



# CS7025

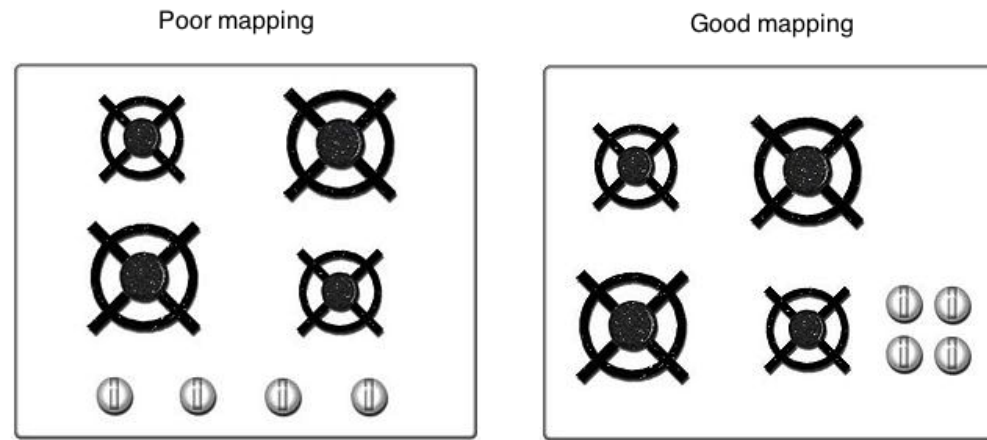
# Programming for Digital Media

Lesson 9 – Programming with JavaScript

# Design Principles

Don Norman

- ▶ Feedback
- ▶ Mapping
- ▶ Visibility
- ▶ Constraints
- ▶ Consistency
- ▶ Affordance



# Problem Solving

- ▶ Drive a car until you approach a traffic light.
- ▶ When the traffic light is red, stop driving the car.
- ▶ When the traffic light is green, continue driving.

Verb -> function(){}  
Noun -> Object or Variable

Condition -> TRUE/FALSE



# Debugging JavaScript

## Comments

Comments in JavaScript can be written in two ways

```
// Single line comment  
/*  
    Multi line comment  
*/
```

The JavaScript interpreter will ignore/not execute comments

Think about comments and don't forget to remove them when they are no longer needed



# Debugging JavaScript

## Console

You can instruct your JavaScript to write information and errors to the browser's console.

The Console API provides functionality to allow developers to perform debugging tasks, such as logging messages or the values of variables at set points in your code, or timing how long an operation takes to complete.

```
console.log(<content>);
```

```
console.error('something went wrong', error);
```

<https://developer.mozilla.org/en-US/docs/Web/API/Console>

<https://medium.com/@anirudh.munipalli/stop-using-console-log-in-javascript-try-these-instead-72490d895a24>

