



*Actually called ECMAScript but the name JavaScript stuck.

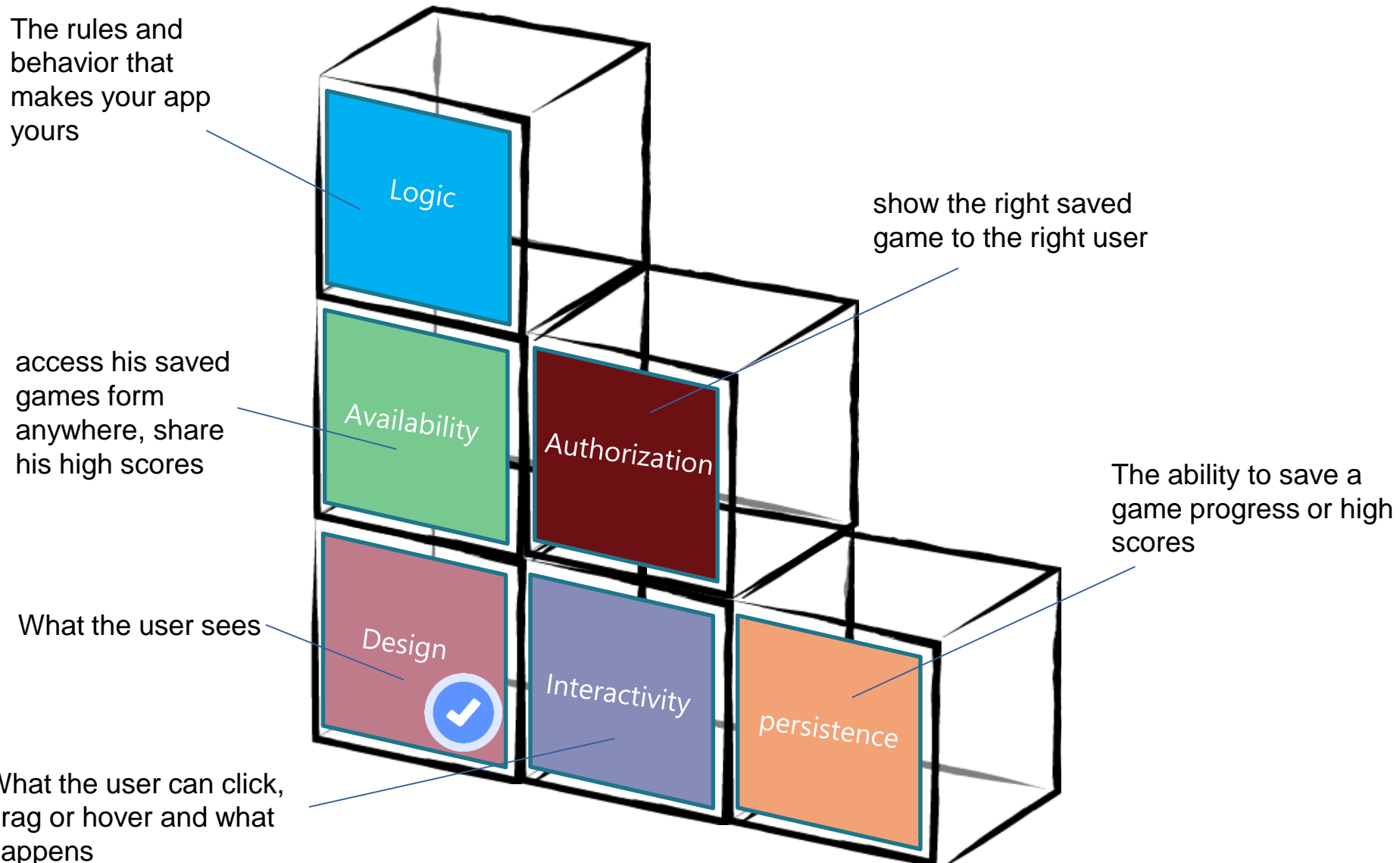
Agenda

- Why JavaScript
- User input/output(io): alert, console.log(), prompt, confirm
- Run JS in the console
- Data types in JS
- Dynamic typing vs Static Typing
- Variables
- For / if / while
- Strings
- Coercion in JS
- Console in Browser, code snippets
- Debugging + Developer tools

Let's Go Back in Time



The elements of a webapp



Interactivity

What?

Interactivity

Ehh.. what?

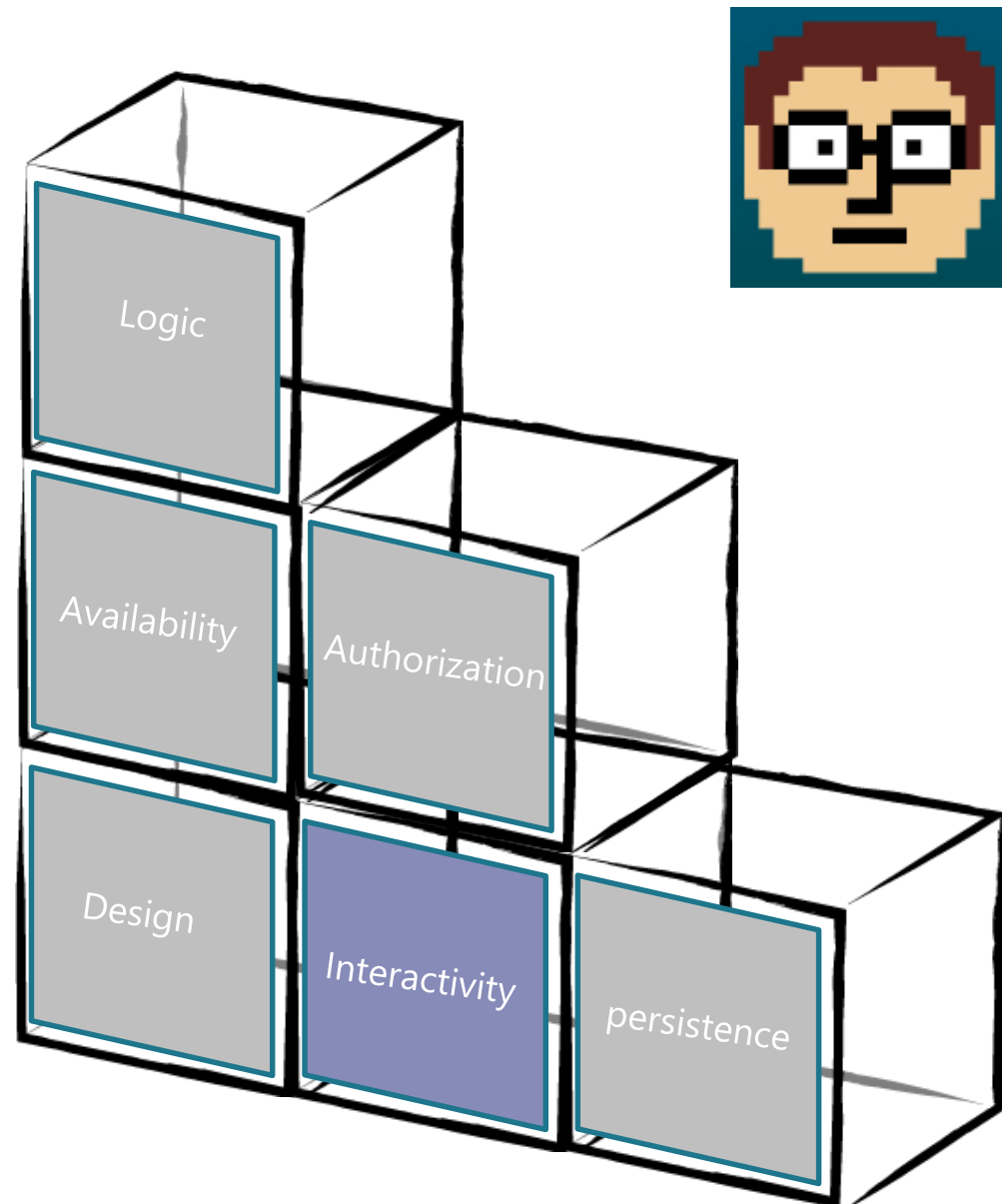
What the user can click, drag or hover
and what happens

How?

