

JS

JavaScript

Fundamentals

Week Summary

Day 1	Core: History, Data Types, DOM, Variables, Functions, Arrays
Day 2	Core ; Mini Projects, Design
Day 3	Objects
Day 4	Objects, JSON, XML, AJAX
Day 5	Problem Day, Retrospective, next steps

JS

JavaScript

Language Core

Core

- Language basics
- Console applications
- Variables, constants, data types
- Selection structures – if-else, switch
- Iteration structures (loops) – while, do-while, for, for each, recursion
- Data structures – arrays, Objects
- Methods
- Error handling
- Algorithms

Objectives

- Candidates should be able to
 - write simple applications in Javascript
 - write a program that uses an if-else statement
 - write a program that uses a loop
 - write a program that uses an array
 - explain what an algorithm is and give an example

Coding good practises



- **Code – lots!**
 - “The only way to learn a new programming language is by writing programs in it.” - Dennis Ritchie, creator of C and Unix
- Read important material at least twice
 - Coding concepts are complex and most people don’t understand them the first time
- Collect working examples of code related to key concepts
 - Play with the code and try out ways you might use it
- Don’t try to code all night
 - Working on screens at night disrupts sleep
 - Evidence at: <https://justgetflux.com/news/2014/12/21/advice.html>
- Ask for help if you need it
 - Tutors will help whenever you need them
 - Excellent programming support websites include: StackOverflow, Code Project
 - Tutorial and reference websites include: Microsoft Developer Network
- Have faith in yourself!
 - Everyone struggles with coding at first, but practice and persistence lead to progress

**What do you already
know about Javascript?**

Background, history and development

JavaScript was invented by Brendan Eich in 1995 (in 10 days!) and became an ECMA standard in 1997.

ECMA-262 is the official name.

ECMAScript 6 (released in June 2015) is the latest official version of JavaScript and is Object Orientated



https://en.wikipedia.org/wiki/Brendan_Eich

- JavaScript is a **programming language**
- JavaScript is **object-based**, Javascript EM6 is object orientated
- JavaScript is **event-driven**
- JavaScript is **platform-independent**
- Javascript is a **Front End** language
- Javascript is a **loosely typed** programming language
- Javascript is an **interpreted language**

What is Javascript?

JavaScript is the programming language of HTML, CSS and the Web.

Javascript is parsed on the client

Javascript is an interpreted programming language

JavaScript is the **most popular** programming language in the world.

JavaScript and Java are completely different languages, both in concept and design.

JavaScript is one of the **3 languages** all **web developers** should learn:

1. **HTML** to define the content of web pages
2. **CSS** to specify the layout of web pages
3. **JavaScript** to program the behaviour of web pages

Tools and Resources

Plunker

<http://plnkr.co/>

Brackets

<http://brackets.io/>

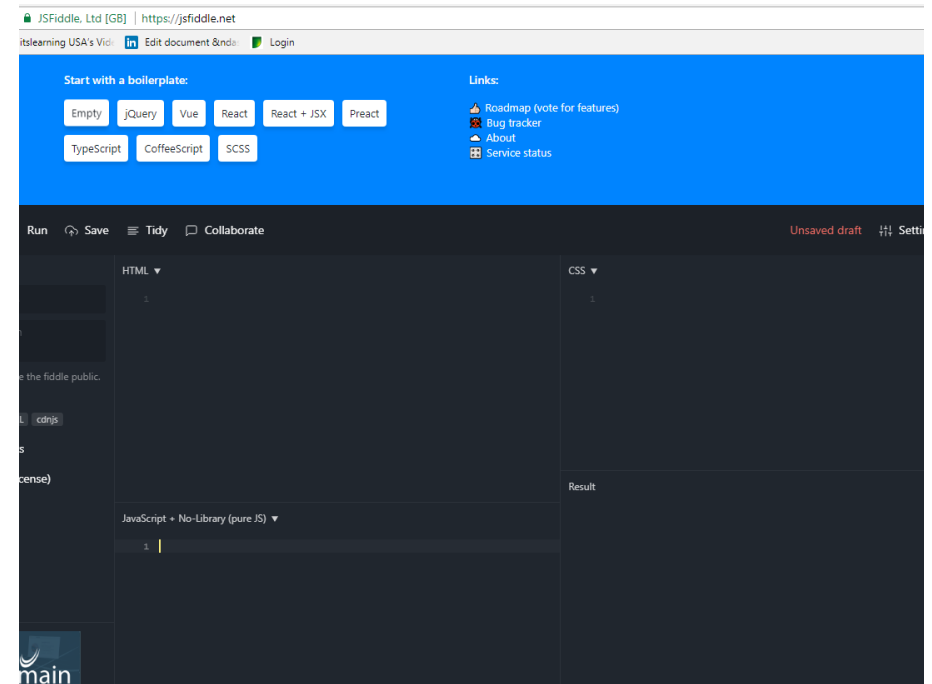
also install Beautify

How to use Brackets

<https://github.com/adobe/brackets/wiki/How-to-Use-Brackets>

<http://www.w3schools.com/js/default.asp>

<https://jsfiddle.net/>



Tips when learning Javascript

Don't Use Short-Hand and Always, Always Use Semicolons!

Technically, you can get away with omitting most curly braces and semi-colons but it can lead to lots of unexpected errors.

Comment Your Code

Get in the habit to comment your code – it also helps you understand what you have done when you return to it at a later date

Test small piece of code to check it works as expected

```
console.log(); window.alert(); Test Test Test
```

Use the right tool for the job

"A common error in JavaScript programs is to use an object when an array is required or an array when an object is required. The rule is simple: when the property names are small sequential integers, you should use an array. Otherwise, use an object.

Use techniques and code that reduce errors

There are many ways of solving the same problem in Javascript. Using techniques and code that reduce your error rate. Learn to learn the good Javascript and avoid the bad Javascript. eg. == and ===

Place scripts at the bottom of the webpage for better performance

The primary goal is to make the webpage load as quickly as possible for the user. When loading a script, the browser can't continue on until the entire file has been loaded. Thus, the user will have to wait longer before noticing any progress. If you have JS files whose only purpose is to add functionality -- for example, after a button is clicked -- go ahead and place those files at the bottom, just before the closing body tag. This is absolutely a best practice.

There will be bugs

Reduce your error rate by coding well. Learn and use features of the language that reduce error rate. Bad parts of Javascript are not useless – they are dangerous!

Separate Javascript from HTML and CSS

It is good practice to separate content structure (HTML), presentation (CSS) and behaviours (Javascript). Keeping separation produces code that is easier to **maintain and reuse**.

Spend time on Design

It is well worth spending time designing your solutions before implementation as your Javascript logic increases

HTML DOM

HTML DOM

What is the HTML DOM?

The HTML DOM is a standard **object** model and **programming interface** for HTML.

It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words:

The HTML DOM is a standard for how to get, change, add, or delete HTML elements.

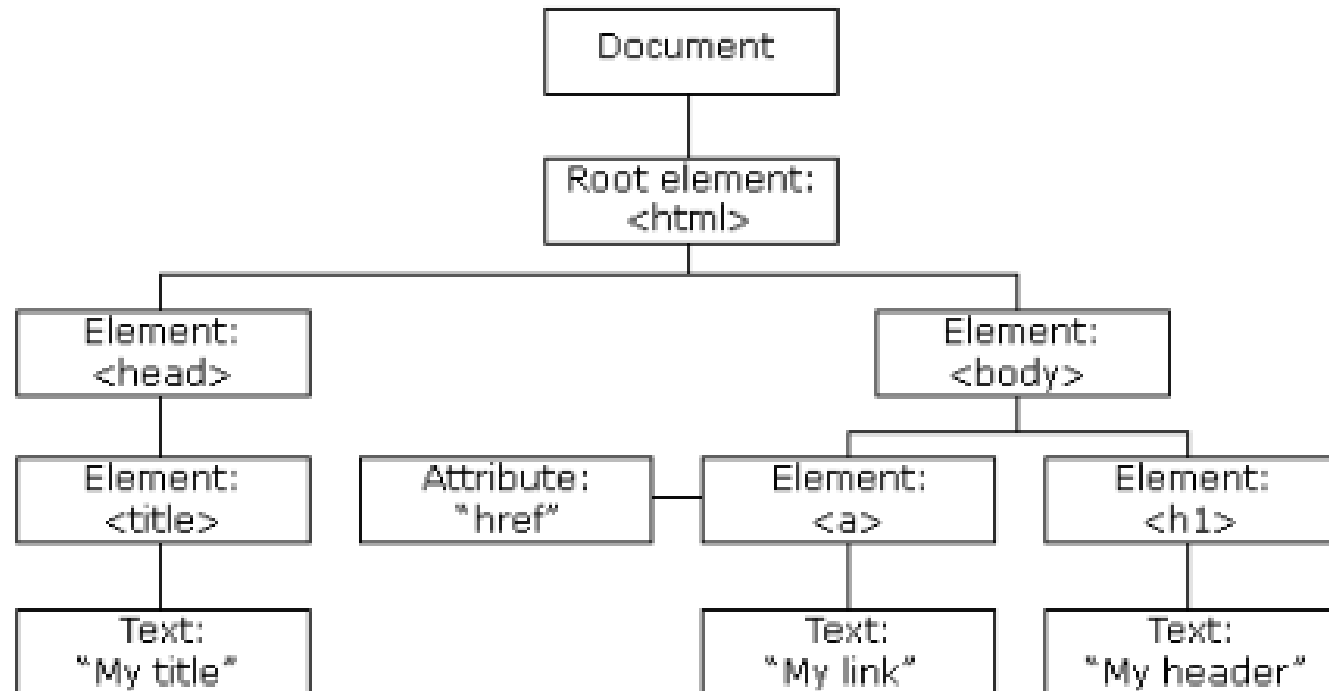
When a web page is loaded, the browser creates a **Document Object Model** of the page.

