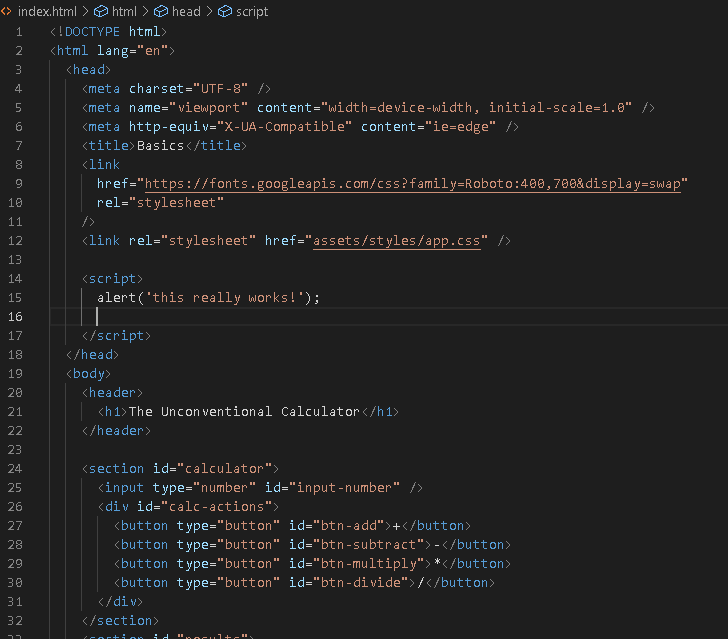
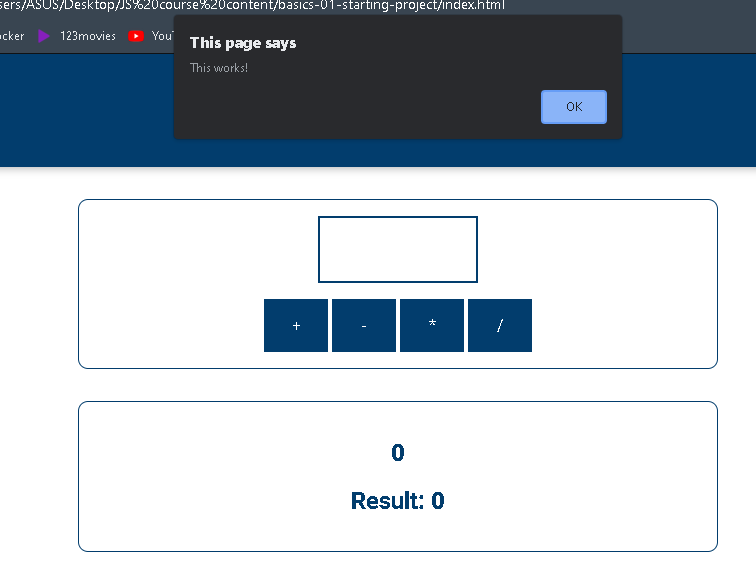
**Basics: Variables, DataTypes, Operators & Functions**

* So a variable is a data container which holds some data and in Javascript, we create a variable by using the let keyword which you see here on the left, then a name of your choice with some naming rules
* Now with a variable created, you can also always reassign it and store a new value in there
* Now I only need the let keyword, which Javascript understands, if you introduce a variable for the first time
* That variable you could say, you have constants,for example let's say you have a total amount of users in your application which is kind of fixed. Now that's still a data container but instead of with the let keyword, you create it with the const

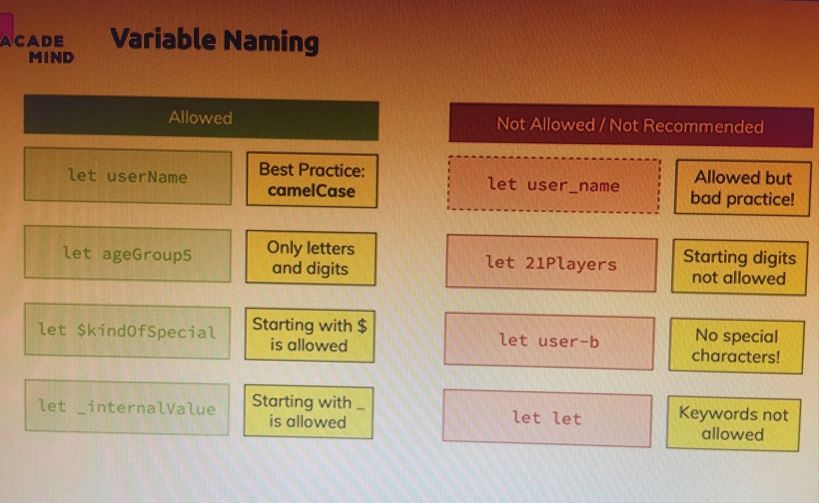
**Adding JavaScript to the Website**

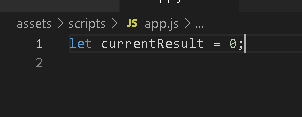




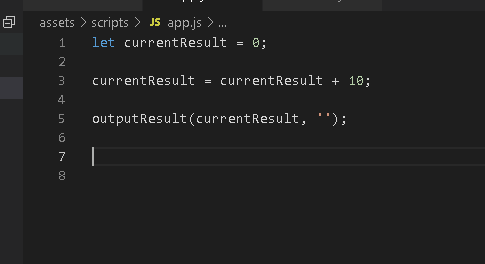
**Declaring & Defining Variables**

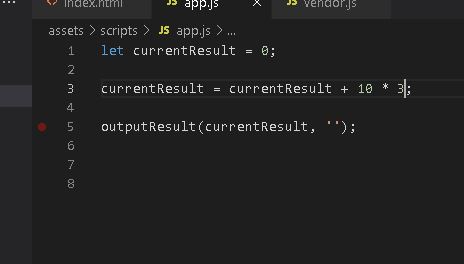
* your name starts with a lowercase character and is only one word
* there are no blanks in there, no whitespace, that would actually be forbidden but every word inside of the word, like name here in userName starts with an upper case character to make that more readable.
* Javascript is case sensitive and this means that it really matters whether a character is a capital character or not
* the casing in the names you choose really matters and that's important to keep in mind.
* You can also use the dollar sign special character and your variable name can even start with it
* variables and constants must not start with a digit. Now there also are special rules when it comes to
* special characters, you can use dollar sign and underscore at the beginning of your variable or anywhere in your variable but you must not use other special characters, neither at the beginning of your variable