- Mostly java script

When?

Later this week



Why learn JavaScript

- Until now we had static web pages 😞
- JS will allow us to be interactive and react to user actions (more about it later).
- Vanilla JS, (JS with no add ons) is the only language which runs in all browsers.

Logic

What?

Logic

Ehh.. what?

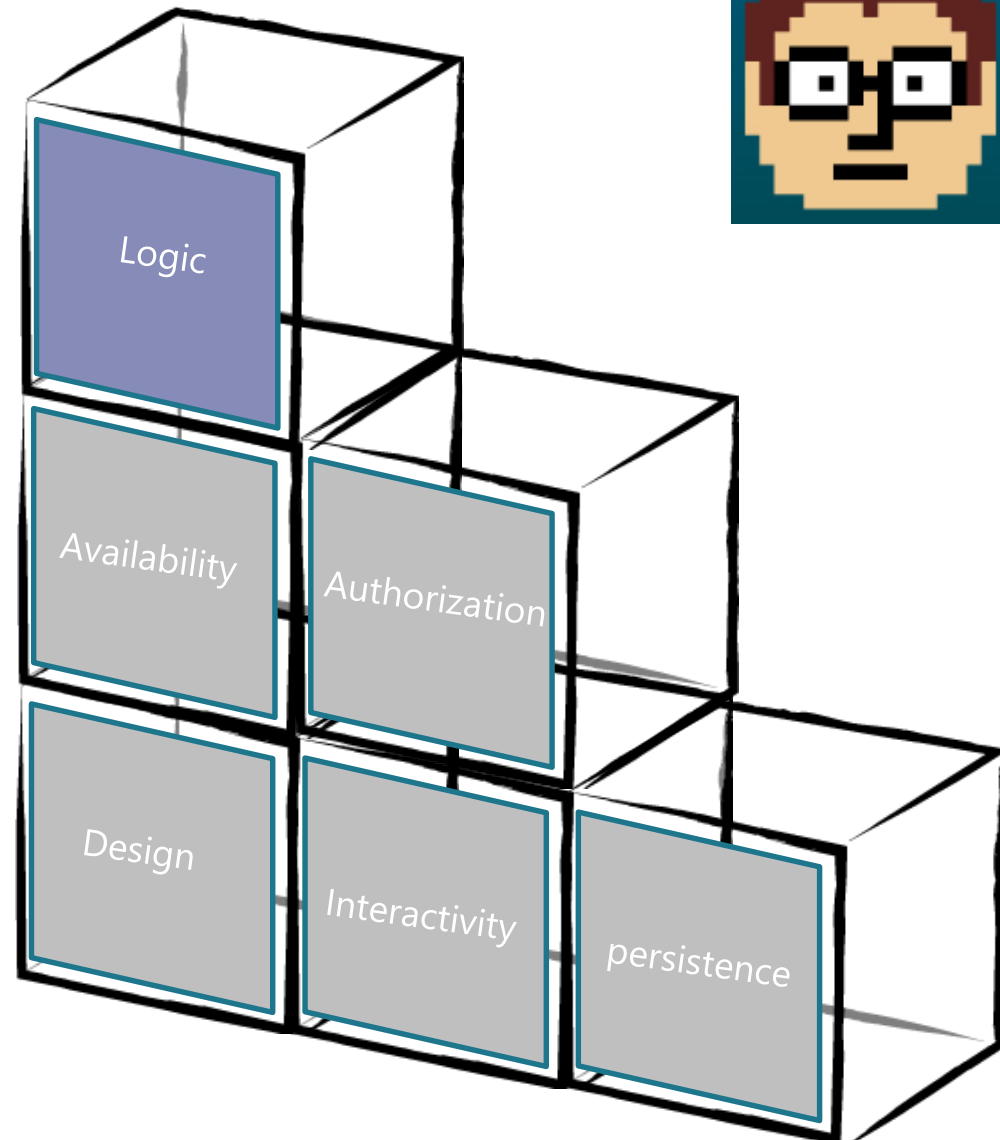
Add custom behavior to the app.

How?

Java script

When?

Starting now



Logic

In the past, most of the logic was in the back end.

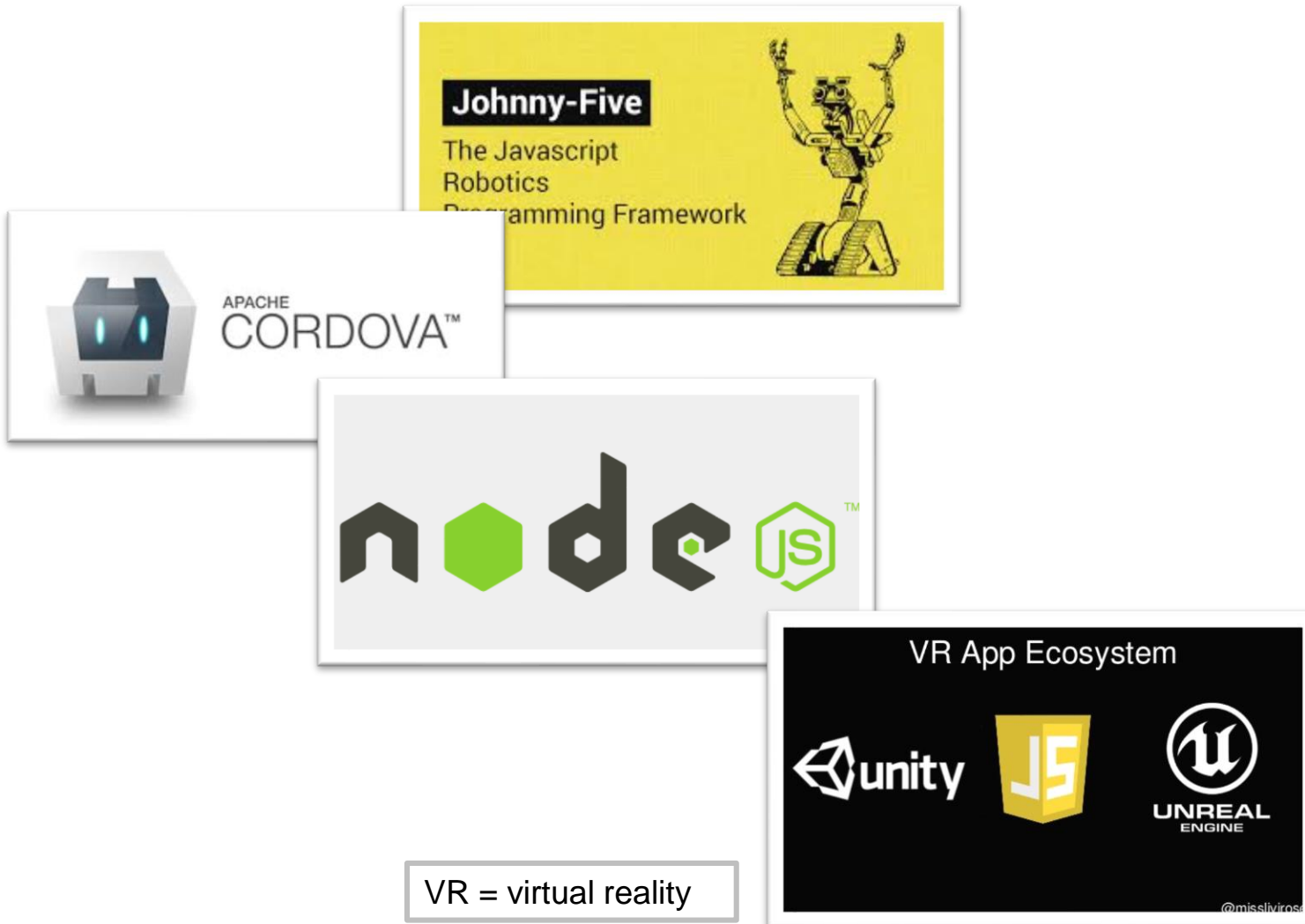
But in recent years the computers and browsers performance have improved significantly, and web development has progressed, and more more logic is moving to the client side.

