JavaScript Basic Syntax

CCAPDEV

Data Types

JavaScript supports the following primitive data types:

- **Numbers** – such as 125, 12.9, etc.
- **Strings** – text such as “Hello World”
- **Boolean** – true or false

Variables

- JavaScript is an untyped language – a variable can be of any data type.
- The data type of a variable can change based on the data type of the value assigned to it.

Variables

```
<script>
	var id;
	var name;
</script>
```

Variables

```
<script>
      var id, name;
</script>
```

Variables

```
<script>
      var id = 113;
      var name;
      name = “Ned”
</script>
```

NOTE

semicolons are not required for writing statements in JavaScript programming, as it includes an *automatic* semicolon feature.

Variables

- **Global Scope** – variables defined anywhere in the JavaScript code. These variables are accessible to all functions that have no duplicate identifier names.
- **Function Scope** – variables declared within a function. These variables are only visible within its function.
- **Block Scope** – variables declared within a block (i.e., { }). These variables are only visible within its block. *New* feature as of ES6 & onwards.

Variables... example

```
<script>
        var myVar = “global”; // global scope
        function sample() {
                var myVar = “func”; //function scope
                document.write(myVar);
                {
                        let myVar = “block”; // block scope
                        document.write(“block” + myVar);
                }
        }
</script>
```

Variables... `var` vs `let`

- There are three keywords associated with declaring a variable (or constant):
  - **`var` –** used to define global and function scoped variables
  - **`let` –** used to define block scoped variables
  - **`const` –** used to declare constants, cannot be reassigned. Can also be used for defining block-scope constants
- `let` and `const` are introduced in ES6 (2015)

Variables... automatically GLOBAL

```
<script>
myfunction();
function myFunction() {
        myVar = “value”;
}
</script>
```

If you assign a value to a variable that has not been declared, it will automatically become a **GLOBAL** variable

Variables... **var** vs **let** in **GLOBAL** and the **lifetime** of a variable

```
<script>
        var var1 = “value1”;
        let var2 = “value2”;
</script>
```

- Global variables declared with **var** belong to the window object, i.e., they will **persist** until you close the browser window (or tab).

- Global variables declared with **let** and function/block variables are deleted when the function/program execution is completed.

IF ELSE

```
<script>
      var x = 3;
      if(x == 1)
            document.write(“x is equal to 1”);
</script>
```

IF ELSE

```
<script>
	var x = 3;
	if(x == 1)
		document.write(“x is equal to 1”);
	else
		document.write(“x is not equal to 1”);
</script>
```

IF ELSE

```
<script>
	var x = 3;
	if(x == 1)
		document.write(“x is equal to 1”);
	else if(x == 2)
		document.write(“x is equal to 2”);
	else document.write(“x is neither 1 nor 2”);
</script>
```

Switch

```
<script>
	var x = 3;
	switch(x) {
		case 1: document.write(“x = 1”); break;
		case 2: document.write(“x = 2”); break;
		default: document.write(“x =/= 1 || 2”);
	}
</script>
```

Loops

```
<script>
	var i = 0;
	var n = 3;
	while (i < n) {
		document.write(i + “<br>”);
		i++;
	}
</script>
```

Loops

```
<script>
	var i = 0;
	var n = 3;
	do {
		document.write(i + “<br>”);
		i++;
	} while (i < n);
</script>
```

Loops

```
<script>
      var i;
      for(i = 0; i < n; i++)
            document.write(i + “<br>”);
</script>
```

Note about == and !=

- In JavaScript, there are two different variations for both the equality (==) and inequality (!=) operators:
  - == is the equal to operator. However, it also results to **true for values of different types.**
    - e.g.,
    - var x = 3;
    - x == 3; // true
    - x == “3”; // true
  - === is the equal to and **equal type** operator.
    - e.g.,
    - var x = 3;
    - x === 3; // true
    - x === “3”; // false
  - For inequality, the equivalent variant is !==

Functions

- To declare a function, use the `function` reserved word
- The return value and the data types of the parameters are not declared.

Functions

```
<script>
	function functionName(parameters) {
		statements
	}
</script>
```

Functions

```
<script>
	function sayHello() {
		alert(“Hello there!”);
	}
</script>
```

Functions

```
<script>
	const sayHello = () => {
		alert(“Hello there!”);
	};
</script>
```

Functions

```
...
<body>
      <form>
            <input type=“button”
            onclick=“sayHello()”
            value=“Say Hello” />
      </form>
</body>
```

Functions

```
<script>
	function sayHello(name) {
		alert(“Hello there, ” + name + “!”);
	}
</script>
```

Functions

```
<script>
	const sayHello = (name) => {
		alert(“Hello there, ” + name + “!”);
	};
</script>
```

Functions

```
...
<body>
      <form>
            <input type=“button”
            onclick=“sayHello(‘Ned’)”
            value=“Say Hello” />
      </form>
</body>
```

Arrays

```
var <array name> = new Array(<size>);
var <array name> = new Array(val1, val2, ...);
var <array name> = [val1, val2, ...];
```