The HTML Document Object Model (DOM) is the browser's view of an HTML page as an object hierarchy, starting with the browser window itself and moving deeper into the page, including all of the elements on the page and their attributes..

The **HTML DOM** model is constructed as a tree of **Objects**.

The top-level object is **window**. The **document** object is a child of **window** and all the objects (i.e, elements or nodes) that appear on the page (e.g, forms, links, images, tables, etc.) are descendants of the **document** object. These objects can have children of their own. For example, **form** objects generally have several child objects, including textboxes, radio buttons, and select menus.

The HTML DOM Tree of Objects

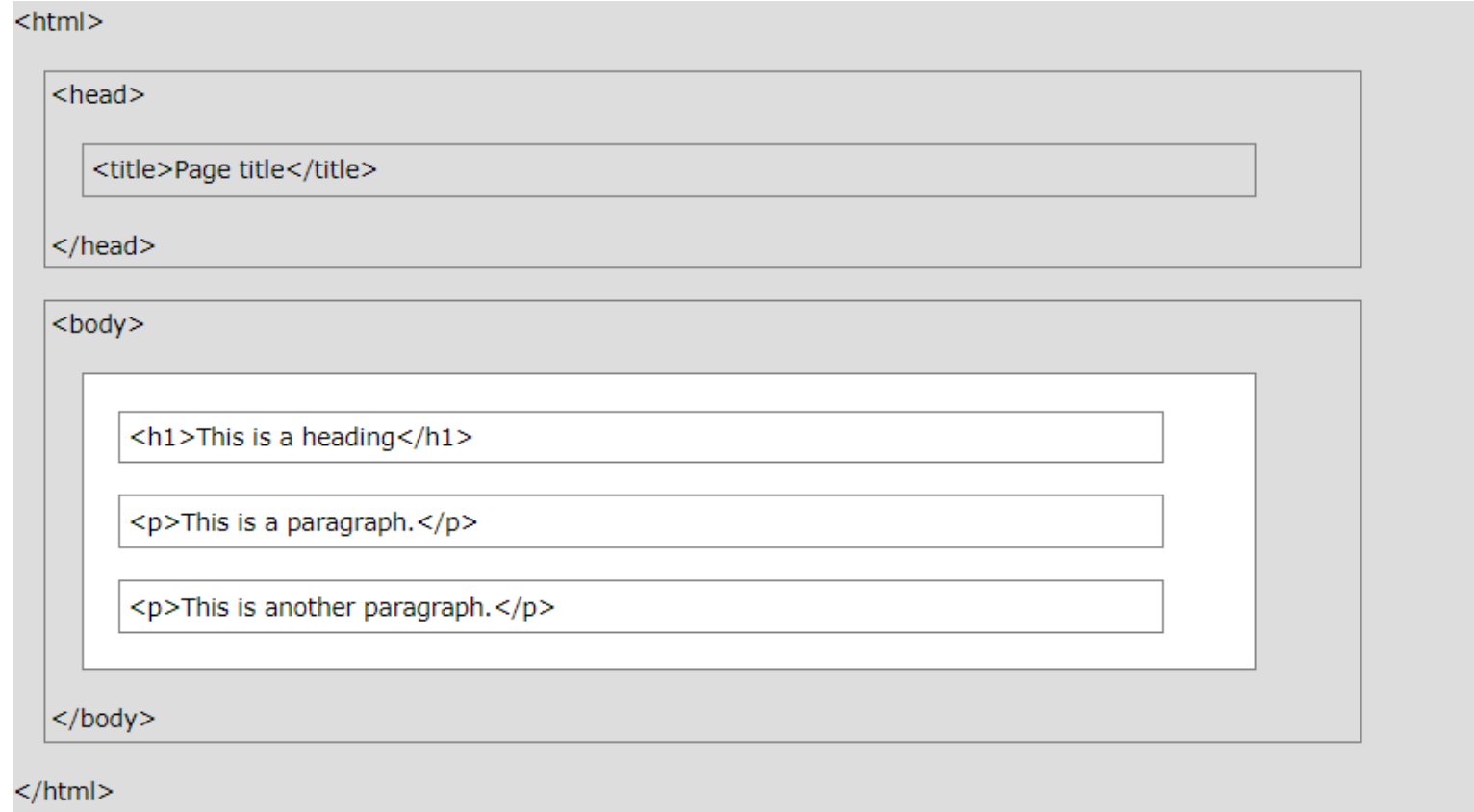


The Basic HTML Page

```
<!DOCTYPE html>
<html>
<head>
<title>Page Title</title>
</head>
<body>

<h1>My First Heading</h1>
<p>My first paragraph.</p>
<p>This is another paragraph.</p>

</body>
</html>
```



The `<!DOCTYPE html>` declaration defines this document to be HTML5
The `<html>` element is the root element of an HTML page
The `<head>` element contains meta information about the document
The `<title>` element specifies a title for the document
The `<body>` element contains the visible page content
The `<h1>` element defines a large heading
The `<p>` element defines a paragraph

HTML Events

HTML Events are "things" that happen to **HTML elements**.

Event Handlers are attributes that force an element to “listen” for a specific event to occur
Event handlers all begin with the letters **"on"**

When JavaScript is used in HTML pages, JavaScript can **"react"** on these events.

An HTML event can be something the browser does, or something a user does.

Here are some examples of HTML events:

An HTML web page has finished loading

An HTML input field is changed

An HTML button is clicked

Often, when events happen, you may want to do something.

JavaScript lets you execute code when events are detected.

Here is a list of some common HTML events:

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

Using Javascript **Event listeners** are considered the best technique to use in **modern code**.

Event listeners keep the Javascript cleanly separate from the HTML

User Events and JavaScript Event Handlers

DOM objects

- *button*
- *reset*
- *submit*
- *radio*
- *checkbox*
- *link*
- *form*
- *text*
- *textarea*
- *select*
- *image*
- *area*
- *window*

User Events

- *abort*
- *blur*
- *click*
- *change*
- *error*
- *focus*
- *load*
- *mouseover*
- *mouseout*
- *reset*
- *select*
- *Submit*
- *unLoad*

Event Handlers

- *onabort*
- *onblur*
- *onclick*
- *onchange*
- *onerror*
- *onfocus*
- *onload*
- *onmouseover*
- *onmouseout*
- *onreset*
- *onselect*
- *onsubmit*
- *onunload*

JS

Language Basics

JS

Data Types

Primitive Data Types

<u>String</u>	A string (or a text string) is a series of characters e.g. "Hello"
<u>Number</u>	represents numeric values e.g. 100
<u>Boolean</u>	Booleans can only have two values: true or false.
Undefined	represents undefined value
Null	represents null i.e. no value at all

Non Primitive Data Types

<u>Object</u>	<code>var cars = ["Saab", "Volvo", "BMW"];</code> represents instance through which we can access members
<u>Array</u>	represents group of similar values
<u>RegExp</u>	represents regular expression

Immutable

Primitive values are immutable (they are hardcoded and therefore cannot be changed).

if x = 3.14, you can change the value of x. But you cannot change the value of 3.14

Value	Type	Comment
"Hello"	string	"Hello" is always "Hello"
3.14	number	3.14 is always 3.14
true	boolean	true is always true
false	boolean	false is always false
null	null (object)	null is always null
undefined	undefined	undefined is always undefined

Variables

Variables are dynamic in Javascript

A dynamic variable is a variable whose address is determined when the program is run. In contrast, a static variable has memory reserved for it at **compilation** time.

