Web Application Development

2019

Javascript

Part I

Javascript

> JavaScript is a scripting or programming language that allows you to implement complex things on web pages — displaying timely content updates, interactive maps, animated 2D/3D graphics etc.It is the third layer of the layer cake of standard web technologies, alongside with HTML and CSS

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

HTML

The markup language that we use to structure and give meaning to our web content.

CSS

A language of style rules that we use to apply styling to our HTML content

Javascript

A scripting language that enables us to create dynamically updating content, control multimedia, animate images, and muchmore

Java != Javascript

Java is to **JavaScript**

as

Car is to **Carpet**

How does it work?

The JavaScript is executed by the browser's JavaScript engine, after the HTML and CSS have been assembled and put together into a web page.

This ensures that the structure and style of the page are already in place by the time the JavaScript starts to run.

Add Javascript to HTML page

Internal Javascript

```
1.  ...
2.  <body>
3.      <script>
4.          // JavaScript goes here
5.      </script>
6.  </body>
7.  ...
```

External Javascript

```
1.  ...
2.  <head>
3.      ...
4.      <script src="script.js"></script>
5.      ...
6.  </head>
7.  ...
```

Inline Javascript Handlers

```
1.  <button onclick="doSomething()">Click me!</button>
2.  <script>
3.      function doSomething() {
4.          // Do Something
5.      }
6.  </script>
```

NB: Please don't do this, however. It is bad practice to pollute your HTML with JavaScript, and it is inefficient

Variables

Variables

A variable is a **container** for a **value**, like a number we might use in a sum, or a string that we might use as part of a sentence. But one special thing about variables is that their contained values can **change**

Declaring a variable

```
1.   let myName;
2.   let myAge;
```

Variable Naming: You can call a variable pretty much anything you like, but there are limitations. Generally, you should stick to just using Latin characters (0-9, a-z, A-Z) and the underscore character.
It is advised to use **lowerCamelCase**

Initializing a variable

```
1.  myName = 'John';
2.  myAge = 26;
3.  let myDog = 'Bobby';
```

Updating a variable

```
1.   myName = 'Bob';
2.   myAge = 27;
3.   myDog = 'Is a Cat now';
```

Variable types

```
1.  let myAge = 17;
2.  let myHeight = 1.79;
3.  let myVision = -20;
4.
5.  let theQuickBrownFox = 'jumped over lazy dog';
6.
7.  let iAmAlive = true;
8.  let test = 6 < 3;
9.
10. let myNameArray = ['Chris', 'Bob', 'Jim'];
11. let myNumberArray = [10, 15, 40];
12.
13. let dog = { name : 'Spot', breed : 'Dalmatian' };
```

Dynamic typing

JavaScript is a "dynamically typed language", which means that, unlike some other languages, you don't need to specify what data type a variable will contain (numbers, strings, arrays, etc).

```
1.  let myNumber = 'five hundred';
2.  myNumber = 500;
3.  myNumber = false;
```

Constants

Many programming languages have the concept of a constant — a value that once declared can never be changed.

```
1.  const daysInWeek = 7;
2.  const hoursInDay = 24;
```

```
1.  daysInWeek = 8;
2.  hoursInDay = 25;
```

Operators

Operator	Name	Purpose	Example
`+`	Addition	Adds two numbers together.	`6 + 9`
`-`	Subtraction	Subtracts the right number from the left.	`20 - 15`
`*`	Multiplication	Multiplies two numbers together.	`3 * 7`
`/`	Division	Divides the left number by the right.	`10 / 5`
`%`	Remainder (sometimes called modulo)	Returns the remainder left over after you've divided the left number into a number of integer portions equal to the right number.	`8 % 3` (returns 2, as three goes into 8 twice, leaving 2 left over).
`**`	Exponent	Raises a **`base`** number to the **`exponent`** power, that is, the **`base`** number multiplied by itself, **`exponent`** times.	`5 ** 2` (returns `25`, which is the same as `5 * 5`).