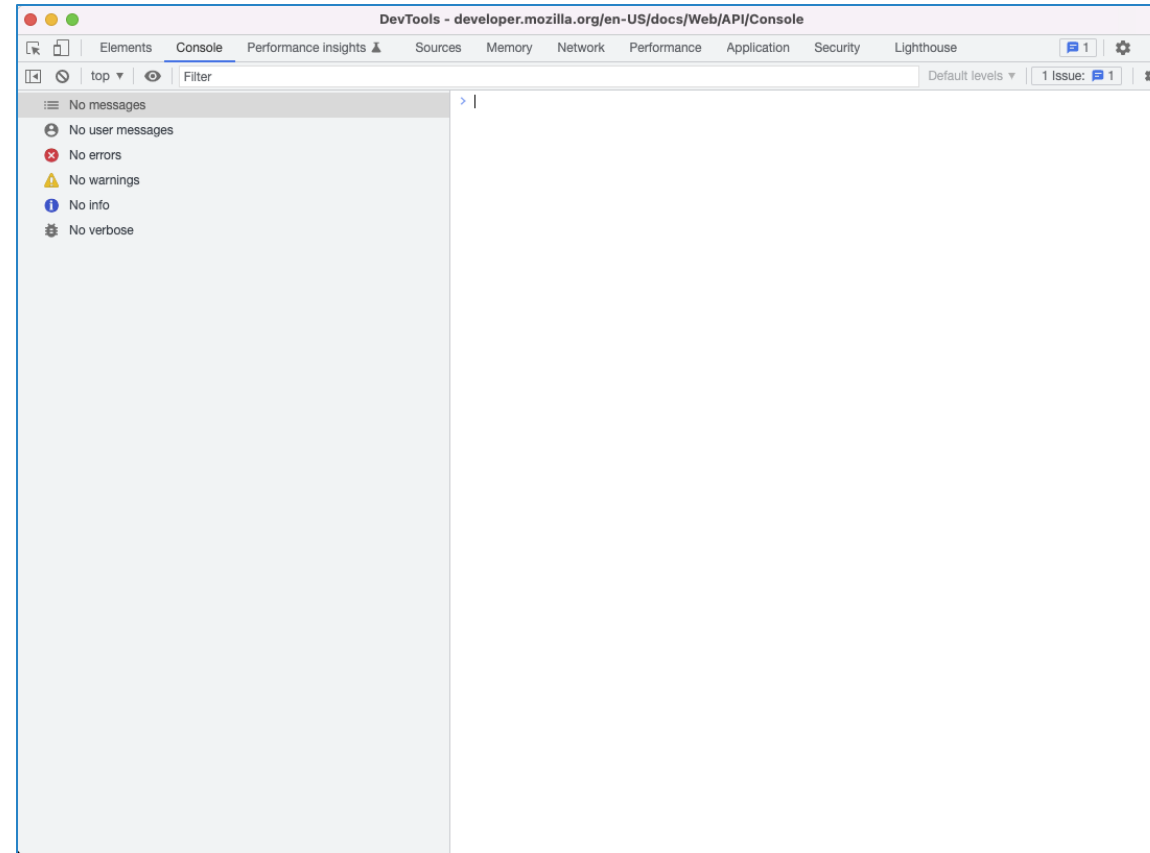


**Trinity College Dublin**  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin

# Debugging JavaScript

## Console

Right Click >  
Inspect >  
Console



<https://developer.mozilla.org/en-US/docs/Web/API/Console>

<https://medium.com/@anirudh.munipalli/stop-using-console-log-in-javascript-try-these-instead-72490d895a24>



**Trinity College Dublin**  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin

# Variable Keyword

## Variable Keywords:

- ▶ `var`      `// global variable accessible everywhere`
- ▶ `let`      `// accessible within scope`
- ▶ `const`    `// immutable, known at compile-time`
- ▶ `final`    `// immutable, known at run-time`
- ▶ `static`    `// accessible class variable without the need to instantiate`



# Variable Types

▶ String	represents textual data	<code>'hello', "hello world!"</code> etc.
▶ Number	an integer or a floating-point number	<code>3, 3.234, 3e-2</code> etc.
▶ BigInt	an integer with arbitrary precision	<code>900719925124740999n , 1n</code> etc.
▶ Boolean	any of two values: true or false	<code>true</code> and <code>false</code>
▶ undefined	variable is not initialized	<code>let a;</code>
▶ null	denotes a null value	<code>let a = null;</code>
▶ Symbol	instances are unique and immutable	<code>let value = Symbol('hello');</code>
▶ Object	key-value pairs of collection of data	<code>let student = { };</code>
▶ Array	list of data	<code>let list = ['milk','bread',...];</code>





# Variable Type String

- ▶ A String is a list of characters surrounded by 'quotes', "double quotes" or `back ticks`

```
let name = 'Ronald';  
let address = 25 + " Herbert Park Road";  
let age = 25; //a number  
age = age.toString();
```