All JavaScript **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

$$\text{nameOfVariable} = \text{value}$$

The **undefined** property indicates that a variable has not been assigned a value, or not declared at all.

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Variables</h2>

<p>Strings are written with quotes.</p>
<p>Numbers are written without quotes.</p>

<p id="demo"> </p>

<script>
var pi = 3.14;
var person = "John Doe";
var answer = 'Yes I am!';

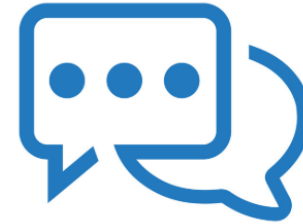
document.getElementById("demo").innerHTML =
pi + "<br>" + person + "<br>" + answer;
</script>

</body>
</html>
```

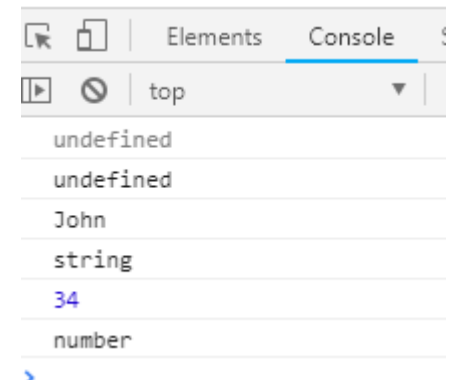
```
<!DOCTYPE html>
<html>
<head></head>
<body>
<h2>JavaScript</h2>
```

```
<script>
  var x; // Now x is undefined x = 5;
  console.log(x);
  console.log(typeof(x));
  x = "John"; // Now x is a String
  console.log(x);
  console.log(typeof(x));
  var x = 34.00; // Written with decimals var x2 = 34;    // Written without decimals
  console.log(x);
  console.log(typeof(x));
</script>
```

```
</body>
</html>
```



console.log
typeof()
undefined
string primitive
number primitive
F12



Operators

Assignment

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Comparison

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Arithmetic

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (Remainder)
++	Increment
--	Decrement

Selection

Conditional statements are used to perform different actions based on different conditions.

If Else

```
if (condition) {  
    block of code to be executed  
    if the condition is true  
}
```

```
var greeting;  
var time = new Date().getHours();  
if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

2 operators to test a condition

if (time > 5 && time < 10)

```
document.getElementById("demo").innerHTML = greeting;
```

Switch

```
switch(expression) {  
    case x:  
        code block  
        break;  
    case y:  
        code block  
        break;  
    default:  
        code block  
}
```

```
<!DOCTYPE html>
<html>
<body>
  <p id="demo"></p>
```

```
<script>
  window.addEventListener('load', age, false);

  function age() {
    var ageAsString = window.prompt("What is your age? ", "");
    var age = Number(ageAsString);
    alert(age);
    if (age < 40) {
      alert("Oh you're young...");
      console.log("Oh you are so young...");
      document.getElementById("demo").innerHTML = "Oh you are so young";
```

```
    } else if (age === 40) {
      alert("Hey, you're 40!");33
      console.log("Hey, you're 40!");40
      document.getElementById("demo").innerHTML = "Hey, you're 40!";
```

```
    } else {
      alert("Don't worry, you're young at heart");
      console.log("Don't worry, you're young at heart");
      document.getElementById("demo").innerHTML = "Don't worry, you're young at heart";
```

```
  }
</script>
</body>
</html>
```

If Else



selection If else
event listener
function

operator ===

document.getElementById

alert

end of statements ;

What is interpreted on page load

Position of script on web page

```
<!DOCTYPE html>
<html>
<body>
<p id="demo"></p>
```

```
<script>
var day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case 6:
    day = "Saturday";
  }
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
```

```
</body>
</html>
```

Switch



Selection
Switch
break - is this good programming?
Date()
DOM write



Iteration

Iteration/Loop logic can execute a block of code a number of times.

For

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

```
var i;  
for (i = 0; i < cars.length; i++) {  
    text += cars[i] + "<br>";  
}
```

Do

While

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

```
do {  
    text += "The number is " + i;  
    i++;  
}  
while (i < 10);
```

While

Do

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

```
while (i < 10) {  
    text += "The number is " + i;  
    i++;  
}
```

For loops are commonly used to do some work to the elements in an array

Var numbers = 1, 2, 3, 4];

Control
variable
initialised

Length of array used to control for loop

Loop continues
while this is
true

```
for (int i = 0; i < numbers.length; i++)  
{  
    total += numbers[i];  
}
```

Control
variable
increment
ed/decre
ment

Loop control variable used as array
index – makes the loop work through
the array

Concatenation operator often useful for calculating totals.
This line means the same as: total = total + numbers[i];

Iteration

```
<!DOCTYPE html>
<html>
  <head> </head>
<body>
<h2>JavaScript For Loop</h2>
<input type="text" id="input1"/>
<div><button id="button1">Button</button> </div>
<div id="demo"></div>
```

```
<script>
var event1 = document.getElementById("button1");
event1.addEventListener('click',forloop,false);
function forloop()
{
  var age = document.getElementById('input1').value;
  alert(age);
  var string = "";
  for ( var theAge = age; theAge > 0; theAge -= 1)
  {
    string += "Happy Birthday\n";
  }
  alert(string);
  document.getElementById("demo").innerHTML = string ;
}
</script>
</body>
</html>
```



For loop
\n
Get from DOM
Write to DOM
Event listener
Function

Iteration - while

- When you want a piece of code to **run 0 or more times**
- **Stop condition is at the top**
 - When it's met, the loop stops running
 - If it's met before the loop has been run, then the code in the loop won't be run at all
- A while loop must contain some **code that will eventually meet the stop condition by changing the value of the variable in the stop condition**, e.g.
 - Increasing a counter
 - Getting user input
 - Otherwise, you get an endless loop
 - Ctl+C breaks out of an endless loop in the console

```
int numberOfTimes = 0;
while (numberOfTimes < 10)
{
    Console.WriteLine(" numberOfTimes is : " + numberOfTimes );
    numberOfTimes ++;
}
```

```
bool happiness = true;
while (happiness)
{
    EatSomeChocolate();
    happiness = CheckIfTheresAnyChocolateLeft();
}
```

```
bool happiness = true;
while (happiness)
{
    EatSomeChocolate();
    moreChocolate = CheckIfTheresAnyChocolateLeft();
}
```

JS

Practise
Variables, Selection,
Iteration, Expressions

```
<!DOCTYPE html>
```

```
<html>
```

```
<head>
```

```
<script>
```

```
window.alert("Hello World");
```

```
alert("Hello World");
```

```
</script>
```

```
</head>
```

```
<body onload="myfirst()"> BAD PRACTICE inline javascript
```

```
<p id="demo">message</p>
```

```
<script>
```

```
window.alert("Hello World");
```

```
alert("Hello World");
```

```
</script>
```

```
</body>
```

```
</html>
```

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
<span id="demo"></span>
<script>
window.addEventListener('load',myfirst,false);
function myfirst(){
window.alert("Hello World");
document.getElementById("demo").innerHTML = "Hello World";
}
</script>
</body>
</html>
```

```
<!DOCTYPE html>  
<html>  
<head></head>  
<body>
```

```
<input type="text" id="input1"/>  
<div><button id="button1">Button</button> </div>  
<div id="demo">Test</div>
```

```
<script>  
//EVENT LISTENER  
var event1 = document.getElementById("button1");  
event1.addEventListener('click',myfirst,false);  
  
function myfirst(){  
//User input 1  
var userinput1 = document.getElementById("input1").value;  
  