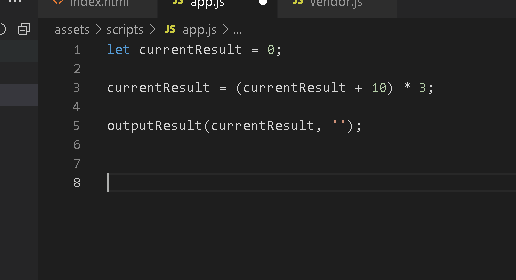


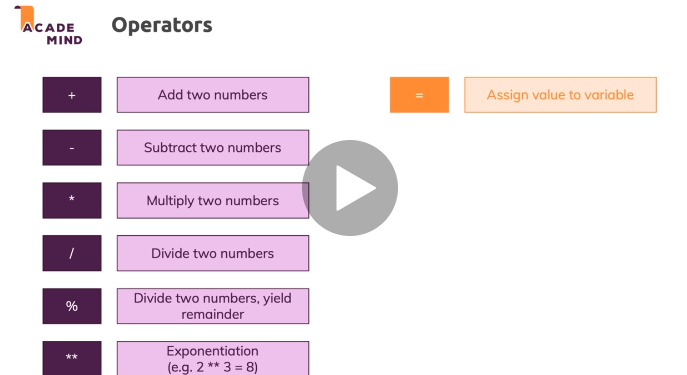


**Working with Variables & Operators**

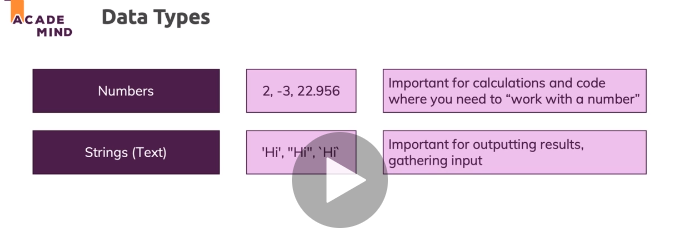


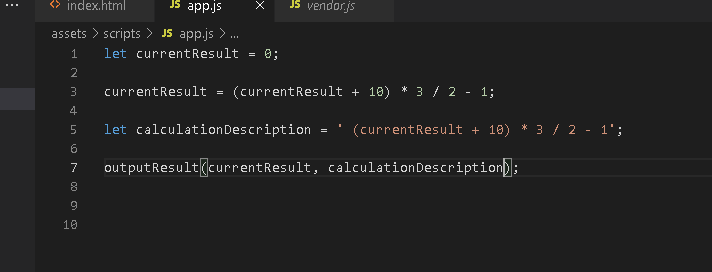


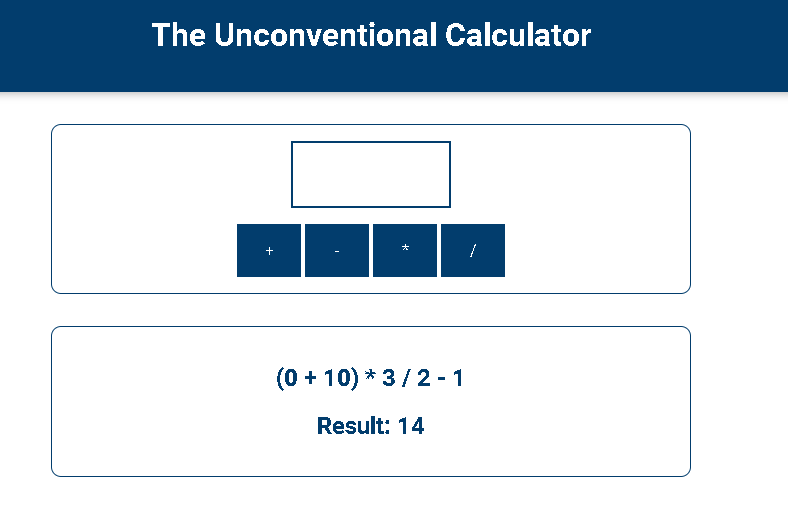




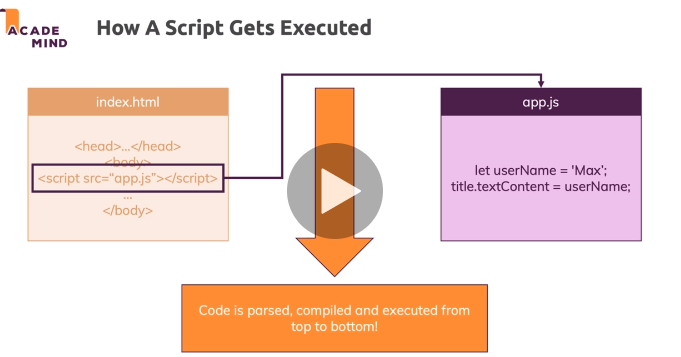
**Data Types: Numbers & Strings (Text)**

