JS short summary

Javascript is used to program the behaviour of webpages

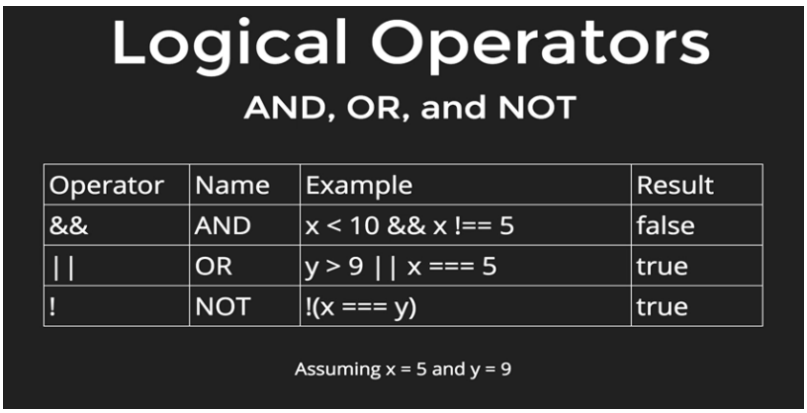
First, let´s go through the ***JavaScript variables*.** JavaScript variables are containers for storing data values. In this example, x, y, z, name and age are variables storing different values:

```
var x = 5;                  // x storing 5, two dashes are also the comment sign in javascript
var y = 6;                  // y storing 6
var z = x + y;              // z storing 11
var name = ”Rusty”;         // name storing ”rusty”
var age;                    // storing null which means ”explicity nothing”
```

We can also update existing variables:

```
x = 8;                  // x now stores 8 instead of 5
name = ”Kitten”         // name now stores ”Kitten” instead of ”Rusty”
```

When we write Javascript we often use **logical operators** to compare values. These are the most common ones:



Now we are going to look at the ***built in functions*** alert and console.log.

If you write:

```
alert(”Hi there!”);
```

the message Hi there! pops up for the user

If you write:

```
console.log("Hi there!")
```

the message Hi there! will show up in the javascript console.

JS short summary

When we write Javascript, just like when we write CSS, we want to do that in a separate file. We do this by using the <script> tag in the head of our html file.

```
<!DOCTYPE html>
<html>
<head>
<title></title>
<script src="script.js"></script>                    //link to script.js where we write our JS
</head>
<body>
<hl> Including JS files </hl>
</body>
</html>
```

We then need to have a script.js file. Inside our script.js file we could have the code:

```
alert("Hello from the JS file");
```

And the result would then be in the browser:

When we write Javascript we often use **comparison operators** to compare values. These are the most common ones:

Comparison Operators

Assuming x = 5

Operator	Name	Example	Result
>	Greater than	x > 10	false
>=	Greater than or equal to	x >= 5	true
<	Less than	x < -50	false
<=	Less than or equal to	x <= 100	true
==	Equal to	x == "5"	true
!=	Not equal to	x != "b"	true
===	Equal value and type	x === "5"	false
!==	Not equal value or equal type	x !== "5"	true

JS short summary

Conditionals control behavior in JavaScript and determine whether or not pieces of code can run. For example,

```
If (age < 18) {
console.log("You are not old enough to enter this venue");
 }
 else if (age < 21) {
console.log("you can enter but can not drink, if you are in America");
}
 else {
console.log("come in, you can drink");
}
```

Example explained:
If age is less than 18 the "You are not old enough to enter this venue" will be printed in the console. If age is less than 21 but bigger than 18 the " you can enter but can not drink, if you are in America" will be printed. And in all other scenarios (age is bigger than 21) it will print " come in, you can drink". Remember that if the first condition is true you will not go thorugh the next ones. And if the second one is true it will not go through the last one.

Loops are handy if you want to run the same code over and over again, each time with a different value.

JavaScript supports different kinds of loops:
• for - loops through a block of code a number of times
• while - loops through a block of code while a specified condition is true

The ***for loop*** has the following syntax:

```
for (statement 1; statement 2; statement 3) {
code to be executed;
 }
```

Statement 1 is executed before the loop starts.
Statement 2 The condition for running the loop. If it returns true, the loop will start over again, if it returns false, the loop will end.
Statement 3 is executed each time after the loop has been executed.

To print out the numbers 1-9 it could look like:

```
for (var = i; i < 10; i++) {
console.log(i);
 }
```

JS short summary

The ***while loop*** has the following syntax:

```
while (condition) {
code block to be executed;
}
```

For example,

```
var i = 1;
 while (i < 10) {
console.log(i);
i++;
}
```

Example explained:
While i is less than 10 it will print the number. In this case it will print numbers 1-9.

Now lets move on to JS functions. **Javascript functions** let us wrap bits of code up into **REUSABLE** packages.

First thing we have to do when working with functions is to declare them. We can do that like this:

```
function doSomething( ) {                  //Name the function doSomething
      console.log("Hi there!");          //What will happen when the function is called
}
```

As you can see in the code above we named this function doSomething and it´s now declared. When it is called it will console.log Hi there.

After we have declared it we have to call/run it before anything happens. The syntax to call it is:

```
doSomething( );
```

Often we want to write functions that take input which is when we use **ARGUEMENTS**. For example,

```
function square(num) {                      //Name the function square and the arguement num
      console.log(num * num);              //Use the arguement to make something happened
}
```

Now when we call square, we need to pass in a value. For example.

```
square(10);                    //prints 100
square(3);                    // prints 9
```

JS short summary

Arrays let us group data together in lists. The lists are indexed and starts at index 0. We can create an array with the name dogs ike this:

```
var dogs = [”Charlie”, ”Rufs”,”Tuffs”,”Matts”];
```

We can use the index to retrieve data. For example,

```
console.log(friends[0]);              //prints out Charlie
```

We can also update arrays with new different data. For example change the first content in a list/array to another. For example,

```
dogs[0] = ”Kitty”;
dogs[1] = ”Katty”;
```

We can also add new data. For example like this:

```
dogs[4] = "Teddy";
```

Arrays come with a few built-in methods. Most important ones:

Push

Put something in the end of an array. For example, if we want to put something in the end of our dogs array

```
dogs.push("Jack");
```

Now the array containes Kitty, Katty, Tuffs, Matts and Jack

Pop

Removes the last item of the array. For example if we want to remove Jack we write.

```
dogs.pop( );
```

Unshift

Add an element in the beginning of the array

```
dogs.unshift("Albert");
```

Shift

Removes the first item of the array.

```
dogs.shift( );
```

IndexOf

Use indexOf to find the index of an item in an array.

```
dogs.indexOf("Kitty");              //returns 1
```

JS short summary

Javascript Objects

An object is a collection of related data and/or functionality — which are called properties and methods when they are inside objects.

Objects store data in key-value pairs.

For example, let's create an object named person and give it properties named name , age and city. In this case name, age and city are keys and Cindy, 32 and Stockholm are values.

```
var person = {
     name: "Cindy",
     age: 32,
     city: "Stockholm"
};
```

To retrieve data from an object you have two choices.

1) Bracket. For example,

```
console.log(person["name"]);                    //prints Cindy
```

2) Dot notation. For example,

```
console.log(person.name);                    //prints Cindy
```

Just like an array you can also update data. For example,

```
person.name = "Anna";          //The name property now have the value Anna instead of Cindy
```

Javascript part is done! And remember that JS is hard. But it will get easier if you just write the code parts many many times. Keep going!

You are awesome!!