//Output 1  
document.getElementById("demo").innerHTML = userinput1;  
}  
</script>  
</body>  
</html>
```

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<p>Please enter 2 numbers below</p>
<input type="number" id="input1"/>
<input type="number" id="input2"/>
<div><button id="button1">Sum of the two numbers</button> </div>
<div id="demo"></div>
```

```
<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',myfirst,false);

function myfirst(){
//User input 1 convert in JS to a number
var userInput1 = parseInt(document.getElementById("input1").value);
//alert(userinput1);

//User input 2 convert in JS to a number
var userInput2 = parseInt(document.getElementById("input2").value);
//alert(userinput2);

//Output 1
document.getElementById("demo").innerHTML = userInput1 + userInput2;
}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<p>Please enter 2 numbers below</p>
<form>
<input type="number" id="input1"/>
<input type="number" id="input2"/>
</form>
<div><button id="button1">Sum of the two numbers</button> </div>
<div id="demo"></div>
```

```
<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',myfirst,false);

function myfirst(){
//User input 1 convert in JS to a number
var userInput1 = parseInt(document.getElementById("input1").value);
//alert(userinput1);

//User input 2 convert in JS to a number
var userInput2 = parseInt(document.getElementById("input2").value);
//alert(userinput2);

if (userinput1 > userinput2){
//Output 1
document.getElementById("demo").innerHTML = userInput1;
} else {
//Output 2
document.getElementById("demo").innerHTML = userInput2;

}

}
</script>
```

```
</body>
</html>
```



```
<!DOCTYPE html>
<html>
<head></head>
<body>
<p>Please enter 2 numbers below</p>

<input type="number" id="input1"/>
<input type="number" id="input2"/>
<div><button id="button1">Sum of the two numbers</button> </div>
<div id="demo"></div>
```

```
<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',myfirst,false);

function myfirst(){
//User input 1 convert in JS to a number
var userInput1 = parseInt(document.getElementById("input1").value);
//alert(userinput1);

//User input 2 convert in JS to a number
var userInput2 = parseInt(document.getElementById("input2").value);
//alert(userinput2);

if (userinput1 === userInput2){
//Output 1

document.getElementById("demo").innerHTML = "number is same";

} else {
//Output 2
document.getElementById("demo").innerHTML = "number is different";
}

}
</script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<p>Please enter 2 numbers below</p>
<form>
<input type="number" id="input1"/>

</form>
<div><button id="button1">Sum of the two numbers</button> </div>
<div id="output1"></div>
<div id="output2"></div>
```

```
<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',myfirst,false);

function myfirst(){
//User input 1 convert in JS to a number
var userinput1 = parseInt(document.getElementById("input1").value);
//alert(userinput1);

document.getElementById("output1").innerHTML = userinput1 + 1;

document.getElementById("output2").innerHTML = userinput1 -1;

}
</script>
```

```
</body>
</html>
```



```
<!DOCTYPE html>
<html>
<head></head>
<body>
<p>Please enter 2 numbers below</p>
<form>
<input type="number" id="input1"/>

</form>
<div><button id="button1">Number</button> </div>
<div id="demo"></div>
```

```
<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',myfirst,false);

function myfirst(){
//User input 1 convert in JS to a number
var userInput1 = parseInt(document.getElementById("input1").value);

if (userinput1 > 10){
document.getElementById("demo").innerHTML = "number is > 10";
alert(userinput1);
} else if (userinput1 > 0) {
document.getElementById("demo").innerHTML = "number is less than 10";
}
else {
document.getElementById("demo").innerHTML = "number is negative";
}

}
</script>
```

```
</body>
</html>
```

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<p>Please enter 2 numbers below</p>
<form>
<input type="number" id="input1"/>
<input type="number" id="input2"/>

</form>
<div><button id="button1">Number</button> </div>
<div id="demo"></div>
```

```
<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',myfirst,false);

function myfirst(){
//User input 1 convert in JS to a number
var userInput1 = parseInt(document.getElementById("input1").value);

//User input 1 convert in JS to a number
var userInput2 = parseInt(document.getElementById("input2").value);

if (userinput1 > 10 && userinput2 > 10){
document.getElementById("demo").innerHTML = "both numbers are > 10";
} else if (userinput1 > 10) {
document.getElementById("demo").innerHTML = "the first number is > than 10";
}
else if (userinput2 > 10){
document.getElementById("demo").innerHTML = "the second number is > 10";
}
else {
document.getElementById("demo").innerHTML = "neither number is greater than 10";
}

}
</script>

</body>
</html>
```

JS

Coding Standards

Coding standards

Coding conventions are **style guidelines for programming**.

They typically cover:

Naming and declaration rules for variables and functions.

Rules for the use of white space, indentation, and comments.

Programming practices and principles

Coding conventions promote QUALITY

Improves code readability

Makes code maintenance easier

Coding conventions can be documented rules for teams to follow, or just be your individual coding practice.

- Use **camelCase** for identifier names (variables and functions)
- Use meaningful (semantic) names for variables
- All names start with a **letter**
- Put spaces around operators (= + - * /), and after commas
- Use 4 spaces for indentation of code blocks
- End all statements with a semicolon
- For readability, avoid lines longer than 80 characters.

Reserved Words

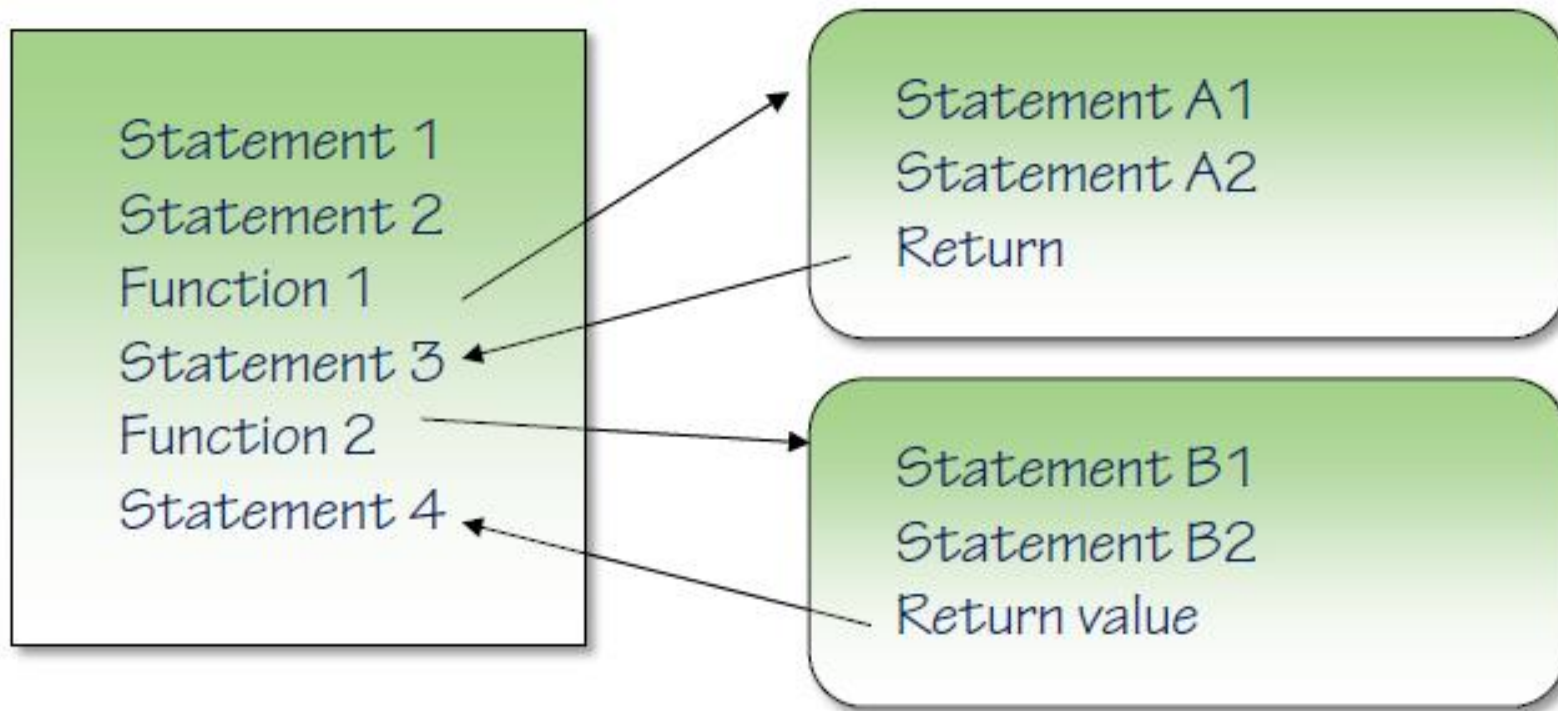
In JavaScript you cannot use these reserved words as variables, labels, or function names:

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

JS

Functions

Functions



Naming a function

Anonymous Functions

Parameters Too many? Ignored

Too few? Undefined

Return and Returning a value No return value?

Undefined

Hoisting

Simple Function that pass in 2 parameters and returns the sum

