JavaScript Coding Basics

Why JavaScript?

JavaScript is a programming language that is best known for its use in web development. That being said, JavaScript does have many additional uses outside of the web development realm. However, we will be focusing on JavaScript as a tool for web development. Additionally, we will use JavaScript as an introduction to coding basics. Static web pages can be useful, but most of today's websites require user interactions. JavaScript enables developers to create dynamic web applications (websites that users can interact with). It is important for web developers to have an understanding of JavaScript because it is used in many web application frameworks.

Hello World

When learning a programming language it is common to start with a "Hello World" program. The "console.log" statement below will print the string "Hello World" to the console. The developer console can be viewed in a browser, and will display any console.log messages that have been triggered by interactions with the webpage. The console will also display error messages. Thus, "console.log" messages can be very useful in debugging web applications.

```
console.log('Hello World');
```

Grammar Basics

JavaScript code consists of a sequence of statements which are separated by semicolons (;). A semicolon is not required after each statement if they are each on a separate line. However, if multiple statements are on the same line, each statement must be followed by a semicolon. Even if a semicolon is not structurally required after a given statement, it is considered best practice to include a semicolon after every statement. Additionally, JavaScript is case-sensitive and uses the Unicode character set. This means that variables with the names "Foo" and "foo" would be considered different.

Comments

Comments can be formatted for a single line or span multiple lines, as shown below. Comments are ignored when the script is executed, and essentially provide developers a way in which they are able to add notes to their code. Commenting your code is especially important when you are working with a group of developers so that another programmer will be able to easily understand your code. That being said, you do not want it to be the case that programmers must rely on your comments to understand your code. You should provide comments for implementations or design choices that are out of the ordinary or may not be immediately understood. However, variable names, which will be discussed later, can improve readability of your code in ways that limit the need for comments.

```
// a one line comment

/* this is a longer,
 * multi-line comment
 */

/* You can't, however, /* nest comments */ SyntaxError */
```

Variables

Scope

When a variable is declared outside a function it has a global scope because it is available to any of the code within the document. On the other hand, if a variable is declared within a function, it has a local scope, because it is only available within the function. In addition, a variable can be block-scoped, meaning that it is only available within the immediate block in which it was declared.

Declaration

In JavaScript, there are three ways in which variables can be declared: var, let, and const.

Important note! Single, double, and triple equals are different

*The single equals (=) is used as an **assignment operator**. The double equals (==) is used to check for **value equality**. The triple equals (===) is used to check that the two entities being compared are of the **same type** and have the **same value**.*

```
const foo = "test"
const bar = "test"
console.log(foo == bar) //true
console.log(foo === bar) //true

const number = 1234
const stringNumber = '1234'
console.log(number == stringNumber) //true
console.log(number === stringNumber)  //false
```

var

Variables can be declared with “var” as either local or global variables. This means that variables declared with “var” will either have a global or function scope. In the example below, the variable “x” is available outside of the if-statement because it was defined with “var.” Thus it is available outside of the direct block in which it is defined (being the if-statement). Whether the declaration initializes the variable to a value is optional.

```
if (true) {
  var x = 5;
}
console.log(x);  // x is 5
```

let

Variables declared with “let” are block-scoped, local variables. This means that the variable defined will only be accessible in the immediate block in which it has been declared. In the example below, because “y” is declared via a “let” declaration it is only available within the block in which it was defined (being the if statement). Whether the declaration initializes the variable to a value is optional.

```
if (true) {
  let y = 5;
}
console.log(y);  // ReferenceError: y is not defined
```

const

Variables declared with “const” are readonly constant values. This means that once the variable is declared its value cannot be changed. Thus, it must be initialized to a value when it is declared.

```
const PI = 3.14;
```

Evaluation

When a variable is declared, but not initialized it has a value of undefined. In the boolean context, undefined will behave as False. In the numeric context, undefined variables will behave as NaN (not a number). When attempting to access variables that have not been declared a ReferenceError results.

```
var a;
console.log('The value of a is ' + a); // The value of a is undefined
console.log('The value of c is ' + c); // Uncaught ReferenceError: c is not
defined

let x;
console.log('The value of x is ' + x); // The value of x is undefined

console.log('The value of y is ' + y); // Uncaught ReferenceError: y is not
```

Naming Rules

Variable names can only start with a letter, underscore (_), or a dollar sign ($). After the first character, the name can contain alphabetic, numeric, underscore, or dollar sign characters. In addition to the strict naming rules, there are best practices that should be followed when naming variables. As a general best practice, variables should be given meaningful names that help you understand their purpose. For example, instead of naming a variable which stores the cost of an item “x,” give it the name “cost.”

