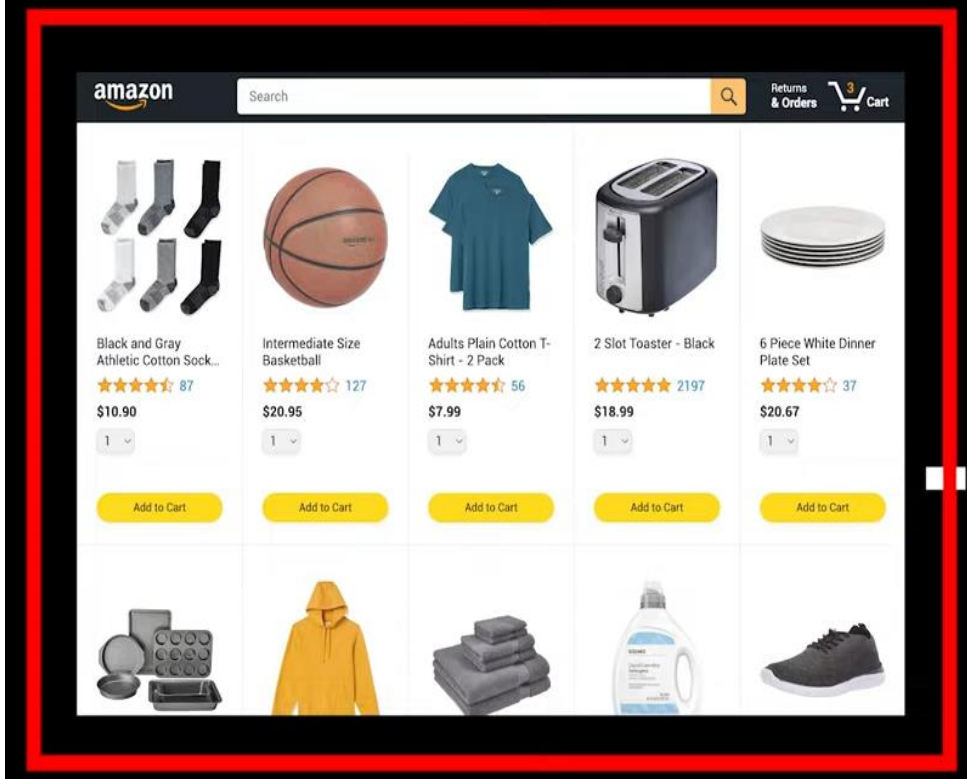


Order Info

Our computer

Owned by Amazon

Amazon Project



Our computer

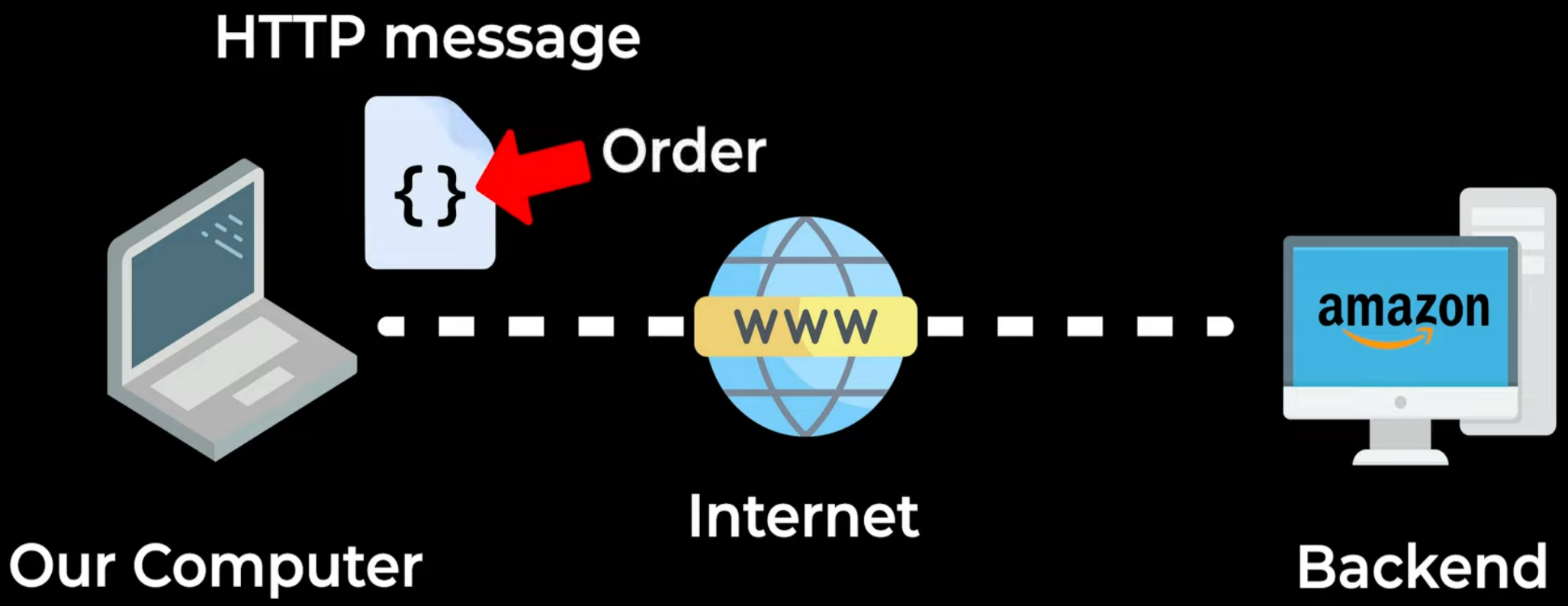
<u>Frontend</u>

Owned by Amazon

<u>Backend</u>

How does our computer send information to the backend?

HTTP = HyperText Transfer Protocol



To send data in a request,
we need to use a different type of request.

```
64    Place your order
```

4 Types of Requests

GET = get something from the backend

POST = create something

PUT = update something

DELETE

```
1  new XMLHttpRequest();
```

Creates a new HTTP message to send to the backend.

message = request

Request-Response Cycle = 1 request, 1 response

List of URL paths

Here is a list of URL paths that are supported by this backend.
Each URL path gives a different response when you send a request to it.

```
GET /
GET /hello
GET /products/first
GET /documentation
GET /images/apple.jpg
GET /products
GET /cart
POST /orders
```

Backend API

API = application programming interface

How we interact with something

```
GET /
```

```
// Runs the function after 3 seconds.
setTimeout(() => {
  console.log('hello');
}, 3000);
```

callback

Promises

- better way to handle asynchronous code
- similar to done() function
- let us wait for some code to finish, before going to the next step

resolve is a function

- similar to done() function
- lets us control when to go to the next step

```
new Promise((resolve) => {
  loadProducts(() => {
    resolve();
  });
})
```

```
beforeAll((done) => {
  loadProducts(() => {
    done();
  });
});
```

e.js JS cart-class.js (Untracked)

ts > JS checkout.js

```
import {renderOrderSummary}
import {renderPaymentSummar
import {loadProducts} from
// import '../data/cart-cla
// import '../data/backe
```

```
new Promise((resolve) => {
  loadProducts(() => {
    resolve();
  });
})

loadProducts(() => {
  renderOrderSummary();
  renderPaymentSummary();
});
```

These 2 groups of code are running at the same time.

```
new Promise((resolve) => {
```

```
loadProducts(() => {
```

```
renderOrderSummary();
```

```
renderPaymentSummary();
```

```
loadProducts(() => {
```

```
resolve();
```

```
// Next step.
```

Allows JavaScript to do multiple things at the same time.

```