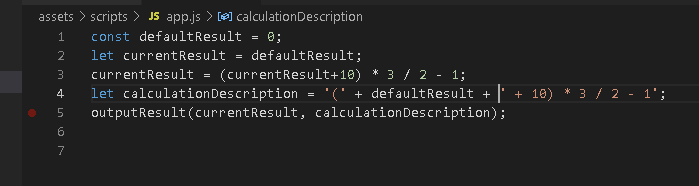


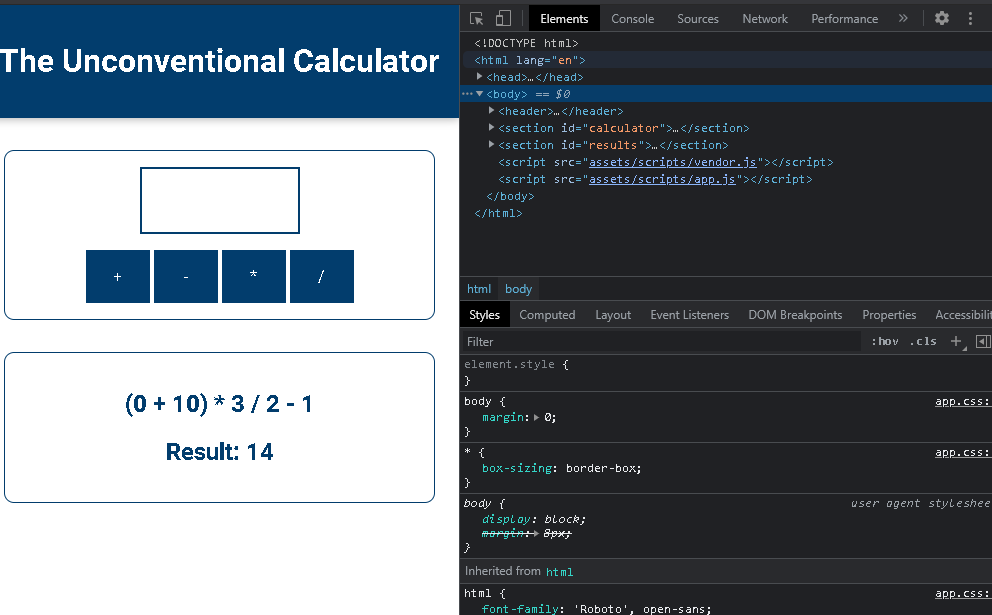




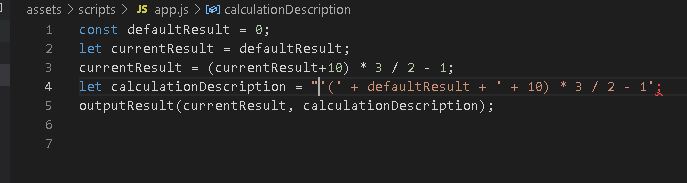
**Using Constants**

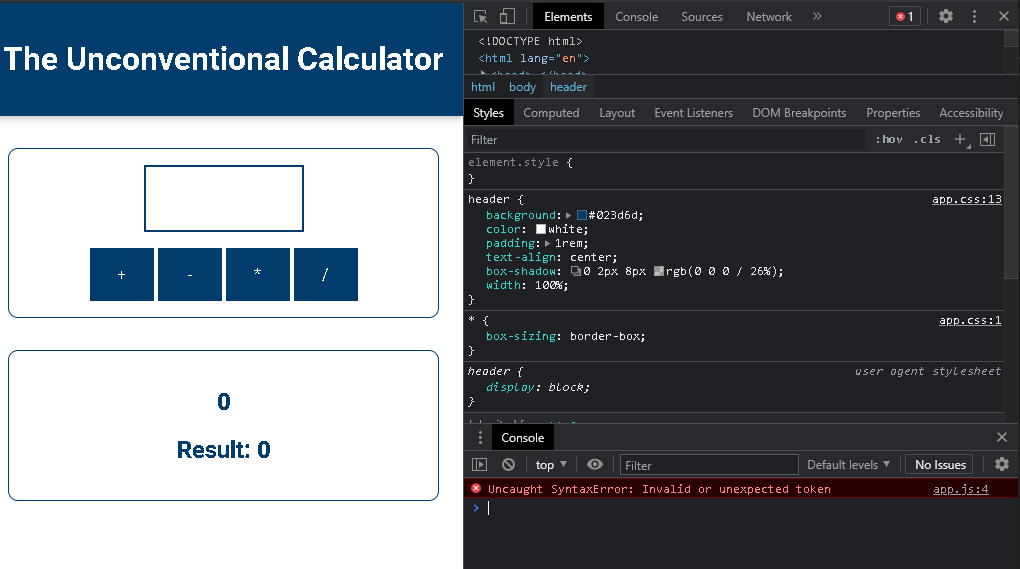


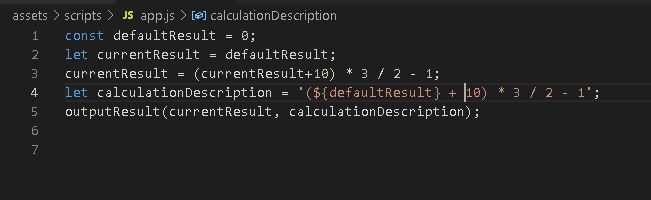


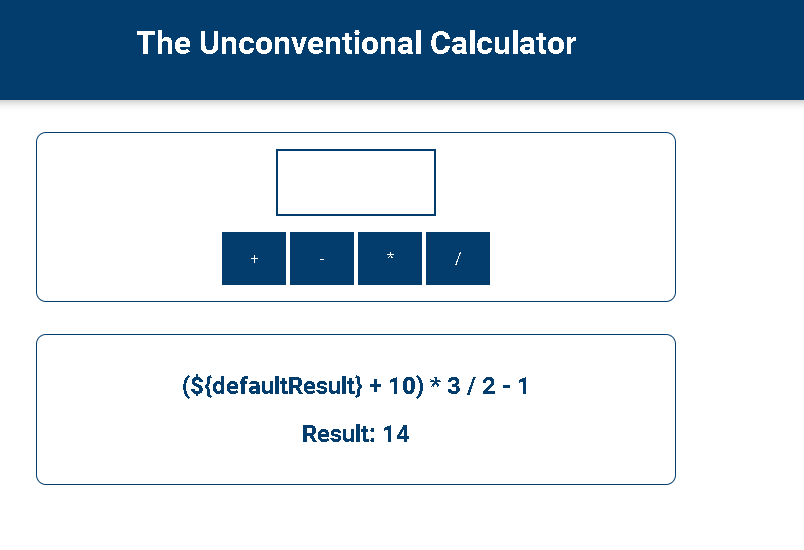


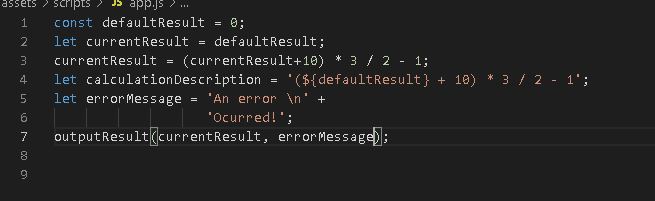
**More on Strings**

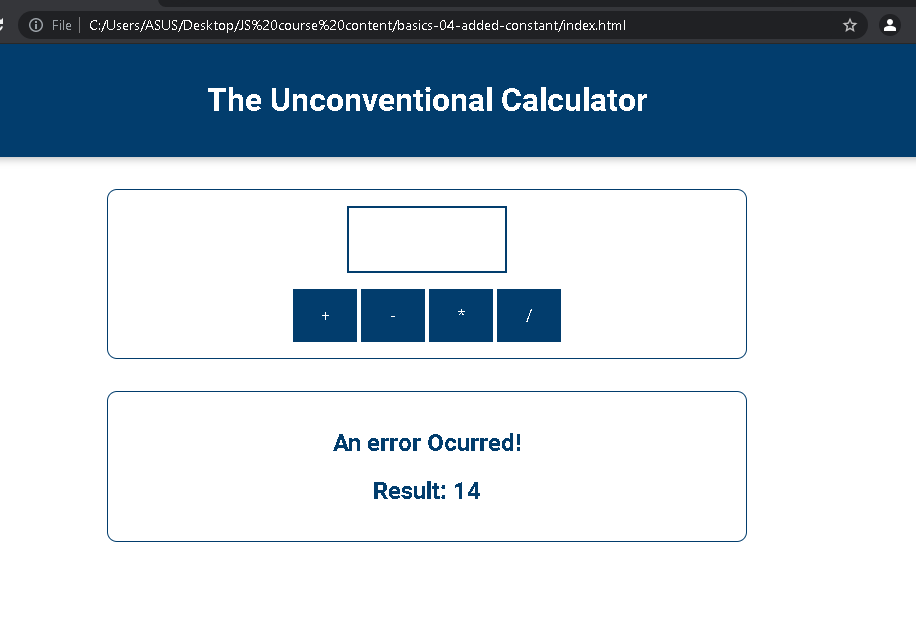




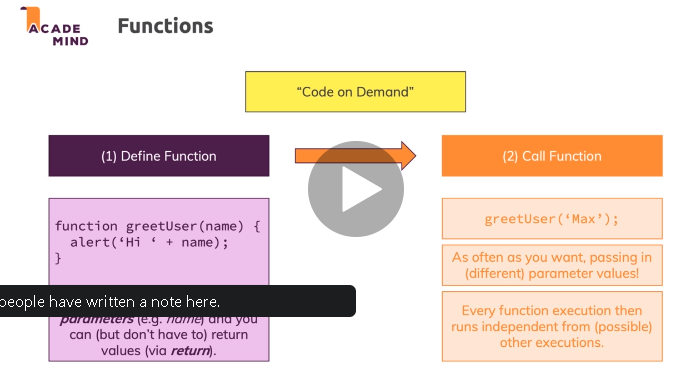




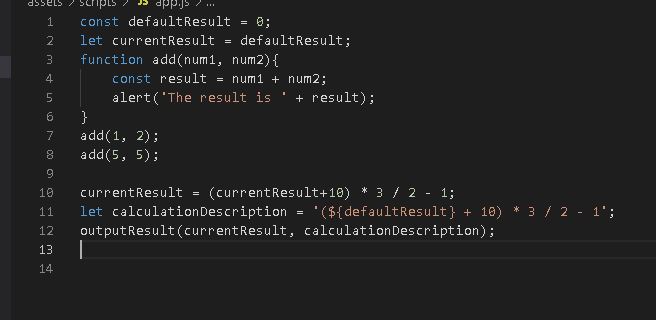


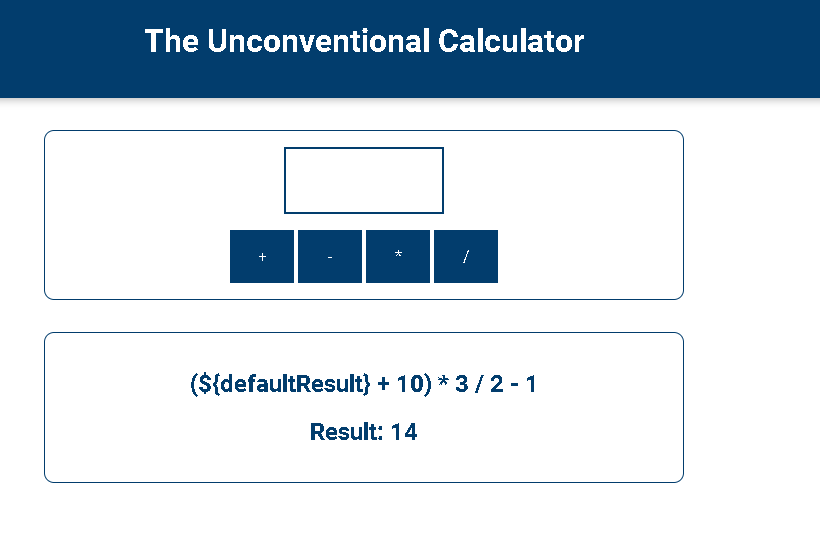


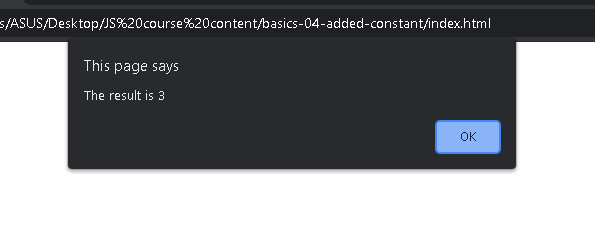
**Introducing Functions**



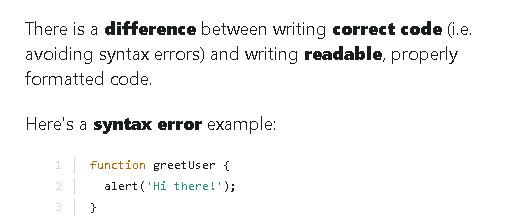
**Adding A Custom Function**



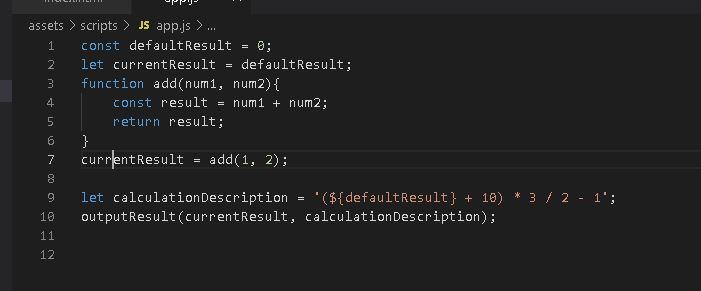


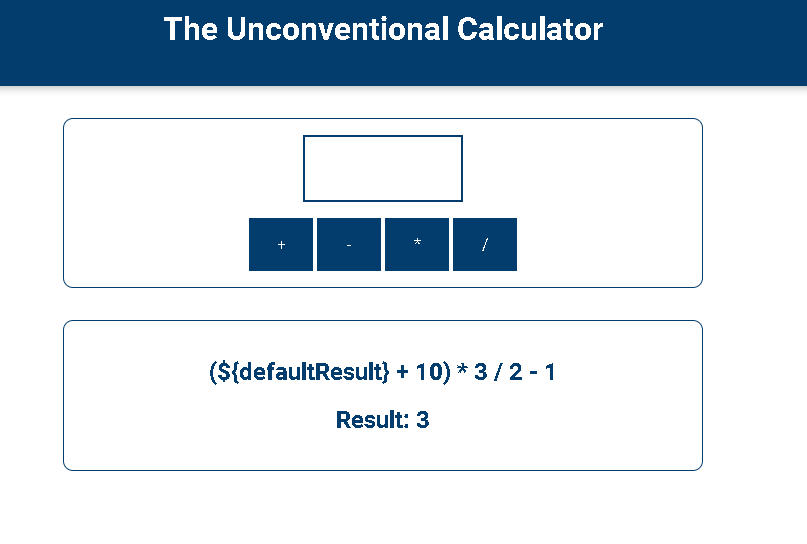


**Code Styles, Conventions & Syntax**



**Returning Values**





**An Introduction to Global & Local Scope**

So the order matters for variables and constants

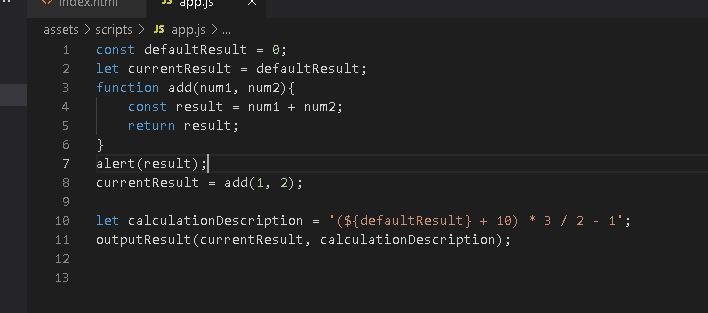
You can access any variable

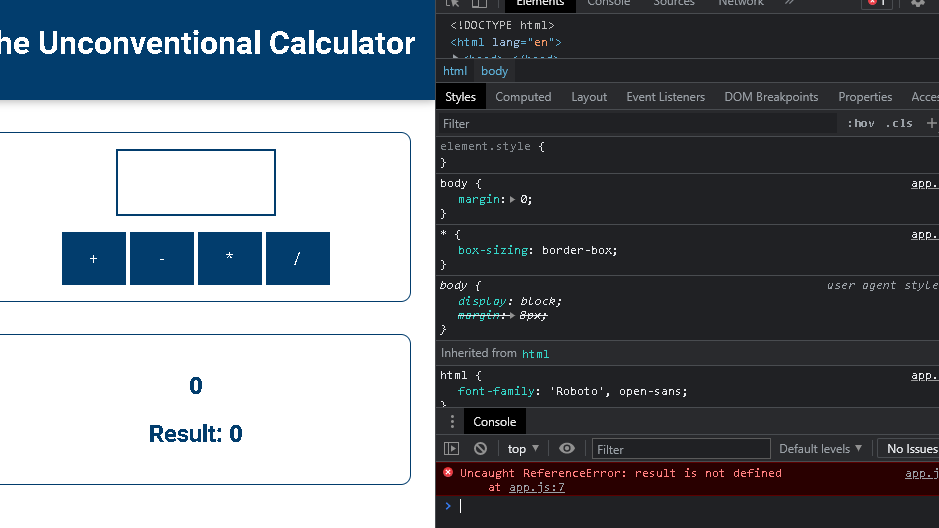
or constant defined outside of the function, so you can read variables or constants defined globally

Variables declared within a JavaScript function, become **LOCAL** to the function.

