WHY THIS CLASS ?

Does this code produce an error ?

```
let user = "name";

user = 4;

console.log(user);
```

Does this code produce an error ?

NO !!

```
let user = "name";

user = 4;

console.log(user);
```

user should still be a string, not an interger !

It’s difficult to find errors with JavaScript

Can you identity easy the type of **dataPost**?

```
login: function () {
  if (this.username !== "") {
    this.loginFound = true;
  } else {
    this.loginFound = false;
  }
  localStorage.setItem("userName", this.username);
  let user = {
    user: this.username,
  };
  axios.post("http://localhost:5000/heartpost", user).then((response) => {
    this.dataPost = response.data;
    console.log(this.dataPost);
  });
},
logout: function () {
  this.loginFound = false;
  this.username = "";
  localStorage.setItem("userName", this.username);
```

Can you identity easy the type of **dataPost**?

NO !!

```
login: function () {
  if (this.username !== "") {
    this.loginFound = true;
  } else {
    this.loginFound = false;
  }
  localStorage.setItem("userName", this.username);
  let user = {
    user: this.username,
  };
  axios.post("http://localhost:5000/heartpost", user).then((response) => {
    this.dataPost = response.data;
    console.log(this.dataPost);
  });
},
logout: function () {
  this.loginFound = false;
  this.username = "";
  localStorage.setItem("userName", this.username);
```

1 It’s **hard** to **code fast** in JavaScript

Because JavaScript in an **untyped l**anguage

untyped = no type declaration

In real life, we talk about **trees, table, pianos**

Not array of strings or dictionaries

2 Write code with real things

We would like to **define structures** to represent the world

A person is composed of a name, a age....

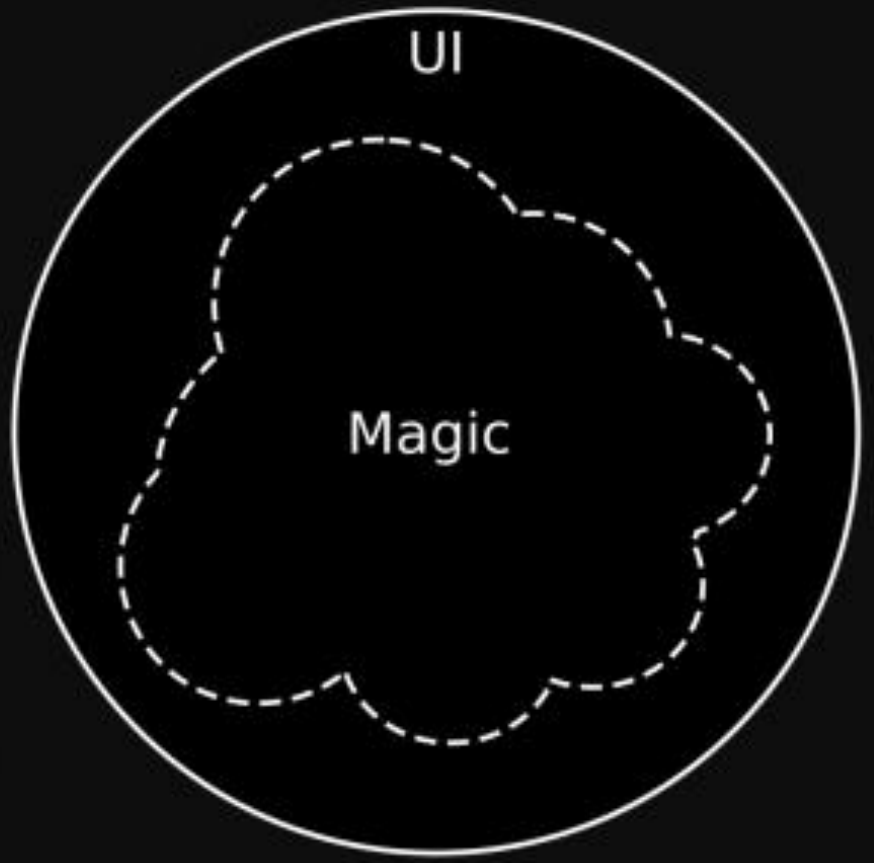
```
var app = new Vue({
  el: "#app",
  data: {
    URL: "http://localhost:3000",
    isNotLogin: true,
    formPost: false,
    loginUser: "",
    userName: "",
    pPost: "",
    filter: "allposts",
    postData: [],
    isPosting: true,
    indexToEdit: -1,
    searchFilter: "",
    resultOfsearchFilter: [],
    file: null,
    imgToUpload: "",
    urlImg: "",
    brainStormUrl: "",
    defaultProfile: "img/userProfile.j
    userProfile: "",
    users: [],
```

How do we manage

Large programs

with a lot of data to manipulate ?