Data Structures & Types

Primitives

JavaScript has seven primitive data types, which is data that is not an object. Primitives are immutable, which means they cannot be altered. A primitive variable may be reassigned a new value, but the existing value cannot be changed in the ways that objects, arrays, and functions can.

```
 // Using a string method doesn't mutate the string
var bar = "baz";
console.log(bar);               // baz
bar.toUpperCase();
console.log(bar);               // baz

// Using an array method mutates the array
var foo = [];
console.log(foo);               // []
foo.push("plugh");
console.log(foo);               // ["plugh"]

// Assignment gives the primitive a new (not a mutated) value
bar = bar.toUpperCase();       // BAZ
```

Boolean

A boolean in a logical data type that can only have the value true or false.

```
var x = true;

if (x) {
    console.log("The value is true.");
}
```

null

The null keyword indicates a null value. When null is used in a numeric context it will be given the value of zero.

```
var n = null;
console.log(n * 32); // Will log 0 to the console
```

undefined

A variable is undefined right after it has been declared if it was not initialized. The undefined property is given to variables that have not been given values.

```
var x; //create a variable but assign it no value

console.log("x's value is", x) //logs "x's value is undefined"
```

Number

The number data type is a numeric data type. Variables of this data type store either integers or floating point numbers in the range between -(2^53 − 1) and 2^53 − 1. This data type also includes three symbolic values: +Infinity, -Infinity, and NaN (not a number).

```
var x = 4;

var y = 3.14159;
```

BigInt

The BigInt data type is used to represent integers with arbitrary precision. This means that large integers (that are not within the Number range) can be stored and operated on safely. The BigInt is created by appending an “n” to the end of an integer or using the BigInt constructor.

```
 > const x = 2n ** 53n;
```

```
9007199254740992n
> const y = x + 1n;
9007199254740993n
```

String

The String data type is used to represent textual data. Each element of a string occupies a position in the String, the first element is at position zero and the last element is at position one less than the length of the string. The length of the String is the number of elements in it. In JavaScript, strings are immutable, which means once a String is created it cannot be changed.

```
var str = "Hello World";
```

Symbol

A Symbol value represents a unique identifier, which is created using the Symbol function.

```
 // Here are two symbols with the same description:
let Sym1 = Symbol("Sym")
let Sym2 = Symbol("Sym")

console.log(Sym1 === Sym2) // returns "false"
// Symbols are guaranteed to be unique.
// Even if we create many symbols with the same description,
// they are different values.
```

Objects

A JavaScript object is a variable that contains many values. The values of the object are specified as a series of name value pairs in which the name and value are separated by a colon. An object is defined with an object literal. As shown below, the spacing and line breaks are not important. Thus, object definitions can look different.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Accessing Properties

The name:value pairs of an object are called object properties and there are two ways in which they can be accessed. Below, the “firstName” property is accessed from the example person object shown above.

```
person.lastName;
```

```
person["lastName"];
```

Adding Properties

JavaScript objects can be defined as empty without any properties, and the properties can be added later.

```
const person = {};
person.firstName = "John";
```

Deleting Properties

Object properties can be deleted using the delete keyword. When an object property is deleted with the delete keyword, the property value and the property itself are both deleted. After deletion, a property cannot be used until it is added back.