Why learn JavaScript

Career Opportunities – JS is on the rise!

- Many popular libraries (frameworks) are written in JS, therefore we should have a strong knowledge of JS so we can use it confidently.
- Just like we had to learn CSS before we could use bootstrap.

Why JavaScript: it's everywhere



Java Vs JavaScript

JAVA *is to* JAVASCRIPT *as* HAM *is to* HAMSTER



About JavaScript

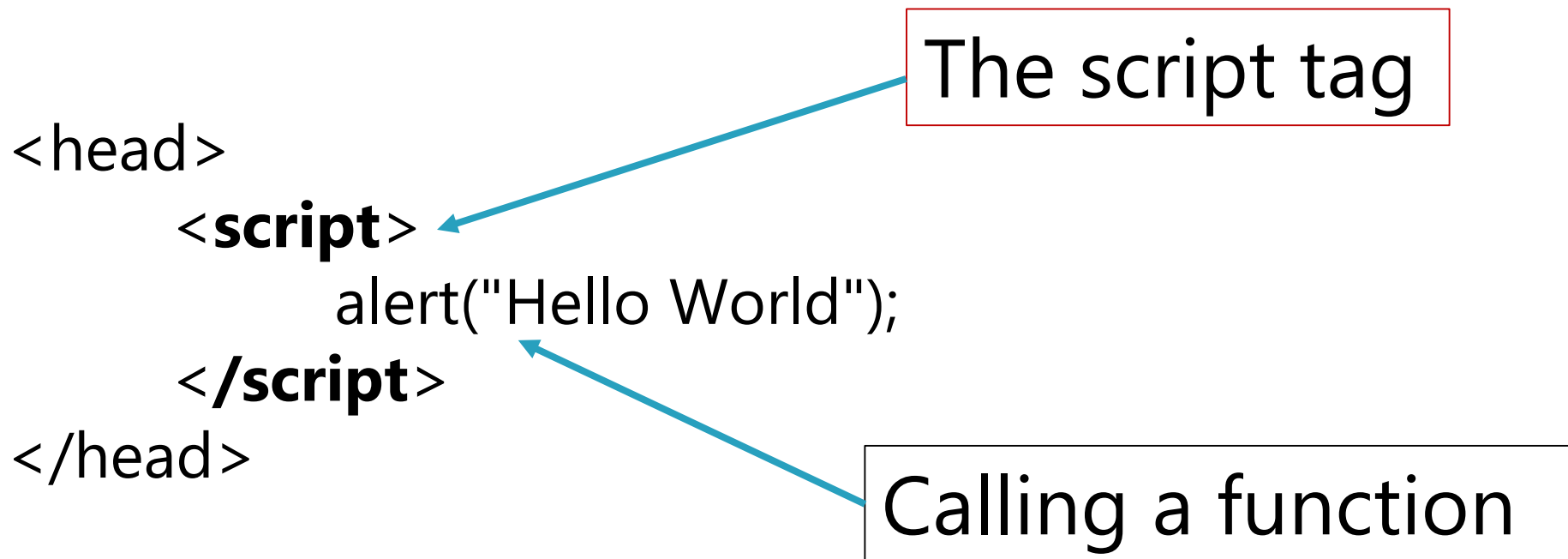
- JavaScript is a programming language interpreted by the browser.
- The "script" suffix implies that this is not a compiled language.
- A script is interpreted by an interpreter, executing the code line by line.

About JavaScript

- As a programming language it has
 - variables
 - flow control (if/for/while)
 - functions
 - reserved words
- It also has syntax and grammar to put it all together

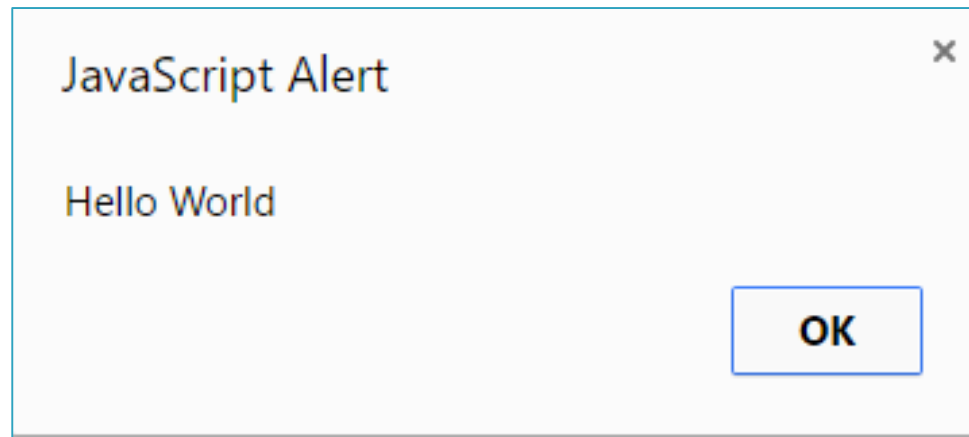
Hello world

How did we run JS?
(for example bootstrap's js file)