# Variable Name

- ▶ Naming is hard!
- ▶ Be descriptive, write for the next programmer who is going to work with your code, or yourself one year from now.
- ▶ Don't write names that are too short:  
Consider `var pr` vs `var patientRecord`
- ▶ or names that are too long:  
`var patientOfThisHospitalRecord`
- ▶ Be consistent, use either `camelCasing` or `underscores_for_naming` variables, when you choose one, keep using that format for the rest of your code



# Variable Name

- ▶ Variable names are made up of Unicode letters
- ▶ Cannot start with a number
- ▶ Cannot contain a space or a hyphen (-)
- ▶ Are case sensitive `firstName != FirstName`
- ▶ Don't use reserved words



# Reserved Words

else	as	const	export	get	null	target
async	continue	extends	in	of	this	while
await	debugger	false	import	return	throw	with
break	default	finally	in	set	true	yield
case	delete	for	instanceof	static	try	
catch	do	from	let	super	typeof	
class	function	new	switch	var	void	



# Semicolon

In Javascript the semicolon ';' is optional. It is good practice however to end a statement with one, to increase legibility of your code and to avoid confusion;

Don't use a semicolon when:

- ▶ Declaring a function
- ▶ A conditional comparison
- ▶ A loop

```
function doSomething(){}  
if(true){}else{}  
for(let i=0; i<5; i++){}
```



# Functions

Functions can return things

```
var studentName = "Robert";
```

```
function doSomething(){  
    return studentName;  
}
```



# Functions

Functions can process things

```
function doSomething(parameter){  
    return parameter;  
}  
console.log(doSomething("Hello World")) // returns "Hello World"
```

```
function sum(a, b){  
    return a + b;  
}  
console.log( sum( 5, 2 ) ); // returns 7
```



# Functions

- ▶ Whatever statement or condition is included within the brackets of a function are executed when the function is called. Functions can be called or nested within other functions.

```
function sum(a, b){  
    console.log("a", a, "b", b);  
    return a + b;  
}  
console.log( sum( 5, 2 ) );    // returns 7
```





# Scope

Variables should only be accessible within scope;

```
let students = 31;
function countAbsentStudents(){
    let absent = 3;
    students -= absent;
}
countAbsentStudents();
console.log("students", students);
console.log("absent", absent); //throws error: absent is not defined
```



# Comparison Operators in JavaScript

==	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



# Conditional Statements: if

```
if ( expression ) {  
    // statement(s) to be executed only if expression is true;  
}  
// statement(s) to be executed whether or not expression is true;
```

```
if (hour < 18) {  
    greeting = "Good day";  
}
```



# Conditional Statements: if ... else

```
if ( expression ) {  
    // statement(s) to be executed only if expression is true;  
} else {  
    // statement(s) to be executed only if expression is false;  
}  
// statement(s) to be executed whether or not expression is true;
```

```
if (hour < 18) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```



# Conditional Statements: chained conditionals

```
if ( expression1 ) {  
    // statement(s) to be executed only if  
    // expression1 is true;  
}  
else if ( expression2 ) {  
    // statement(s) to be executed only if  
    // expression2 is true;  
}  
else {  
    // statement(s) to be executed only if  
    // neither expression is true;  
}  
  
// statement(s) to be executed  
// whether or not the expressions are true;
```

```
function greeting(theTime){  
    let greeting;  
    if (theTime < 10) {  
        greeting = "Good morning";  
    } else if (theTime < 20) {  
        greeting = "Good day";  
    } else {  
        greeting = "Good evening";  
    }  
    return greeting;  
}
```



# Conditional Statements: nested conditionals

```
if ( expression1 ) {  
    // statement(s) to be executed only if expression1 is true;  
} else {  
    if ( expression2 ) {  
        // statement(s) to be executed if expression 1 is false and expression2 is true;  
    } else {  
        // statement(s) to be executed only if expression1 and expression2 are false;  
    }  
}  
  
