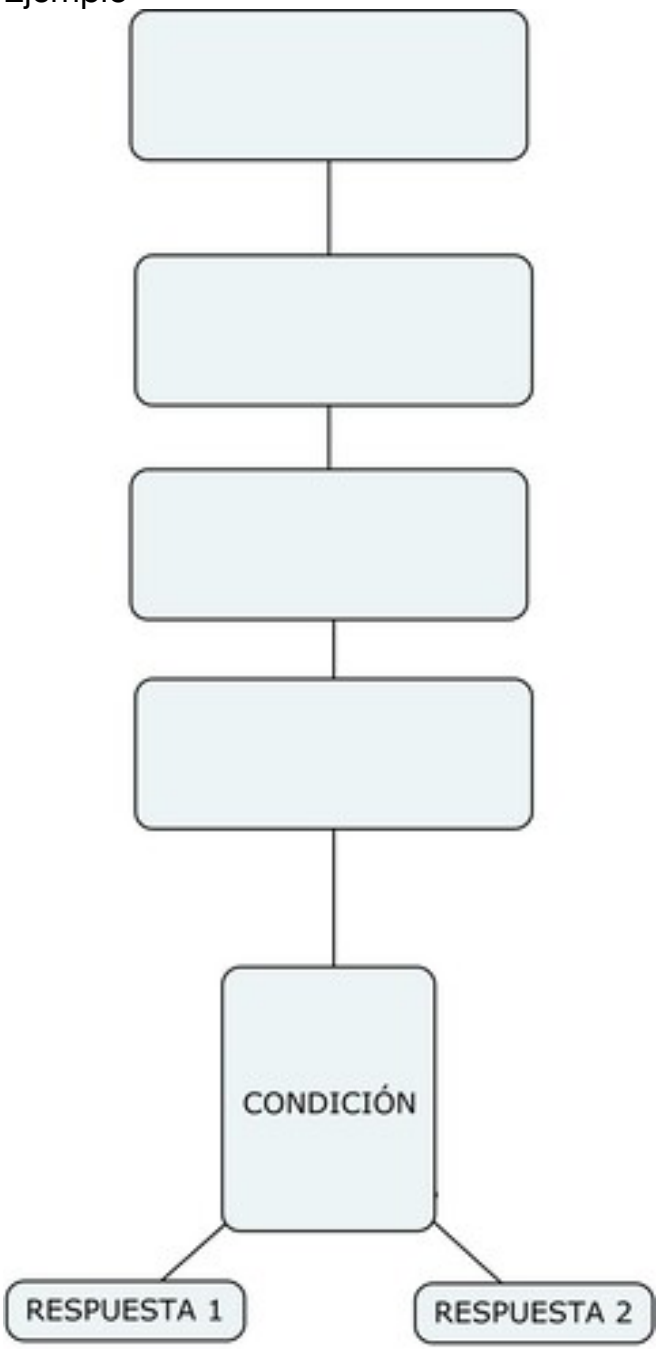
3.Paradigmas de programación

3.1 Programación estructurada

Programación por secuencia de instrucciones. Comienza de arriba hacía abajo.
Ejemplo