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

delete person.age;
```

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

delete person["age"];
```

Object Methods

In addition to properties, objects can also have methods, which are actions that can be performed on objects. In the function definition below, the “this” keyword refers to the “owner” of the function. In this case, the owner of the function is the person object.

```
const person = {
  firstName: "John",
  lastName : "Doe",
  id       : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
```

```
// with the parentheses, the return value of the function will be
returned
name = person.fullName();
// name = "John Doe"

// without the parentheses, the function definition will be returned
name = person.fullName;
// name = function() { return this.firstName + " " + this.lastName; }
```

Nested Objects

Values in an object can be another object. The nested objects can be accessed using either the bracket notation or the dot notation.

```
myObj = {
  name:"John",
  age:30,
  cars: {
    car1:"Ford",
    car2:"BMW",
    car3:"Fiat"
  }
}

myObj.cars.car2;

myObj.cars["car2"];
```

```
myObj["cars"]["car2"];

let p1 = "cars";
let p2 = "car2";
myObj[p1][p2];
```

Arrays can also be nested in objects. Arrays will be discussed in more detail later, but below is an example of how arrays can be nested in objects and how the array elements can be accessed.

```
// defining the nested array
 const myObj = {
  name: "John",
  age: 30,
  cars: [
    {name:"Ford", models:["Fiesta", "Focus", "Mustang"]},
    {name:"BMW", models:["320", "X3", "X5"]},
    {name:"Fiat", models:["500", "Panda"]}
  ]
}

// accessing the array
for (let i in myObj.cars) {
  x += "<h1>" + myObj.cars[i].name + "</h1>";
  for (let j in myObj.cars[i].models) {
    x += myObj.cars[i].models[j];
  }
}
```

Data Type Conversion

JavaScript is a dynamically typed language, which means you do not need to specify the data type of a variable when you declare it. This also means that variables are converted as needed during the execution of the script.

```
// answer is initialized with a number
var answer = 42;
// answer is later set to a String value
answer = "Hello!";
```

Arrays

An array is a list of elements that are accessible by their indexed position.

```
// creating an array
let fruits = ['Apple', 'Banana']
// finding the length of the array
console.log(fruits.length)
// 2
```

```
// accessing the first element of the array (notice that the array is zero
indexed)
let first = fruits[0]
// Apple
// accessing the final element of the array
let last = fruits[fruits.length - 1]
// Banana
```

```
// looping over the array
fruits.forEach(function(item, index, array) {
  console.log(item, index)
})
// Apple 0
// Banana 1
```

```
// adding an element to the end of the array
let newLength = fruits.push('Orange')
// ["Apple", "Banana", "Orange"]
```

```
// removing an element from the end of the array
let last = fruits.pop() // remove Orange (from the end)
// ["Apple", "Banana"]
```

```
// removing an item from the beginning of the array
let first = fruits.shift() // remove Apple from the front
// ["Banana"]
```

```
// adding an item to the beginning of the array
let newLength = fruits.unshift('Strawberry') // add to the front
// ["Strawberry", "Banana"]
// finding the index of an item in the array
fruits.push('Mango')
// ["Strawberry", "Banana", "Mango"]

let pos = fruits.indexOf('Banana')
// 1
```

```
// removing an item by index position
let removedItem = fruits.splice(pos, 1) // this is how to remove an item
// ["Strawberry", "Mango"]
```

```
// making a copy of the array
let shallowCopy = fruits.slice() // this is how to make a copy
// ["Strawberry", "Mango"]
```

Control Flow Statements

Conditional Statements

There are two types of conditional statements supported in JavaScript: if...else and switch.

ℹ️ **What is a condition?**

The condition part of a conditional is a statement that evaluates to true or false. All individual values that are not being compared will evaluate to true as long as they are not defined as falsy values. Falsy values include false, 0, -0, 0n, “” (empty string), null, undefined, and NaN. The conditional statement can include comparisons and/or multiple statements combined with logical operators.

*Equality comparisons are achieved through the use of the double (==) or triple (===) equals. The double equals (==) is used to check for **value equality**. The triple equals (===) is used to check that the two entities being compared are of the **same type** and have the **same value**. The other comparison operators are listed below.*

- *Equal to (==)*
- *Equal value and type (===)*
- *Not equal (!=)*
- *Not equal value or not equal type (!==)*
- *Greater than (>)*
- *Less than (<)*
- *Greater than or equal to (>=)*
- *Less than or equal to (<=)*

Multiple conditions can be combined through the use of logical operators.

- *And (&&)*
- *Or (||)*
- *Not (!)*

E.g. x = 6 and y = 5

(x == 5 || y == 5) → false

if...else

Below if the conditional statement evaluates to true, statement_1 is executed. Otherwise, statement_2 is executed.

```
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

Multiple conditions can be added using else if. In the example below, the statement which corresponds to the first conditional which evaluates to true will be the only statement that is executed.