Variables declared **Globally** (outside any function) have **Global Scope**.

**Global** variables can be accessed from anywhere in a JavaScript program.





**Shadowed Variables**

You learned about local ("function-internal") variables and global variables.

What happens if you have this code?

    1. let userName = 'Max';

    2. function greetUser(name) {

    3.   let userName = name;

    4.   alert(userName);

    5. }

    6. userName = 'Manu';

    7. greetUser('Max');

This **will actually show** an alert that says 'Max' (NOT 'Manu').

You might've expected that an error gets thrown because we use and declare userName more than once - and as you learned, that is not allowed.

It indeed is **not allowed on the same level/ in the same scope.**

So this would fail:

    1. let userName = 'Max';

    2. let userName = 'Manu';

Why does it work in the first code snippet though?

Because we first create a global variable userName via

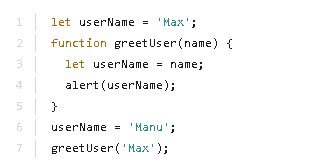
    1. let userName = 'Max';

But then we never re-declare that on the global level (that would not be allowed).

We only declare another variable inside of the function. But since variables in functions get their **own scope**, JavaScript does something which is called **"shadowing"**.

It **creates a new variable on a different scop**e - this variables does not overwrite or remove the global variable by the way - **both** **co-exist**.

When referring to userName inside of the greetUser function we now **always refer to the local, shadowed variable**. Only **if no such local variable existed**, JavaScript would **fall back to the global variable**.



It indeed is **not allowed on the same level/ in the same scope.**

We only declare another variable inside of the function. But since variables in functions get their **own scope**, JavaScript does something which is called **"shadowing"**.

**More about the "return" Statement**

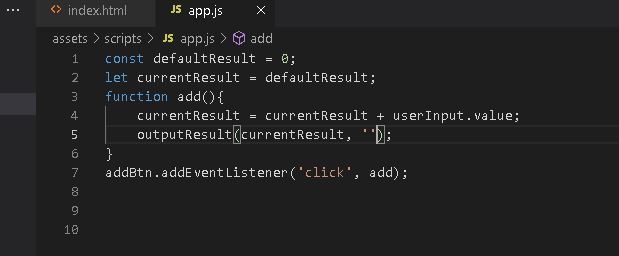
The return statement stops the execution of a function and returns a value from that function.

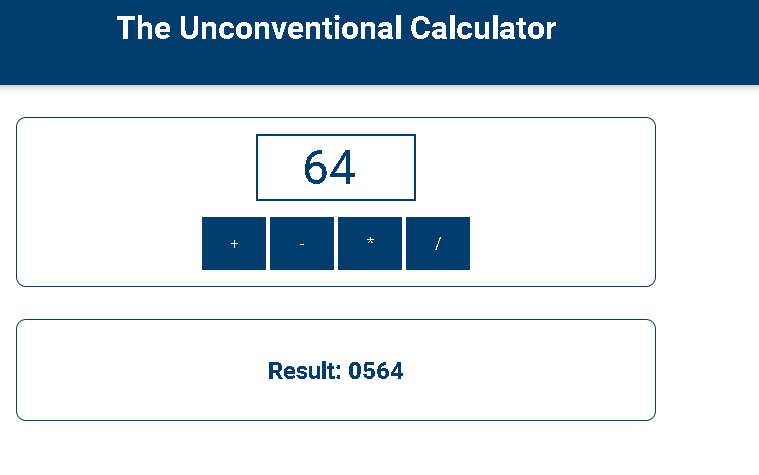
**"Indirect" vs "Direct" Function Execution Summary**

**Sometimes however**, you **don't want to execute the function immediately**. You rather want to "tell JavaScript" that it should execute a certain function **at some point in the future**

someButton.addEventListener('click', add);

This snippet would tell JavaScript: *"Hey, when the button is clicked, go ahead and execute add."*.