ce.js        JS cart-class.js (Untracked)
pts > JS checkout.js
import {renderOrderSummary}
import {renderPaymentSummar
import {loadProducts} from
// import '../data/cart-cla
// import '../data/backe

new Promise((resolve) => {
  loadProducts(() => {
    resolve();
  });
})

loadProducts(() => {
  renderOrderSummary();
  renderPaymentSummary();
});
```

```
new Promise((resolve) => {
```

```
loadProducts(() => {
renderOrderSummary();
renderPaymentSummary();
```

```
loadProducts(() => {
resolve();
// Next step.
```

If we have lots of callbacks, our code will become more and more nested:

```
loadProducts(() => {
  loadCart(() => {
    loadOrders(() => {
      loadAccount(() => {
        loadHistory(() => {
          renderOrderSummary();
          renderPaymentSummary();
        });
      });
    });
  });
});
```

```
new Promise((resolve) => {
  loadProducts(() => {
    resolve();
  });

}).then(() => {
  return new Promise((resolve) => {
    loadCart(() => {
      resolve();
    });
  });

}).then(() => {
  return new Promise((resolve) => {
    loadOrders(() => {
      resolve();
    });
  });

}).then(() => {
  return new Promise((resolve) => {
    loadAccount(() => {
      resolve();
    });
```

Promise.all()

- lets us run multiple promises at the same time
- and wait for <u>all</u> of them to finish

```
import {loadCart} from '../data/c
// import '../data/cart-class.js'
// import '../data/backend-practi

async function loadPage() {
  console.log('load page');

  await loadProductsFetch();

  return 'value2';
}
```

```
function loadPage() {
  return new Promise((resolve) => {
    console.log('load page');
    resolve();

  }).then(() => {
    return loadProductsFetch();

  }).then(() => {
    return new Promise((resolve) => {
      resolve('value2');
    });
  });
}
```

<u>await</u>

= lets us write as
like normal cod

```
8  async function loadPage() {
9    console.log('load page');
10 }
11
```

```
function loadPage() {
  return new Promise((resolve) => {
    console.log('load page');
    resolve();
  });
}
```

async = makes a function return a promise

<u>async await</u>

= shortcut for promises

= lets us write asynchronous code like normal code.

<u>await</u>

= lets us write asynchronous code like normal code.