JavaScript

Discuss

We have learn the basics of static content:

What can we do with HTML?

What does CSS allow us to do?

How can we add dynamic actions to our web pages? (changes on web pages, interaction user)

History JS

- JS was introduced by Nescape. Later, other browsers did the same.
- Google and other big companies supported the JS development to make it run fast in browsers.
- JS is not going away anytime soon.

Backend using JS or Python?

- Python has more libraries
- Python is considered number 1 languages in the TIOBE index (Programming Community index is an indicator of the popularity of programming languages) https://www.tiobe.com/tiobe-index/
- Python is used in many different sectors: scientific, soft. development, education, finance,.....
- Python is simple, syntax is clean, readable
- Django is a high-level Python web framework. Django is cool, well documented, capable of asynchronous programming.

Front-End

HTML

CSS

JavaScript

Back-End

Python

Django

JavaScript

Javascript is like python, an interpreted language.
JavaScript gets most of its syntax from Java, but has some differences.

Convention

Variables and functions use camelCase format

Basic syntax

First, go to the dev tool of your web browser, an empty page, and check the console tab. The console is a javascript console.

Let’s type: alert('Hello!')

Do you see the message?

Declaring a variable

Use keyword let:

```
let person = "Bob"
```

The keywork **let** did not exist when javascript was created, but in 2015 (with the famous ES6, a version you will hear a lot because it is a version where they inserted significant changes to the Javascript language).

With let, you cannot redeclare the same variable name multiple times. Just once.

```
let person = "Bob"
let person = "Peter"
```

You will get a syntax error. Try it.

Numbers

In javascript, all the numbers (integer, float) are type number.

let studentNumber = 35

typeof studentNumber => "number"

studentNumber + 0.1 it will equal to 35.1

Booleans

let longLiveToPython = true

Important to know

If you declare a variable without any value (empty), it will be considered as Undefined

let doIHaveAValue
On the console, type doIhaveAValue, it will return undefined

In JS, we also have the following:

Null, we will see it later.
Symbol, we may see it later.
BigInt => let bigValue = BigInt(589345340589058 9089)

So far we have covered the primitive data types in javascript.
Javascript is like python, dynamic typing, you do not need to manually define the data type of the value stored in the variable, like we have to do in Java. JS will determine it automatically, and of course, like python, you can change the data type under a variable if you would like to do so.

Let's create an html file and add a script in the body

```
<!DOCTYPE html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Movie Theater</title>
  <link rel="stylesheet" href="styles.css">
</head>

<body>
  <h1>Movie Theater</h1>

 <script>
   let var_test = "Hi"
   if (var_test == "Hi") alert('Hello')
 </script>

  </body>

</html>
```

Is anything missing here? Maybe ;?

How can we print in the console?

```
<script>
    ....
    console.log(40+10+8)
</script>
```

Let's keep our code organized and create an js file

Create a folder js and create a file learn.js inside that folder.

You must add the js file in your html file:

```
<body>
 <h1>Movie Theater</h1>
 <script src="js/learn.js"></script>
</body>
```

Variables

let - We already learned how to create a variable

let day = "today"   You can change the variable value: day = "tomorrow"

const - We can also create constant variables

const city = "NYC"

When a constant is defined, you cannot redeclare it.

You cannot declare a const variable without assigning a value (it works for any no const variable - let iHaveNoValue)

var - the old way to declare a variable before JS created the let syntax. But the var syntax can be useful in certain conditions that we will see later.

var howAreYouToday = "I am good"
howAreYouToday = "Not bad"

Do we always need let, const or var to declare variables?

You can use nothing and put directly:

```
iAmARebel = "Oh yeah"
```

This would have an impact that we will see later (maybe or maybe not)

So for the moment, if you need a constant variable, use const variable_name.
Otherwise use let variable_name

Basic Operators

JS uses the operator you already know: + - * / % ** (power)

let x = 10

x += 10 (shorter than x = x + 10)

x *= 100

x++ (also work, but do not make a mistake with the return value of x++)

x –

Comparison operator

You know these ones: > < >= <= (x < 100)

Operator Precedence

It is the same as Java and Python

Try:
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Operator_precedence

Add a number to a string

It is possible. It will be converted to string

```
let price = 3.75
let myString = "The croissant price is " + price
myString
```

String interpolation with `(backtick) instead of '

```
`The croissant price is ${price}`
```

You can also do:

```
`The croissant price is ${price+1}`
```

Conditional syntax (same as Java)

```
let goingToEat
price = 4.5

if (price < 4) {
        goingToEat = "yes"
} else if (price < 5) {
        goingToEat = 'maybe'
} else {
        goingToEat = "no"
}
goingToEat
```

Check for a condition and assign a value to a variable directly

```
goingToEat = price < 4? true : false
```

This will be useful when you want to place a value in the html code directly (when using vuejs for example) and you do not want to write a function to get the value for just a simple condition. Otherwise in a function, it is better to write the condition with if.. else.. It is more readable. But up to you.