```
if (condition_1) {
  statement_1;
} else if (condition_2) {
  statement_2;
} else if (condition_n) {
  statement_n;
} else {
  statement_last;
}
```

Switch Statement

A switch statement evaluates the value of an expression and attempts to match the result to a series of case labels. If a match is found, the corresponding statement is executed. Otherwise, the default statement is executed. The option break statement will ensure that the program breaks out of the switch statement after the statement is executed. If the break statement is not included then the execution of the switch statement will continue.

```
switch (expression) {
  case label_1:
    statements_1
    [break;]
  case label_2:
    statements_2
    [break;]
    ...
  default:
    statements_def
    [break;]
}
```

```
switch (fruittype) {
  case 'Oranges':
    console.log('Oranges are $0.59 a pound.');
    break;
  case 'Apples':
    console.log('Apples are $0.32 a pound.');
    break;
  case 'Bananas':
    console.log('Bananas are $0.48 a pound.');
    break;
  case 'Cherries':
    console.log('Cherries are $3.00 a pound.');
    break;
  case 'Mangoes':
    console.log('Mangoes are $0.56 a pound.');
    break;
  case 'Papayas':
    console.log('Mangoes and papayas are $2.79 a pound.');
    break;
  default:
   console.log(`Sorry, we are out of ${fruittype}.`);
}
console.log("Is there anything else you'd like?");
```

Loops and Iteration

Loops allow you to perform operations repeatedly. There are a few different kinds of loops. In this section we will cover for, while, and do…while statements.

for Statement

A for loop will repeat until a specified condition evaluates to false.

```
for ([initialExpression]; [conditionExpression]; [incrementExpression])
  statement
```

```
let sum = 0;
for (let i = 0; i < 6; i++) {
    sum = sum + i;
}
```

for Loop Steps:

1. Execute the initial expression, which is often used to initialize loop counter variables.
2. Evaluate the conditional expression.
    a. If the condition is true, the for loop statements execute. To execute multiple statements, they must be contained in a block ({}). If the condition expression is omitted, it defaults to true.
    b. If the condition is false, the for loop terminates.
3. Execute the increment expression if it is included.
4. Return to step 2.

while Statement

A while statement will execute its statements as long as the condition evaluates to true. When the statement becomes false, the program will pass over the statements within the loop and move to the next statement following the loop. Again, if multiple statements are included in the loop they must be placed in a block ({}).

```
while (condition)
  statement
```

```
let n = 0;
let x = 0;
while (n < 3) {
  n++;
  x += n;
}
```

do…while Statement

A do…while statement will execute until the given condition evaluates to false. The statement is always executed once before the condition is checked. Again, if you would like to include multiple statements in the loop, they must be grouped in a block statement ({}).

```
do
  statement
while (condition);
```

```
let i = 0;
do {
  i += 1;
  console.log(i);
} while (i < 5);
```

Project

- https://www.learn-js.org
  - Contains basic exercises, tutorials, and web-based IDE, definitely a day or multi-day activity

Sources

JavaScript uses: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript
Introduction to JavaScript:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction
Hello world: https://www.geeksforgeeks.org/javascript-course-printing-hello-world-in-javascript/
Developer console:
https://support.airtable.com/hc/en-us/articles/232313848-How-to-open-the-developer-console
BigInt constructor:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt/BigInt
JavaScript Objects: https://www.w3schools.com/js/js_objects.asp
Object Properties: https://www.w3schools.com/js/js_object_properties.asp

Double vs. Triple Equals
https://www.freecodecamp.org/news/javascript-triple-equals-sign-vs-double-equals-sign-comparison-operators-explained-with-examples/#:~:text=Double%20Equals%20(%20%3D%3D%20)%20checks%20for,does%20not%20perform%20type%20coercion.
Condition
https://en.wikiversity.org/wiki/Programming_Fundamentals/Conditions#:~:text=Conditions%20are%20statements%20that%20are,if%20it's%20true%20or%20false.
Truthy values https://developer.mozilla.org/en-US/docs/Glossary/Truthy
Comparisons and logical operators https://www.w3schools.com/js/js_comparisons.asp