Hello world with Alert

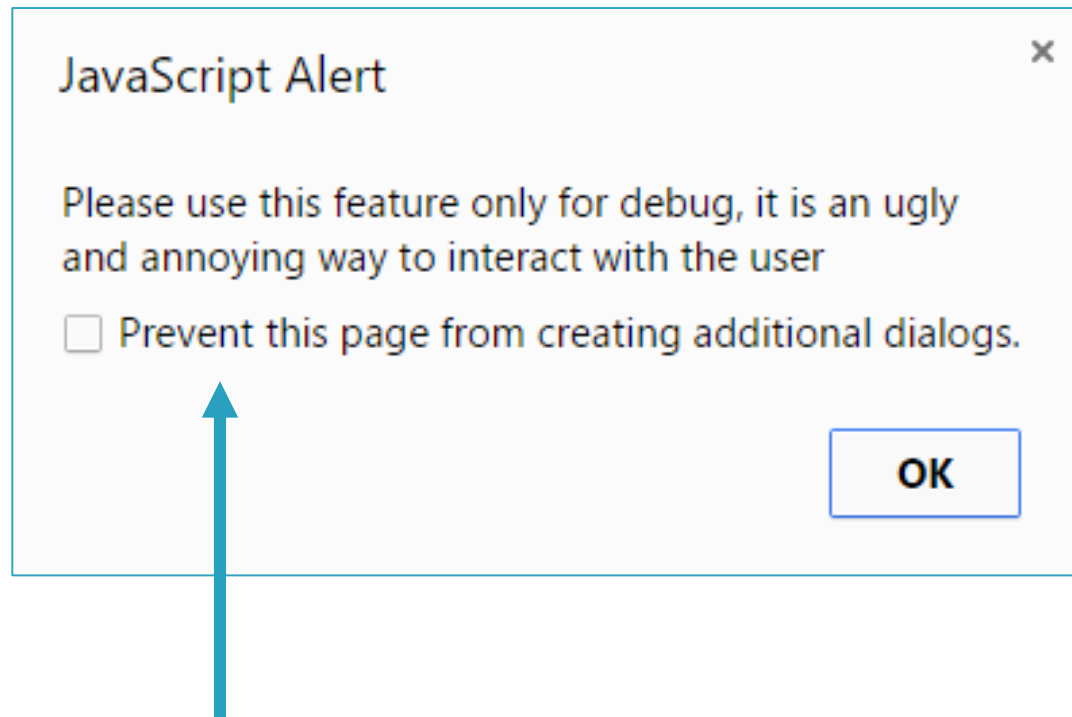
When we open our document, the script we wrote will be executed



Alert is a built-in function that the browser knows.

It receives a parameter of type string that holds the message we want to display

Alert disadvantage



And the user can disable them...

Add JS file to the HTML

We saw how to write js in a `<script>` tag in the html file, BUT JavaScript can be written in a separate JS file.

Just like CSS is written inside a `<style>` tag or in a CSS file .

The file extension is *.js

In order to include it we will use the following tag

```
<script src="./js/myscripts.js"></script>
```

This tag can be in the head or the body tags

Note: the location will affect the execution order.

Basic structure

Our html file looks now like this:

```
<!DOCTYPE html>
<HTML>
<head>
  <!-- link to css file -->
  <link rel="stylesheet" href="./css/main.css">
  <!-- css inside a style tag -->
  <style>
  </style>
  <!-- link to js file -->
  <script src="js/main.js"></script>
  <!-- javascript inside a script tag -->
  <script>
    alert("hi");
  </script>
</head>

<body>
  <div id="menu"></div>
</body>
</HTML>
```

JS file

```
alert("hi from js file");
```

Write js in a file and link to it in the html file

Write js inside a script tag in



File Structure

Our website folder will now look like this



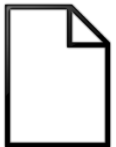
images



js



css



mysite.html

3 ways to run JS

So we've got 3 ways to run js
In the html file:

```
<!DOCTYPE html>
<HTML>
<head>
  <!-- link to css file -->
  <link rel="stylesheet" href="./css/main.css">
  <!-- css inside a style tag -->
  <style>
  </style>
  <!-- link to js file -->
  <script src="js/main.js"></script>
  <!-- javascript inside a script tag -->
  <script>
    alert("hi");
  </script>
</head>

<body>
  <div id="menu"></div>
  <div id="pages"></div>
</body>
</HTML>
```

Write js in a file and link to it in the html file

Write js inside a script tag in the html file

Another way to run js is in the console

Hello world with console.log

Another option for the hello world program is

The script tag

<script>

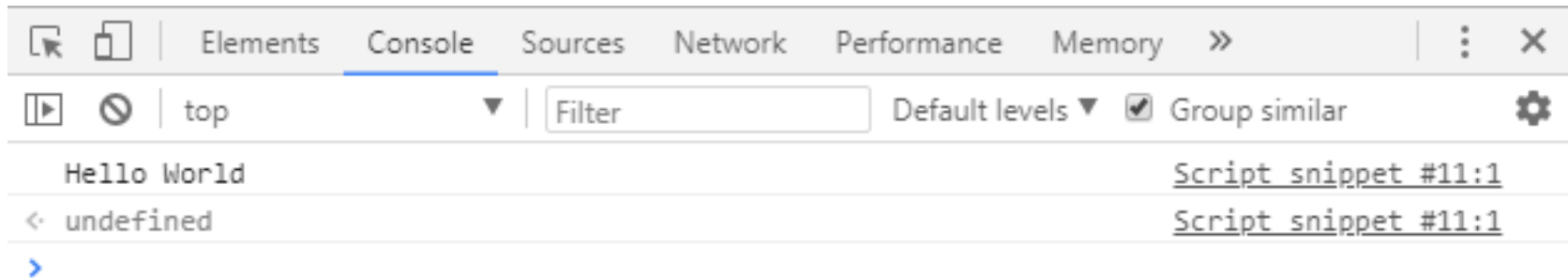
console.log("Hello World");