// statement(s) to be executed whether or not expressions are true
```



# Loops: for

```
for (let i = 0; i < 5; i++) {  
    // statement(s) to be until i >= 5;  
}
```

// ++ is a shorthand way to say **i = i + 1;**

```
// statement(s) to be executed after the for loop has been completed;  
// i.e. when i is no longer < 5
```

```
for (let i = 0; i < 5; i++) {  
    console.log("The number is " + i );  
}
```



# Loops: for/In

```
let sequence = [1,2,3,4,5];
let position;
for ( position in sequence ) {
    // statement(s) to be executed once for each element in the sequence;
    // note that you need to use the syntax sequence[position] to access
    // the value at that position in the sequence
}
// statement(s) to be executed after each element has been processed;
```

```
let numbers = [34,2,56,8,10];
let x;
for (x in numbers) {
    console.log("num " + x + " is "+numbers[x]);
}
```

```
let student = {name: 'John Doe', age: 25};
for (i in students) {
    console.log("student: " + student[i]);
}
```





# Loops: while

```
while (expression) {
```

```
    // statement(s) to be executed while the expression is true;
```

```
}
```

```
// statement(s) to be executed if the expression is false or after it becomes false;
```

(The statements inside the loop may or may not be executed)

```
let i = 0;
while (i < 10) {
    console.log("The number is " + i );
    i++;
}
```



# Loops: for vs. while

- ▶ **for** is great when the program knows in advance the number of times it will need to go through the loop
- ▶ **for/in** is used when you want to repeat some code for each element of a string or array (list of values) or an object
- ▶ **while** is useful for situations where the program won't know until later on how many times it needs to loop
- ▶ **do/while** is similar to while, but better if you always want to execute the statements in the loop at least once



# Events

Events are the things that are happening on your web page or in your application.

Examples:

- ▶ Loading a page
- ▶ Clicking on a hyperlink
- ▶ Entering text in a text field / pressing keys on the keyboard
- ▶ Checking a checkbox
- ▶ Submitting a form

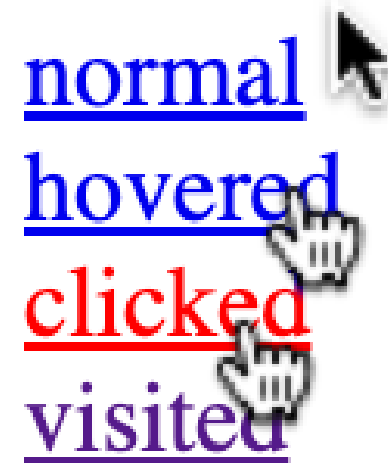


# State

State goes hand in hand with events. Events usually alter the state of your web page or application. State is whatever describes the UI at a given point in time or when you are in control of your own states, a specific stage of a process.

In the case of a hyperlink for instance, it can be

- ▶ 'normal' the link is there but no interaction
- ▶ 'hovered' the mouse cursor is over the link
- ▶ 'clicked'
- ▶ 'visited'



# The Document Object Model

The Document Object Model (DOM) represents the structure of a document (HTML).

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
  </head>
  <body>
  </body>
</html>
```



# Manipulating the DOM

Let the browser wait with executing code within scope until the document is completely loaded into memory

```
document.addEventListener("DOMContentLoaded", () => {  
    //do things  
});
```



# Manipulating the DOM

<https://developer.mozilla.org/en-US/docs/Web/API/Document>

The dot notation is used to traverse through the DOM

```
document.querySelector('#theSelector');
```

In English:

"Of the document object, use the query selector for the element with an ID of 'theSelector' "



**Trinity College Dublin**  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin

# Manipulating the DOM

Different ways of addressing elements within the DOM

`getElementById()`

```
// <div id="student"></div> document.getElementById('student');
```

`getElementsByClassName()`

```
// <div class="student"></div> document.getElementsByClassName('student');
```

