

{ Angular }

Framework para la creación de Aplicaciones Web y Apps

Formador: Ezequiel Llarena Borges

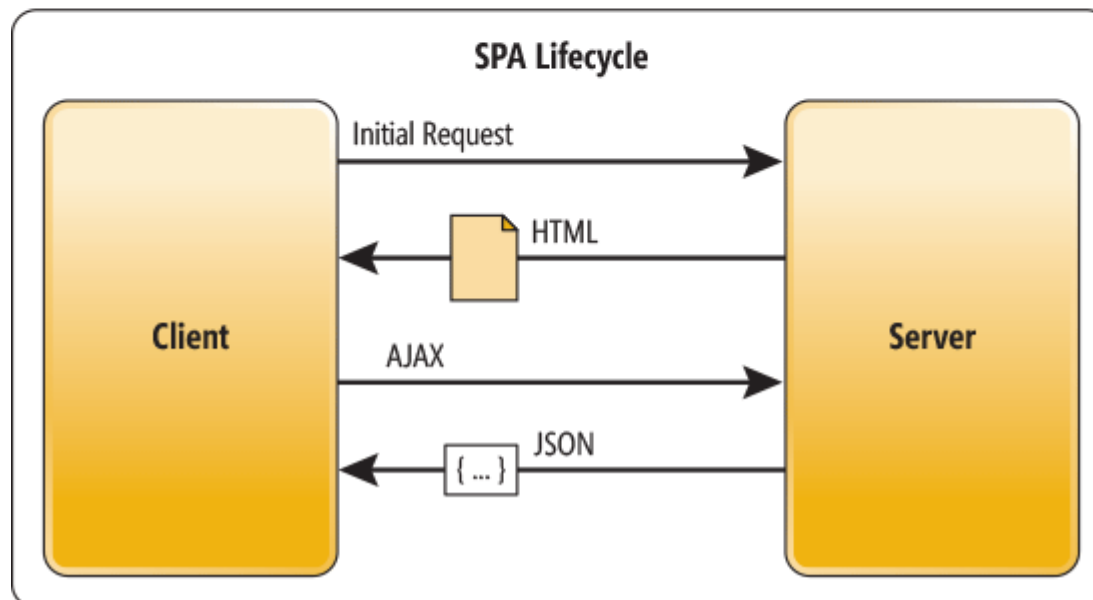
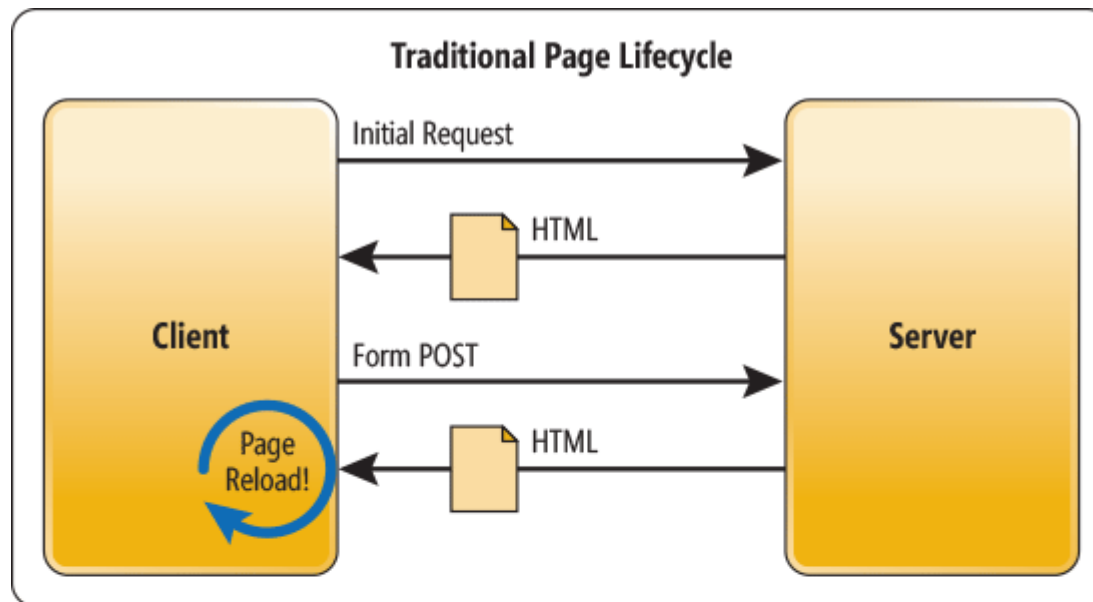
{ Angular }

