Contents

[Contents 1](#_Toc42690697)

[The Daily Brick 2](#_Toc42690698)

[JavaScript Video games development 2](#_Toc42690699)

[The Odin Project 3](#_Toc42690700)

[Gatsby JS - The Great Gatsby Bootcamp [Full Tutorial] 4](#_Toc42690701)

[Udemy - CSS - The Complete Guide 2020 5](#_Toc42690702)

[Hacking 101 5](#_Toc42690703)

[CSS - Studies 7](#_Toc42690704)

[The Odin Project Studies 8](#_Toc42690705)

# The Daily Brick

This section is to keep track of the many tutorials and tasks I am trying to learn. Due to the sheer amount of content, it is becoming hard to keep it all in my brain.

## JavaScript Video games development

The last Git Commit is below:

[[218 warrior legend step 6 walk like a warrior - slide to move complete]](https://github.com/mathew461/Udemy_JS_Games/commit/db10a9a203ff6c30479804ec1351aca7752585cd)

This shows the title of the book.

Here is the last recording viewed: **17 004 Rework Drive Motion Into Walking**

**For Pixel art follow:** Udemy - Learn Professional Pixel Art & Animation for Games

**Completed:** 3. creating props and animating them: 4. Animating your diamond

My next task will revolve around taking some reference from Pinterest and recreating something. I want to study their choice in values and their colour theory.

## The Odin Project

Currently working on Dev Tools 2: <https://www.theodinproject.com/lessons/developer-tools-2>

Which has lead to reading Chrome Developer Tools: <https://developers.google.com/web/tools/chrome-devtools>

Currently on tab: Console Overview

## Gatsby JS - The Great Gatsby Bootcamp [Full Tutorial]

Installed relevent software using Ubuntu’s Terminal and managed to set up Gatsby in Visual Studio Code.

Next up: 10. Sourcing Content from the File System - Timestamp: [1:51:32](https://www.youtube.com/watch?v=8t0vNu2fCCM&t=6692s)

**Link to video is:** <https://www.youtube.com/watch?v=8t0vNu2fCCM&t=1s>

**Logo Design**

During doing the Great Gatsby tutorial I decided if I was going to make a blog I would need to study how to design a logo, and will need further education on using Illustrator to do this. Therefore I have decided to follow this tutorial below

<https://www.udemy.com/course/professional-logo-design-crash-course/learn/lecture/2322176#overview>

First Shot



Second Shot



## Udemy - CSS - The Complete Guide 2020

**Working on the UDEMY CSS tutorial**

**Completed:** 3. Diving Deeper into CSS: 9 9. Understanding Box Sizing

## Hacking 101

This will be a hobbie, as hacking culture as always been interesting to me. First I believe educating myself on the windows command line is the best place to start.

Research these: command prompt, PowerShell, registry editor, group policy editor, and if you can, the api calls for windows itself.

<https://www.reddit.com/r/HowToHack/comments/9pc71e/my_guide_to_what_you_need_to_become_a_hacker/>

# CSS – Studies

## Class vs ID

One thing I have been confused on is the use of full stops and hashtags when setting up a CSS command. Essentially what is the difference between a class and an ID

In the CSS, a class selector is a name preceded by a **full stop** (“.”) and an ID selector is a name preceded by a **hash character** (“#”).

The difference between an ID and a class is that an ID can be used to identify one element, whereas a class can be used to identify more than one.

I am still unsure what the purpose for an ID is when you could theoretically use a class. My understanding is that you would use an ID for a single element, but my point is could a class not be used for that single element as well? If so would that not make the ID useless?

**Conclusion**

I began to study CSS again, I felt I had an adequate understanding of CSS at a level that I can build a website with it, but there are still many questions I need answered. Therefore I began watching the ‘Udemy - CSS - The Complete Guide 2020 (incl. Flexbox, Grid & Sass) tutorials. During the 2. Diving into the Basics of CSS section, I reached the sixth episode and thought I had an answer to my Class vs ID question.

