# Info general

* Curso: React
* Horario: 9 a 14 de lunes a viernes
  + ~10:45 y ~12:45
* Nombre: Raul Lapeira
* Link al drive:
  + (Sin acceso) <https://drive.google.com/drive/folders/1XnBONcNUG1BsnPtylbDa2F5vJS2etbky?usp=sharing>

# Pendiente/ solicitudes alumnos

* Compartir por onedrive
* React con typescript
* Mirar scaffolding y [deprecación](https://github.com/facebook/create-react-app)

# Preguntas

## Cuando se habla del back de react, ¿es realmente un back como el que tienes tú en tus desarrollos?

El término es genérico pero en general será un backend programado en otra cosa. React tiene algunos components que “simulan” un backend, pero siempre corre en el navegador

## Tengo una cierta funcionalidad (función flecha) que no sé si se puede usar en funciones privadas o públicas (??). ¿Se puede usar en ambos?

Una cosa es el ámbito de definición y resolución de “nombres” (métodos, variables, tipos,...) y otra cosa es la ejecución en el motor javascript de dichos nombres una vez se ha resuelto la signatura y el cuerpo de esos nombres. Es decir: usar “funciones flecha” no es un mecanismo de seguridad, sino una forma de escribir el código. No es como la visibilidad en java (public, private,...)

## ¿Podemos tener un listado de scaffoldings de proyectos en React típicos?

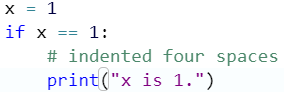
# Conceptos

* Monolítico: Un sistema que no se divide en partes
  + Del mainframe siempre se ha dicho que es monolítico
  + Del java “2 capas” (web + logica) actualmente también se dice que es monolítico
* Scaffolding: Andamiaje, usar una libreria que te simplifica el “setup” de tu proyecto
* SSR: Server side rendering, es darle cosas “ya procesadas y maquetadas” al cliente
* Escalable:
  + A nivel de rendimiento: Si tienes un servidor y das servicio a 100 clientes, al añadir un segundo servidor deberías dar servicio a casi 200
  + A nivel de requisitos: Que el dia de mañana será fácil introducir nuevos requisitos debido a que el “codebase” es fácilmente mantenible
* Functions as first class citizens: la idea de usar funciones como si fueran “tipos potentes” del lenguaje, es decir, que puedas
  + Asignarlo a una variable
  + Meterlo en un array
  + Pasarlo como parámetro
  + Componerlo con otra función
  + …
* Syntactic Sugar: Movidas del lenguaje para escribir menos (facilitarle la vida), en algunas ocasiones no es tán fácil saber si una determinada característica del lenguaje es “azúcar” o es “algo más profundo”.
  + Sería ofensivo decir que la programación orientada a objetos es azucar sintáctico
* División de responsabilidades: Es la idea de que resulta mejor tener piezas de SW especializadas en ámbitos concretos
  + React se salta la “división de responsabilidades” entre el HTML y el JS que manipula el HTML pq han visto (empíricamente) que el HTML y el JS suelen ser totalmente interdependientes

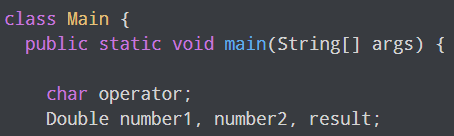
# Versus

## Weakly typed versus strongly typed language

* En un lenguaje débilmente tipado no se especifican los tipos al programar y se van pillando a medida que se declaran variables (ver también ‘dinámicamente tipados’)



* En un lenguaje fuertemente tipado, el tipo debe especificarse y NO se puede cambiar una vez creado el objeto



## Cohesionado versus acoplado

* Cohesionado: Cuando tienes módulos que cooperan bien entre sí pero uno de ellos es fácilmente intercambiable
  + Un coche está cohesionado con sus ruedas (pq gracias a que cumplen un estándar se pueden sustituir por ruedas de otro fabricante)
* Acoplado: Cuando tienes módulos que dependen completamente el uno del otro
  + Un coche está acoplado con su motor (pq el diseño del motor no es estándar y por lo tanto no puedes poner un motor seat a un coche mercedes)

# Link

# Anexo I. Batallitas del profe

* Puede sonarnos que un cierto lenguaje tiene funcionalidades que otro lenguaje no tiene pero debido a la filosofía multiparadigma tenemos librerías para hacer casi lo mismo en Java, Scala, NET, Python y PHP