Introducción, Instalación y Arquitectura de un proyecto Angular 4

Formador: Ezequiel Llarena Borges

Características

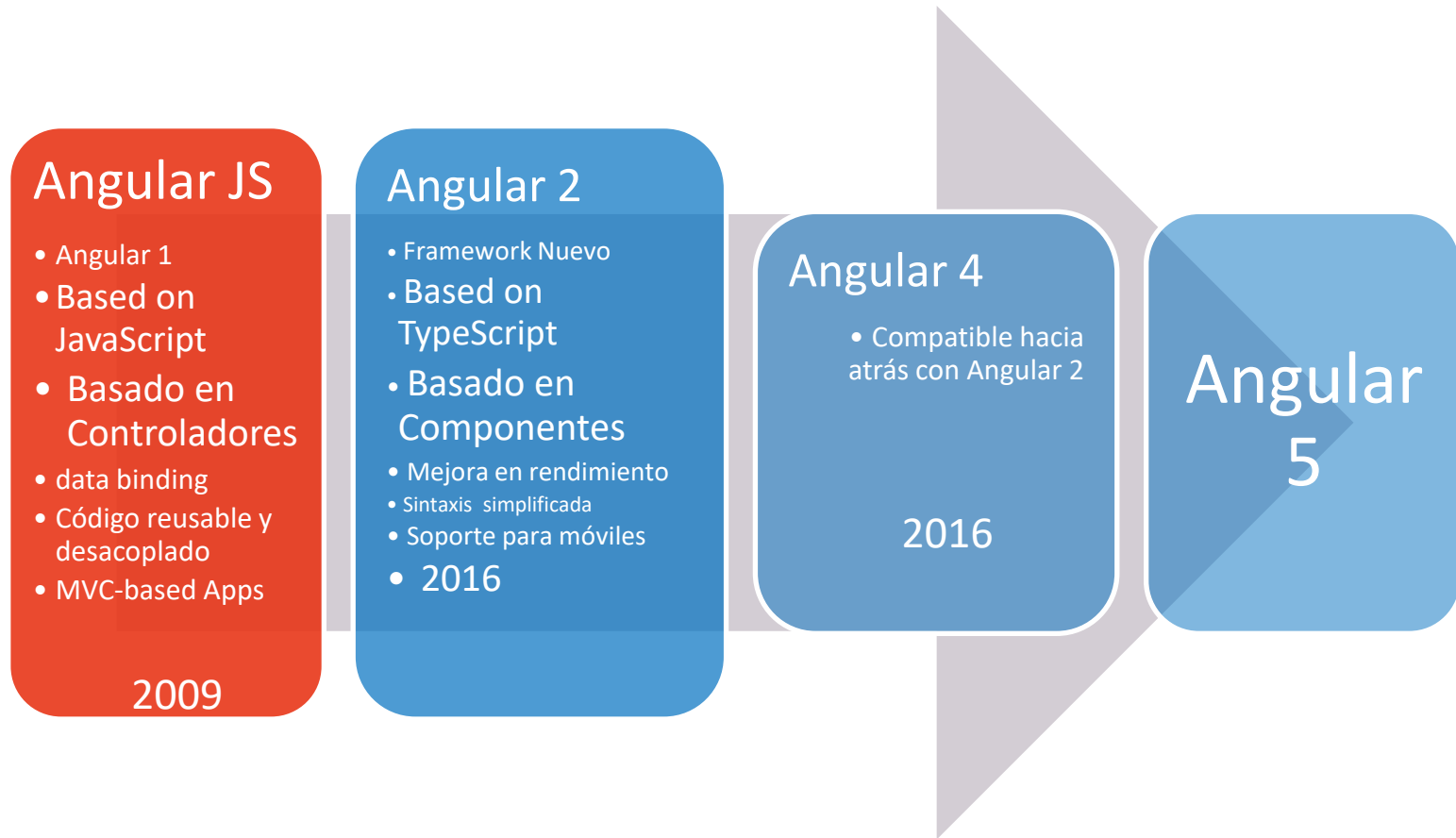
- Framework desarrollado y soportado por **Google**
- Proyectos **SPA** (Single-Page Application): carga de datos **asíncrona**
- Basado en el lenguaje **TypeScript** (desarrollado por Microsoft)
- TypeScript **basado en JavaScript** mejorado (funciones tipado y POO)