</script>

Calling
console.log
function

Console

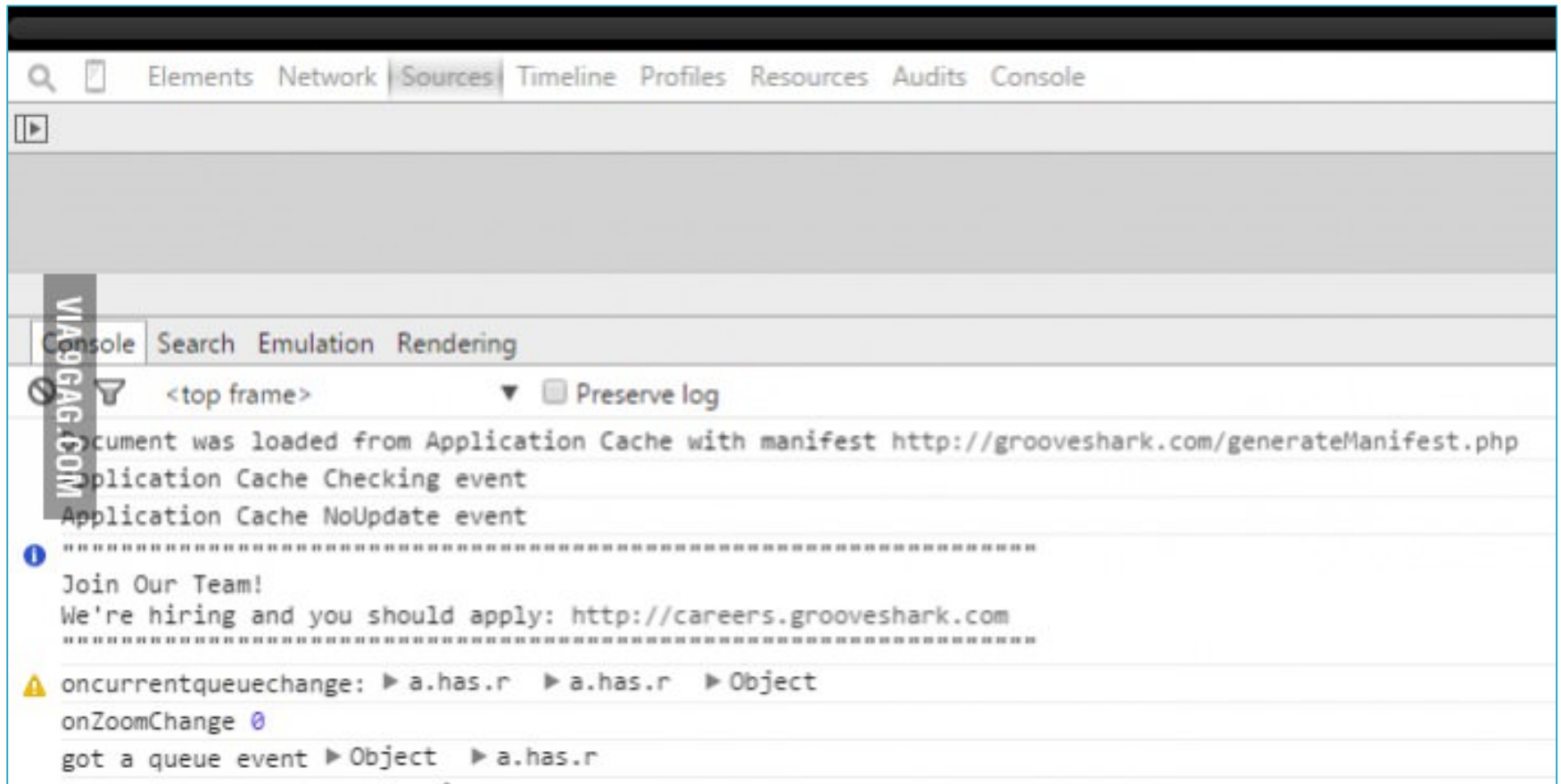
The output of the console.log method can be found in the developer tools, under the console tab



It is used for debugging purpose

Console



Or as a place for hiring new programmers ☺



Chrome Devtools Console

run Javascript in the Console

1 Open the chrome browser

2  Press f12 Or Ctrl+Shift+I
 (mac: option+cmd+J, ⌘ + ⌥ + J)

3 The dev tools window will open.
Under the tab called "console" we have a something called a "REPL".

Easiest way to run Javascript

The REPL stands for "Read, evaluate, Print, Loop"

Read: the browser reads the code.

Evaluate: the Browser runs the code (evaluates the written expression).

Print: the result of the evaluation gets printed into the console.

Loop: go back to the start, you can write more code.

The REPL is a text input that lets us write code, run it, get the result, and repeat this process until we had enough.

After we write some code, we hit enter.

What is an expression ?

An expression contains at least one value.

```
122
"hello"
true
```

Type these in your console!

An expression can contain operators

```
3 + 4
```

and functions

```
Math.min(3, -4)
```

After evaluating an expression the console prints the result.

An expression always evaluates and returns a single value.

Semicolon ;

- In JavaScript we end expressions with a semicolon:

`4 + 6 ;`

- If we don't, the code will still work.
- most of the time.



- Once in a long while, leaving the ';' out will have an unexpected effect that will ruin your day.



- While we are in the console, it's ok to leave it out.
Once we start writing blocks of code, use it!

Assigning a value to a variable

A variable is like a box that stores a value.

```
var code-lines-today = 100;
```

We can use that value later:

```
var code-lines-tomorrow = code-lines-today + 1000;
```

```
var var1 = 3; //var1 has now the value of 3
```

The value of a variable can be a result of an expression:

```
var var2 = var1 * 3; //var2 has now the value of 9
```

```
var var3 = var2 > 5; //var3 has now the value of true
```

This operation is called **assignment**.

Comments inside code

If we want to have text in our code that the computer will ignore, we can write that text as a comment

Line comments

Simply start the line with //

//All that line of text will be ignored.

Block comments

To comment a block of code surround it with /*:

Example: */* Comments can go here */*

Commenting code

```
// I have no effect  
// on the program!  
  
/*  
You can also add comments  
like this :)  
  
Nothing in this block  
can effect the program!  
  
LALAALALLALALALAL  
*/
```

Commenting code

When to use?




1. For debugging – you can comment out parts of code that you **temporarily** want to avoid.
2. To explain intention (but not your code)

Commenting code

Keep Your Code Clean – When not to use

Don't leave commented out code on your project.
And definitely don't submit it

```
function hideAllDivs() {  
  var pages = document.getElementById("pages");  
  // var should_remove = pages.firstChild ||  
    pages.getAttribute("class") === "initial";  
  if (pages.firstChild) {  
    pages.firstChild.remove();  
  }  
};
```



Why is this line commented out? Do we need this code?

Of course nobody knows, or it takes about half a day to find out why.

Result: this code will never be deleted. Every time some one will run into it (even the same person) he will waste at least 10 minutes, to try realize what is going on.

Commenting code

Keep Your Code Clean – When not to use

Don't explain your code!

```
//sets Player 2 score
Function setP2(number) {
  ...
};
```

Let your code explain itself
Instead of the comment rename the function!

```
Function setPlayer2Score(number) {
  ...
};
```



Declaring a variable

We can declare a variable without assigning a value to it.

```
var a;
```

If we do not define the value of a variable, the value of the variable will be undefined.

```
// a is undefined
```

```
console.log(a);  
// undefined
```

Declaring a variable

```
// let's have a console session
```

```
myVar
```

```
// what will be the result?
```

```
// the variable is undeclared
```

```
// we will get a error and the rest of the code will not run
```

```
//let's try again
```

```
var myVar; // variable is declared but no value is defined
```

```
// what will be the result?
```

```
myVar // undefined
```

```
myVar = 10; // assigning a value
```

```
myVar // evaluating the variable
```

```
// 10
```

Using a variable

Once we declared a variable and assigned a value to it, we can use it in other expressions.

```
var x = 3;
```

```
var y = 18 / x;
```

```
// y is now 6
```

Using a variable

Never forget to use the **var** keyword.

If you don't use it:



```
global_var = "noooooooooooooooooooooo !!!";
```

this will create a global variable

We will explain later why this is bad

Practice

What will be the output?

```
var numberX = 5;
```

```
var numberY = 6;
```

```
console.log(numberX * numberY);
```

```
var numberA = 1.5;
```

```
var numberB = 2.5;
```

```
console.log(numberA + numberB);
```

```
var firstName = "Amy";
```

```
var lastName = "Winhouse";
```

```
console.log(firstName + " " + lastName);
```

Let's take it line
by line

Practice

What will be the output?

```
var numberX = 5;  
var numberY = 6;
```

```
console.log(numberX * numberY);
```

Result:
30

```
var numberA = 1.5;  
var numberB = 2.5;
```

```
console.log(numberA + numberB);
```

Result:
4

```
var firstName = "Amy";  
var lastName = "Winehouse";
```

```
console.log(firstName + " " + lastName);
```

Result:
Amy Winehouse

Using variables

- In JavaScript, variables don't require declaration of a specific type

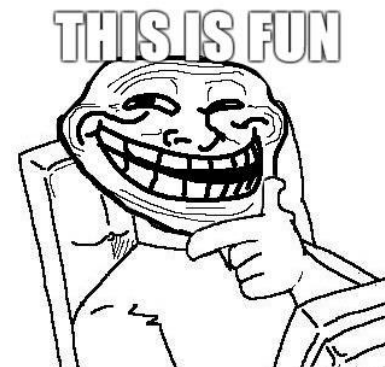
Reserved word

`var` userName = "Gilad";

`var` userAge = 25;

`var` userHeight = 1.86;

`var` userGrades = [95, 63, 41, 88, 100];



JavaScript

But also



JavaScript is not as strict as java

You can omit ; sometimes, You can forget to write var, You can pass more parameters than a function asks for...