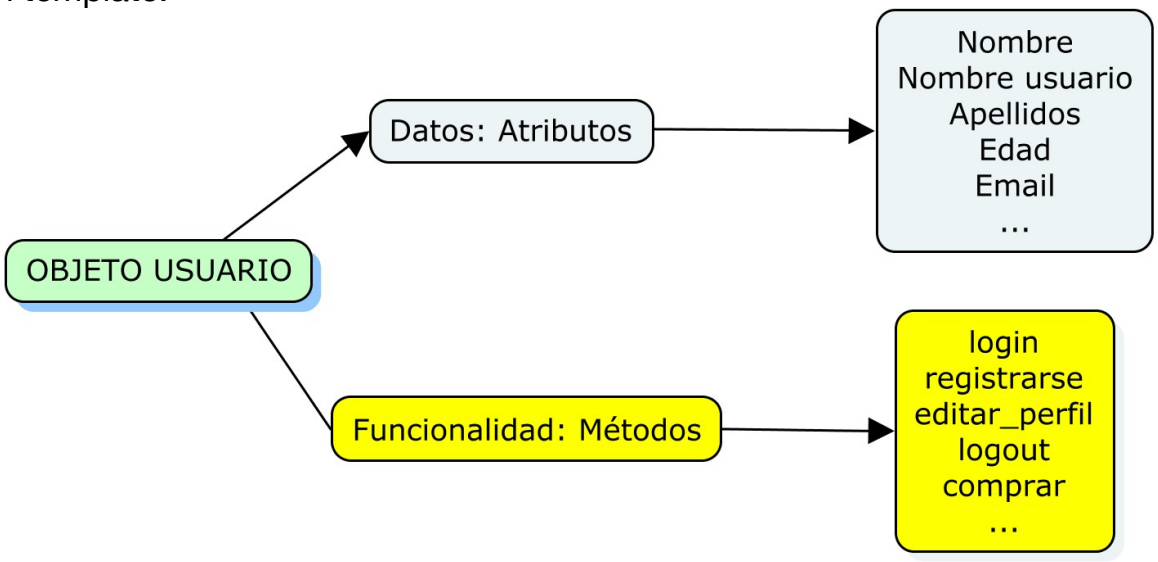


En esencia cada línea de código va ejecutado correlativa a la anterior. Es una secuencia lineal, es un método de desarrollo no apto para webs.

3.2 Programación orientada a objetos

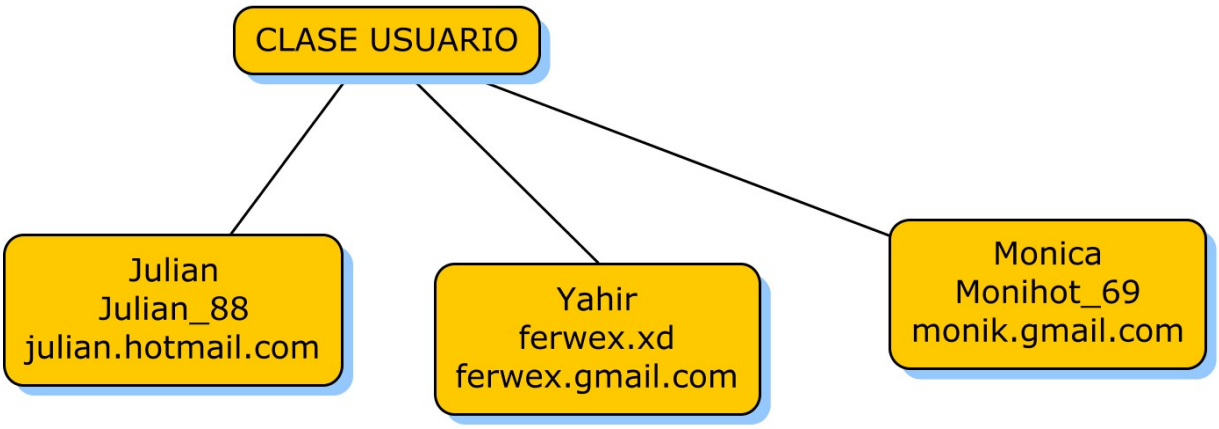
El objetivo es que los objetos se interrelacionen entre ellos; el usuario puede acceder a diferentes páginas, categorías, productos; las categorías deben poder acceder a los productos. Los objetos se van relacionando entre sí. Deja de ser lineal y pasa a ser interactivo como lo es una página web o una tienda web.

Cada objeto tiene 2 características, por un lado tiene los atributos y por otro la funcionalidad que se le conoce como métodos.
¿Cómo creamos los objetos? Hacemos uso de lo que se conoce como clases, son un template.



La clase tiene estos atributos y estos métodos, de esta plantilla. A partir de ahí podemos ir creando objetos diferentes.