Funcionamiento

- **Componentes** (controlador en Angular JS) que representan la vista/presentación de la webapp
- Componentes soportados por **Clases**
- Utiliza **TypeScript** para lógica de las aplicaciones
- Sintaxis y desarrollo incorpora mejoras y características de **ECMAScript 6** (estandarización de JavaScript)
- Angular **transpila** el código TypeScript directamente a JavaScript ya que ECMAScript 6 aún no es soportado por los navegadores web
- Permite **trabajar en tiempo real** sobre la aplicación (podemos levantarla mediante un servido disponible con **Node JS**)

Angular JS vs Angular



TypeScript

- 85% JavaScript normal
- Nuevas características de **EcmaScript 6** (ES6)
- **Tipado** fuerte
- **Orientación a Objetos** mejorada (**Clases**)
- Lenguaje interpretado (se **transpila** a JS)
- Extensión de los archivos **.ts**
- Superset de JS desarrollado por MicroSoft

API Angular

<https://angular.io/api>

Herramientas de desarrollo

- **Node JS** (Entorno de ejecución para JavaScript)
- **NPM** (Gestor paquetes; *el Maven de Angular*)
- **Angular CLI** (Agiliza y facilita instalación de Angular + Inicio proyecto Angular)
- **Sublime Text / Visual Studio Code** (Editor de código / IDE)
- **Angular** (Framework)

Node JS

- Librería y Entorno de ejecución del lado del **servidor**
- Basado en **eventos** → Programación **Asíncrona**
- Compila y ejecuta JavaScript
- Utiliza motor V8 desarrollado por Google
- Open source (código abierto)

Pasos

1. Instalar **Node JS**
2. Instalar Angular con *quickstart* de su repositorio GitHub
3. “Hola Mundo” de Angular
4. Crear proyecto con **Angular CLI**
5. Componentes: **AppComponent**
6. Creación de componentes
7. Trabajar con múltiples componentes (**Binding** y **Selector Tag**)

Ezequiel.formacion@gmail.com

Comandos básicos Angular CLI

- `npm -v / npm --version`
- `node -v / node --version`
- `tsc -v`
- `ng new <nombre_proyecto>` Crear aplicación nueva
- `ng serve [--port <puerto>|--open]` Lanzar aplicación en el servidor