`getElementsByName()`

```
// <input type="text" name="up" /> document.getElementsByName('up');
```

`getElementsByTagName()`

```
// <h1>A Title</h1> document.getElementsByTagName('h1');
```

`querySelector()`

```
// <div id="student"></div> document.querySelector('#student');
```

`querySelectorAll()`

```
// <div class="student"></div> <div id="location">...</div>  
// document.querySelectorAll('.student #location');
```





# Manipulating the DOM

## Adding and removing elements

The way to add new elements to your document is by creating a new element (store it in memory)

```
let paragraph = document.createElement("p");  
paragraph.textContent = "Hello World!";
```

And then add it to the document (make it available)

```
document.body.appendChild(paragraph);
```

To remove an element, use the `remove()` function:

```
document.getElementById(<THE ID>).remove();
```



# Manipulating the DOM

## Attributes

```
<div id="students" class="cs7025" data-counter="3">...</div>
```

Get the value of an attribute by invoking the `getAttribute(attribute)` function

Set the value of an attribute by invoking the `setAttribute(name, value)` function

```
document.getElementById('students').setAttribute('id','teachers');
```



# Arrays

## Zero-indexed

Zero-indexed means that the first element in the array is accessed by the index 0

```
let week =
```

```
["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"];
```



0



1



2



3



4



5



6



# Arrays

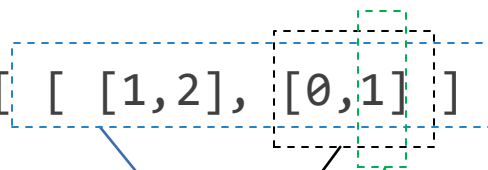
## Multi-dimensional

Array of arrays

```
let m = [ [0,1,2,3], [2,3,4,5], [1,2,3,4,5] ];
```

```
console.log( m[0][2] );
```

```
let q = [ [ [1,2], [0,1] ], [ [4,5], [2,3] ] ];
```



```
console.log( q[0][1][1] );
```



# JSON vs JavaScript Objects

## JSON

- ▶ key must be in double quotes
- ▶ JSON can't contain functions
- ▶ JSON can be used and created by other programming languages
- ▶ use `JSON.parse()` to convert it to a JavaScript Object

## JavaScript Objects

- ▶ No quotes needed for key
- ▶ JavaScript Objects can contain functions
- ▶ JavaScript Objects can only be used in JavaScript
- ▶ use `JSON.stringify()` to convert it to JSON



# JavaScript Objects

```
{  
  key: value[,  
  key: value,  
  ...  
  key: value]  
}
```



# JavaScript Objects

JavaScript Object values can hold any valid JavaScript structure

```
const STUDENT = {  
  id: 12345,  
  firstName: "Christiane",  
  lastName: "Castillo",  
  age: calc(20,5)  
}  
  
function calc(a, b){ return a + b; }
```



# JavaScript Objects

```
const STUDENT = {  
  id: 12345,  
  firstName: "Christiane",  
}
```

There are 2 ways to call the value of a key in an object

```
STUDENT["firstName"] // returns "Christiane"
```

```
STUDENT.firstName // returns "Christiane"
```





# JSON

JavaScript Object Notation

Similar to objects, but not the same

Uses key:value pairs, but the key must be surrounded by quotes

```
{  
  "name": "Christiane",  
  "age": 64  
}
```



# JSON

JSON is purely a string with a specified data format — it contains only properties, no methods.

JSON requires double quotes to be used around strings and property names. Single quotes are not valid other than surrounding the entire JSON string.

