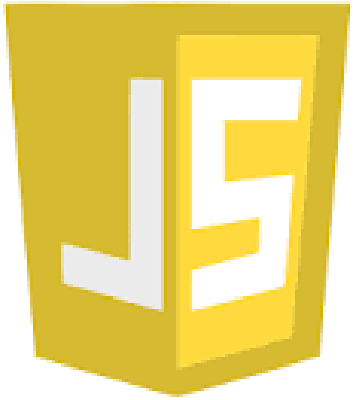


Introducción a JS

UD1: Introducción a JS

Al terminar la lección…

- Habrás recordado elementos básicos de programación comunes a todos los lenguajes
  - Comentarios, variables, entrada/salida
  - Operadores
  - Estructuras de control
  - Funciones y objetos [adelanto]
- Habrás aprendido cómo programar dichas estructuras en JavaScript

Indice

Aspectos básicos

Sintaxis básica

- Case-sensitive
  - Sensible a mayúsculas-minúsculas.
- Formato libre
  - Los saltos de línea y espacios en blanco no aportan significado.
- Fin de instrucción con punto y coma [;]
  - Opcional pero recomendable.
- Tipado blando: No se define el tipo de las variables

Comentarios

- Comentario de una línea:

```
//comentario de una línea
alert('Hola mundo');
```

- Comentario de varias líneas:

```
/*
 * Comentarios
 * de
 * varias
 * líneas
 */
alert('Hola mundo');
```

El intérprete los ignora, pero se descargan con el resto del script

Declaración de variables

- Anteponemos la palabra reservada **`let`** (accesible desde el bloque {} donde se ha declarado)o **`var`** (accesible desde toda la función donde se ha declarado)
  - No se declara el tipo de las variables
  - El tipo de una variable puede cambiar dinámicamente
  - Puedo asignar un valor al declarar o posteriormente

Ámbito de visibilidad: `let`

- **`let`** nos permite declarar una variable de alcance limitado al bloque, declaración o expresión donde se está usando (entre llaves)

```
function varTest() {
  var x = 31;
  if (true) {
    var x = 71;  // misma variable!
    console.log(x);  // 71
  }
  console.log(x);  // 71
}
```

```
function letTest() {
  let x = 31;
  if (true) {
    let x = 71;  // Variable distinta
    console.log(x);  // 71
  }
  console.log(x);  // 31
}
```

Declaración de constantes

- Son variables cuyo valor no puede cambiar
- Se declaran con la palabra reservada **`const`**

```
const CAPITAL="Oviedo";
alert(CAPITAL + " es la capital de Asturias");

CAPITAL = "Gijón";

alert(CAPITAL + " es la capital de Asturias");
```

var

var apple =

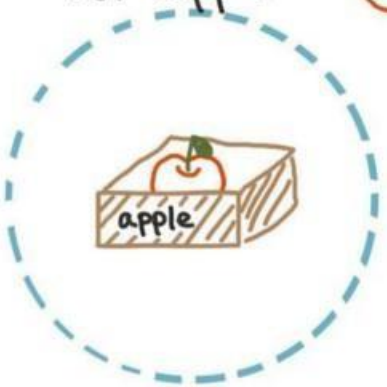
a thing in a box named "apple"

apple =

you can swap item later

let

let apple =



a thing in a box named "apple" w/ protection shield

apple = NG

apple = ok!

you can swap item only if you ask inside of the shield

const

const apple =

a thing in LOCKED cage named "apple"

apple = NG

you can't swap item later.

apple.multiply(3) ok!

... but you can ask the item to change itself (if the item has method to do that)

Buenas prácticas

- No es obligatorio declarar las variables
  - Pero sí recomendable, pues las variables no declaradas son implícitamente globales
- No es obligatorio **inicializar las variables**
  - Pero sí **recomendable** ya que si no tienen el valor `undefined`
    - No es deseable que las variables tengan dicho valor.
    - No confundir con `null`, que es el valor que asignamos a objetos que aún no han sido instanciados.

La sentencia **`use strict`** al comienzo de un programa fuerza una sintaxis "restrictiva", que obliga, por ejemplo, a declarar las variables.

