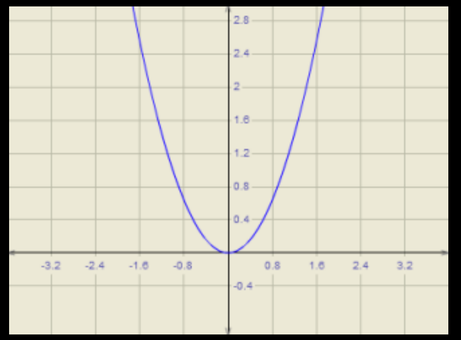
Funciones puras

Functional Core @functionalJS

Funciones

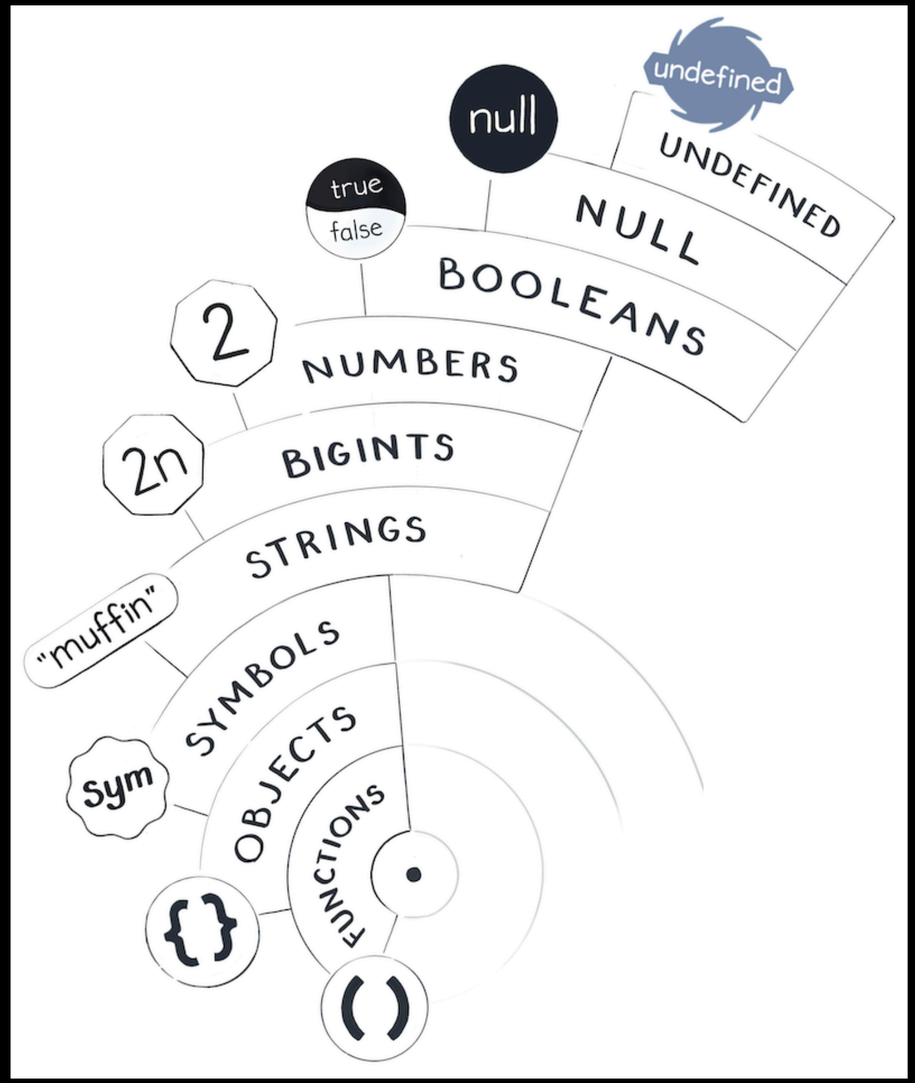
- ¿qué es una función en programación funcional?
  - Una función es la **relación semántica** entre una **entrada** (input) y la **salida** computada (output)
    f(x) = 2*x + 1



  - CONCLUSIÓNES:
    - Un mismo valor de entrada aplicado a una función debería devolver el mismo valor de salida
    - Una función en programación funcional siempre nos debería devolver un valor (asociado a un tipo)

Tipos

- ¿qué son los tipos?



Funciones puras

- La definición de una función pura, sería en primer término una función que ni genera ni consume efectos secundarios

- ¿qué es un efecto secundario? *AKA: efecto lateral , efecto colateral, side effect*
  Cualquier cosa que produzca un cambio en el estado de la aplicación, que es observado fuera de la llamada a la función más allá del valor devuelto por ella

Funciones puras

… que no consume efectos secundarios

Esto significa que cuando ejecutemos la función, el resultado que obtenemos será calculado sólo con los datos internos de esa función

```
// OK
function sayHelloTo(name) {
    return 'Hi, ' + name;
}1

// KO
let now = new Date();

function whatTimeIsIt() {
    return 'it is ' + now.getTime();
}
```

Funciones puras

… que no genere efectos secundarios

Esto significa que cuando ejecutemos la función, no modifiquemos internamente una variable global

```
var score = 10;
function addPoints(num) {
    return score += num;
}
```

Funciones puras

Si no usamos ni generamos efectos secundarios podríamos decir además que, una función pura sería una función que su resultado podría ser predecible ya que no usa nada ajeno a su cálculo y nuestra definición de función pura sería:

Una función pura es una función que ni usa ni genera efectos secundarios y además para los mismos parámetros de entrada siempre generará los mismos parámetros de salida

Funciones puras


```
function addOne(num){
    return num + 1
}
```

```
let n = 1;
function addOne(num){
    return n + num
}
```

```
let id = 1;
function customItemFactory() {
    return {
        id: ++id,
        name: 'item'
    }
}
```

Funciones puras


```
function returnNum(num) {
    return num
}

function waitAndReturnNum(wait, num) {
    setTimeout(returnNum , wait, num)
    return true;
}
```

```
function sayHiTo(name){
    console.log(`Hi, ${name}`);
    return true;
}
```

Todo depende del contexto

```
const COST_OF_ITEM = 10;
function totalCost(numOfItems) {
    return numOfItems * COST_OF_ITEM;
}
```

```
Var COST_OF_ITEM = 10;
function totalCost(numOfItems) {
    return numOfItems * COST_OF_ITEM;
}
```

```
function perimeter(r) {
    return  r * 2 * Math.PI
}
```

```
const ITEM = {
    cost: 10,
};
```

```
function totalCost(numOfItems) {
    return numOfItems * ITEM.cost;
}
```

```
const PI_NUM  = Math.PI;
function perimeter(r) {
    return  r * 2 * PI_NUM
}
```

Funciones puras


```
function getId(obj) {
    return obj.id;
}
```

```
const myObj = {
    id: Math.random()
};
```

Conclusión

La pureza de una función es una cuestión del nivel de confianza de nuestro código

Procrastinar es la palabra clave, tenemos que intentar sacar nuestras impurezas de nuestras funciones y ejecutarlas de forma aislada