Type Conversion

```
Number('19.45')

String(19.45)
```

Expressions

What is the result of this '23' + '10' + 3 ?

"23103"

What is the result of this '23' - '10' + 3?

16

Same behavior with / and *, because for string we only have the operator +.

Of course 'bob' - '10' is going to be NaN (null value in javascript)

Be careful in case like that: 2+3+4+'5'
Can someone explain what is going on here?

Truthy and Falsy Values

This is a little bit annoying in javascript.
In the condition, to trigger a false value, you can have:
false, 0, '', undefined, null or NaN

Boolean(0) => false
Boolean(false) => false
Boolean("") => false
Boolean(undefined) => false
Boolean(null) => false
Boolean('Bob') => true
Boolean({}) => true
Boolean([]) => true

Equality operator == vs ===

```
let price = 3.5
let goForIt
if (price === 3.5) {
        goForIt = true
} else {
        goForIt = false
}
```

(if you have an error about redeclaration of price, just remove the let or reload the page)

=== mean strict equality, while **==** can be more loose
In this example, it does not change

```
if (price == 3.5) {
        goForIt = true
} else {
        goForIt = false
}
```

Equality operator == vs ===

```
let a = true;
let b = 'true';

a == b        => What is the output?
```

These are the rule for the loose equality and how javascript is going to convert the value to check if there is a loose equality (https://www.freecodecamp.org/news/loose-vs-strict-equality-in-javascript/)
So, here are the rules for type coercion in JavaScript:

- If either operand is a `string`, the other operand will be converted to a `string`.
- If either operand is a `number`, the other operand will be converted to a `number`.
- If either operand is a `boolean`, it will be converted to a `number` (`true` becomes `1` and `false` becomes `0`).
- If one operand is an `object` and the other is a primitive value, the object will be converted to a primitive value before the comparison is made.
- If one of the operands is `null` or `undefined`, the other must also be `null` or `undefined` to return `true`. Otherwise it will return `false`.

Equality operator == vs ===

```
let a = true;
let b = 'true';

a == b => False
```

How it works: a = true, is going to be converted to 1, and 'true' stays 'true' => 1 === 'true' => false

```
a= 0
b = false
a == b
a===b
```

This case is different. On the loose equality (==), we have true. in the strict equality (===), it going to check that the objects are on the same type of object, so it will be false.

Try:

```
a = '100'            a = true
b = 100              b = 1
a == b               a == b
```

Equality operator == vs ===

In general you want to use the strict equality (to not have surprise when you code runs), and you face some problem in your code and that you need to change it to a loose equality, then do it at this moment.

Operators: and, or, not

The **and, or, not operator** (&, ||, !) are the same as in java, to we can skip that
switch statement is the same as in java

Functions

Piece of code we can reuse many times in our code.

```
function logger(){
	console.log("JS is cool")
}
// call function
logger()
```

Passing data and returning data

```
function printFullName(firstName, lastName){
    console.log(firstName, LastName)
    const fullNameStr = `My name is ${firstName} ${lastName}`
    return fullNameStr
}
```

Function Declarations vs. Expressions

Function Declaration

```
function calcAge1(birthYear){
    return 2023 - birthYear
}
const age1 = calcAge1(1991)
console.log(age1)
```

Function Expression: Function with no name. Anonymous function

```
const  calcAge2 = function(birthYear){
    return 2023 - birthYear
}
const age2 = calcAge2(1991)
console.log(age1, age2)
```

Functions are values in JS.

Difference between function declarations and function expressions

You may call a function declaration before it is defined in the code.

That is not possible with a function declaration

Which one should you use?

Anyone you prefer.

Arrow Function

It is a shorter function expression.

No curly braces, no parenthesis and return value is implicit.

These functions are helpful in certain situations.

Could get complicated with more parameter and more code.

```
const calcAge3 = birthYear => 2023 - birthYear

const age3 = calcAge3(1991)

console.log(age3)
```

Years until retirement function

```
const yearsUntilRetirement = birthYear =>{
        cont age = 2024 - birthYear
        cons retirement = 65 - age
        return retirement
}
console.log(yearsUntilRetirement(1991))
```

More than one line in the body of the function requires curly braces and return statement.