**Converting Data Types**

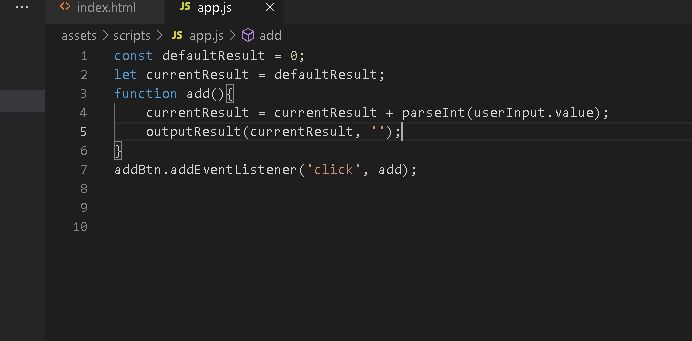
JavaScript variables can be converted to a new variable and another data type:

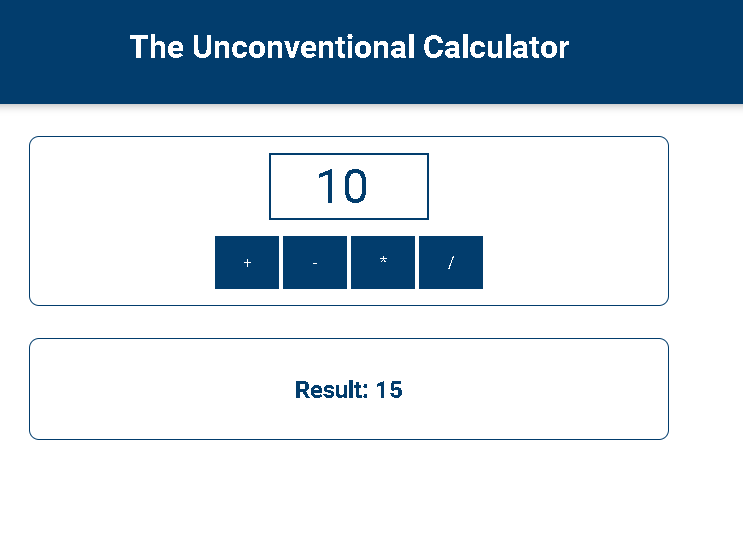
* By the use of a JavaScript function
* **Automatically** by JavaScript itself

 So parseInt and parseFloat are functions built into Javascript

that will in the end take a string as an input, take a string as a parameter you pass in and will then

convert this text into a number





**Mixing Numbers & Strings**

3 + '3' => '33'

in JavaScript.

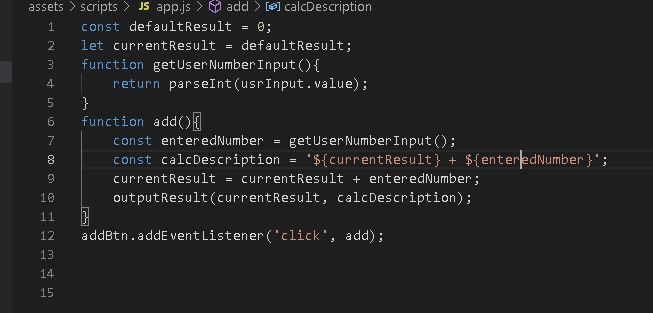
That happens because the + operator also supports strings (for string concatenation).

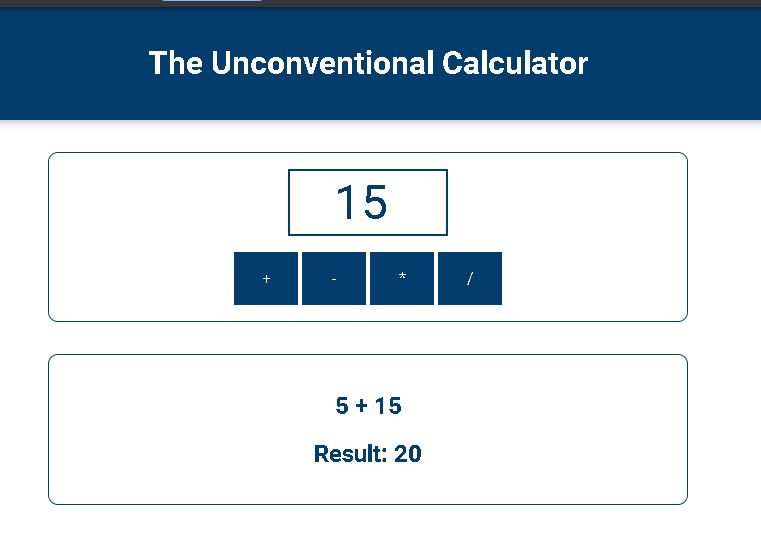
'hi' - 'i' => NaN

NaN is covered a little later, the core takeaway is that you can't generate a string of 'h' with the above code. Only + supports both strings and numbers.

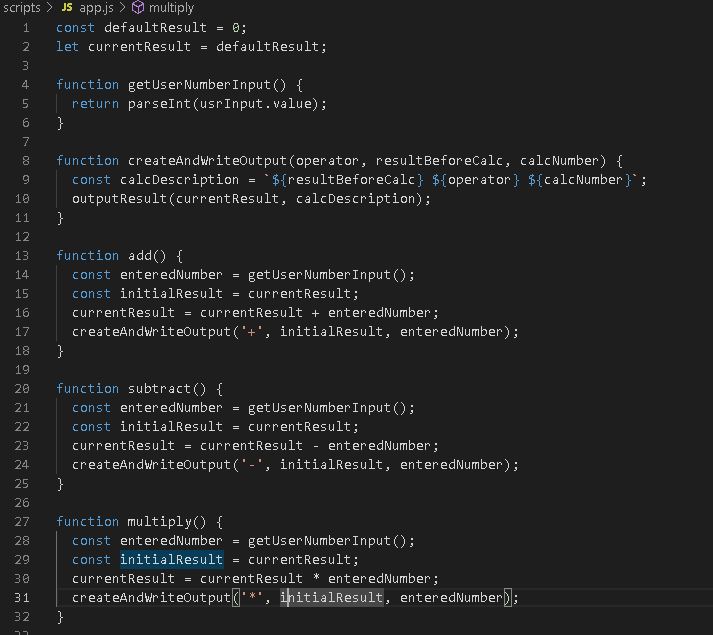
Just 3 + '3' yields '33' because here JavaScript uses the *"I can combine text"* mode of the + operator and generates a string instead of a number.