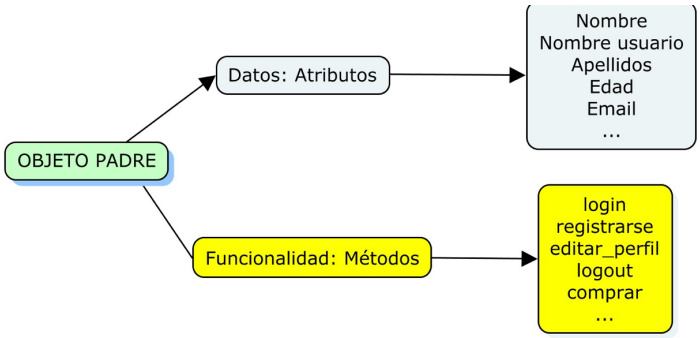
Clase usuario de donde partirán todos los demás usuarios diferentes. De esta plantilla “clase usuario” partirán diferentes usuarios con diferentes atributos;
Esto en el mundo de la programación orientada a objetos se llama “instanciar objetos”
Cada uno de los objetos se le conoce como “instancia” de la clase usuario/objeto usuario.

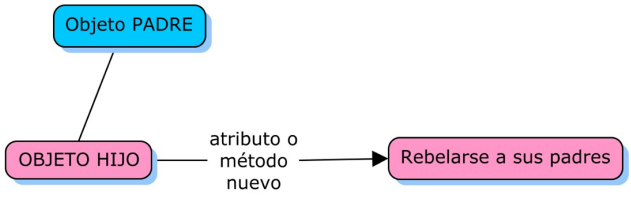


Conceptos clave de los objetos

- Abstracción: tenemos que definir qué atributos representar ese usuario. Los atributos y también la funcionalidad que tendrán estos objetos.

- Encapsulamiento: dentro de un mismo objeto tener todos los atributos y todos los métodos para poder tratar ese objeto únicamente como una variable y/o registro dentro de la memoria, mientras que cada uno de ellos tiene sus propios atributos y métodos.
- Herencia: Clase padre que podría ser “objeto padre” y de este objeto padre podemos derivar clases del objeto padre. Este objeto padre tiene sus propios atributos y métodos, de este mismo podemos instancias objetos padre. A partir de la programación orientada a objetos podemos crear nuevas clases que se llamen “clase/objeto hijo”, que puede tener por parte de la herencia, hereda todos los atributos y métodos del padre, pero nosotros en algún momento podemos añadirle métodos extras, además de los heredados.

