# Typescript

Autor: Esp. Ing. Ernesto Gigliotti. UTN-FRA

# Fat Arrow =>

- Permite crear funciones anónimas
- Evita escribir “function”.
- Captura el valor de “this” al ejecutarse.

# Fat Arrow =>

- TS

```
let inc:Function = function(x:number){return x+1;};
let r = inc(6);
console.log(r);
```

- JS

```
let inc = (x) => {return x+1;};
let r = inc(6);
console.log(r);
```

- TS

```
let inc:Function = (x:number):number => {
                                    return x+1;};
let r = inc(6);
console.log(r);
```

# Fat Arrow =>

```
class MyClass{
    counter:number=0;
    constructor(c:number){
        this.counter = c;
    }

    incFactory():Function{
        return function(){return this.counter+1;};
    }
}

let m:MyClass = new MyClass(5);
let f:Function = m.incFactory();

console.log(f());
```

# Fat Arrow =>

```
class MyClass{
    counter:number=0;
    constructor(c:number){
        this.counter = c;
    }

    incFactory():Function{
        return function(){return this.counter+1;};
    }
}

let m:MyClass = new MyClass(5);
let f:Function = m.incFactory();

console.log(f());
```

**Cannot read property 'counter' of undefined**

# Fat Arrow =>

```
class MyClass{
    counter:number=0;
    constructor(c:number){
        this.counter = c;
    }

    incFactory():Function{
        return ()=>{return this.counter+1;};
    }
}

let m:MyClass = new MyClass(5);
let f:Function = m.incFactory();

console.log(f());  // 6
```

# map

- Invoca a una función pasándole cada elemento del array como argumento.
- Lo que devuelva la función se agregará a un nuevo array.

```
let users:string[] = ["Juan","Jose","Carlos"];

console.log(users); // ["Juan", "Jose", "Carlos"]


let res= users.map((item)=>{return item+"."});

console.log(res); // ["Juan.", "Jose.", "Carlos."]
```

# join

- Toma todos los elementos de un array y los concatena con un separador.
- Si son objetos les llama a toString()
- Devuelve un string

```
let users:string[] = ["Juan","Jose","Carlos"];

console.log(users); // ["Juan", "Jose", "Carlos"]


let res= users.join("*");

console.log(res); // "Juan*Jose*Carlos"
```

# Template Strings

- Literales de texto con expresiones incrustadas.
- Permiten más de una línea.
- Usan la tilde invertida `

```
let tmpl:string = `hola mundo`;

let tmpl:string = `hola ${variable} mundo`;

//let tmpl:string = "hola "+ variable +"mundo";
```

# Template Strings

- Nos permiten iterar fácilmente

```
let tmpl:string=`${users.map(
                    (u)=>{return `<li>${u.name}</li>`}
                          ).join("")
                 }`;

let tmpl2:string=`${users.map(
                          (u)=>`<li>${u.name}</li>`
                          ).join("")
                  }`;
```

**<li>Juan</li><li>Jose</li><li>Carlos</li>**

# Interfaces

- Definen tipos de datos.
- Permite que un objeto sea de más de un tipo.
- Obligan a las clases a tener ciertos atributos y/o métodos.

# Interfaces

```
interface Hablador {
   hablar():void;
}

class Cat implements Hablador {

   hablar():void {
      console.log("Miau");
   }
}

class Humano implements Hablador {

   hablar():void{
      console.log("Hola mundo");
   }
}
```

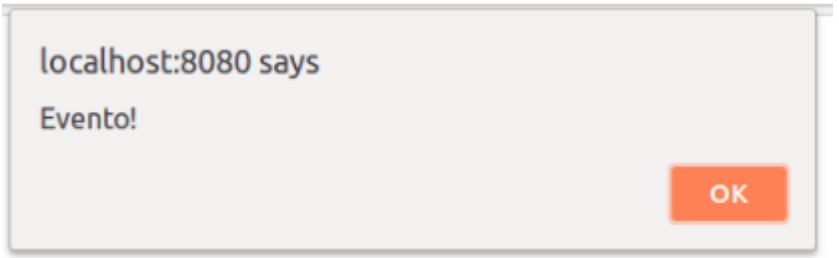
# Ejemplo 1

**HTML:**

```
<input type="submit" id="boton"/>
```

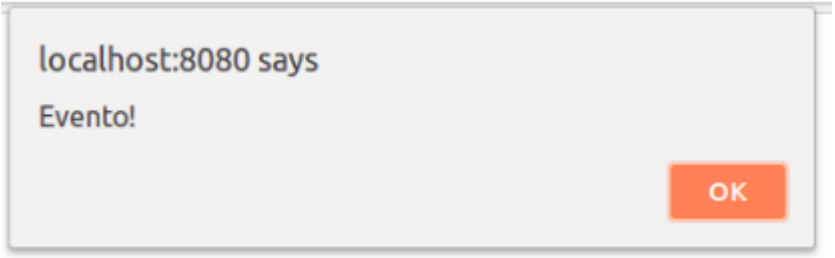
**TS:**

```
let b:HTMLElement = document.getElementById("boton");

b.addEventListener("click",()=>{alert("Evento!");});
```