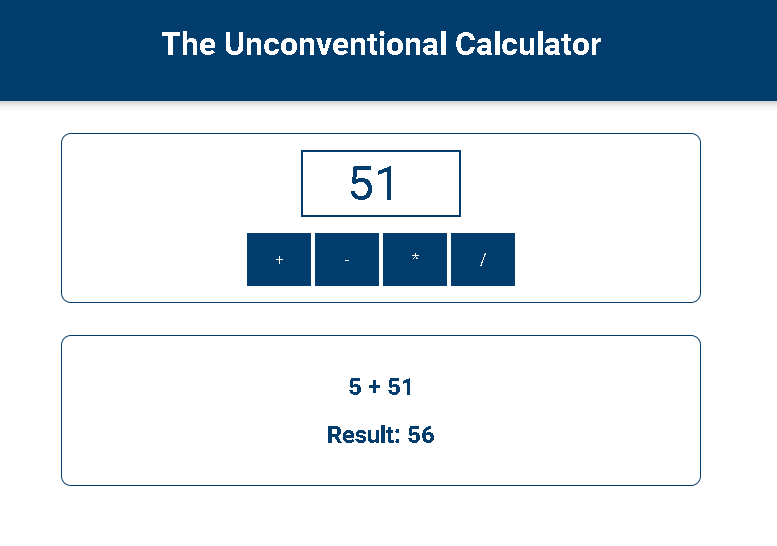
**Splitting Code into Functions**





**Connecting all Buttons to Functions**





**Working with Code Comments**

 Now a comment in Javascript can be added in two ways,

you can add it here, like you add regular code by adding two forward slashes,

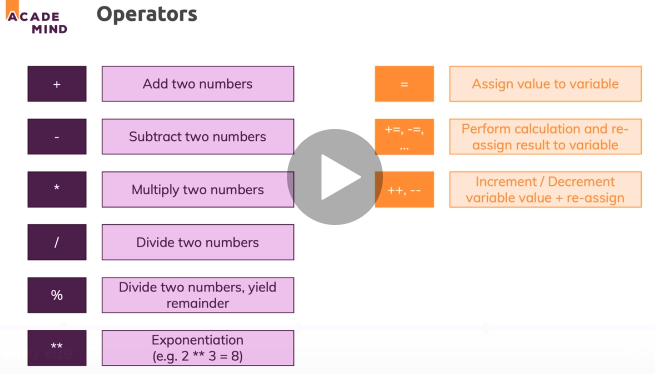
this is a comment now

it's not executed as code, instead this is just there to comment your code, to make it more understandable.

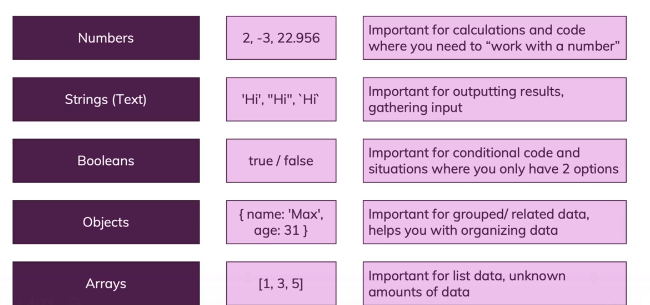
If you have a longer a comment to write which should be split across multiple lines

you can of course add another line of comments here with two other forward slashes but you can also add a block comment by adding a forward slash and then a star and now you see everything is green down there because now everything after that is treated as a comment,