JSON values can only be of the following data types:

- ▶ strings
- ▶ numbers
- ▶ objects (JSON object)
- ▶ arrays
- ▶ Boolean
- ▶ null

<https://www.json.org/json-en.html>



**Trinity College Dublin**  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin

# Object Oriented Programming

## In JavaScript

### Principles of OOP

- ▶ Abstraction // blueprint
- ▶ Inheritance // sub classes
- ▶ Encapsulation // private members/methods
- ▶ Polymorphism // different behaviour same name
- ▶ Method Overloading / Overriding // different method depending on  
// number of arguments



# Object Oriented Programming

## In JavaScript

In OOP you use classes to create a (reusable) blueprint of an object with the **data** and the **methods** it needs

```
class Person{
  name;
  constructor(name){
    this.name = name;
  }
  speak(){
    console.log(`Hello, my name is ${this.name}`);
  }
}
```



# Object Oriented Programming

## In JavaScript

When you create an instance of a class, you create an object with the characteristics of that class, all its members (variables and methods) are available.

```
let joris = new Person("Joris");  
joris.speak();
```



# Object Oriented Programming

## In JavaScript

### Inheritance

You can extend an existing class to give it more specificity, new members or change existing ones

```
class Fish extends Animal{
  move(){
    console.log("swim");
  }
}

let guppy = new Fish();
guppy.move(); // "swim"
```



# Object Oriented Programming

## In JavaScript

### Encapsulation

You can control who has access to members by making them private

```
class Person{
```

```
  #name;
```

```
  constructor(name){
```

```
    this.#name = name;
```

```
  }
```

```
  getName(){
```

```
    return this.#name;
```

```
  }
```

```
}
```

```
let jim = new Person("Jim");
```

```
jim.name      // undefined
```

```
jim.#name     // Error
```

```
jim.getName() // "Jim"
```



# Object Oriented Programming

## In JavaScript

### Polymorphism

```
class Animal{
  name;
  constructor(name){
    this.name = name;
  }
  move(){
    console.log("moving!");
  }
}
```

```
class Bird extends Animal{
  move(){
    console.log("Fly");
  }
}
```

```
class Fish extends Animal{
  move(){
    console.log("Swim");
  }
}
```





# Object Oriented Programming

## In JavaScript

### Static Class Members

Sometimes you just want to access a class member without instantiating the class. `Math.random()`

```
class Car{  
    static drive(){  
        console.log("vroom, vroom");  
    }  
}
```

```
Car.drive(); // "vroom, vroom"
```



# Data Storage

## Client-side

Browsers allow you to store data on the client. There are three types of storage available to us:

### **Cookies**

- ▶ Have an expiry date

### **Session Storage**

- ▶ Gets discarded when the browser is closed

### **Local Storage**

- ▶ Persist for longer (until local storage is cleared)



# Local Storage

- ▶ Accessible via the Window interface in JavaScript.
- ▶ `window.localStorage` is a read-only property that returns a reference to the local storage object used to store data that is only accessible to the origin that created it.
- ▶ Stores data in key-value pairs as strings



# Local Storage

Local Storage has 5 methods

- ▶ `setItem()`: Add key and value to `localStorage`
- ▶ `getItem()`: This is how you get items from `localStorage`
- ▶ `removeItem()`: Remove an item by key from `localStorage`
- ▶ `clear()`: Clear all `localStorage`
- ▶ `key()`: Pass an index to retrieve the key of a `localStorage`



# Local Storage

## Storing Content

An example of storing data in local storage with JavaScript is:

```
window.localStorage.setItem('name', 'John Doe');
```

`localStorage` only wants Strings. You can store an object in `localStorage`, but you'll have to convert it to a string first. JSON has a function for that.

```
let person = { name: "John", age: 28 };
```

```
window.localStorage.setItem('person', JSON.stringify(person));
```

```
// '{"name": "John", "age": "28" }'
```



# Local Storage

## Retrieving Content

An example of retrieving data in local storage with JavaScript is:

```
window.localStorage.getItem('name');
```

```
JSON.parse(window.localStorage.getItem('person'));
```



# Try it yourself

Scratch



**Trinity College Dublin**  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin

# Thank You



**Trinity College Dublin**  
Coláiste na Tríonóide, Baile Átha Cliath  
The University of Dublin