Identificadores

- **Reglas**:
  - Formado por letras, números, y los símbolos $ y _
  - El primer carácter NO puede ser un número
  - Se recomienda convención **`camelCase` para variables** y **mayúsculas para constantes**

```
let numero1;
let $numero1;

let letra
let $letra;
let $_letra;
```

```
let 1Nombre; /* Empieza por número */
let numero;1; /* Contiene ; */
```

Mostrando y leyendo variables

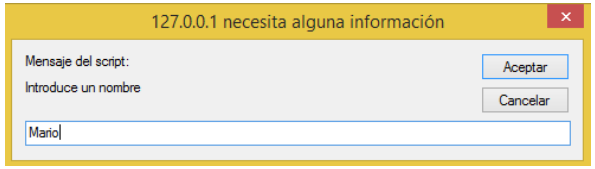
- La comunicación con el usuario se realiza a través de la página web (formularios, párrafos, etc.), pero de momento…

`alert(variable)`	Muestra el valor de la variable o literal en una ventana emergente
`variable=prompt("Mensaje")`	Muestra un formulario que permite dar valor a una variable.

```
var nombre="Mario";

console.log(nombre);
console.log ("Mario Bros");
```

```
var nombre=prompt("Introduce un nombre");
```

No usar en una Web "real"!



Tipos de variables

- Dependen del valor que tienen asignado:
  - numéricas/**number** (entero o real)

```
var partidas=99;
var fuerza=77.3;
```

  - Cadenas de texto/**string** (entre cadenas dobles o simples)
    - Para introducir caracteres especiales es necesario usar caracteres de escape (\n, \t, etc.)

```
var mensajeBienvenida="Bienvenido Mr. Marshall";
var nombreJugador='El Fary\n';
var letraPulsada='c';
```

  - Booleanos/**boolean** [true/false]

```
var deseaContinuar=true;
var haPagado=false;
```

Identificador de tipos

- typeof
  - Devuelve una cadena que representa el tipo de dato contenido en una variable
- isNaN
  - Devuelve verdadero si le pasamos algo que no sea un número

```
var numero=25;
alert(typeof(numero));
```

number

```
var numero="25";
alert(typeof(numero));
```

string

```
var numero=2.5;
alert(typeof(numero));
```

number

```
var numero=false;
alert(typeof(numero));
```

boolean

```
isNaN("hola")
```

true

Conversión de tipos: cadena a número

- `parseInt(cadena)` o `parseFloat(cadena)`

```
var numero=parseInt("30");
console.log(numero);
```
30

```
var numero2=parseInt("40px");
console.log(numero);
```
40

```
var numero3=parseInt("5pepe");
console.log(numero);
```
5

```
var numero4=parseInt("8+2");
console.log(numero);
```
8

```
var numero=parseFloat("3,2");
console.log(numero);
```
3

```
var numero2=parseFloat("3.2");
console.log(numero);
```
3.2

Coerción

- Es posible convertir un valor de un tipo a otro

IMPLÍCITA

```
let numero = 5;
console.log(numero);
```

EXPLÍCITA

```
console.log(numero.toString());
```

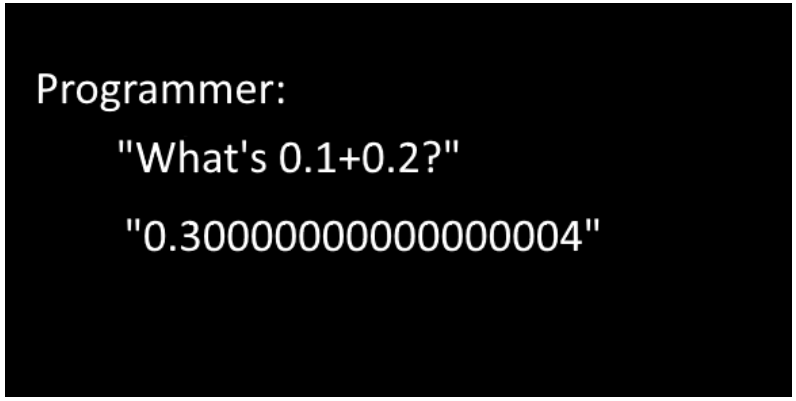
- Es importante saber cómo funcionan las **coerciones implícitas**:

```
let a = "2", b = 5;
console.log( typeof a + " " + typeof b); // string number
console.log( a + b ); // nos muestra 25
```

UD1 ACT1 Ejercicio1

PARA SABER MÁS

- Lee el artículo "Cosas extrañas de JavaScript"



Si no me crees…
¡Pruébalo!