Operator precedence

What will be the value of variable **num3**?

- 12
- 5
- 2.4

```
1.  let num1 = 16;
2.  let num2 = 8;
3.  let num3 = num2 + num1 / 8 + 2;
```

Operator precedence

What will be the value of variable **num3**?

- 12
- ~~5~~
- ~~2.4~~

```
1.  let num1 = 16;
2.  let num2 = 8;
3.  let num3 = num2 + num1 / 8 + 2;
```

Increment and decrement operators

```
1.  let num1 = 4;
2.  num1++;
3.  //5
4.  let num2 = 6;
5.  num2--;
6.  //5
```

Operator	Name	Purpose	Example	Shortcut for
`+=`	Addition assignment	Adds the value on the right to the variable value on the left, then returns the new variable value	`x = 3;` `x += 4;`	`x = 3;` `x = x + 4;`
`-=`	Subtraction assignment	Subtracts the value on the right from the variable value on the left, and returns the new variable value	`x = 6;` `x -= 3;`	`x = 6;` `x = x - 3;`
`*=`	Multiplication assignment	Multiplies the variable value on the left by the value on the right, and returns the new variable value	`x = 2;` `x *= 3;`	`x = 2;` `x = x * 3;`
`/=`	Division assignment	Divides the variable value on the left by the value on the right, and returns the new variable value	`x = 10;` `x /= 5;`	`x = 10;` `x = x / 5;`

Concatenating strings

```
1.  let one = 'Hello, ';
2.  let two = “how are you?”;
3.  let joined = one + two;
4.  // Hello, how are you?
```

Concatenating strings

```
1.  let one = 'Hello, ';
2.  let two = 404;
3.  let joined = one + two;
4.  // Hello, 404?
```

Arrays

Arrays are generally described as "list-like objects"; they are basically single objects that contain multiple values stored in a list. We can access each value inside the list individually, and do useful and efficient things with the list, like loop through it and do the same thing to every value.

```
1.  let shopping = ['bread', 'milk', 'cheese', 'eggs'];
2.  let sequence = [1, 1, 2, 3, 5, 8, 13];
3.  let random = ['tree', 795, [0, 1, 2]];
```

Accessing and modifying array items

You can then access individual items in the array using bracket notation

```
let shopping = ['bread', 'milk', 'cheese', 'eggs'];
shopping[0];
// returns "bread"
shopping[0] = 'noodles';
// shopping list will now contain
[ "noodles", "milk", "cheese", "eggs" ]
```

Length of an array

You can find out the length of an array (how many items are in it)

```
1.  let shopping = ['bread', 'milk', 'cheese', 'eggs'];
2.  shopping.length;
3.  // returns 4
```

Strings are arrays

Strings are an array of characters

```
1.  let myString= 'Hello World!';
2.  myString[0];
3.  // returns 'H'
4.  myString.length;
5.  // returns 12
```

Conditionals

if ... else statements

```
if (condition) {
  code to run if condition is true
}
else {
  run some other code instead
}
```

```
let shoppingDone = false;
if (shoppingDone === true) {
    let moneyForBeer = 10;
}
else {
    let moneyForBeer = 0;
}
```

else ... if statements

```
1.  let wifeAngerLevel = 10;
2.  let haveEnoughMoney = false;
3.  if (wifeAngerLevel > 9 ) {
4.      let buyGift = 'Chocolate';
5.  }
6.  else if (wifeAngerLevel > 5 ) {
7.      let buyGift = 'flowers';
8.  }
9.  else {
10.     let buyGift = false;
11.     if (haveEnoughMoney) {
12.         let buyMyselfBeer = true;
13.     }
14. }
```