But that might lead to **UNEXPETED RESULTS!**

Exercise variables

1. Create a var named seven with the value of 7.
2. Create a var named houdini with the value of 3.99
3. Create a new var named sum and set it's value to the value of the two previous variables summed together.

What is the result?

Exercise variables Example

```
var seven = 7;  
var houdini = 3.99;  
  
var sum = seven + houdini;  
console.log(sum); // 10.99
```

Types in Javascript

There are 6 different **primitive data types** in JS.

Primitive Type – a built in type in the language with a specific behavior.

1. undefined
2. null
3. Boolean
4. Number
5. String

Undefined

1. undefined

A primitive value automatically assigned to variables that have just been declared (but not assigned).

Represents a variable that has no value assigned to it.

Usually we will not set a variable to it!

Undefined

Example – Array pop method

Description

The **pop()** method removes the **last** element from an array and returns that element.

Syntax

```
arr.pop()
```

Return value

The removed element from the array; `undefined` if the array is empty.

```
var arr = [4];
```

```
var popped = arr.pop();  
// popped is 4
```

```
popped = arr.pop();  
// popped is undefined
```

Types in Javascript

2. null

Represents a reference that points to a nonexistent or invalid value. You can set a variable to it.

Null

Example – String match method

Description

The **match()** method retrieves the matches when matching a *string* against a *regular expression*.

Syntax

```
str.match(regex)
```

Parameters

regex

A regular expression object. If a non-RegExp object `obj` is passed, it is implicitly converted to a `RegExp` by using `new RegExp(obj)`. If you don't give any parameter and use the `match()` method directly, you will get an `Array` with an empty string: `[""]`.

Return value

If the string matches the expression, it will return an `Array` containing the entire matched string as the first element, followed by any results captured in parentheses. If there were no matches, `null` is returned.

Null

Example – String match method

Live example

```
var str = "javascript";  
var match = str.match(/script/);
```

```
> match  
◀ ▶ ["script", index: 4, input: "javascript", groups: undefined]
```

```
match = str.match(/css/);  
// match is null
```

Null vs Undefined

Example

```
function openAccount(name, businessType){
  if (businessType === undefined){
    console.log("Error. No business type was specified");
    return;
  } else if (businessType === null){
    businessType = "private account";
  } else if (businessType === "company") {
    // do something
  } else if (businessType === "small business") {
    // do something
  }
}
```

Something is wrong!

We set a default value

Types in Javascript

3. **Boolean** – true or false, similar to Java

4. **Number** – in JS, there is only one type of 'number'. A floating point number (there is always some decimals). This can make math weird in JS.

5. **String** – a sequence of characters.

Typeof

The **typeof** operator returns a string indicating the type of the expression.

Examples:

```
console.log(typeof 42);  
// output: "number"
```

```
console.log(typeof "bla bla");  
// output: "string"
```

```
console.log(typeof true);  
// output: "boolean"
```

```
console.log(typeof undefined);  
// expected output: "undefined"
```

One special case:

```
console.log(typeof null);  
// output: "object"
```

**Note: the return type
is a string**

Typeof usage

We can use typeof operator to validate the type of a variable:

Examples:

```
function getEmptyValue(myVar){  
  if (typeof myVar === "string"){  
    return "";  
  } else if (typeof myVar === "number"){  
    return 0;  
  }  
}
```

Questions ?

```
console.log("Questions?");
```

Operators

Operator – A special function that is syntactically written differently.

- Generally, operators take two parameters and return one result.
- We saw that we can use mathematical **operators** with **numbers**.

Example: `var sum = 3 + 4;`

Operators Precedence and Associativity

Operator Precedence:

Which operator function gets called first.

Operators Precedence

The following [table](#) is ordered from highest (20) to lowest (0) precedence.

Precedence	Operator type	Associativity	operators
20	Grouping	n/a	(...)
19	Member Access	left-to-right
	Function Call	left-to-right	... (...)
17	Postfix Increment	n/a	... ++
16	Logical NOT	right-to-left	! ...
	Prefix Increment	right-to-left	++ ...
14	Multiplication	left-to-right	... * ...
	Division	left-to-right	... / ...
	Remainder	left-to-right	... % ...
13	Addition	left-to-right	... + ...
11	Less Than	left-to-right	... < ...
10	Strict Equality	left-to-right	... === ...
	Strict Inequality	left-to-right	... !== ...
6	Logical AND	left-to-right	... && ...
5	Logical OR	left-to-right
3	Assignment	right-to-left	... = ...
			... += ...

Operators Precedence and Associativity

// what is the precedence?

var result = 6 % 4 + 2 *// what is the result?*



// how about now?

var result = (6 % 4) + 2



Which do you think is a better coding style?

It is better to write code that can be interpreted in only 1 way.

Operators Precedence and Associativity

Definition: Associativity determines the way in which operators of the same precedence are parsed.

Could be: right-to-left, or left-to-right.

// what is the associativity?

var result = 12 / 6 / 2 *// what is the result?*



// we might remember that / associativity is left-to-right

// but why confuse our colleagues?

// solution 1: parentheses

var result = (12 / 6) / 2

// solution 2: variable extraction

var firstResult = 12 / 6

var result = firstResult / 2

Strict vs Loose Equality

"==" is known as **Loose Equality**



"===" is known as **Strict Equality**
and can prevent potentially confusing situations.

Strict vs Loose Equality

Loose Equality "=="

Definition: Loose equality compares two values for equality, *after* converting both values to a common type.

Strict Equality "==="

Definition: Strict equality compares two values for equality in type and value. If the values have different types, the values are considered unequal. Otherwise, if the values have the same type, they're considered equal if they have the same value.

* Some special cases for numbers.

Strict and Loose Equality in Javascript

Strict Equality "==="

```
3 === 3
//true
"3" === "3"
//true
3 === "3"
//false
```

Loose Equality "=="

```
3 == 3 // true
3 == "3" // ?
// true after coercion
```

Questions?



Coercion in Javascript

Definition: Converting a value from one type to another.

This happens quite often in JS because it's weakly typed.

```
var result = 1 + 3;
// 4
result = "hello " + "world";
// "hello world"
result = 1 + "2";
// "12"
```

When using + operator with a string and a number the number is converted to string before the operator is applied.

Coercion in Javascript

A Test Case:

```
var a = 0;  
var b = false;  
  
if (a == b){  
    console.log("equal");  
} else {  
    console.log("not equal");  
}
```

What is the result?