Arrays

- Es una colección de elementos que pueden ser **del mismo o distinto tipo**

```
var dias = ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
            "Sábado", "Domingo"];
```

- Podemos acceder a los elementos del array a través de un índice
  - Las posiciones de un array comienzan a contarse en 0 y no en 1

```
var diaSeleccionado = dias[0]; // diaSeleccionado = "Lunes"
var otroDia = dias[5]; // otroDia = "Sábado"
```

```
const colores = ["#ff0000","#00ff00","0000ff"];
const [rojo, verde, azul] = colores;
```

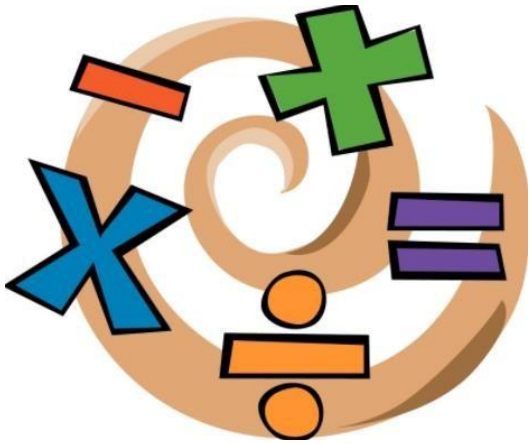
Desestructuración: para inicializar variables de forma rápida

Operadores

Indice

Operadores

- Permiten manipular el valor de las variables, realizar operaciones matemáticas con sus valores y comparar valores
- Tipos:
  - Asignación
  - Lógicos
  - Matemáticos
  - Relacionales
  - Identificación de tipos



Asignación (=)

- Se utiliza para guardar un valor en una variable
- Dos partes:
  - Izquierda del igual: Nombre de variable.
  - Derecha del igual: Expresión (variables, valores, condiciones lógicas…)

```
var numero1=3;
var numero2=4;

numero1=5;
numero1=numero2;
```

Incremento (++) y decremento (--)

- Se aplican a tipos numéricos y permiten sumar 1 a una variable de una forma simplificada

  - ✓ Si va delante de la variable su valor se incrementa antes de ejecutar la sentencia
  - ✓ Si va después de la variable su valor se incrementa después de ejecutar la sentencia

```
var numero1=5;
var numero2=2;
numero3=numero1++ + numero2;
```

numero3=7, numero1=6

```
var numero1=5;
var numero2=2;
numero3=++numero1 + numero2;
```

numero3=8, numero1=6

Operadores lógicos

- Devuelven un valor de tipo booleano
- Realizan una operación lógica
  - Negación (!)
  - “Y” Lógico(AND, &&)
  - “O” Lógico (OR, ||)

Operadores lógicos: Ejemplos

AND

```
var valor1=true;
var valor2=false;

resultado=valor1 && valor2; //resultado=false

valor1=true;
valor2=true;

resultado=valor1 && valor2; //resultado=true
```

NOT

```
var visible;
console.log(!visible); // "false"
```

OR

```
var valor1=true;
var valor2=false;

resultado=valor1 || valor2; //resultado=true

valor1=false;
valor2=false;

resultado=valor1 || valor2; //resultado=false
```

Operadores matemáticos

- Realizan operaciones matemáticas elementales
  - Suma (+)
  - Resta (-)
  - Multiplicación (*)
  - División (/)
  - Resto de la división entera (%).
- Puedo combinarlos con el operador de asignación para **acumular** sobre una variable (+=, -=, *=, etc.)

Operadores matemáticos: Ejemplos

Operaciones elementales

```
var num1=10;
var num2=5;

resultado=num1/num2; //resultado=2
resultado=3 + num1; //resultado=13
resultado=num2-4; //resultado=1
resultado=num1*num2; //resultado=50
```

Operaciones acumuladas

```
var num1=5;

num1+=3; //num1=num1+3;
num1-=1; //num1=num1-1;
num1*=2; //num1=num1*2;
num1/=5; //num1=num1/5;
num1%=4; //num1=num1%4;
```

Resto

```
var num1=10;
var num2=5;
resultado=num1%num2; //resultado=0

num1=9;
num2=5;
resultado=num1%num2; //resultado=4
```