Arrays

```
var colors = new Array(3);
var colors = new Array(“blue”, “red”, “yellow”);
var colors = [“blue”, “red”, “yellow”];
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// accessing the value “blue”
colors[0];
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// changing the 2nd element
colors[1] = “green”;
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// add element at the end of the array
colors[3] = “pink”;
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// add element at the end of the array
colors.push(“pink”);
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// add elements at the end of the array
colors.push(“pink”, “white”);
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// add element at the start of the array
colors.unshift(“grey”);
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// add elements at the start of the array
colors.unshift(“grey”, “black”);
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// insert element after the 1st element
colors.splice(1, 0, “silver”);
// inserts at index 1:
// [“blue”, “silver”, “red”, “yellow”]
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// insert elements after the 1st element
colors.splice(1, 0, “silver”, “green”);
// inserts at index 1:
// [“blue”, “silver”, “green”, “red”, “yellow”]
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// insert element after the 1st element
colors.splice(1, 1, “silver”);
// inserts at index 1, delete 1 element after:
// [“blue”, “silver”, “yellow”]
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// remove the last element in the array
colors.pop();
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// remove the last element in the array
colors.length = 2;
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// remove the first element in the array
colors.shift();
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// remove the 2nd element in the array
colors.splice(1, 1);
// [“blue”, “yellow”]
```

Arrays

```
var colors = [“blue”, “red”, “yellow”];

// remove the 2nd and 3rd elements in the array
colors.splice(1, 2);
// [“blue”]
```

Loops... for.. of loop (mainly used for arrays)

```
<script>
      const array1 = [‘a’, ‘b’, ‘c’];
      for (let item of array1)
            document.write(item + “<br>”);
</script>
```

Objects

- Objects are composed of attributes.
- If an attribute contains a function, it is referred as a *method* of the object. Else, the attribute is referred to as a *property*.
- To add a new field/property, simply assign a value to it. If the field doesn't exist, JS will create it automatically.

Objects

```
var p = {
    firstName: “Jimmy”,
    lastName: “Neutron”,
    age: 50
};
```

```
var p = new Object();
p.firstName = “Jimmy”;
p.lastName = “Neutron”;
p.age = 50;
```

Objects

```
var p = {
    firstName: “Jimmy”,
    lastName: “Neutron”,
    age: 50
};
```

```
// This will not create a copy of p
// but rather create a reference to p
var x = p;
x.firstName = “Hugh”;
```

Objects

```
var p = {
    firstName: “Jimmy”,
    lastName: “Neutron”,
    age: 50
};
```

```
// access properties
p.firstName
p[“firstName”]
```

Loops... for.. in loop (loops through properties of an object)

```
<script>
	const person = {
		fname: “Jimmy”,
		lname: “Neutron”
	}
	for (let x in person)
		document.write(person[x] + “<br>”);
</script>
```

Objects

```
var p = {
    firstName: “Jimmy”,
    lastName: “Neutron”,
    age: 50
};
```

```
// add new property
p.eyeColor = “brown”;
```

Objects

```
var p = {
    firstName: “Jimmy”,
    lastName: “Neutron”,
    age: 50
};
```

```
// delete property
delete p.age;


// the property will be
released from memory only
after nothing else references
it
```

Objects

```
var p = {
   hours: 40,
   ratePerHour: 500
};

p.getSalary = function() {
   return this.hours * this.ratePerHour;
}
```

Objects

```
var p = {
   hours: 40,
   ratePerHour: 500,
   getSalary() {
       return this.hours * this.ratePerHour;
   }
};
```

Objects

```
var p = {
   hours: 40,
   ratePerHour: 500,
   getSalary: function() {
       return this.hours * this.ratePerHour;
   }
};
```

Objects

```
var p = {
   hours: 40,
   ratePerHour: 500,
   getSalary: function() {
       return this.hours * this.ratePerHour;
   }
};

p.getSalary(); // calls the method
```

Objects

```
var p = {
    hours: 40,
    ratePerHour: 500,
    get salary() {
        return this.hours * this.ratePerHour;
    }
};

p.salary //calls the getter (no parenthesis)
```

Objects

```
var p = {
    hours: 40,
    ratePerHour: 500,
    set rate(val) {
        this.ratePerHour = rate;
    }
};

p.rate = 600; //calls the setter (no parenthesis)
```

NOTE

val can be replaced with any identifier

Objects

```
function person(firstName, lastName, age) {
   this.firstName = firstName;
   this.lastName = lastName;
   this.age = age;
}

// create objects
var p1 = new person(“Hugh”, “Neutron”, 50);
```

Objects

```
class Person {
   constructor (firstName, lastName, age) {
      this.firstName = firstName;
      this.lastName = lastName;
      this.age = age;
   }
}

// create objects
var p1 = new person(“Hugh”, “Neutron”, 50);
```

Further Reading

- **`async`**, **`Promises`**, and **`await`**
  - Allows for handling asynchronous behaviour
  - Useful for some APIs such as the `Fetch API`
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function
  - https://www.youtube.com/watch?v=PoRJizFvM7s

- array.forEach
  - call a function for every item in the array
  - https://www.w3schools.com/jsref/jsref_forEach.asp

JavaScript Basic Syntax

CCAPDEV