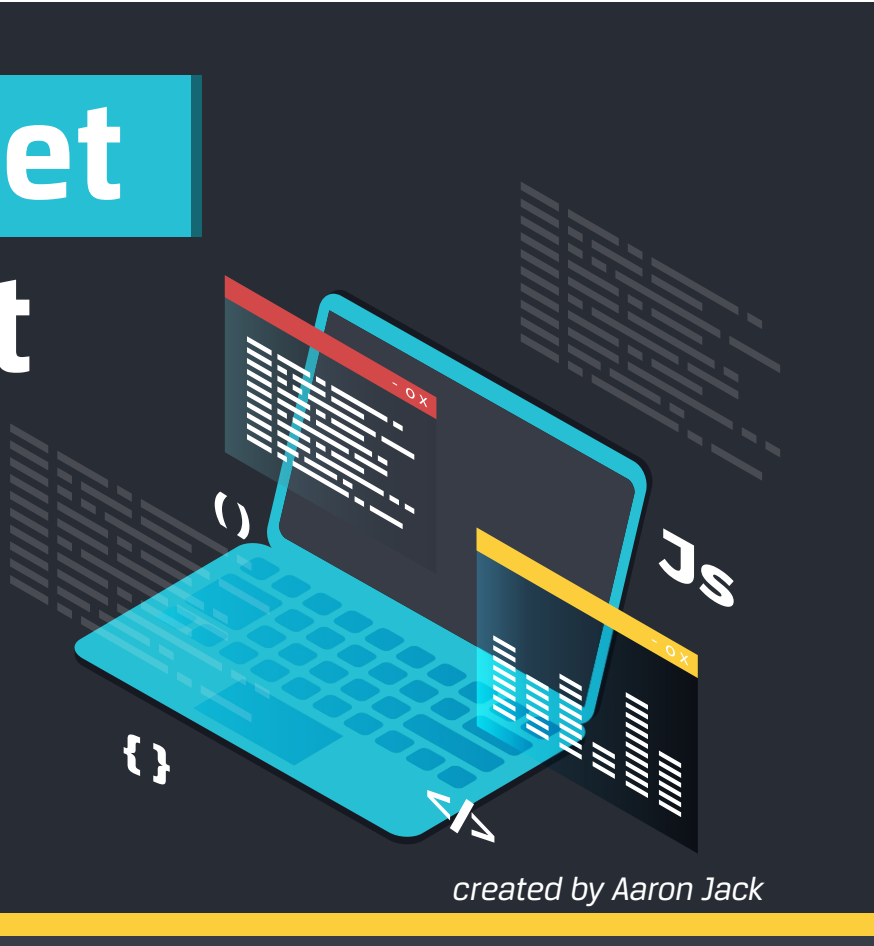
Don't Forget JavaScript

JS



created by Aaron Jack

String & Array

These methods create copies of the original (except for splice)

SLICE

```
[1, 2, 3].slice(0, 1)
```

returns [1]

Creates a new arr / str from first index, until second (or end, if no second arg)

SPLIT

```
"Jun-1".split("-")
```

returns ["Jun", "1"]

Convert a string to an array, split on the character you provide

JOIN

```
["Jun", "1"].join("-")
```

returns "Jun-1"

Convert an array to a string, separated by character you provide (or nothing using empty string)

SPLICE

```
[4, 2, 3].splice(0, 1)
```

returns [4]
original array is set to [2, 3]

Modifies the array in place by removing 2nd arg # of items at index of first arg. You can also add items with a third argument. Finally, returns removed items

Arrow Functions

Replace the function keyword with => after your arguments and you have an arrow function.

ARROW FUNCTION DECLARATION

```
const add = (num1, num2) => {
    return num1 + num2
}
```

Unlike a regular function, you must store an arrow function in a variable to save it

AUTO RETURNS I

```
(num1, num2) => num1 + num2
```

If your arrow function can fit on one line, you can remove the brackets AND return statement (return is automatic)

AUTO RETURNS II

```
(num1, num2) => (
    num1 + num2
)
```

Using a parenthesis on the same line (not bracket) is also an automatic return

SINGLE PARAMETER

```
word => word.toUpperCase()
```

If you just have a single parameter, you don't need the parentheses around your params