How to avoid **spaghetti code** ?



In Object-Oriented-Programming:

We <u>group</u> things in logical **<u>units</u>** (objects)

data

actions
on data

OOP

1. Typed languages
2. Object/Class
3. Encapsulation / Aggregation
4. Polymorphism
5. Abstraction

OOP

1. **Typed languages**
2. Object/Class
3. Encapsulation / Aggregation
4. Abstraction
5. Polymorphism

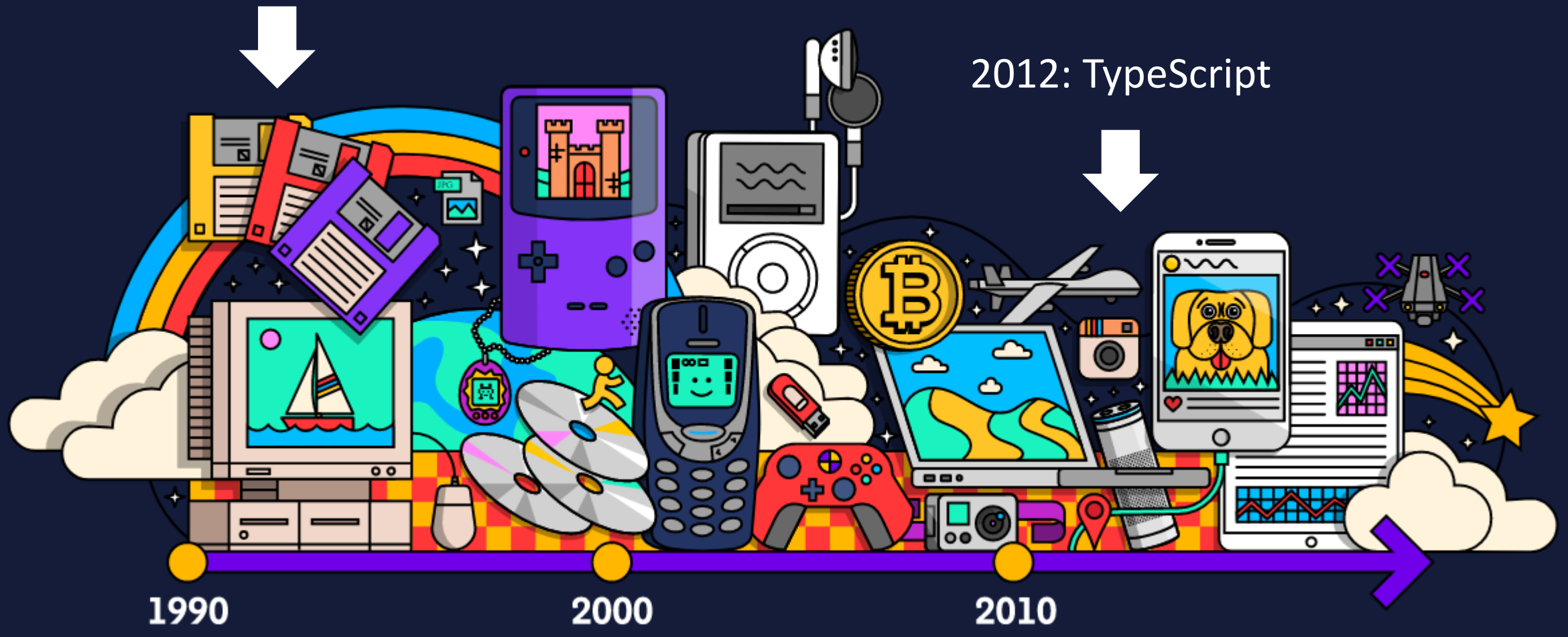
TYPED LANGUAGES

PREVENT ERRORS WITH TYPES

From **JavaScript** to **TypeScript**

1995 : JavaScript

2012: TypeScript



On VSCode, open a terminal and perform those 4 steps :

1 - Add NPM to the PATH

```
setx PATH "%PATH%;C:\Users\pros.nob\AppData\Roaming\npm;"
```

Your window account !!