**6. Theory Time – Selectors:** This episode pretty much confirmed my theory. That an ID is only usable once, therefore if you know you are only going to use CSS on that specific tag one time then it is fine to use. If you want to use the element more than once however, then a class is for you. One thing I learned here you are fine in using a Class even if you only use the Element one time, however you can only use the ID one time. ~~This has led me to believe that ID’s are pointless, since a Class can do the same thing and more.~~

**From watching 7. Understanding the Cascading Style & Specificity:** This video went into detail on the specifity and the cascading elements of CSS, basically different rules have different levels of importance. A class for example has a higher importance to a generic element. This prevents the same attributes being applied to the same asset twice, for example applying font-colour blue in the element, and then font-colour red in the class. In this example the red would be applied because the Class element is superior to the generic one. Above this however is the ID element, therefore if I was to write font-colour yellow, in the id element, then that would be the colour shown. This has led me to believe that the ID element has some importance.

# The Odin Project Studies

**Part 1**

* How do you declare a variable?

It would seem there are three ways to declare a Variable. The var: statement would seem to be the old way to do it, but now you can also use let: and const:. From what I have read it is best to use let:

* What are three different ways to declare a variable?

Let: and const: see above.

* Which one should you use when?

The var keyword is almost the same as let. It also declares a variable, but in a slightly different, “old-school” way.

Let seems to be the method to use default. The advice seems to state that if in doubt use let: var seems out-dated.

Variables declared using const are called “constants”. They cannot be reassigned. An attempt to do so would cause an error:

* What are the rules for naming variables?

A variable name should have a clean, obvious meaning, describing the data that it stores.

* Use human-readable names like userName or shoppingCart.
* Stay away from abbreviations or short names like a, b, c, unless you really know what you’re doing.
* Make names maximally descriptive and concise. Examples of bad names are data and value. Such names say nothing. It’s only okay to use them if the context of the code makes it exceptionally obvious which data or value the variable is referencing.
* Agree on terms within your team and in your own mind. If a site visitor is called a “user” then we should name related variables currentUser or newUser instead of currentVisitor or newManInTown.
* What are operators, operands, and operations?

We know many **operators** from school. They are things like addition +, multiplication \*, subtraction -, and so on.

An **operand** – is what operators are applied to. For instance, in the multiplication of 5 \* 2 there are two operands: the left operand is 5 and the right operand is 2. Sometimes, people call these “arguments” instead of “operands”.

An **operator** is unary if it has a single operand. For example, the unary negation - reverses the sign of a number:

The basic arithmetic operations are addition, **subtraction**, **multiplication** and division, although this subject also includes more advanced operations, such as manipulations of percentages, square roots, exponentiation, logarithmic functions, and even trigonometric functions, in the same vein as logarithms

* What is concatenation and what happens when you add numbers and strings together?

In formal language theory and computer programming, string concatenation is the operation of joining character strings end-to-end. For example, the concatenation of "snow" and "ball" is "snowball".

* What are the different types of operators in JavaScript?

Arithmetic Operators

Comparison (Relational) Operators

Bitwise Operators

Logical Operators

Assignment Operators

Special Operators

* What is the difference between == and ===?

10 == Equal x == y

10 === Strict equal x === y

When using the == operator, equal numbers are equal:

When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

* What are operator precedence values?

If an expression has more than one operator, the execution order is defined by their precedence, or, in other words, the default priority order of operators.

From school, we all know that the multiplication in the expression 1 + 2 \* 2 should be calculated before the addition. That’s exactly the precedence thing. The multiplication is said to have a higher precedence than the addition.

Parentheses override any precedence, so if we’re not satisfied with the default order, we can use them to change it. For example, write (1 + 2) \* 2.

There are many operators in JavaScript. Every operator has a corresponding precedence number. The one with the larger number executes first. If the precedence is the same, the execution order is from left to right.

* What are the increment/decrement operators?

Increasing or decreasing a number by one is among the most common numerical operations.

So, there are special operators for it:

Increment ++ increases a variable by 1:

Decrement -- decreases a variable by 1:

* What is the difference between prefixing and post-fixing them?

When the operator goes after the variable, it is in “postfix form”: counter++.

The “prefix form” is when the operator goes before the variable: ++counter.

Both of these statements do the same thing: increase counter by 1.

Is there any difference? Yes, but we can only see it if we use the returned value of ++/--.

Let’s clarify. As we know, all operators return a value. Increment/decrement is no exception. The prefix form returns the new value while the postfix form returns the old value (prior to increment/decrement).

* What are assignment operators?

Let’s note that an assignment “=” is also an operator.

It is listed in the precedence table with the very low priority of 3.

* What is the “Unary +” Operator?

