Front-end Web Development

March 3rd 2020

Dorset College Dublin

Revision

How JS makes web pages interactive

Access Content - select elements, attributes or text
Modify Content -add elements, attributes or text
Program Rules - specify a set of steps the browser should follow
React to Events

Rules for Naming Variables

- The name must begin with a letter, dollar sign($), or and underscore(_). It must **not start** with a number.
- The name can contain letters, numbers, dollar sign($) or an underscore(_). **Do not use** a dash(-) or a dot (.) in a variable name.
- You cannot use **keywords** or **reserved** words.
- All variables are case sensitive and should be named appropriately
- Name the variable in a way that describes the information it holds
- Use a capital letter if there are multiple words in variable names

JavaScript

- JS code can be **referenced** in HTML or **embedded** directly into it
- JS is **dynamically typed**(loosely typed) data-types are bound to **values** not variables

Primitive Data Types

Numeric - numbers and decimals
String - letters and characters written in single/double quotes
Boolean - true or false

var, let and const

var - defines a global variable regardless of the block scope
let - declares a **block scope local** variable
const - Block scoped similar to let however the value **cannot be changed** or redeclared.

Control Flow

The order in which the computer executes statements in a script.

- All conditions are evaluated to be **truthy or falsy**
- We can use an else if to add more conditional statements to if/else statements
- Switch statements can be used to achieve the same result as if/else statements

Ternary Operator

```
isLingtOn ? console.log('Turn on the lights!') : console.log('Turn off the lights!');
```

Falsey Values

- false
- 0 and -0
- “” and ‘’ (empty strings)
- null
- undefined
- NaN (Not a Number)
- document.all(something you will rarely encounter)

Relational Expressions

- The === operator is known as a **strict equality** operator (identity operator) which checks if two operands are “identical”.
- The == operator is known as the **equality operator** which checks if two operands are “equal” using less strict definition of sameness. It allows type conversions.

Variable Hoisting

Hoisting is JS default behavior of **moving declarations** to the top.

- A variable (var) can be used before it has been declared
- The variable declaration is said to have been hoisted.
- var variable declarations are **hoisted to the top** of the current script.

not defined, undefined and null

- not defined variables don't exist
- undefined variables exist but are not assigned and hold no value
- null variables exist and have null assigned

Arrow Functions (ES6)

The process of **refactoring code** without changing its external behaviour to make it more readable, maintainable and extensible.

```
const multiply = x => x * x;
```

```
const isLesserThan = function(numOne, numTwo){
    if(numOne > numTwo){
        return true;
    } else {
        return false;
    }
};
```

```
const isLesserThan = (numOne, numTwo) => numOne > numTwo;
```

Switch statements

A switch statement is usually **more efficient** than a set of nested ifs.

- Not as much syntax with braces and parentheses
- Creates **clarity** and readability
- Inherits C's syntax which means you have you have to use **break** to separate each single scope for variables

```
let moonPhase = 'full';

switch (moonPhase){
    case 'full':
        console.log('Howl!');
        break;
    case 'mostly full':
        console.log('Arms and legs are getting hairier');
        break;
    case 'mostly new':
        console.log('Back on two feet');
        break;
    default:
        console.log('Invalid moon phase');
    break;
}
```

Functions

A function is a block of JS code that is defined once but may be executed or invoked any number of times.

Functions pass **parameters** and use **arguments** which are passed to the function.

Example:

Return

Sends a value back from the function to where it was invoked

- There are 2 times you'll want to use return in a function:
- When you literally want to **return a value**
- When you want the function to stop running
- If we return a value inside a function, it can be used anywhere else in the code

```
// Declare a function that takes on parameter
function multiplyNumber(inputNumber){
    return inputNumber * 4;
}

// Pass 10 as an argument to the multiplyNumber function
console.log( multiplyNumber(10) );
```

Function Declaration

A function declaration is a function that is bound to an **identifier** or name.

```
function calculateTaxRate(cost) {
    const taxRate = .23;
    return cost * taxRate;
}

costOfLaptop = 540;

console.log(calculateTaxRate(costOfLaptop));
```

Function Expression

Similar to a function declaration, with the exception that the identifier can be **omitted** which creates an anonymous function. Often **stored** in a variable.

```
const calculateTaxRate = function (cost) {
    const taxRate = .23;
    return cost * taxRate;
};

let costOfLaptop = 540;

console.log(calculateTaxRate(costOfLaptop));
```

Function Hoisting

Function declarations are “hoisted” to the top of the enclosing script or enclosing function.