Years until retirement function with an extra parameter

```
const yearsUntilRetirement1 = (birthYear, firstName) =>{
	const age = 2024 - birthYear
	const retirement = 65 - age
	return `${firstName} retires in ${retirement}`
}
console.log(yearsUntilRetirement1(1991, "Bob"))
console.log(yearsUntilRetirement1(1991, "Robert"))
```

Calling a function inside another function

```
function cutFruitPieces(fruit){
        return fruit * 4
}
function fruitProcessor(apples, oranges){
        applePieces = cutFruitPieces(apples)
        orangesPieces = cutFruitPieces(oranges)
        const juice = `Juice with ${applePIeces} pieces of apple and ${orangePieces} pieces of orange`
        return juice
}
console.log(fruitProcessor(2, 3));
```

Arrays

```
const friends = ["Peter", "Sophie", "Simon"]

console.log(friends)
```

Another way:

```
const years = new Array(2001, 2002, 2003)
```

Get data:

```
console.log(friends(0))
```

Length:

```
console.log(friends.length)
```

Mutate:

```
friends[2] = "Mary"

console.log(friends.length)
```

How can we change the value of a constant variable?

Arrays

arr.push(element): Append an element. When you push an element to an array, JS will return the new length of the array.

arr.push('Bob')

arr.unshift(element): Add at first position

arr.unshift('Pipo')

arr.pop(): Remove the last element: JS will return the element you just removed from the array

arr.shift(): Remove first element.

Arrays

arr.push('Bob')
arr.push('Pipo')
arr.push('Tango')

indexOf(element): It returns the position of the element. If the element does not exist, it returns -1.

arr.indexOf('Pipo') = > 1

arr.includes(element): Check if the element is present in the array and return true or false.

arr.includes('Pipo) => true

Objects

In JS, dictionaries are objects. The **keys** are called **properties**, because it is an object.

```
teachers = {
        'DW': ['intro', 'NeXTCS', 'graphics'],
        'Alonso': ['intro'],
        'Mouzakitis': ['intro'],
        'K': ['apcsa', 'systems', 'security'],
        'Platek': ['intro'],
        'Novillo': ['apcsa', 'soft dev']
}
```

teachers.DW, teachers.K, ….. => These are the properties because the dictionary is an object itself, and each key is a property (in python, this is not the case). So, let's call this an object from now on.

This will also work teachers['DW'], teachers['K']

Add properties to an object

```
teachers['Dillon'] = ['intro']

teachers['Alonso'] = ['foundations']
```

Add functions to objects

```
teachers = {
    'DW': ['intro', 'NeXTCS', 'graphics'],
    'Alonso': ['intro'],
    'Mouzakitis': ['intro'],
    'K': ['apcsa', 'systems', 'security'],
    'Novillo': ['apcsa', 'soft dev'],
    hasHoliday: function(date) {
        if (date.getMonth() == 11 && date.getDate() == 25) {
            return true
        } else if (date.getMonth() == 0 && date.getDate() == 1) {
            return true
        } else {
            return false
        }
    }
}
teachers.hasHoliday(new Date(2024, 0, 2)) => false
teachers.hasHoliday(new Date(2023, 11, 25)) => true
teachers.hasHoliday(new Date(2024, 0, 1)) => true
```

Call the own property of the object in a function.

```
teachers = {
	'DW': ['intro', 'NeXTCS', 'graphics'],
	'Alonso': ['intro'],
	'Mouzakitis': ['intro'],
	'K': ['apcsa', 'systems', 'security'],
	'Novillo': ['apcsa', 'soft dev'],
	hasHoliday: function(date) {
		if (date.getMonth() == 11 && date.getDate() == 25) {
			return true
		} else if (date.getMonth() == 0 && date.getDate() == 1) {
			return true
		} else {
			return false
		}
	},
	checkCourse: function(teacherName) {
		if (this[teacherName]) {
			return this[teacherName]
		} else {
			return null
		}
	}
```

With the keyword **'this'** you have access to the current object.

```
teachers.checkCourse('Novillo')
```

Iteration

```
for (let i=1; i<10; i++) {
        console.log("this is a loop, ", i)
}

const dogs = ['Pipo', 'Tango', 'Allison']
for (let dog in dogs) {
        console.log('One dog name is :', dog)
}

for (let i in dogs) {
        console.log('One dog name is :', dogs[i])
}

for (let dog of dogs) {
        console.log('One dog name is :', dog)
}
```

Loops have the keyword **continue** and **break** as in Java and Python.

While loop

```
let j = 0
while (j <= 10) {
	console.log('you know this ', j)
	j++
}
```