An operator is unary if it has a single operand. For example, the unary negation - reverses the sign of a number:

**Part 2**

* What are the eight data types of JavaScript?

In JavaScript there are two different kinds of data: **primitives**, and **objects**. A primitive is simply a data type that is not an object, and has no methods.

* + **Number:** There is only one type of Number in JavaScript. Numbers can be written with or without a decimal point. A number can also be +Infinity, -Infinity, and NaN (not a number).
  + **Bigint:** is for integer numbers of arbitrary length.
  + **Strings:** are used for storing text. Strings must be inside of either double or single quotes. In JS, Strings are immutable (they cannot be changed).
  + **Boolean**: represents only one of two values: **true**, or **false**. Think of a Boolean as an on/off or a yes/no switch.
  + **Null:** has one value: **null.**It is explicitly nothing.
  + **Undefined:** for unassigned values – a standalone type that has a single value undefined.
  + **Object:** for more complex data structures.
  + **Symbol:** for unique identifiers. (relatively new)
* Which data type is NOT primitive?

**Object** is not a primitive data type

An object is a collection of properties. These properties are stored in key/value pairs. Properties can reference any type of data, including objects and/or primitive values.

**BigInt** is a built-in object that provides a way to represent whole numbers larger than 253 - 1, which is the largest number JavaScript can reliably represent with the [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number) primitive and represented by the [Number.MAX\_SAFE\_INTEGER](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/MAX_SAFE_INTEGER) constant. **BigInt** can be used for arbitrarily large integers.

* What is the difference between single, double, and backtick quotes for strings?

In JavaScript, there are 3 types of quotes.

1. Double quotes: "Hello".
2. Single quotes: 'Hello'.
3. Backticks: `Hello`.

Double and single quotes are “simple” quotes. There’s practically no difference between them in JavaScript.

Backticks are “extended functionality” quotes. They allow us to embed variables and expressions into a string by wrapping them in ${…}, for example:

let name = "John";

// embed a variable

alert( `Hello, ${name}!` ); // Hello, John!

// embed an expression

alert( `the result is ${1 + 2}` ); // the result is 3

* Which type of quote lets you embed variables/expressions into a string?

Backticks, See above.

* How do you embed variables/expressions into a string?

by wrapping them in ${…}, See information on Backticks above.

* How do you escape characters in a string?

The backslash ( \ ) **character** is used to **escape characters** that otherwise have a special meaning, such as newline, backslash itself, or the quote **character**.

* What is the difference between slice/substring/substr?

slice() works like substring() with a few different behaviors.

Syntax: string.slice(start, stop);

Syntax: string.substring(start, stop);

**What they have in common:**

1. If start equals stop: returns an empty string
2. If stop is omitted: extracts characters to the end of the string
3. If either argument is greater than the string's length, the string's length will be used instead.

**Distinctions of** [substring()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/substring)**:**

1. If start > stop, then substring will swap those 2 arguments.
2. If either argument is negative or is NaN, it is treated as if it were 0.

**Distinctions of** [slice()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/slice)**:**

1. If start > stop, slice() will return the empty string. ("")
2. If start is negative: sets char from the end of string, exactly like substr() in Firefox. This behavior is observed in both Firefox and IE.
3. If stop is negative: sets stop to: string.length – Math.abs(stop) (original value), except bounded at 0 (thus, Math.max(0, string.length + stop)) as covered in the [ECMA specification](https://www.ecma-international.org/ecma-262/9.0/index.html#sec-string.prototype.slice).

* What are methods?

JavaScript methods are actions that can be performed on objects. A JavaScript method is a property containing a function definition.

|  |  |
| --- | --- |
| **Property** | **Value** |
| firstName | John |
| lastName | Doe |
| age | 50 |
| eyeColor | Blue |
| fullName | function() {return this.firstName + " " + this.lastName;} |

* What are the three logical operators and what do they stand for?

There are **three logical operators** in JavaScript: || (OR), && (AND), ! (NOT). Although **they** are called “**logical**”, **they can** be applied to values of any type, not only boolean. Their result **can** also be of any type.

* What are the comparison operators?