{ Angular }

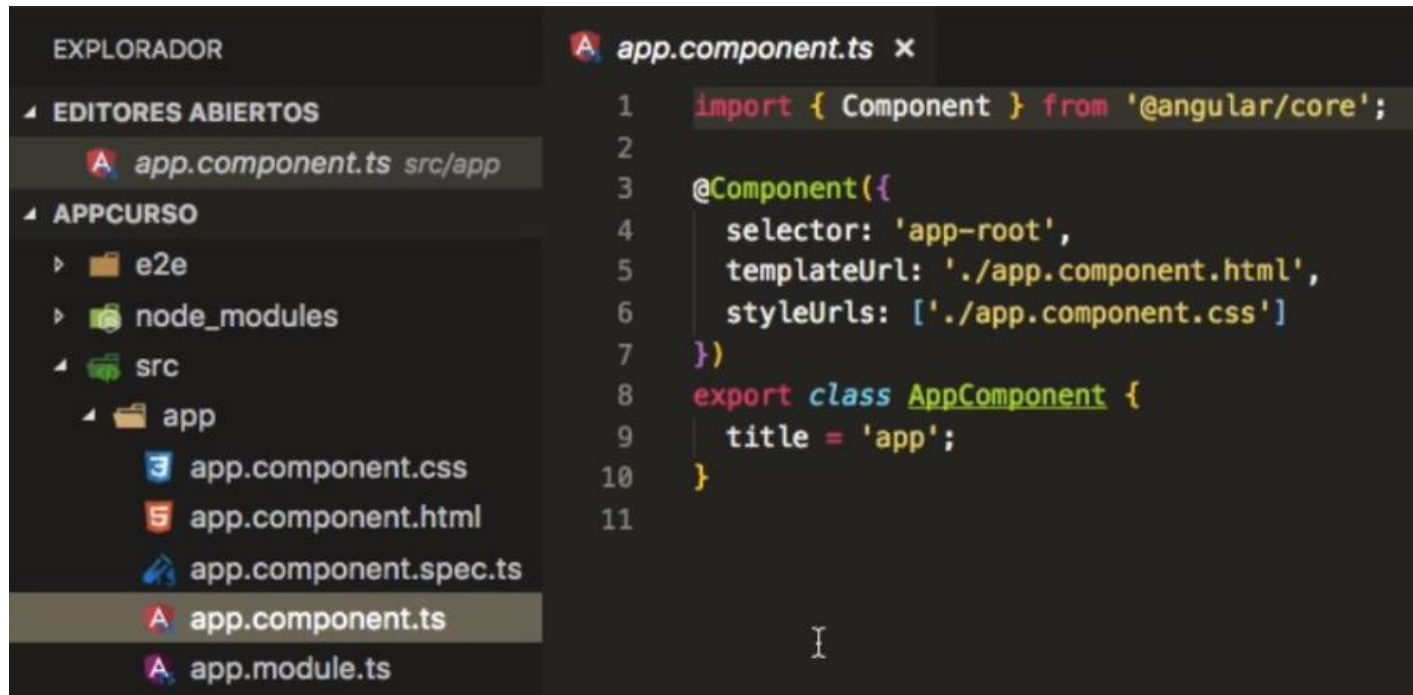
Componentes

Formador: Ezequiel Llarena Borges

Componente

Conjunto de archivos que realizan una determinada función en la aplicación
(nombre_comp.component.ts)

- .html** (Plantilla del componente)
- .ts** (Lógica de negocio del componente)
- .css** (Estilos del componente)
- .spec.ts** (Testing)



The screenshot shows an IDE interface. On the left, the 'EXPLORADOR' (Explorer) panel displays the project structure under 'APPCURSO'. The 'src/app' directory is expanded, showing files: 'app.component.css', 'app.component.html', 'app.component.spec.ts', 'app.component.ts' (selected), and 'app.module.ts'. The main editor area shows the content of 'app.component.ts' with the following code:

```
1 import { Component } from '@angular/core';
2
3 @Component({
4   selector: 'app-root',
5   templateUrl: './app.component.html',
6   styleUrls: ['./app.component.css']
7 })
8 export class AppComponent {
9   title = 'app';
10 }
11
```

Componente

```
app.component.ts x
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.css']
7  })
8  export class AppComponent {
9    title = 'app';
10 }
```

Importa la clase **Component** del paquete de Angular instalados en la carpeta **node_modules**:

Importaciones = Aplicaciones ligeras

Comandos Angular CLI

ng generate component <nuevo-componente>

- Crear un nuevo componente

ng g c <nuevo_comp> --spec false

- Versión simplificada anterior
- Omite archivo de pruebas `app.component.spec.ts`

{ Angular }

Data Binding

Formador: Ezequiel Llarena Borges

Data Binding

One Way Binding

de la fuente
de datos



a la vista
HTML

- Interpolación `{{ data }}`
- Property Binding `[placeholder]="data"`



Interpolación de strings vs Binding a **Propiedad**

```
<p>  </p>
```



```
<p> <img [src]="urlImagen"> </p>
```

```
<ul>
```

```
<li [class]="valorClass">item 1</li>
```

```
<li class="{{valorClass}}">item 2</li>
```

```
</ul>
```



```
<p>{{texto}}</p>
```

```
<p [innerHTML]="texto"></p>
```