```
<!DOCTYPE html>
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>Function</title>
</head>
<body>

<h1>Sum</h1>
<h2 id="demo"></h2>

<!-- <script src="tech1.js" type="text/javascript"></script> -->
<script>
var sum = add(5,6);
function add(x, y)
{
var z = x + y;
return z;
}
document.getElementById("demo").innerHTML = sum;
</script>

</body>
</html>
```

JS

Functions

Program Flow

2.1

```
<!DOCTYPE html>
<html>
<body>
  <p>Hello World:</p>
  <p id="demo"></p>

  <script>
function hw() {
  alert("Hello World");
  return;
}
hw();
  </script>

</body>
</html>
```

```
<!DOCTYPE html>
<html>
<body>
<p>This example calls a function which performs a calculation,
and returns the result:</p>
<p id="demo"></p>
  <script>
function myFunction(a, b) {
  return a * b;
}
document.getElementById("demo").innerHTML =
myFunction(4, 3);
  </script>
</body>
</html>
```

2.2

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title> Interest Calculator</title>
</head>
<body>
  <h1>Sum</h1>
  <h2>Add three variables with a function</h2>
  <input id="input1" type="number" /> +
  <input id="input2" type="number" /> +
  <input id="input3" type="number" />

  <input type="button" id="answer2" value="Answer2" />

  <span id="result"></span>
```

```
<script>
//BUTTON EVENT LISTENER when click runs the main function
    var e2 = document.getElementById('answer2');
    // invoke main function when button 2 is clicked
    e2.addEventListener('click',main,false);

    //MAIN function runs the procedures
    function main(){
//CALCULATE – this invokes the function summit, takes in 3 arguments (real values) from user input,
    // converts to integer and then returns the total which initialises the value of the global variable sum
    var sum = summit(parseInt(input1.value), parseInt(input2.value), parseInt(input3.value));
    // alert(sum);

//DISPLAY this function accepts 1 parameter – the argument is the value of the global variable sum and
    // displays that value on the webpage
    displayit(sum);
    }

    function summit(x, y, z){
    //alert(x + " " + y + " " + z);
total = x + y + z;
    return total;
    }

function displayit(sum){
    //display on webpage
document.getElementById("result").innerHTML = sum;
    }
</script>
</body>
</html>
```

2.3

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title>Javascript local and global variables</title>
</head>
<body>
  <h1>Javascript</h1>
  <h2>Local and global variables</h2>
  <p>a = 5</p>
  <span id="globalvar"></span><br /><br />
  <span id="localvar"></span><br /><br />

  <script>
//EVENT LISTENER
window.addEventListener('load',localvar,false);

var a = 5; //global variable
document.getElementById("globalvar").innerHTML = "Global variable: " + a;

function localvar() {
var a = 4; //local variable
document.getElementById("localvar").innerHTML = "Local variable: " + a;
}

  </script>

</body>
</html>
```

2.5

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title>Javascript loop from 1 to 50</title>
</head>
<body>
  <h1>Javascript</h1>
  <h2>For Loop from 1 to 50</h2>
  <input type="button" value="Display 1 to 50" onclick="forloop()" />

  <p id="result"></p>

  <script>
function forloop() {
var counter = 0;
  for (vari = 1; i<= 50; i++){
    counter = counter + i + ", ";
  }
  document.getElementById("result").innerHTML = counter;
}
  </script>
</body>
</html>
```


2.8

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta charset="utf-8" />
<title>Javascript while loop from 1 to 10</title>
</head>
<body>
<h1>Javascript labs</h1>
<h2>While loop from 1 to 10</h2>
<input type="button" value="Display 1 to 10" onclick="lab66()"/>
<span id="result"></span>
```

```
<script>
//EVENT LISTENER
window.addEventListener('load',numberloop,false);
```

```
function numberloop() {
//test condition first, then execute code within loop
vari = 1; counter = 0;
while (i<= 10) {
    counter = counter + i + ", ";
    i++;
}
document.getElementById("result").innerHTML = counter;
}
</script>
```

```
</body>
</html>
```

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta charset="utf-8" />
<title>Javascript do...while loop from 1 to 10</title>
<script>
//EVENT LISTENER
window.addEventListener('load',newloop,false);

//do while – executes the loop once then tests the condition
function newloop() {
    vari = 1;
    var counter = 0;
    do{
        counter = counter + i + ", ";
        i++;
    }while(i<= 10);
    document.getElementById("theresult").innerHTML = counter;
}
</script>
</head>
<body>
<h1>Javascript labs</h1>
<h2>Do...While loop from 1 to 10</h2>
<input type="button" value="Display 1 to 10" onclick="lab67()"/>
<span id="theresult"></span>
</body>
</html>
```

2.10

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title>Javascript nested loop</title>
  <script>
    function lab30() {
      var i = 0;
      var j = 1;

var counter = 0;
      var counter2 = 0;

      for(;i<= 10; i++){
        for (; j <= 10; j++){
          //runs nested loop first
          counter = counter + j + " , ";
          alert("j=" + j + " i=" + i + " counter=" + counter); //test variable values when running code
        }

        //then runs this loop
        counter2 = counter2 + i + " , ";
      }

      //then displays text after code in both loops have executed
document.getElementById("lab30result").innerHTML = counter + " " + counter2;
    }
  </script>
</head>
<body>
  <h1>Javascript labs</h1>
  <h2>Lab 30 - Nested loop</h2>
  <input type="button" value="Display 1 to 10" onclick="lab30()" />
  <p id="lab30result"></p>
</body>
</html>
```

JS

Strings

```
<!DOCTYPE html>
<html>
  <!-- this is the strings exercise - 7.1 -->
  <head>
    <link rel="stylesheet" href="stylesStrings.css">
```

```
</head>
```

```
<body>
  <div>
    <h2>Strings - exercise 1 - convert to upper and lower case</h2>
```

```

    <span>Please enter some text:</span>
    <input type = "text" id="inputTextBox"/>
    <br/>
    <br/>
    <button type="button" id="submitText">Submit Text </button>
    <br/>
    <p>Here is the text that you entered:
    <span id="textEntered" class = "outputBox"></span>
    </p>
    <p>Here it is in upper case:
    <span id="textUpperCase" class = "outputBox"></span>
    </p>
    <p>Here it is in lower case:
    <span id=textLowerCase class = "outputBox"></span>
    </p>
    <br/>
    <br/>
  </div>
```

```
<script>
```

```
  //objects - strings exercise 1 - convert to upper and lower case
```

```
//set up an event on the Submit Quiz button
```

```
eventSubmit = document.getElementById("submitText");
eventSubmit.addEventListener('click', convertText, false);
```

```
function convertText()
{
  var textInput = document.getElementById("inputTextBox").value;
  document.getElementById("textEntered").innerHTML = textInput;
  document.getElementById("textUpperCase").innerHTML = textInput.toUpperCase();
  document.getElementById("textLowerCase").innerHTML = textInput.toLowerCase();
}
```



JS

Validation

Data Validation

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

has the user filled in all required fields?

has the user entered valid data?

has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input.

Validation can be defined by many different methods, and deployed in many different ways.

Server side validation is performed by a web server, after input has been sent to the server.

Client side validation is performed by a web browser, before input is sent to a web server.