Operadores relacionales

- Devuelven un valor booleano
  - Mayor que (>)
  - Menor que (<)
  - Mayor o igual (>=)
  - Menor o igual (<=)
  - Igual que (==)
  - Distinto de (!=)

```
var num1=3;
var num2=5;

resultado=num1>num2; //resultado=false;
resultado=num1<num2; //resultado=true;

num1=5;
num2=5;

resultado=num1 >= num2; //resultado=true;
resultado=num1 <= num2; //resultado=true;
resultado=num1 == num2; //resultado=true;
resultado=num1 != num2; //resultado=false;
```

Error común

- El operador *igual* (==) es origen de un gran número de errores de programación
- No confundir con el operador de asignación (=).

```
//El operador "=" asigna valores
var num1=5;
resultado=num1=3;
//num1 vale 3 y resultado vale3

//El operador "==" compara variables
var num1=5;
resultado=num1==3;
//num1 vale 5 luego resultado vale false
```

Comparación estricta

- === y !== permiten comparación **estricta**
  - Se compara el tipo de la variable además de su valor
  - Los operadores de comparación == y != son de comparación **relajada**
    - Si los tipos son distintos trata de trasnsformarlos para que sean comparables
    - La operación relajada tiene bastantes reglas que no siempre son fáciles de recordar, por eso por lo general se recomienda el uso de la **comparación estricta**

Comparación estricta vs relajada

```
var numero=10;
var numero2="10";

if(numero==numero2){
    /* .... */
}
```

El **if** se evalúa como verdadero

```
var numero=10;
var numero2="10";

if(numero===numero2){
    /* .... */
}
```

El **if** se evalúa como falso

Operadores con cadenas

- El operador + también se utiliza para **concatenar** cadenas

```
var frase1="Ojos que no ven";
var frase2="corazón que no siente";

var refran1=frase1+frase2;
var refran2=frase1+" castañazo que te pegas";

console.log(refran1);
console.log(refran2);
```

- Los operadores lógicos también pueden usarse para comparar cadenas de texto (alfabéticamente)

```
var  texto1="Hola";
var  texto2="Hola";
var texto3="Adios";

resultado=texto1==texto3;  //false
resultado=texto1!=texto2;  //false
resultado=texto1>=texto2;  //false
```

String literals

- Ofrecen una forma limpia de insertar variables (sustituyéndolas por su valor) en strings con ${} entre comillas invertidas:

```
var nombre="Jose";
var apellidos="Gonzalez";

console.log(`Mi nombre completo es ${nombre} ${apelldos}`);
```

Estructuras de control

Indice

Estructura condicional (`if … [else …]`)

- Las sentencias se ejecutan si se cumple la condición
  - Las llaves no son obligatorias si hay una única sentencia
  - La parte del else se ejecuta si no se cumple la condición
- Puedo encadenar varias condiciones simples o complejas