A [comparison operator](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators) compares its operands and returns a logical value based on whether the comparison is true. The operands can be numerical, string, logical, or object values. Strings are compared based on standard lexicographical ordering, using Unicode values. In most cases, if the two operands are not of the same type, JavaScript attempts to convert them to an appropriate type for the comparison. This behavior generally results in comparing the operands numerically. The sole exceptions to type conversion within comparisons involve the === and !== operators, which perform strict equality and inequality comparisons. These operators do not attempt to convert the operands to compatible types before checking equality. The following table describes the comparison operators in terms of this sample code:

|  |  |  |
| --- | --- | --- |
|  |  |  |
| **Operator** | **Description** | **Examples returning true** |
| [Equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Equality) (==) | Returns true if the operands are equal. | 3 == var1  "3" == var1  3 == '3' |
| [Not equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Inequality) (!=) | Returns true if the operands are not equal. | var1 != 4 var2 != "3" |
| [Strict equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Identity) (===) | Returns true if the operands are equal and of the same type. See also [Object.is](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/is) and [sameness in JS](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Equality_comparisons_and_sameness). | 3 === var1 |
| [Strict not equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Nonidentity) (!==) | Returns true if the operands are of the same type but not equal, or are of different type. | var1 !== "3" 3 !== '3' |
| [Greater than](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Greater_than_operator) (>) | Returns true if the left operand is greater than the right operand. | var2 > var1 "12" > 2 |
| [Greater than or equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Greater_than_or_equal_operator) (>=) | Returns true if the left operand is greater than or equal to the right operand. | var2 >= var1 var1 >= 3 |
| [Less than](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Less_than_operator) (<) | Returns true if the left operand is less than the right operand. | var1 < var2 "2" < 12 |
| [Less than or equal](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Comparison_Operators#Less_than_or_equal_operator) (<=) | Returns true if the left operand is less than or equal to the right operand. | var1 <= var2 var2 <= 5 |

* What is nesting?

function x () {  
function y() {  
// something;  
}  
}

* What are truthy and falsy values?

**Truthy and falsy** in JavaScript. In JavaScript, **truthy** are expressions which evaluates to boolean true **value** and **falsy** evaluates to boolean false **value**. Unlike other languages, true and false **values** are not limited to boolean data types and comparisons.

* What are the falsy values in Javascript?

A falsy value is something which evaluates to FALSE, for instance when checking a variable. There are only six falsey values in JavaScript: undefined, null, NaN, 0, "" (empty string), and false of course.

* What is the syntax for an if/else if/else conditional?

if (*condition1*) {  
  *// block of code to be executed if condition1 is true*} else if (*condition2*) {  
  *// block of code to be executed if the condition1 is false and condition2 is true*  
} else {  
  *// block of code to be executed if the condition1 is false and condition2 is false*}

* What is the syntax for a switch statement?

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

* What is the syntax for a ternary operator?

The **conditional (ternary) operator** is the only JavaScript operator that takes three operands: a condition followed by a question mark (?), then an expression to execute if the condition is [truthy](https://developer.mozilla.org/en-US/docs/Glossary/truthy) followed by a colon (:), and finally the expression to execute if the condition is [falsy](https://developer.mozilla.org/en-US/docs/Glossary/falsy). This operator is frequently used as a shortcut for the [if](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else) statement.

*condition* ? *exprIfTrue* : *exprIfFalse*

***condition***

An expression whose value is used as a condition.

***exprIfTrue***

An expression which is evaluated if the *condition* evaluates to a [truthy](https://developer.mozilla.org/en-US/docs/Glossary/truthy) value (one which equals or can be converted to true).

***exprIfFalse***

An expression which is executed if the *condition* is [falsy](https://developer.mozilla.org/en-US/docs/Glossary/falsy) (that is, has a value which can be converted to false).

* What is the relationship between null and undefined?

**Null** means an empty or non-existent value. Null is assigned, and explicitly means nothing.

*null is also an object. Interestingly, this was actually an error in the original JavaScript implementation:*

**Undefined** means a variable has been declared, but the value of that variable has not yet been defined. Unlike null, undefined is of the type undefined:

* What are conditionals?

Conditional statements control behavior in JavaScript and determine whether or not pieces of code can run.

There are multiple different types of conditionals in JavaScript including:

“If” statements: where if a condition is true it is used to specify execution for a block of code.

“Else” statements: where if the same condition is false it specifies the execution for a block of code.

“Else if” statements: this specifies a new test if the first condition is false.

Now that you have the basic JavaScript conditional statement definitions, let’s show you examples of each.