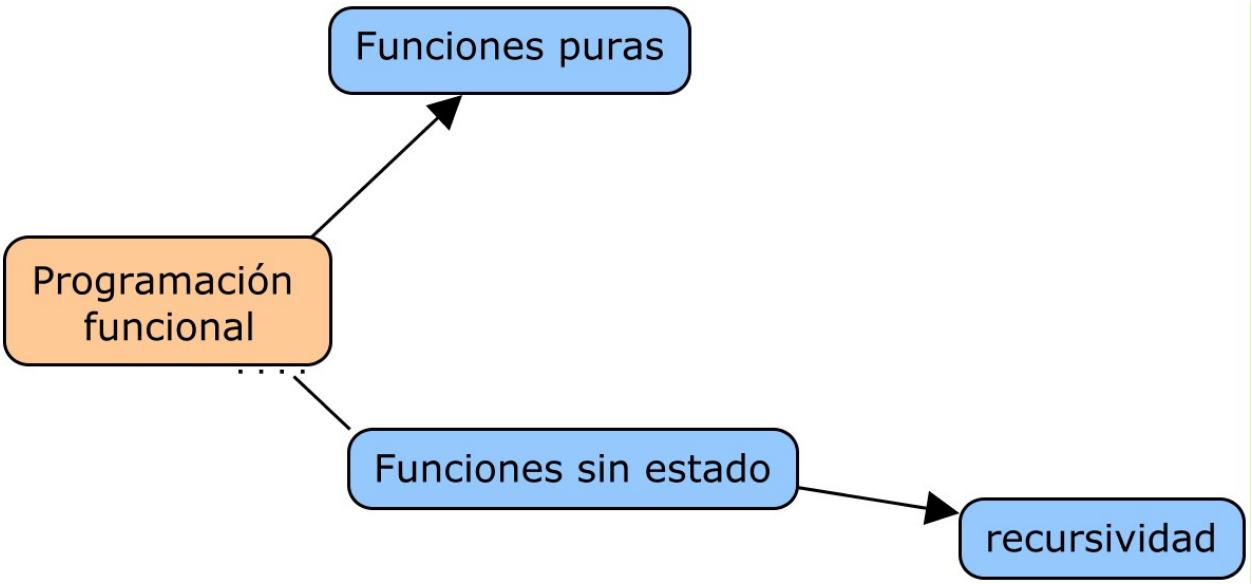




• Polimorfismo: Significa que un mismo método/función puede tener diferentes formas en función de sus atributos o en función del atributo del propio objeto.

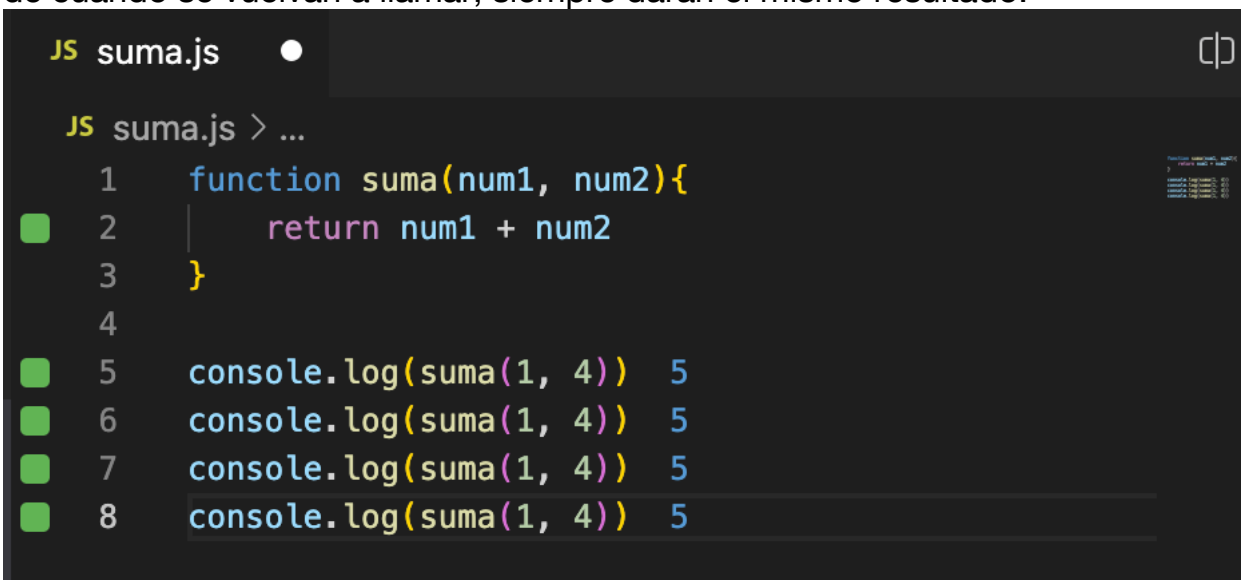
3.3 Programación funcional

Es un paradigma de programación que se puede utilizar tanto en paradigmas estructurados como en programación orientada a objetos. No es más que ir encapsulando diferentes funciones para poder reutilizarlas.