2 - Add NPM to your PATH

```
npm install -g typescript
```

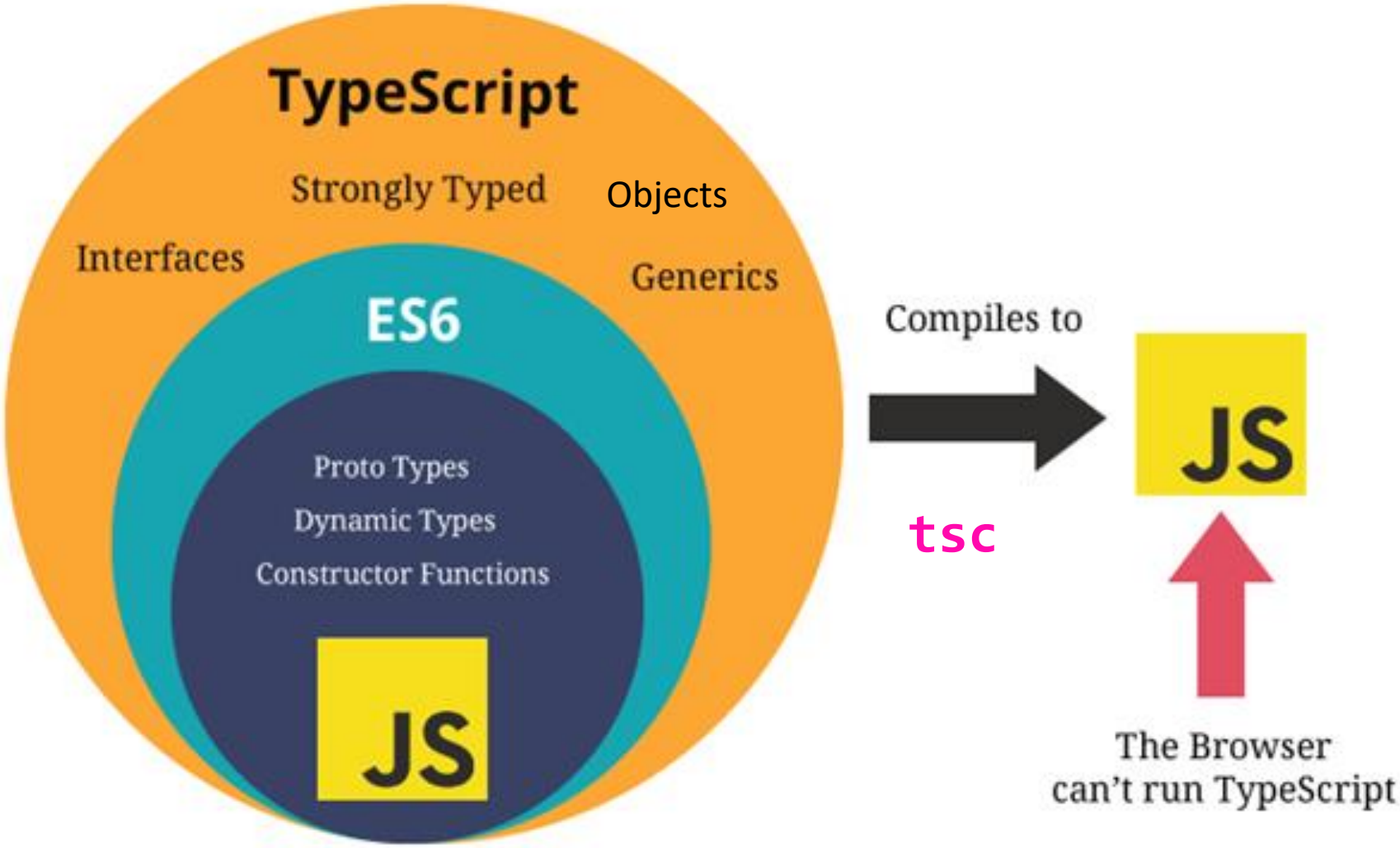
3 - Allow scripts to execute

```
Set-ExecutionPolicy -Scope CurrentUser  Unrestricted
```

4 – DONE !! Just check TypeScript works :

```
tsc --version
```

What is Typescript ?



What is Typescript ?

1- You can write **JavaScript** in Typescript !

2- You need to **compile** your Typescript!

A compiler is a small program
To transform a language into
Another language

RULE 1- Typescript can **guess your type**

```
let user = "ronan";
user = 45;

console.log(user);
```

Typescript detects a type script

Typescript detects an error :
Cannot assign a integer to a string

RULE 2- If Typescript **cannot guess**, the type is : ***any***

```
Let user;
```

Typescript does not detect anything, type is any

RULE 3 - Variables can have **annotations** after their names

```
let mustBeAString : string;
```

TYPE

ACTIVITY

1- Open activity3.ts

2- Add an **annotation** to type phone Number

String or number !