Function Expressions however **cannot be hoisted**.

Optional Parameters

If a function is called with **missing arguments**(less than declared), the missing values are set to undefined

```
function connect(hostname, port, method){
    hostname = hostname || "localhost";
    port = port || 80;
    method = method || "HTTP";
}
```

Default Parameters (ES6)

```
function multiply(a, b = 1){
    return a * b;
}

multiply(5, 2);
multiply(5, 1);
multiply(5);
```

Rest Parameters (ES6)

Introduced to ES6 to clean up variable-length argument list work.

- Rest parameters are the only ones that haven’t been given a separate name
- Rest parameters are **real arrays**

Rest parameters example

```
function pizzaBuilder(base, ...toppings){
    console.log('Number of toppings: ' + toppings.length);
    console.log(`You ordered a ${base} based pizza with the
    following toppings: ${toppings}`);
}

pizzaBuilder('thin', 'mushroom', 'pepperoni', 'peppers');
```

Rest (Spread) Operator

Allows an array to be **expanded** in places where zero or more arguments (in function calls) or elements (in array literals) are expected.

```
let arr = [1,2,3];

example(...arr);

function example(var1, var2, var3){
    console.log(var1);
    console.log(var2);
    console.log(var3);
}
```

Arrays

An array is an ordered collection of values where each value is called an element.

- JS arrays are **untyped:** can be of any type (string, numerical, boolean)
- Array elements can be objects or other arrays
- JS arrays are always **zero-based:** index of the first element is 0

Array Literal

```
var colors;
colors = ['red', 'orange',
'yellow', 'green', 'blue',
'indigo', 'violet'];
```

Array Constructor

```
Let a = new Array('red',
'orange', 'yellow', 'green',
'blue', 'indigo', 'violet');
```

length

The length property tracks the **highest index** in an array and not the number of elements.You can use it to **add or delete** array elements.

Objects

Objects group together a **set of variables and functions** to create a model. Variables and names in an object take on a few names.

- If a variable is part of an object, it is called a **property** of that object
- Properties are the **characteristics** of an object
- If a function is part of an object, it is called a **method**
- Methods represent the tasks that are associated with an object
- Properties and methods have a **name (key)** and a **value**

Object Literal Notation

```
var hotel = {
    name: 'Clayton',
    rooms: 110,
    booked: 25,
    gym: true,
    roomTypes: ['double', 'twin', 'suite', 'king'],
    checkAvailablity: function() {
        return this.rooms - this.booked;
    }
};
```

Object Constructor Notation

```
var hotel = new Object();

hotel.name = 'Calyton';
hotel.rooms = 45;
hotel.booked = 24;
hotel.checkAvailablity = function(){
    return this.rooms - this.booked;
};
```

Many Objects

- Object constructors can use a function as a **template** for creating objects
- You create **instances** of the object using the constructor function
- The new keyword followed by a call to the function creates a **new object**
- The properties of each object are given as **arguments** to the function

```
function Hotel(name, rooms, booked){
    this.name = name;
    this.rooms = rooms;
    this.booked = booked;
    this.checkAvailablity = function(){
        return this.rooms - this.booked;
    };
}

var clyatonHotel = new Hotel('Clayton', 128, 25);
var lemonHotel = new Hotel('Lemon', 76, 14)
```

this

This keyword is commonly used inside functions and objects. Depending on where the function is declared, the meaning of this changes. Always **refers to one object**.

- A function declared at the top level of a script (not inside an object or a function)is in the global scope/context
- The this keyword is a reference to the object that the function is created inside

Good Luck!