Strings in JS

Previously, we said that a string is a sequence of characters.

```
var myString = "hello";
```

- Strings (like arrays) have indexes that are zero-based: The first character is in position 0, the second in 1, and so on.
- Character access:

```
// first approach: built-in function  
myString.charAt(0); // what will be the result?  
//will return "h"
```

```
// second approach: array-like  
myString[1];  
myString[4];
```

**Note the
different syntax!**

A bit about strings and operators

- We saw that we can use mathematical operators with **numbers**.
- We can use operators with other types as well.
- The + operator lets us combine (concatenate) two strings together and get a third string.

```
var helloMessage = "hello " + "everybody!";  
helloMessage === "hello everybody!";//?  
//true
```



Unlike in Java, you don't have to use the .equals method for comparing strings.

Mutable and Immutable

Some functions changes the value of an object.
For example:

```
var myArray = [1,2,3];  
myArray.push(4); // adds an element at the end  
myArray; // [1,2,3,4]
```

What can we conclude?

1. Array can change => it is a mutable type
2. Push function (and others) can change the array
=> push is a mutable function

Strings in JS - Mutable and Immutable

What about strings?

```
var myStr= "hello";
myStr.concat("world"); // combines strings, like +
myStr; // hello
```

What happened?

String is an **immutable** type.
It means that it's value will never change.

```
var myStr= "hello";
var newStr = myStr.concat(" world");
myStr; // hello
newStr; // hello world
```

Questions?

```
console.log("Questions?");
```

Control flow : if

Example:

```
var eggs = 12;  
var multiplyEggs = true;  
  
if (multiplyEggs){  
    eggs = eggs * 2;  
}
```

The condition must be a Boolean variable,
or an expression that will be evaluated into a Boolean value.

```
if (true)  
if (2 < -3)  
if (counter < 10 && isValid)
```

Control flow : if

If the condition is not a Boolean, JavaScript will convert it to a Boolean using coercion.

```
var userName = getUser_name();  
if (userName) {  
    //do something  
}
```

```
var height = getHeight();  
if (height){  
    //do something  
}
```

- For example the number 0 is treated like the value false and all other numbers are converted to true

Control flow : if else

- The keyword 'else' lets us define a piece of code to run if the condition is not true

```
if (true) {
    // if the condition is true the code inside these
    //Curly brackets will execute
} else {
    // if the condition is false the code inside these
    //Curly brackets will execute
}
```

Control flow : if, else

Example:

```
var eggs = 12;
var multiplyEggs = true;

if (multiplyEggs){
    eggs = eggs * 2;
} else {
    console.log("we left the eggs alone");
}
```

Control flow : if else if

We can add another `else if` to the if clause.
That code will run if both the first condition is false and the second one is true

```
if(condition1) {  
  
} else if (condition2) {  
  
} else {  
  
}
```

Control flow : if, else if, else

Example:

```
var eggs = 12;
var multiplyEggs = true;

if (multiplyEggs){
    eggs = eggs * 2;
} else if (eggs < 20){
    eggs = eggs + 1;
} else {
    console.log("we left the eggs alone");
}
```

The 'while' loop

The while loop lets us define a condition and a piece of code.

```
while (condition){  
    //run this code  
}
```

The code will run again and again as long as the condition is true.

Example

```
var sum = 8;  
  
while (sum % 6 !== 0){  
    sum += 2;  
}
```

The 'while' loop

If the condition start off being false, the code will not run even once!

Example

```
var sum = 8;

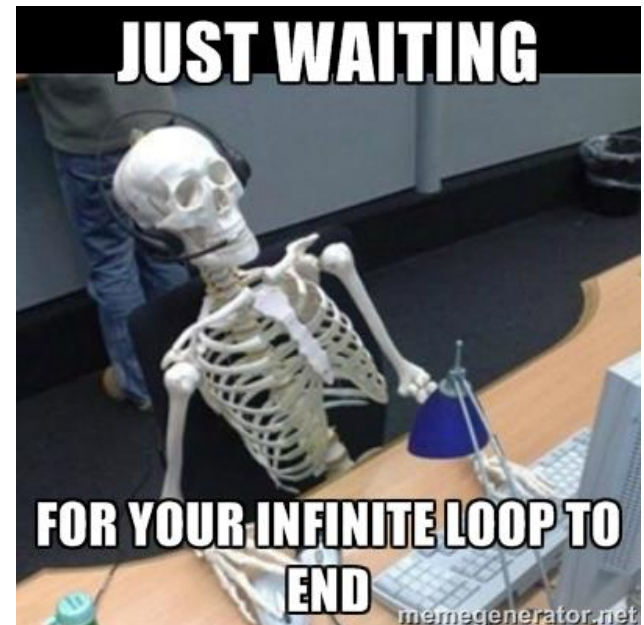
while (sum !== 10){
    sum += 2;
}
```

What is the value of `sum` ?

The 'while' loop

- **Infinite loop:**
If the condition never changes from true to false, the code will never stop running!
- Example?

```
while (1 < 3){
  console.log("always true");
}
```



Another Example

```
var counter = 0;

while (counter < 10){
  console.log(counter);
  counter++;
}
```

- This loop will run 10 times, the first time counter will be 0, and on the last run it will be 9.
- This pattern is very common, so the language has a loop called "for" that makes it clearer.

The 'for' loop

```
for (var counter=0; counter < 10; counter++){
  console.log(counter);
}
```

Same result, but our intention is clearer and we are less likely to make the mistake of forgetting to increment the counter.

Let's see how it works!

The 'for' loop order

Init index

Check condition

update index

```
for (var counter=0; counter < 10; counter++){
  console.log(counter);
}
```

Execute code

1. Initialize the index
2. Execute the steps
 1. Check the condition, if false exit loop
 2. Execute the code inside the loop
 3. Update the index.

For Loop



```
for ( a = 1;    a < 5;    a ++ )
{
    printf( "%d", a );
}
```

Tutorial4us.com

a	Output
1	

Questions?

```
console.log("Questions?");
```

User Interaction

Basic user interaction - Prompt

Let's go back to getting input from the user

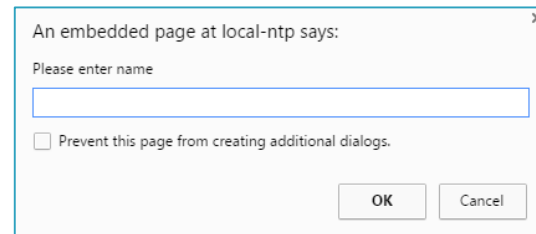
In JS we have the prompt function (sister of alert)

It allows us to display a modal dialog that contains an input field:

```
var userInput = prompt("Please enter your name");
```