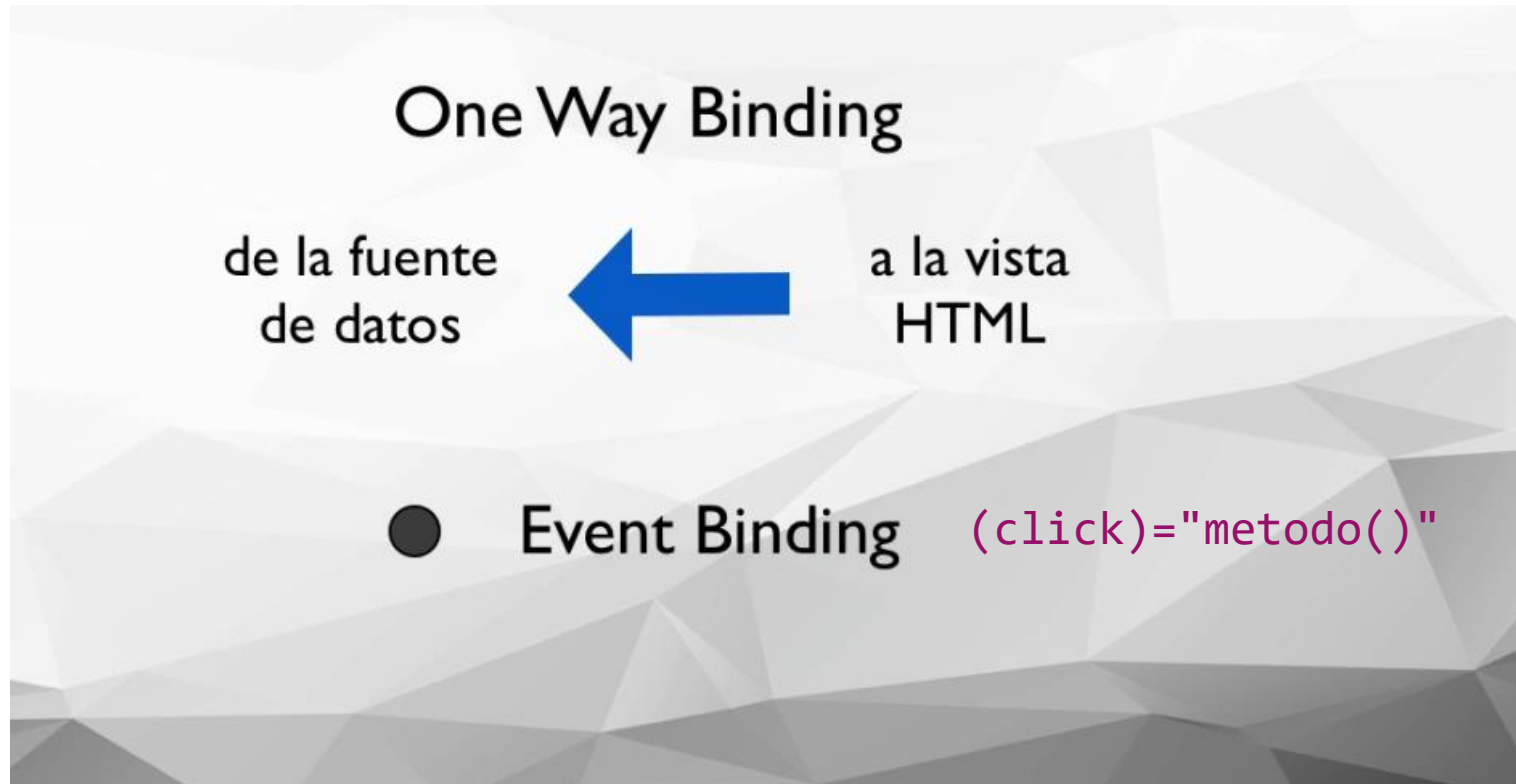
```
<button [disabled]="activo">Clic aquí</button>
```



```
<button disabled="{{activo}}">Clic aquí</button>
```



Data Binding



Data Binding

Two Way Binding

de la fuente
de datos



a la vista
HTML

- Two Way Binding `[(ngModel)]="data"`

{ Angular }

Directivas

Formador: Ezequiel Llarena Borges

Directivas

Clases Angular con código para crear, formatear e interaccionar con elementos HTML con el DOM.