```
<!DOCTYPE html>
<html>
<head></head>
<body>
  <input type="text" id="input1" />
  <div><button id="button1">Button</button> </div>
  <div id="demo"></div>
  <script>
    //EVENT LISTENER
    var event1 = document.getElementById("button1");
    event1.addEventListener('click', validation, false);
    function validation() {
      //User input 1
      var userInput1 = document.getElementById("input1").value;
      if (userInput1 == "") {
        alert("You must enter a value");
      }
      else {
        //console.log(userinput1);
        myfirst(userinput1);
      }
    }
    function myfirst(theinput) {
      console.log(theinput);

      document.getElementById("demo").innerHTML = theinput;
    }
  </script>
</body>
</html>
```



Validation



JS

Arrays

An **array** is a special variable, which can hold more than one value at a time.

Arrays in Javascript have variable length

All elements in an array can have different data types

First element is:
arrayElement[0]
5th element in an array is:
arrayElement[4]

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>

<script>
var fruits, text, fruitLength, i;

fruits = ["Banana", 1, "Apple", "Mango"];
fruitLength = fruits.length;
text = "<ul>";
for (i = 0; i < fruitLength; i++) {
    text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



Array
Index
For Loop

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Arrays</h2>

<p>The best way to loop through an array is using a standard for loop:</p>

<p id="demo"></p>

<script>
var fruits, text, fruitLength, i;

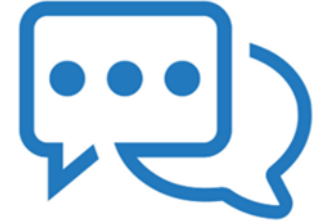
fruits = ["Banana", 1, "Apple", "Mango"];
fruitLength = fruits.length;
text = "<ul>";
for (i = 0; i < fruitLength; i++) {
    text += "<li>" + fruits[i] + "</li>";
}
text += "</ul>";
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```



Array
Length property
Array Index
For Loop
Data Types stored

The **Array object** enables you to do things to the items in the array



Methods

The **join()** method of the *Array* object

The **reverse()** method of the *Array* object

The **sort()** method of the *Array* object

The **pop()** method removes the last element from an array

The **push()** method adds a new element to an array (at the end)

The **shift()** method removes the first array element and "shifts" all other elements to a lower index.

The **unshift()** method adds a new element to an array (at the beginning), and "unshifts" older elements:

The **splice()** method can be used to add new items to an array in a specific position

The **concat()** method creates a new array by merging (concatenating) existing arrays:

The **toString()** method converts an array to a comma separated string when a primitive value is expected

Properties

The *Array* object **length** property

JS

Array Exercises

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<meta charset="utf-8" />
```

```
<title>Javascript allocate to array</title>
```

```
</head>
```

```
<body>
```

```
<h1>Javascript - Arrays</h1>
```

```
<h2>Input, Create an array and display</h2>
```

```
<input id="input1"/>
```

```
<input id="input2"/>
```

```
<input id="input3"/>
```

```
<input type="button" value="Create array" onclick="myarray(input1.value, input2.value, input3.value)"/>
```

```
<p id="arrayresult"></p><br>
```

```
    <p id="arrayresult1"></p><br>
```

```
    <p id="arrayresult2"></p>
```

```
<script>
```

```
function myarray(x, y, z) {
```

```
//declare and create array
```

```
varmyArray = [];
```

```
myArray[0] = x;
```

```
myArray[1] = y;
```

```
myArray[2] = z;
```

```
    //display array on webpage with 2 different approaches
```

```
    document.getElementById("arrayresult").innerHTML = myArray;
```

```
    document.getElementById("arrayresult1").innerHTML = myArray.join(", ");
```

```
    //display item in array with index 1
```

```
    document.getElementById("arrayresult2").innerHTML = myArray[1];
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```

```
<!DOCTYPE html>
```

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
```

```
<meta charset="utf-8" />
```

```
<title>Javascript - Array Length</title>
```

```
</head>
```

```
<body>
```

```
<h1>Javascript - Arrays</h1>
```

```
<h2>Array Length</h2>
```

```
<input id="input1" />
```

```
<input type="button" value="Push value to array" onclick="addtoarray(input1.value)" />
```

```
<p id="result"></p>
```

```
<p id="lengthcounter"></p>
```

```
<script>
```

```
var myArray = [];
```

```
function addToarray(x) {
```

```
myArray.push(x);
```

```
document.getElementById("result").innerHTML = myArray.join(", ");
```

```
document.getElementById("lengthcounter").innerHTML = "Number of elements: " + myArray.length; }
```

```
</script>
```

```
</body>
```

```
</html>
```

```

<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title>Javascript add and display 10 members of an array</title>

</head>
<body>
  <h1>Javascript</h1>
  <h2>Display 10 members of an array using for loop</h2>
  <!--  <button type="button" id="myBtn1" onclick="looparray()">Display member
from country array</button> -->
  <button type="button" id="myBtn1">Display member from country array</button>
  <input type="text" id="guess" />

  <button type="button" id="mybee">Search</button>

  <p id="myarray"></p>

  <p id="myarray3"></p>
  <p id="myarray4"></p>
  <p id="myarray5"></p>

</body>
</html>

```

```

<script>
// ADD EVENT LISTENERS
//      var but1 = document.getElementById('myBtn1');
//      var but1 = document.querySelector("#myBtn1");
//      but1.addEventListener('click', looparray,false);

// var but2 = document.getElementById('mybee');
//var but2 = document.querySelector("#mybee")

      var but2 = document.getElementById('mybee');
but2.addEventListener('click', three,false);

// CREATE ARRAY
var myCountryArray = ["England", "France", "Italy", "Mexico", "Poland", "Russia", "China", "Greece", "Egypt", "India"];
// alert(myCountryArray);
//loop array and display on webpage
function looparray () {
  for(var i = 0; i<myCountryArray.length; i++){
document.getElementById("myarray").innerHTML += myCountryArray[i] + " , ";
  }
};

//LOOP - get user input, then search array for a match
function three () {
  var x = document.getElementById("guess").value;
  alert(x);

  for(var j = 0; j <myCountryArray.length; j++){
    if (x == myCountryArray[j]) {
document.getElementById("myarray3").innerHTML = " Success!";
      // alert(myCountryArray[j]);
      document.getElementById("myarray4").innerHTML = "Value: " + myCountryArray[j];
      // alert(j);
      document.getElementById("myarray5").innerHTML = "Position: " + j;

      return true;
    }
    else {
document.getElementById("myarray3").innerHTML = " Failure";
    }
  }
};

</script>

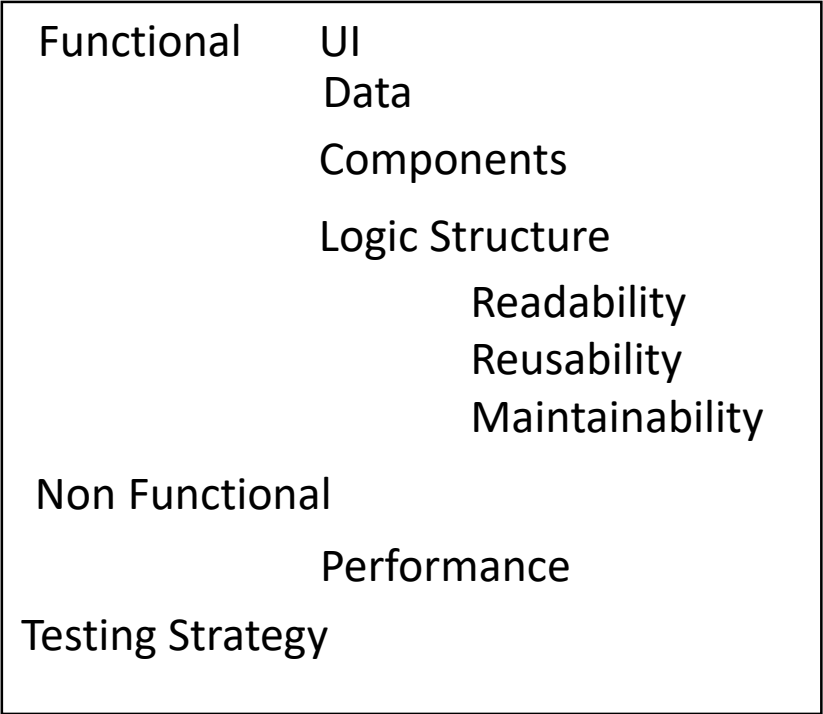
```


JS

Mini Projects

Requirements

Design



Development

Testing

Deployment

Vending Machine Change (Core version)

User enters money number value between £1-£100 to 2 decimal places

The correct and lowest number of money denominations are listed to represent the amount

Following denominations accepted: 5,10, 20, 50 cents £1, £2, £5, £10, £20, £50

Test case 1
Input£47.50

Output
2 x £20
1 x £5
1 X £2
1 x 50 cents

Test case 2
Input£105

Output
Invalid amount
Amount must be
between £1-£100

Test case 3
Input£91

Output
1 X £50
2 X £20
1 X £1

```
var num = 5.56789;
    var n = num.toFixed(2);
    console.log("n = " + n);
```

Math.random(

//Regular expression to check if input is a number

```
var str = document.getElementById("input1").value;
var reg = /^\d+$/;
console.log(str.search(reg));
```

<script>

Convert string to number

```
function myFunction() {
    var a = parseInt("10") + "<br>";
    var b = parseInt("10.00") + "<br>";
    var c = parseInt("10.33") + "<br>";
    var d = parseInt("34 45 66") + "<br>";
    var e = parseInt(" 60 ") + "<br>";
    var f = parseInt("40 years") + "<br>";
    var g = parseInt("He was 40") + "<br>";
```

```
    var h = parseInt("10", 10)+ "<br>";
    var i = parseInt("010")+ "<br>";
    var j = parseInt("10", 8)+ "<br>";
    var k = parseInt("0x10")+ "<br>";
    var l = parseInt("10", 16)+ "<br>";
```

```
    var n = a + b + c + d + e + f + g + "<br>" + h + i + j + k + l;
    document.getElementById("demo").innerHTML = n;
```

}

</script>

END

Output Javascript

There are 4 ways to output Javascript:

Writing into an HTML element, using **innerHTML**.