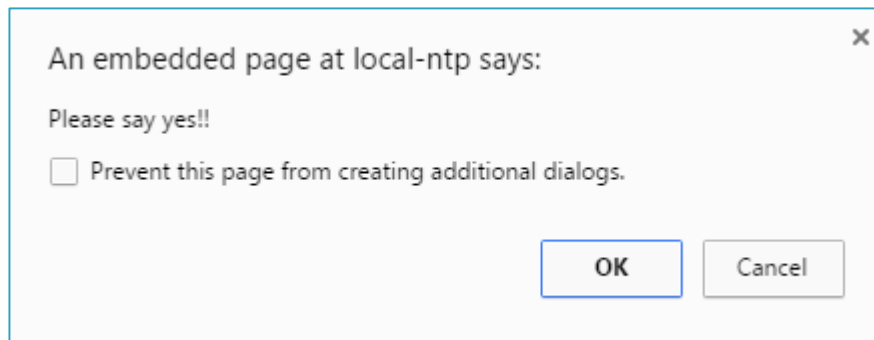
This will hold the user name
after this line executes,
Just like Scanner next in Java



Basic user interaction - Confirm

In case we only have a yes/no question we can use the **confirm function** that reads a true or false value (Boolean)

```
var userInput = confirm('Please say yes!!');  
userInput; //prints true or false depending on what the user chose;
```



- Clicking OK returns true
- Clicking cancel returns a false value

Example – User interaction

Let's create a simple app that adds 1 to a user input

Here is our code:

```
var input = prompt("Insert a Number");
console.log(input + 1);
```

Let's try it

And this is what we get:

31



The input from prompt is a string

We need to convert the string to a number before we add 1

Parsing

The value that we got from the input was a string.

But we need a number in order to do a calculation.

- We need to transform input from a string format to a number (in other cases maybe to Boolean...)
- We can use existing JS functions for that!
 - parseInt: string → whole number
 - parseFloat: string → decimal number
 - Both are of type number, but parseInt ignores what comes after the decimal point

```
var num1 = parseInt("1234"); //will be 1234
var num2 = parseInt("123.999"); //will be 123
var num3 = parseFloat("123.999"); //will be 123.999
var num4 = parseInt(""); //will be NaN
var num5 = parseInt("One"); //will be NaN
var num6 = parseFloat("One point nine"); //will be NaN
```

Example – User interaction

Let's fix the code

```
var input = prompt("Insert a Number");  
var parsedInput = parseInt(input);  
console.log(parsedInput + 1);
```

Let's Try it again!

Questions?

```
console.log("Questions?");
```

Summary

```
var userName = prompt("Hello what is your name?");
var startGame = confirm(userName + ", do you want to play a game?");
if (startGame){
  var userOption = "easy";
  do{
    userOption = prompt("Choose level of difficulty (easy/hard)");
  }while(userOption.toLocaleLowerCase() != "easy" && userOption.toLocaleLowerCase() != "hard");

  var numOfGuesses = 3;
  var numberRange = 5;
  if (userOption == "hard"){
    numOfGuesses = 2;
    numberRange = 10;
  }

  var userWon = false;
  var randomNumber = Math.floor(Math.random() * 10) + 1;
  while(numOfGuesses > 0 && !userWon){
    var userGuess = prompt("I am thinking of a number, can you guess it?");
    if (userGuess == randomNumber){
      userWon = true;
    }else{
      numOfGuesses--;
    }
  }
  if (userWon){
    console.log("You Won!")
  }else{
    console.log("You Lost :(")
  }
}
```

JavaScript

- Using more than one programming language can be confusing...
- A good programmer can switch between languages like a surgeon switches between his tools




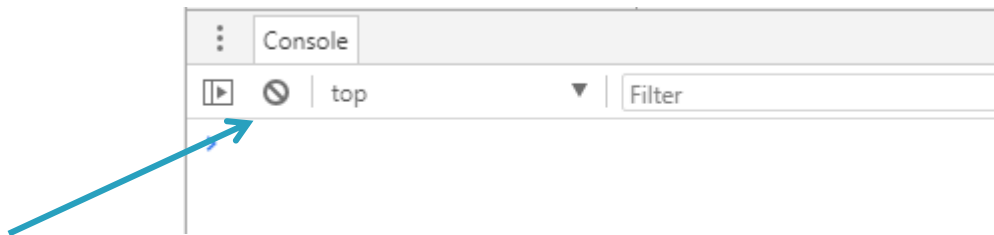
Console tips

- You can get the latest thing you wrote by pressing up key ↑
- If the browser is suggesting some text with autocomplete, you can accept it by hitting tab



Console tips

- If you want to see what methods or properties an object has, you can type in a dot after the object name and the browser will list the options.
- If you want to clear the console click the  clear button



- Or type: `clear();`



Printing

We saw 2 ways to print variable in order to find out their value

In the console:

```
var myStr= "hello"  
myStr // will print "hello"
```

When will we use printing?

In a js file or in the <script> tag:

```
var myStr= "hello";  
console.log(myStr); // will print "hello"
```



Moving from console to snippets

- Open the dev tools again
 - go to the pane called "sources". In it, there's a tab called snippets, click on it.
 - Right click on the empty space and select 'new'.
- We can now write multi line statements and run all of them.
 - This way we can start writing a real program in JS.
 - After you write code you can ctrl + enter to run it.
- Snippets are saved in the browser for later use.
- This is a learning/experimenting tool!

Moving from console to snippets

In a snippet:

- Convert English to code:
- Let's have:
 - speed of 10.
 - time of 5.
 - distance will be the multiplication of speed by time.

Moving from console to snippets 2

```
var speed = 10;  
var time = 5;  
var distance = speed * time; // 50
```

Moving from console to snippets 3

When we were running code in the console, the result of the last expression was always printed. This is still true when running a snippet.

```
3 + 4;
```

```
2 - 1;
```

The result in the console will be: 1

Moving from console to snippets 3

- The code you write in the snippet runs in the same context as the code you put into the console!
- Variables you create one way are accessible from both.

Lets try it:

In the snippet type:

```
var number = 3 + 4;  
number - 1;
```

And in the console type:

```
number
```

Summary

You needed to understand:

- Value have types
- Control flow in JS

You need to remember:

- Javascript syntax
- When to use var and typeof
- How to use the console and snippets
- How to debug

You need to be able to do:

- Use mathematical operations
- Concatenate strings

Questions?

```
console.log("Questions?");
```

Cheat Sheet

Prompt

```
var userInput = prompt("Please enter your name");
```

Saving a value into a variable

```
var fruit = "pear";
```

Printing

```
console.log("hello"); // will print "hello"
```

JS types

1. Undefined | `var empty = undefined;`
2. null | `var error = null;`
3. Boolean | `var bool = true;`
4. Number | `var number = 314;`
5. String | `var str = "Hello";`

Equality comparison == vs ===

Loose Equality "==": compares two values after converting both values to a common type.

Strict Equality "===": compares two values in type and value.

For loop

```
for (var counter=0; counter < 10; counter++){
  console.log(counter);
}
```

While loop

```
while (counter < 10){
  counter++;
}
```

Basic html file structure

```
<HTML>
<head>
  <!-- link to css file -->
  <link rel="stylesheet"
        href="./css/main.css">
  <!-- css inside a style tag -->
  <style></style>
  <!-- link to js file -->
  <script src="js/main.js"></script>
  <!--js inside a script tag-->
  <script>
    alert("hi");//displays popup for the user
  </script>
</head>
</HTML>
```

Get character from a string

```
"hello".charAt(1); //e
```

String Concatenation

```
var msg = "Hi "+"All";//"Hi ALL"
```

If else flow

```
if(condition1) {
  // do something 1
} else if (condition2) {
  // do something 2
} else {
  // do something 3
}
```