Funciones puras

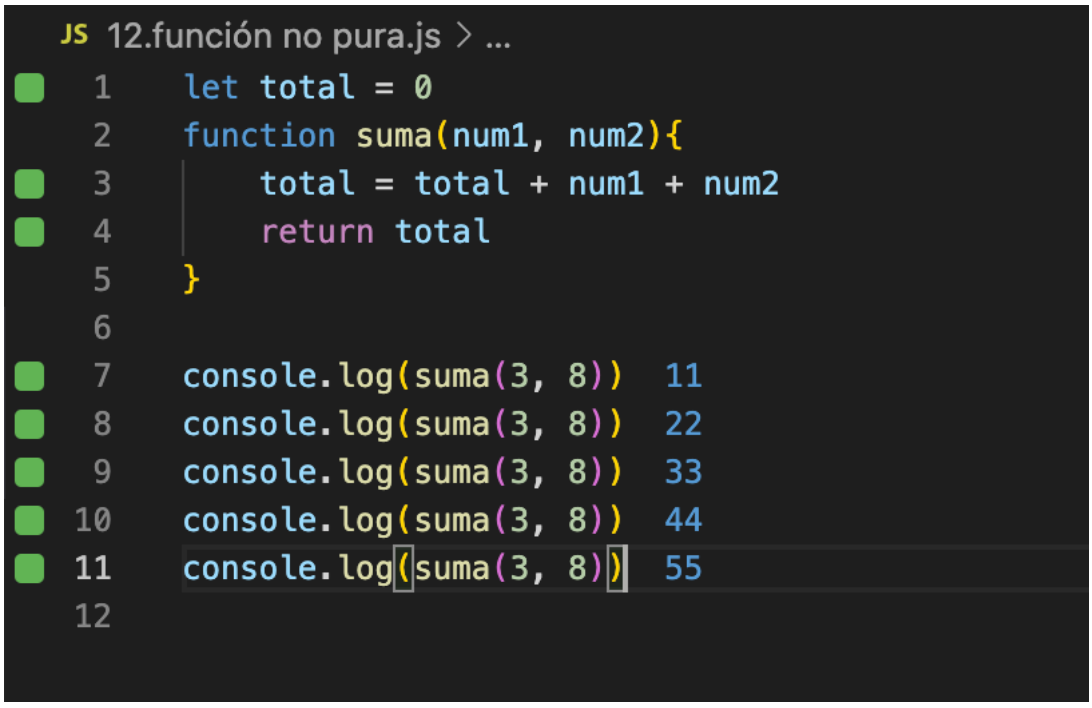
Funciones que al llamarse una vez se obtiene un resultado, independientemente de cuándo se vuelvan a llamar, siempre darán el mismo resultado.



Función no pura

Es una función no pura, porque lo que hace es mutar una variable y está no se comporta de la misma manera todas las veces que la llamamos. Quiere decir que hay alguna variación de algún valor y estas funciones que no son puras son mucho más difíciles de solucionar si hay algún problema, de hacer algún "debug"

No nos devuelve el mismo valor al ser llamadas.



Funciones sin estado

Funciones sin variables internas, sin estado interno que se estén utilizando dentro de las propias funciones.

14. función sin estado.js — Release Notes: 1.78.0

```js
function suma(num1, num2) {
    return num1 + num2
}

function multiplica(num1, num2){
    return num1 * num2
}

function suma_y_multiplica(num1, num2){
    return suma(num1, num2) * multiplica(num1, num2)
}
// Una función pura que se compone unicamente de funciones puras, sigue siendo una funcion pura
console.log(suma(1, 4))  5
console.log(multiplica(1, 4))  4
console.log(suma_y_multiplica(1, 4))  20
```

Para ello se utiliza algo que se llama **RECURSIVIDAD:**

13. Recursividad.js > factorial

```js
//  Recursividad
// Cálculo factorial de un número entero
// Factorial de 5 = 5 * 4 * 3 * 2 * 1 = 120

// Función NO recursiva -> Tiene un estado interno (en este caso la variable fact que es la que devolvemos al final)
// Programación estructurada
function factorial(num){
    let fact = num
    for (let i = num - 1; i > 0; i = i - 1) {
        fact = fact * i
    }
    return fact
}

console.log(factorial(5))  120

//Programación funcional, donde se hace uso de la recursividad
function factorial_rec(num){
    if (num === 1) return 1
    return num * factorial_rec(num - 1)
}

console.log(factorial_rec(5))  120
```

La recursividad no es más que una función se llame así misma de nuevo.