Tipos:

- **Componentes** (@Component)
- **Estructurales** (modifican el DOM del elemento HTML en el que se encuentran)
- **Atributos** (funcionan como un atributo HTML, similar al *property binding*)

Directivas Estructurales

`*ngIf`

`*ngIf + else + ng-template`

`*ngFor`

Directivas de Atributos

`[ngStyle]`

`[ngClass]`

`[ngSwitch]`

{ Angular }

Servicios

Formador: Ezequiel Llarena Borges

Servicios

- Clases que realizan un determinada función
- Facilitan la **reutilización** de código
- Separan lógica del componente y llevarla al servicio
- Uso a través de **Inyección de Dependencias**
- Definir el decorador **@Injectable**

{ Angular }

Routing

Formador: Ezequiel Llarena Borges

Routing

Aplicaciones Angular basadas en **SPA** → consecuencias:

- Se utiliza una **única página**
- Los componentes se van renderizando sin necesidad de que se refresque la página
- SPA ≠ todos los componentes en la misma página → Muy compleja y difícil de manejar por el usuario con muchos componentes → solución:
 - Routing con ayuda de barra de navegación del navegador
 - Se establecen URLs para cargar dinámicamente cada componente

Implementación del Routing

1. Ir a módulo `app.module.ts`

2. Importar los paquetes del routing:

```
import { Routes, RouterModule } from '@angular/router';
```

3. Crear array de rutas (path's):

```
routes = [{path: ' ', component: },  
          {path: ' ', component: } ... ];
```

4. En la plantilla raíz `app.component.html`:

```
<router-outlet></router-outlet>
```

5. En `index.html` global: `<base href="/">`

{ Angular }

Formularios

Formador: Ezequiel Llarena Borges

Técnicas de control de formularios

- **Template Driven**
 - Lógica de captura de datos + Validación desde HTML
 - **ngForm**
 - **ngModel**
- **Reactive**
 - Gestión del Formulario desde TypeScript
 - Mayor control de la gestión de formularios
 - En tiempo real se van actualizando los valores de los campos del formulario
 - Clases: **FormControl**, **FormGroup**, **FormBuilder**

Validación HTML

- Por defecto, Angular elimina la validación HTML5
- **NgForm** tiene su propio sistema de validación mediante **Estados**:
 - Dirty
 - Pristine
 - Touched
 - Untouched
 - Valid
 - Invalid

{ Angular }

Modularización

Formador: Ezequiel Llarena Borges

Módulo

- Angular permite crear aplicaciones modularizables
- Módulo raíz: `app.module.ts`
- Comando Angular CLI: **ng generate module nuevo**

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

@NgModule({
  imports: [
    CommonModule
  ],
  declarations: []
})
export class NuevoModule { }
```

Módulos

- Disponibilidad del nuevo módulo en el proyecto:
- Registrarlo en **app.module.ts**

```
import { NuevoModule } from '../nuevo/nuevo.module';
import { NuevoComponent } from '../nuevo/nuevo/nuevo.component';

@NgModule({
  declarations: [ ... ],
  imports: [
    ...
    NuevoModule
  ],
  providers: [ ],
  bootstrap: [AppComponent]})

export class AppModule { }
```

{ Angular }

Programación Reactiva (RxJs)

Formador: Ezequiel Llarena Borges

Programación Reactiva

- Programación **Asíncrona**
- Considera el procesamiento de datos de manera asíncrona
- Procesamiento de datos = **Flujo de datos asíncrono** (Stream)
- Establece los mecanismos para manipular streams (**Observable** y **Observer**)
- Implementación **ReactiveX**: API for asynchronous programming with observable streams
- Soporte para distintos lenguajes de programación
- **RxJX** Reactive API para lenguaje JavaScript

Programación Reactiva

"Todo es un stream"

ReactiveX



Programación Reactiva

ReactiveX - RxJS

```
Rx.Observable.from(["Reactive", "Extensions",  
  "Java"])  
  .take(2)  
  .map(function(s) { s + " : on " + new  
    Date()})  
  .subscribe(function(s) {console.log(s)});
```

Resultado Final

Result:

```
Reactive : on Wed Jun 17 21:54:02  
  GMT+02:00 2015  
Extensions : on Wed Jun 17 21:54:02  
  GMT+02:00 2015
```


Programación Reactiva

Observables y Observers

```
function write(text) {  
    document.body.innerHTML = '<div>' + text + '</div>'  
    + document.body.innerHTML;  
}
```

Rx.Observable

.from(['One', 'Two', 'Three'])

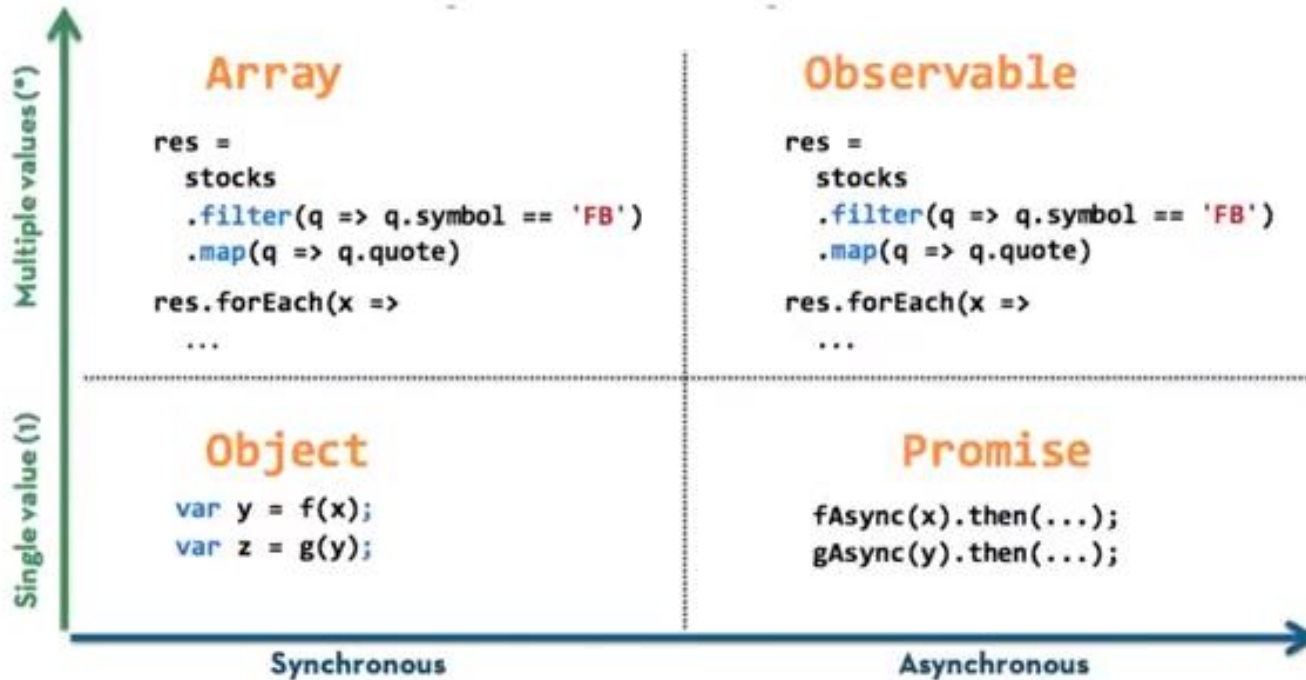
.subscribe(
 function (dato) { write('Next: ' + dato); },
 function (error) { write('Error: ', error); },
 function () { write('Completed'); }
);

Crea un Observable

Crea un Observador

- onNext
- onComplete
- onError

Programación Reactiva



{ Angular }

Despliegue a Producción

Formador: Ezequiel Llarena Borges

Build de la aplicación

- Comando Angular CLI
`ng build -prod`
- Paquetes resultantes
 - Código optimizado
 - Sólo se incorporan los módulos que se utilicen en el proyecto
 - Crea directorio **dist**
 - Archivos **.JS** + **index.html** → publicar en un servidor