3- Check now, you have a mistake

And fix it !

```
let phoneNumber;

if (Math.random() > 0.5) {
  phoneNumber = "ronan";
} else {
  phoneNumber = 7167762323;
}
```

ACTIVITY

1- Open activity4.ts

```
function addNumber(a, b) {
  return a + b;
}

// Should print: 10
console.log(addNumber("6", "4"));
```

2- **Compile** : tsc activity4.ts

3- **Run** : node activity4.js

Check the result is not correct !

4- To fix this : add some type to the parameter a and b, to specify you want 2 numbers

5 – Fix the problem then with the “6” and “4”

ACTIVITY

```
function sayManyTimes(name, count) {
  for (let i = 0; i < count; i += 1) {
    console.log(`${name}!`);
  }
}

// Say 'Muriel!' six times
sayManyTimes(6, "Muriel");
```

1- Open activity5.ts

2- **Compile** : tsc activity5.ts

3- **Run** : node activity5.js

Check the result is not correct !

4- To fix this : add some type to parameters name and count in the function

5 – Then fix the problem in the function call : sayManyTimes(6, "Muriel");

RULE 4- You can choose if parameter are : **mandatory** or **optional**

MANDATORY

```
function printInfo (name: string, age: number) {
  console.log("name is " + name  + " age is " + age );
}

printInformation("ronan");
```

ERROR : because age is not provided

OPTIONAL

```
function printInfo (name: string, age?: number) {
  console.log("name is " + name  + " age is " + age );
}

printInformation("ronan");
```

NO ERROR because age is optional

RULE 5- Inference of types : TypeScript can guess types

1

User is a **String**

2

lastIndexOF

String --→ **number**

```
let user = "ronan";

let n = user.lastIndexOf("r")


print(n)
```

3

So n is a number

Type code to **easier the edition**

```
const myName = "ronan";

const size = myName.l
                    lastIndexOf
                    length          (property) String.length: number
                    link
                    localeCompare
                    toLocaleLowerCase
                    toLocaleUpperCase
```

- ✓ myName is a **string**
- ✓ So while editing code, Typescript will display the **properties /functions related to string only**

ACTIVITY

1- Open activity6.ts

2- Right now the type of “a” is **any**

Because no types are specified in the code

```
function getMax(n1, n2) {
  let max;
  if (n1 > n2) {
    max = n2;
  } else {
    max = n2;
  }
  return max;
}

function numberFromString(word) {
  return parseInt(word);
}

let a: any
let a = getMax(4, numberFromString("5"));
```

3- Improve this : add types to parameters, function return, variables ….

4– Then check that the display type *(when mouse hover)* of a is : **number**

How to **type** in TypeScript ?

```
let name : string = "ronan";
```

A string

```
let nums : number[] = [5, 8];
```

An array of numbers

```
let nums : (number|string)[] = [5, " 8 "];
```

An array of numbers or string

```
let student : { x: number; y: number } = {x : 45, y : 55};
```

An object composed of 2 numbers

Does this code produce an error ?

```
let student : { age: number; address: string} =
                {    age : 19,
                     address : 45
                };
```

A - YES B - NO

Does this code produce an error ?

```
let nums : (number|boolean) [] = [5, false, 58];
```

A - YES B - NO

Does this code produce an error ?

```
let nums : (number|boolean) [] = [5, false, " 58 "];
```

A - YES B - NO

Does this code produce an error ?

```
let students : { age: number; address: string} [] =
               [  {    age : 19, address : “paris “},
                  {    age : 15, address : “chicago “},
               ];
```

A - YES B - NO

TO SUM UP

On this course we will <u>improve your code style</u> :

- **Type data** to avoid mistakes
- **Group data into objects** to avoid spaghetti code

RULE 1 - Typescript can **guess the type** of your variable

RULE 2 – If no type, the type is: **any**

RULE 3 – You can specify the type using **annotation** *age* **:** *number*

RULE 4 – You can decide a parameter is **optional** *function test(name* **?** *: string)*

RULE 5 – **Inference of** types : Typescript can guess type even of multiple function calls

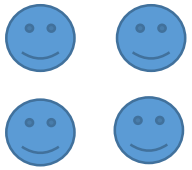
ORGANISATION OF PRACTICE

SESSION S2

INDIVIDUALY

SESSION S3

REVIEW S2 — 45”



BE AGREE ON 1 SOLUTION

15’

1 STUDENT PRESENT

EXPLANTION ON INTERFACE — 15”

PRACTICE WITH INTERFACE — 30”

Week to finish their work

WANT TO GO FURTHER ?

BASICS TO START TYPESCRIPT

https://www.typescriptlang.org/docs/handbook/2/basic-types.html