Objects

Powerful, quick storage and retrieval

KEY LITERALS

```
obj.a OR obj["a"]
```

This literally gives you the value of key "a"

KEY WITH VARIABLE

```
obj[a]
```

But this gives you the value of the key stored in variable a

FOR IN... LOOPS

```
for (let key in obj) ...
```

Loop over an object's keys with a for...in loop, and access its values using `obj[key]`

OBJECT.KEYS

```
Object.keys({a: 1, b: 2})
```

returns ["a","b"]

Easily get an object's keys in an array with Object.keys( ), or values with Object.values( )

DESTRUCTURING

```
const {a} = {a: 1}
```

variable a is set to 1

Destructuring lets you pull values out of objects, the key becomes its variable name

DESTRUCTURING II

```
const a = 1
const obj = {a}
```

variable obj is set to {a: 1}

It goes the other way too, assuming the variable a was already 1, this creates an object

Array Methods

These methods ALSO create copies of the original (except for sort)

MAP

```
[1, 2, 3].map(n => n + 1)
```

returns [2, 3, 4]

Runs the function once per item in the array.
Saves each return value in a new array, in the same place

FOREACH

```
[1, 2, 3].forEach(n=>console.log(n))
```

Same as map, but does not save results, it always returns undefined

FILTER

```
[1, 2, 3].filter(n => n > 1)
```

returns [2, 3]

Runs function once per item, if false, the item will not be ncluded in the new array, if true, it will

REDUCE

```
[1, 2, 3].reduce((a,val) => a + val)
```

returns 6

Runs function once per item, your return value becomes the accumulator arg on the next iteration. The accumulator starts at 0 by default but you can change it with an optional 2nd arg.

SORT

```
[3, 1, 2].sort()
```

returns [1, 2, 3]

Sorts array in place, by default in ascending numerical (or alphabetical) order. Passing in a 2 argument comparison function (optional) can arrange items in a descending, or custom order

the DOM

For every HTML tag there is a JavaScript DOM node

CREATE ELEMENT

```
document.createElement('div')
```

Create an HTML element with JavaScript, returns a <Node> object

SET STYLE

```
<Node>.style.color = "blue"
```

You can change a <Node> object's CSS styles

ADD CLASS

```
<Node>.classList.add(".myClass")
```

Add or remove a Node's CSS classes

INNER HTML

```
<Node>.innerHTML = "<div>hey</div>"
<Node>.innerText = "hey"
```

You can set a Node's HTML or text contents

ADD CHILD

```
<Node1>.appendChild(<Node2>)
```

You can nest nodes as children to existing nodes

QUERY SELECTOR

```
document
.querySelector("#my-id")
```

Search the DOM for the first Node that matches - both ".classes" and "#ids" work!

QUERY SELECTOR A LL

```
document
.querySelectorAll(".my-class")
```

Same as above, but returns all matches (in a node list)

ADD EVENT LISTENER

```
<Node>.addEventListener("click",
function() {...}
)
```

Add listeners to user events, like clicks. The function will run when the event happens.

Async Programming

Usually network requests, these functions happen outside of the normal "flow" of code

FETCH

```
fetch('https://google.com')
```

Fetch returns a promise , which is non blocking, in other words, your code keeps going

PROMISE.THEN

```
.then(result => console.log(result))
```

When it finally does return, use a .then method to capture its result in the first argument

.THEN CHAINING

```
.then(...).then(...)
```

A then block may also return a promise, in which case we can add another .then to it

PROMISE.CATCH

```
.catch(err => console.error(err))
```

Add a "catch" method to a promise, or chain of promises to handle any errors that may occur

PROMISE.ALL

```
Promise.all([fetch(...),fetch(...)]
   then(allResults => ...)
```

You can pass multiple promises into the Promise.all function. Attaching a .then block will give you the result of all the promises, in a single array.

ASYNC / AWAIT I

```
const res = await fetch(URL)
```

Async await is a cleaner syntax for promises, instead of .then blocks simply use the await keyword, which will block your code until the promise returns ..however...

ASYNC / AWAIT II

```
const getURL = async (URL) => (
  await fetch(URL)
```

Await keywords must be inside of an "async" function -- simply attach the async keyword before any function, or arrow function definition

Aaron Jack
Coding, Career, Remote lifestyle
Aaron Jack
@aaronjack
@ aaronjack