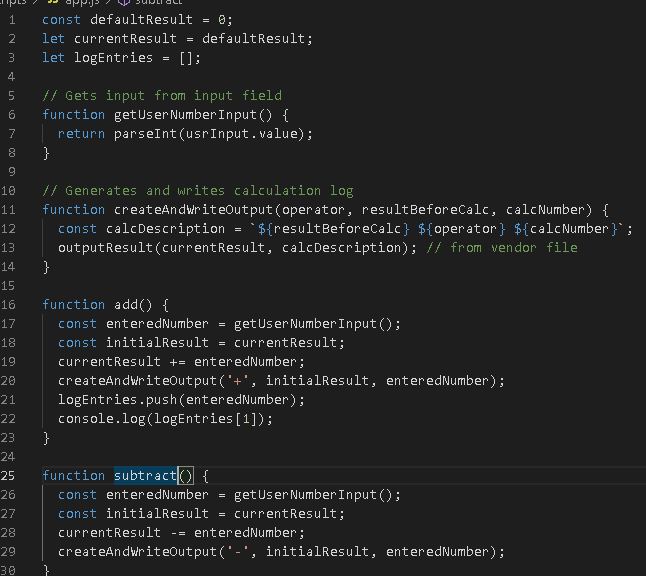
**More Operators!**

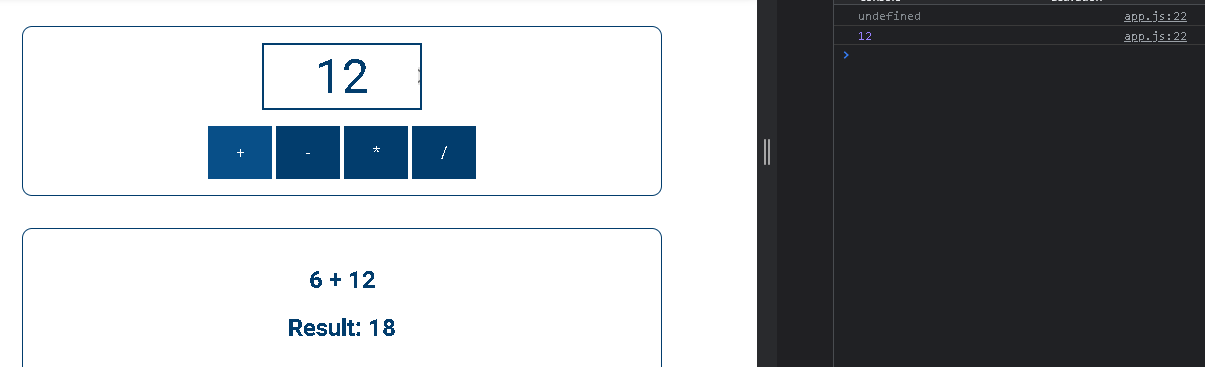


**More Core Data Types**

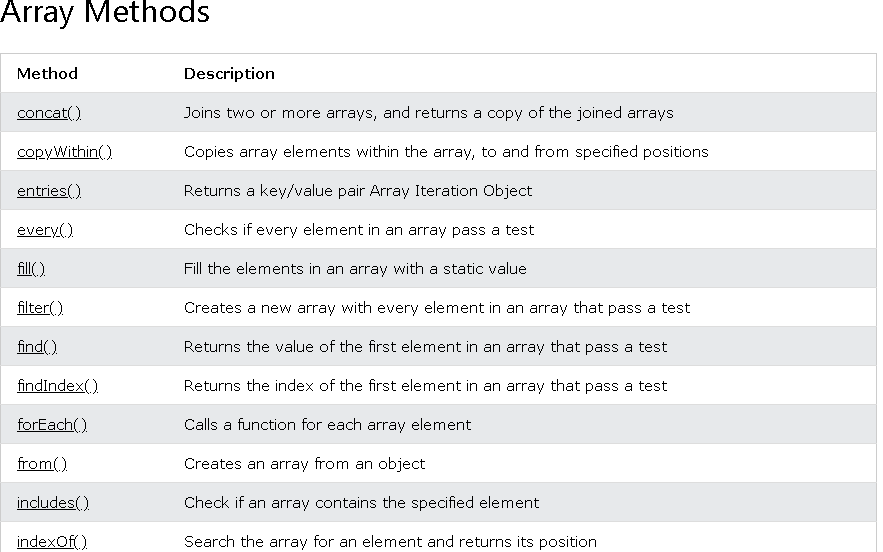


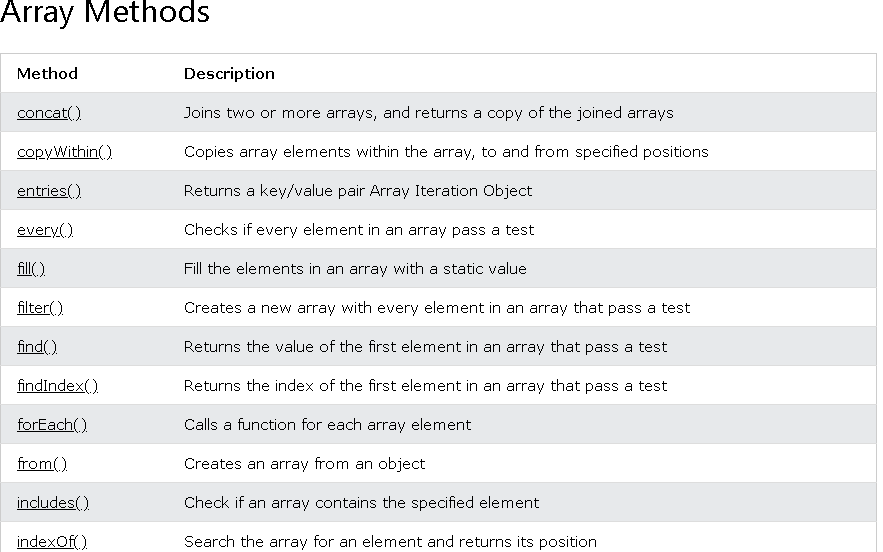
**Using Arrays**



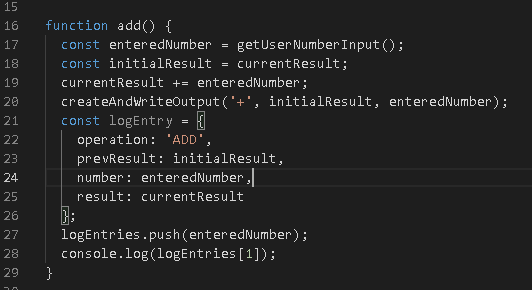


The Array object is used to store multiple values in a single variable:





**Creating Objects**



**Objects - Common Syntax**

You learned the correct syntax for creating an object in the last lecture:

    1. const user = {

    2.     name: 'Max',

    3.     age: 30

    4. };

A couple of important things:

1. You use {} to "group the data" - a semicolon (;) is used after the closing }. On functions, we didn't do that. As a rule of thumb, you can keep in mind that **a semicolon is used after {} if the {} are on the right side of the equal sign**!
2. key-value pairs are **separated via a comma** (,), **NOT** via a semicolon. Using a semicolon inside of an object (i.e. between {}), would be a **syntax error**!
3. Values are assigned to keys/ properties **via a colon** (:), NOT via an equal sign (=). Using an equal sign inside of an object (i.e. between {}), would be a **syntax error**!

I.e. this would be WRONG and would throw an error:

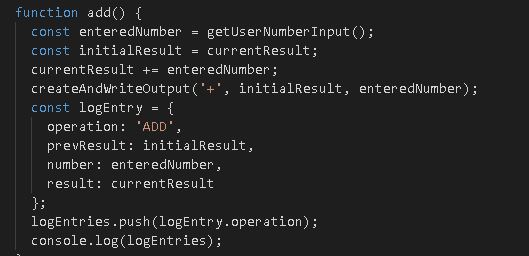
    1. const worstPossibleUser = {

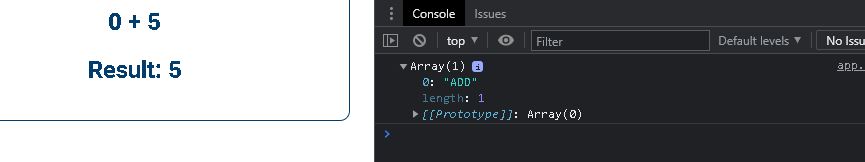
    2.     name = 'Max';

    3.     age = 30;

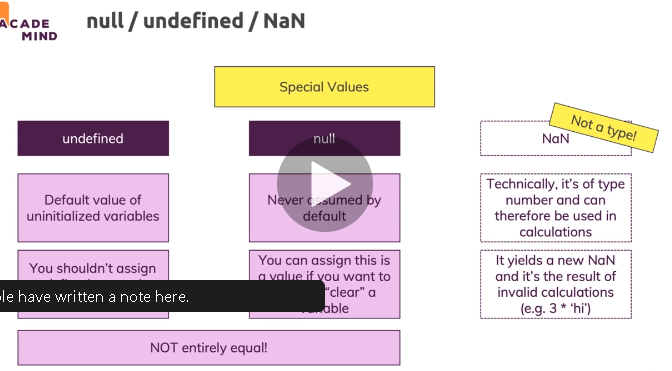
    4. };

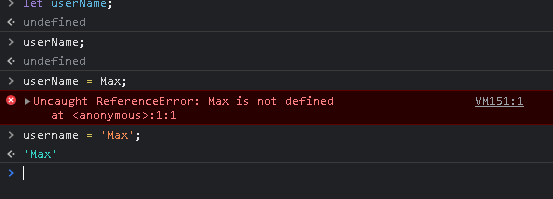
**Accessing Object Data**





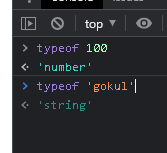
**undefined, null & NaN**





**The "typeof" Operator**

You can use the typeof operator to find the data type of a JavaScript variable.



**Importing Scripts Correctly with "defer" & "async"**