Writing into the HTML output using **document.write()**.

Writing into an alert box, using **window.alert()**.

Writing into the browser console, using **console.log()**

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>Javascript</h2>
```

```
<p>Output Javascript</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
document.write(5 + 6);
```

```
window.alert(5 + 6);
```

```
console.log(5 + 6);
```

```
document.getElementById("demo").innerHTML = 5 + 6;
```

```
</script>
```

```
</body>
```

```
</html>
```

Invoking Javascript

In HTML Block

In HTML, JavaScript code must be inserted between <script> and </script> tags.

JavaScript function is placed in the <body> section of an HTML page.

The function is invoked (called) when a button is clicked:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body onload=age()>
```

```
<p id="demo"></p>
```

```
<script>
```

```
document.getElementById("demo").innerHTML = "My First JavaScript";
```

```
</script>
```

```
</body>
```

```
</html>
```

Event Listeners

Event Listeners are the modern approach to invoking Javascript on HTML events.

Event listeners support the development of separated code and decoupling from the HTML document.

HTML
Javascript

X

HTML

Javascript

```
<!DOCTYPE html>
<html>
<head></head>
<body>
<form>
<input type="text" id="input1"/></form>
<div><button id="button1">Button</button> </div>
<div id="demo"></div>

<script>
//EVENT LISTENER
var event1 = document.getElementById("button1");
event1.addEventListener('click',writeinput,false);

function writeinput(){
//User input 1
var userinput1 = document.getElementById("input1").value;

//Output 1
document.getElementById("demo").innerHTML = userinput1;
}
</script>

</body>
</html>
```


Scope

Scope determines the accessibility of variables, objects, and functions from different parts of the code.

The lifetime of a JavaScript variable starts when it is declared.

Local variables are deleted when the function is completed.

```
<!DOCTYPE html>
<html>
<body onload="forloop()">

<p>The local variable carName cannot be accessed from code outside
the function:</p>

<p id="demo"></p>

<script>
myFunction();
document.getElementById("demo").innerHTML =
"The type of carName is " + typeof carName;

function myFunction() {
  var carName = "Volvo";
}
</script>

</body>
</html>
```

The *window* Object

The window object represents an open window in a browser. There are many properties and methods to access

The localStorage and sessionStorage properties allow to save key/value pairs in a web browser.

The sessionStorage object stores data for only one session (the data is deleted when the browser tab is closed).

The localStorage and sessionStorage properties allow to save key/value pairs in a web browser.

The localStorage object stores data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

The localStorage property is read-only

https://www.w3schools.com/jsref/o bj_window.asp

```
<!DOCTYPE html>
<html>
<body>
<div id="result2"></div>
<div id="result"></div>
<script>
// Check browser support
if (typeof(Storage) !== "undefined") {
    // Store
    // localStorage.setItem("lastname", "Smith");
    // Retrieve
    document.getElementById("result").innerHTML = localStorage.getItem("lastname");
} else {
    document.getElementById("result").innerHTML = "Sorry, your browser does not support
Web Storage...";
}
</script>

<script>
//comment out session storage and reopen in new browser window to show local storage
retains value

// Check browser support
if (typeof(Storage) !== "undefined") {
    // Store
    // sessionStorage.setItem("lastname2", "Brown");
    // Retrieve
    document.getElementById("result2").innerHTML = sessionStorage.getItem("lastname2");
} else {
    document.getElementById("result2").innerHTML = "Sorry, your browser does not support
Web Storage...";
}
</script>
</body>
</html>
```

The *document* Object

When an HTML document is loaded into a web browser, it becomes a **document object**.

Provides the properties and methods to work with the current document

- The *bgColor* and *fgColor* properties
- The *title* property
- The *lastModified* property
- Referencing remote *window* and *document* objects

```
<!DOCTYPE html>
<html>
<body>

<div class="example">First div element with class="example".</div>

<div class="example">Second div element with class="example".</div>

<p>Click the button to change the text of the first div element with class="example" (index 0).</p>

<button onclick="myFunction()">Change Text</button>

<p><strong>Note:</strong> The getElementsByName() method is not supported in Internet Explorer 8 and earlier versions.</p>

<script>
function myFunction() {
    //returns a node list (array) of all elements in HTML page with class=example
    var x = document.getElementsByClassName("example");
    //change value of node[0]
    x[0].innerHTML = "Hello World!";
}
</script>

</body>
</html>
```

The *location* Object

Allows you to specify URLs in a script

Location Object

The location object contains information about the current URL.

The location object is part of the window object and is accessed through the window.location property.

Note: There is no public standard that applies to the location object, but all major browsers support it.

The *history* Object

Allows the user to move backward or forward through the stored history of your Web page

The *image* Object

Allows you to manipulate images in browsers

Handling *image* object events

JavaScript and image maps

The *navigator* Object

Determines the brand and version of the browser in use

Identifies the user's operating system

Redirecting the browser with the *navigator* and *location* objects

Debugging

X

Errors can (will) happen, every time you write some new computer code.

code might contain syntax errors, or logical errors

F12 key, and select "Console"

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<h2>My First Web Page</h2>
```

```
<p>
```

Activate debugging in your browser (Chrome, IE, Firefox) with F12, and select "Console" in the debugger menu.

```
</p>
```

```
<script>
```

```
a = 5;
```

```
b = 6;
```

```
c = a + b;
```

```
console.log(c);
```

```
</script>
```

```
</body>
```

```
</html>
```

Error Handling

Errors can (will) happen, every time you write some new computer code.

code might contain syntax errors, or logical errors

F12 key, and select "Console"

- The **try** statement lets you test a block of code for errors
- The **catch** statement lets you handle the error
- The **throw** statement lets you create custom errors
- The **finally** statement lets you execute code, after try and catch, regardless of the result.

try catch

The **try** statement allows you to define a block of code to be tested for errors while it is being executed.

The **catch** statement allows you to define a block of code to be executed, if an error occurs in the try block.

The JavaScript statements **try** and **catch** come in pairs:

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}
```

```
<!DOCTYPE html>  
<html>  
<body>  
  
<p id="demo"></p>  
  
<script>  
try {  
    adddlert("Welcome guest!");  
}  
catch(err) {  
    document.getElementById("demo").innerHTML = err.message;  
}  
</script>  
  
</body>  
</html>
```

throw

When an error occurs, JavaScript will normally stop and generate an error message.

The technical term for this is: JavaScript will **throw an exception (throw an error)**.

The **throw** statement allows you to create a custom error.