Logical operators

- AND - & &
- OR - | |
- NOT - !

```
1.  if (choice == 'sunny' && temperature < 25) {
2.      wearJacket()
3.  }
4.  else if (choice == 'sunny' || temperature >= 25) {
5.      wearTshirt()
6.  }
7.  else if (!(choice == 'sunny' || temperature >= 0)){
8.      wearTshirt()
9.  }
```

switch...case

```
switch (choice) {
    case 'sunny':
        wearTshirt();
        break;
    case 'rainy':
        wearJacket();
        break;
    case 'snowing':
        wearCoat();
        break;
    default:
        stayHome();
}
```

Ternary operator

```
( condition ) ? run this : run this instead
```

```
let number = (isEven) ? 'Even' : 'Odd';
```

Loops

Loops

- A **counter**, which is initialized with a certain value — this is the starting point of the loop.
- An **exit condition**, which is the criteria under which the loop stops — usually the counter reaching a certain value.
- An **iterator**, which generally increments the counter by a small amount on each successive loop, until it reaches the exit condition.

Loops

Pseudo code

```
loop(food = 0; foodNeeded = 10) {
    if (food >= foodNeeded) {
        exit loop;
    }
    else {
        food += 2;
    }
}
```

The standard `for` loop

1. An **initializer** — this is usually a variable set to a number, which is incremented to count the number of times the loop has run.
2. An **exit-condition** — this defines when the loop should stop looping.
3. A **final-expression** — this is always evaluated (or run) each time the loop has gone through a full iteration.

```
1.  const cats = ['Bill', 'Jeff', 'Jasmin'];
2.  let info = 'My cats are called ';
3.  for (let i = 0; i < cats.length; i++) {
4.      info += cats[i] + ', ';
5.  }
6.  // My cats are called Bill, Jeff, Jasmin,
```

Exiting loop

If you want to exit a loop before all the iterations have been completed, you can use the `break` statement.

```
1.  const cats = ['Bill', 'Jeff', 'Jasmin'];
2.  let info = 'My cats are called ';
3.  for (let i = 0; i < cats.length; i++) {
4.      if (i == 1) {
5.          break;
6.      }
7.      info += cats[i] + ', ';
8.  }
9.  // My cats are called Bill,
```

Skipping iterations

The `continue` statement works in a similar manner to `break`, but instead of breaking out of the loop entirely, it skips to the next iteration of the loop.

```
1.  const cats = ['Bill', 'Jeff', 'Jasmin'];
2.  let info = 'My cats are called ';
3.  for (let i = 0; i < cats.length; i++) {
4.      if (i == 1) {
5.          continue;
6.      }
7.      info += cats[i] + ', ';
8.  }
9.  // My cats are called Bill, Jasmin,
```

`while` loop

This works in a very similar way to the for loop, except that the initializer variable is set before the loop, and the final-expression is included inside the loop after the code to run

```
let i = 0;
while (i < cats.length) {
    if (i == cats.length - 1) {
        info += 'and ' + cats[i] + '.';
    }
    else {
        info += cats[i] + ', ';
    }
    i++;
}
// My cats are called Bill, Jeff and Jasmin.
```

`do...while` loop

The differentiator here is that the exit-condition comes after everything else, wrapped in parentheses and preceded by a while keyword. In a do...while loop, the code inside the curly braces is always run once before the check is made to see if it should be executed again

```
let i = 0;
do {
    if (i == cats.length - 1) {
        info += 'and ' + cats[i] + '.';
    }
    else {
        info += cats[i] + ', ';
    }
    i++;
}while (i < cats.length)
// My cats are called Bill, Jeff and Jasmin.
```

Errors

Error types

Syntax errors: These are spelling errors in your code that actually cause the program not to run at all, or stop working part way through — you will usually be provided with some error messages too. These are usually okay to fix, as long as you are familiar with the right tools and know what the error messages mean!

Logic errors: These are errors where the syntax is actually correct but the code is not what you intended it to be, meaning that program runs successfully but gives incorrect results. These are often harder to fix than syntax errors, as there usually isn't a resulting error message to direct you to the source of the error.

References

https://developer.mozilla.org/en-US/

Questions?

Next: Javascript Part II