# Ejemplo 2

```
function configClick(id:string,callback:any):void {

    let b:HTMLElement = document.getElementById(id);

    b.addEventListener("click",()=>{callback();});
}

function evento():void {
    alert("Evento!");
}

configClick("boton",evento);
```



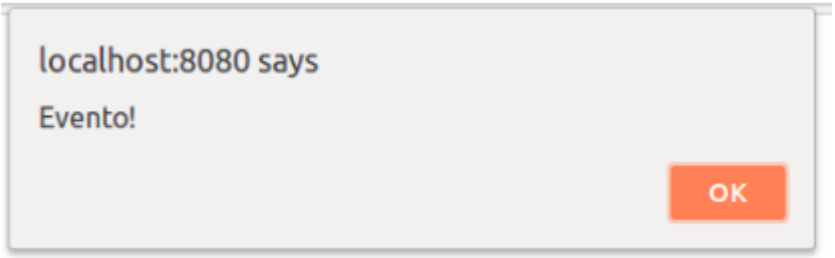
# Ejemplo 3

```
function configClick(id:string,callback:any):void {
    let b:HTMLElement = document.getElementById(id);
    b.addEventListener("click",()=>{callback();});
}

class MyClass{
    msg:string="Evento!";

    evento():void{
        alert(this.msg);
    }
}

let o:MyClass = new MyClass();

configClick("boton",o.evento);
```

**Cannot read property ‘msg’ of undefined**

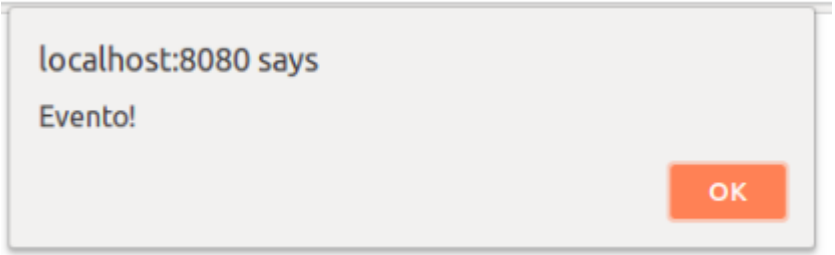
# Ejemplo 3

```
function configClick(id:string,callback:any):void {
    let b:HTMLElement = document.getElementById(id);
    b.addEventListener("click",()=>{callback();});
}

class MyClass{
    msg:string="Evento!";

    evento():void{
        alert(this.msg);
    }
}

let o:MyClass = new MyClass();

configClick("boton",()=>(o.evento())); // solucion
```



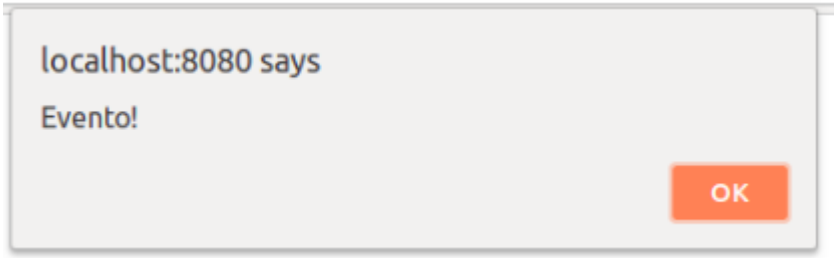
# Ejemplo 4

```
interface BtnListener{
    handleClick():void;
}

function configClick(id:string,listener:BtnListener):void {
    let b:HTMLElement = document.getElementById(id);
    b.addEventListener("click",()=>{listener.handleClick();});
}

class MyClass implements BtnListener{
    msg:string="Evento!";

    handleClick():void{
        alert(this.msg);
    }
}


let o:MyClass = new MyClass();

configClick("boton",o);
```



# Ejemplo 5

```
function
configClick(id:string,listener:EventListenerObject):void {
    let b:HTMLElement = document.getElementById(id);
    b.addEventListener("click",listener);
}

class MyClass implements EventListenerObject{
    msg:string="click!";

    handleEvent(evt:Event):void{
        alert(this.msg);
    }
}

let o:MyClass = new MyClass();
configClick("boton",o);
```



# Bibliografía


- https://code.visualstudio.com/docs/typescript/typescript-tutorial
- https://www.typescriptlang.org/docs/handbook/basic-types.html
- https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/template_strings