GDSC 7-Day BootCamp

Basics Of JavaScript

JavaScript is an object based programming language that is used mainly for changing HTML content on a web page.

Javascript is written inside the script tags in an HTML document:

```
<script>

 document.getElementById('mainHeading')...

</script>
```

ECMAScript is the official name of javascript and we use the ECMAScript 2015 or ES6 standard of JS with react.

Identifiers in JavaScript

With ES6, there are three ways of defining your variables: `var`, `let`, and `const`.

Data Types

JavaScript has 8 Datatypes

1. String
2. Number
3. Bigint
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

1. An object
2. An array
3. A date

Functions

A JavaScript function is a block of code designed to perform a particular task.

A JavaScript function is executed when "something" invokes it (calls it).

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.

Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:
(*parameter1, parameter2, ...*)

The code to be executed, by the function, is placed inside curly brackets: **{}**

Function **parameters** are listed inside the parentheses () in the function definition.

Function **arguments** are the **values** received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Function Invocation

The code inside the function will execute when "something" **invokes** (calls) the function:

- When an event occurs (when a user clicks a button)
- When it is invoked (called) from JavaScript code
- Automatically (self invoked)

You will learn a lot more about function invocation later in this tutorial.

Function Return

When JavaScript reaches a return statement, the function will stop executing.

If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- String Operators
- Logical Operators
- Bitwise Operators
- Ternary Operators
- Type Operators

Variable Scope

If you use `var` outside of a function, it belongs to the global scope.

If you use `var` inside of a function, it belongs to that function.

If you use `var` inside of a block, i.e. a for loop, the variable is still available outside of that block.

`var` has a *function* scope, not a *block* scope.

`let` is the block scoped version of `var`, and is limited to the block (or expression) where it is defined.

If you use `let` inside of a block, i.e. a for loop, the variable is only available inside of that loop.

`let` has a *block* scope.

`const` is a variable that once it has been created, its value can never change.

`const` has a *block* scope.

The keyword `const` is a bit misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

- Reassign a constant value
- Reassign a constant array
- Reassign a constant object
- But you CAN:
- Change the elements of constant array
- Change the properties of constant object

Arrow Functions

Arrow functions allow us to write shorter function syntax.

Destructuring

To illustrate destructuring, we'll make a sandwich. Do you take everything out of the refrigerator to make your sandwich? No, you only take out the items you would like to use on your sandwich.

Destructuring is exactly the same. We may have an array or object that we are working with, but we only need some of the items contained in these.

Destructuring makes it easy to extract only what is needed.

```
function calculate(a, b) {
  const add = a + b;
  const subtract = a - b;
  const multiply = a * b;
  const divide = a / b;
  return [add, subtract, multiply, divide];
}
const [add, subtract, multiply, divide] = calculate(4, 7);
```

Spread Operator

The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.

```
const numbersOne = [1, 2, 3];

const numbersTwo = [4, 5, 6];

const numbersCombined = [...numbersOne, ...numbersTwo];
```

The spread operator is often used in combination with destructuring.

```
const numbers = [1, 2, 3, 4, 5, 6];

const [one, two, ...rest] = numbers;
```

Modules

JavaScript modules allow you to break up your code into separate files.

This makes it easier to maintain the code-base.

ES Modules rely on the `import` and `export` statements.

Export

You can export a function or variable from any file.

Let us create a file named `person.js`, and fill it with the things we want to export.

There are two types of exports: Named and Default.

Named Exports

You can create named exports two ways. In-line individually, or all at once at the bottom.

```
export const name = "Jesse"
export const age = 40
```

OR

```
const name = "Jesse"
const age = 40

export { name, age }
```

Default Exports

Let us create another file, named `message.js`, and use it for demonstrating default export.

You can only have one default export in a file.

```
const message = () => {
  const name = "Jesse";
  const age = 40;
  return name + ' is ' + age + 'years old.';
};

export default message;
```

Import

You can import modules into a file in two ways, based on if they are named exports or default exports.

Named exports must be destructured using curly braces. Default exports do not.

```
import { name, age } from “./person.js";

OR

import message from "./message.js";
```