Technically you can **throw an exception (throw an error)**.

```
<!DOCTYPE html>
<html>
<body>

<p>Please input a number between 5 and 10:</p>

<input id="demo" type="text">
<button type="button" onclick="validateFunction()">Test Input</button>
<p id="message"></p>

<script>
function validateFunction() {
  var message, x;
  message = document.getElementById("message");
  message.innerHTML = "";
  x = document.getElementById("demo").value;
  try {
    if(x == "") throw "empty";
    if(isNaN(x)) throw "not a number";
    x = Number(x);
    if(x < 5) throw "too low";
    if(x > 10) throw "too high";
  }
  catch(err) {
    message.innerHTML = "Input is " + err;
  }
}
</script>

</body>
</html>
```


HTML Validation

HTML Validation

Modern browsers will often use a combination of JavaScript and built-in HTML validation, using predefined validation rules defined in HTML attributes:

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Please input a number between 5 and 10:</p>
```

```
<input id="demo" type="number" min="5" max="10" step="1">
```

```
<button type="button">Test Input</button>
```

```
<p id="message"></p>
```

```
</body>
```

```
</html>
```

External Javascript File

```
<!DOCTYPE html>
<html>
<body>

<h2>External JavaScript</h2>

<p id="demo">A Paragraph.</p>

<button type="button" onclick="myFunction()">Try it</button>

<p>(myFunction is stored in an external file called
"myScript.js")</p>

<script src="myScript.js"></script>

</body>
</html>
```



```
function myFunction() {
    document.getElementById("demo").innerHTML = "Javascript
does something.";
}
```

Mini Projects 2.1

Common coding errors

JavaScript programs may generate **unexpected results** if a programmer accidentally uses an assignment operator (=), instead of a comparison operator (==) in an if statement.

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>
<p id="demo1"></p>

<script>
var x = 0;
document.getElementById("demo").innerHTML = Boolean(x = 10);
document.getElementById("demo1").innerHTML = Boolean(x == 10);
</script>

</body>
</html>
```

JavaScript programs may generate **unexpected results** if a programmer accidentally uses regular comparison == and not strict ===.

Use strict only === when comparing to minimise errors

```
<!DOCTYPE html>
<html>
<body>

<p id="demo"></p>
<p id="demo1"></p>

<script>
var w = 10 + "5";
document.getElementById("demo1").innerHTML = w + " " + typeof(w);

var x = 10;
var y = "5";
var z = x + y;
document.getElementById("demo").innerHTML = z + " " + typeof(z);
</script>

</body>
</html>
```

In a web browser, global variables are deleted when you close the browser window (or tab), but remain available to new pages loaded into the same window.

Always use local variables where you can.

```
<!DOCTYPE html>
<html>
<body>

<p>
Global variables will become window variables.
</p>

<p id="demo"></p>

<script>
var carName = "Volvo";

// code here can use window.carName
document.getElementById("demo").innerHTML = "I can display
" + window.carName;
</script>

</body>
</html>
```

Algorithms

Algorithms

- Algorithms
- Flowcharts
- Decision tables
- Sorting algorithms
- An algorithm is a set of ordered and finite steps to solve a given problem

Recursion

- Where a function or method calls itself
- It MUST have a stop condition
 - Called the **base condition**
- Mainly mathematical uses



Recursive Function

To move x discs from peg A to peg C, using peg B as an "aux" peg:

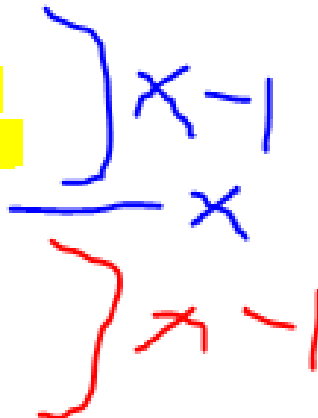
Move $x-1$ discs from peg A to peg B, using peg C as the aux peg.

Move the x 'th disc (last) from peg A to peg C (no aux peg needed because you're only moving one disc).

Move the $x-1$ discs from peg B to peg C, using peg A as the aux peg.

Move disc 1 from src to dst
Move disc 2 from src to aux
Move disc 1 from dst to aux

Move disc 3 from src to dst
Move disc 1 from aux to src
Move disc 2 from aux to dst
Move disc 1 from src to dst



```
<!DOCTYPE html>
<html><head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<head>
<meta charset="utf-8">
<meta http-equiv="X-UA-Compatible" content="IE=edge">
<title>Recursion</title>
</head>

<body onload="hanoi(3,"src","aux","dst");">

<h1>Recursive Function</h1>
<h2>Tower of Hanoi</h2>

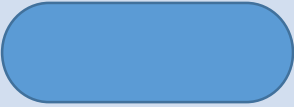

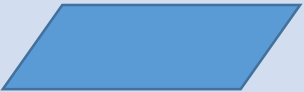


<!-- <script src="tech1.js" type="text/javascript"></script> -->
<script>
var hanoi = function(disc,src,aux,dst)
{
    /*base case*/
    if (disc > 0) {
        hanoi(disc - 1,src,dst,aux);
        document.write("Move disc " + disc + " from " + src + " to " + dst + "<br />");
        hanoi(disc - 1,aux,src,dst);
    }
};

hanoi(3,"src","aux","dst");
</script>

</body></html>
```

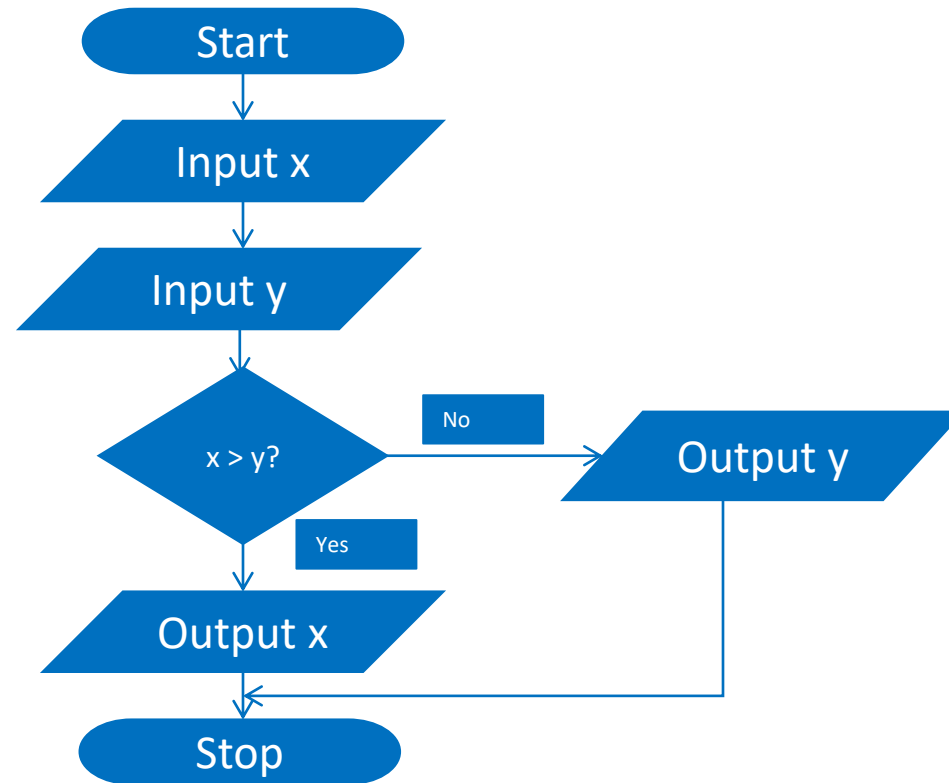
Flowcharts

- A flowchart is a graphical representation of an algorithm

Common Flowchart Symbols	
	Start or end of an algorithm
	Process or computational operation
	Input or output operation
	Decision making operation
	Direction of the flow of control

Flowcharts - example

- A flowchart that compares two numbers



Sorting algorithms

- Quick sort
 - Probably the fastest sorting algorithm
 - Uses divide and conquer idea – list is subdivided into halves, quarters, etc., until sorts are easily done
- Example

		5,1,3,2,4			Choose a pivot element, e.g. 3, and divide the list into 3
	2,1	3	5,4		Move everything less than 3 to the left list and everything greater than 3 to the right list
1	2	3	4	5	Choose a pivot element in the left and right lists and move everything less than the pivot to the left and everything greater to the right. The lists now consist of single elements, which are sorted.
		1,2,3,4,5			Put the lists back together to form a sorted list

Quicksort

- Animation
- <https://commons.wikimedia.org/wiki/File:Quicksort-example.gif>
- Quicksort vs. bubblesort
- <https://www.youtube.com/watch?v=vxENKlcs2Tw>

Regular Expressions

A regular expression is a sequence of characters that forms a search pattern.

The search pattern can be used for text search and text replace operations.

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<p>Search a string for "w3Schools", and display the position of  
the match:</p>
```

```
<button onclick="myFunction()">Try it</button>
```

```
<p id="demo"></p>
```

```
<script>
```

```
function myFunction() {
```

```
    var str = "Javascript is fun";
```

```
    var n = str.search(/fun/i);
```

```
    document.getElementById("demo").innerHTML = n;
```

```
}
```

```
</script>
```

```
</body>
```

```
</html>
```