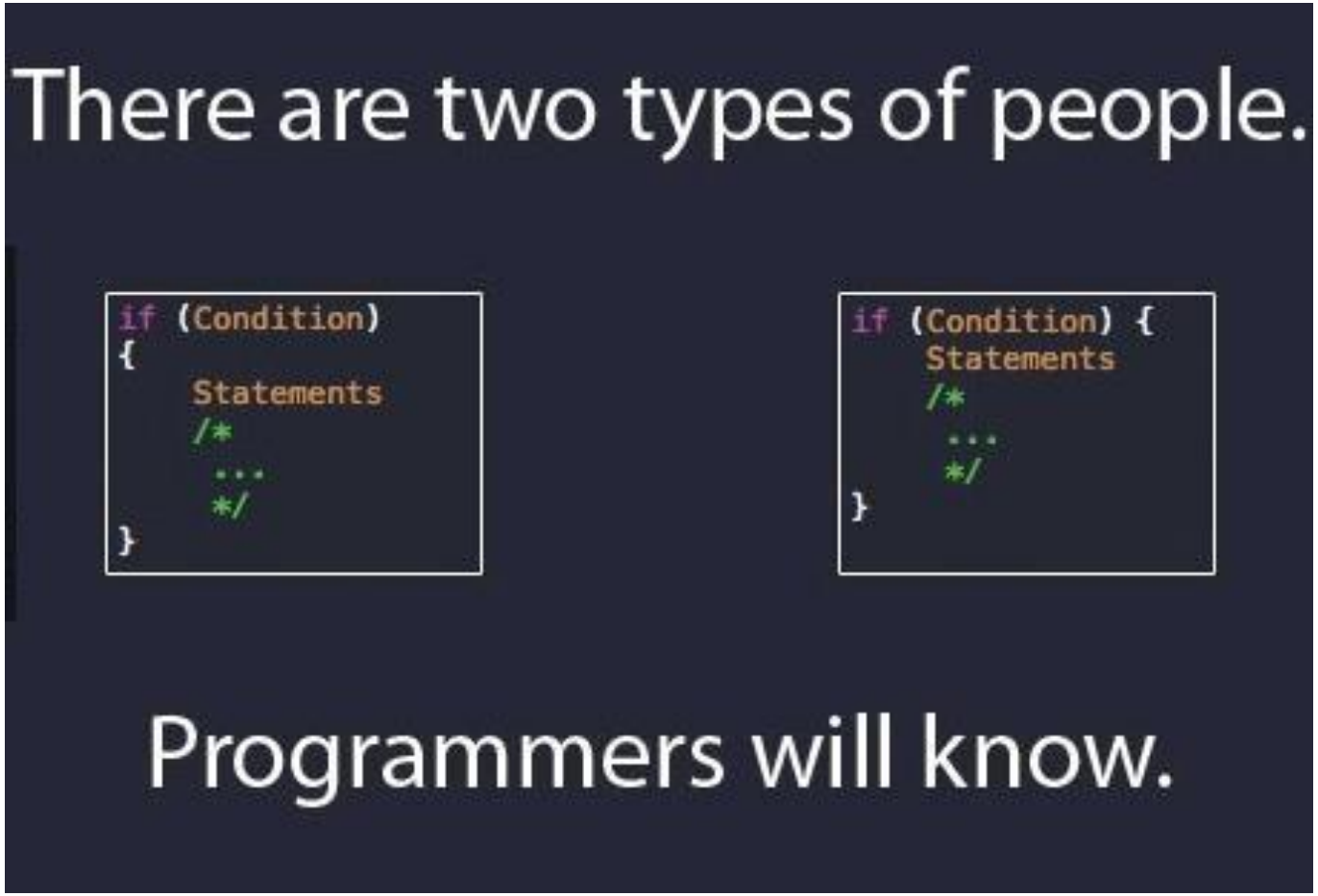
```
if(condicion){
sentencias;
}
else{
  otrasSentencias;
}
```

Equivalente a `condicion ==true`

Opcional

```
var mostrado=false;
var usuarioPermiteMsj=true;

if(!mostrado && usuarioPermiteMsj)
  console.log("PUM!");
```

Estructura condicional



Estructura condicional alternativa: Ejemplos

```
var edad=18;
if(edad>=18){
 console.log("Felicidades, ya eres mayor de edad");
}
else{
 console.log("Todavía eres menor de edad, pringao");
}
```

```
if(edad<=12)
console.log("Eres un niño")
else if (edad<19)
 console.log("Eres adolescente");
else if(edad<35)
console.log("Sigues siendo joven");
else
 console.log("Piensa en cuidarte un poco más");
```

Operador ternario

- Permite evaluar una condición y ejecutar una de dos expresiones en función de la misma

```
condición ? expresión1 : expresión2;
```

- **Condición**: Expresión que podemos evaluar como verdadero o falso.
- **Expresión 1**: Se ejecuta si condición es verdadero
- **Expresión 2**: Se ejecuta si condición es falso.

```
var miEdad = 24;
var mayorEdad = (miEdad > 18) ? "Sí, eres mayor de edad" : "No, sigue intentando";
```

Estructura múltiple (`switch`)

- Se basa en evaluar una expresión con resultado escalar, para decidir el punto de entrada en la estructura

```
switch(expresion){
 case valor1:
      sentencia1;
   break;

 case valor2:
      sentencia2;
   break;

 case valor3:
      sentencia3;
   break;
}
```

Bucle for

- Permite repetir una o varias sentencias un número determinado de veces
  - **inicialización**: Valor inicial de la variable que controla la repetición.
  - **condición**: Que debe cumplirse para que las sentencias se ejecuten.
  - **actualización**: Se ejecutan después de cada repetición. Normalmente actualiza la variable que controla la repetición.

```
var mensaje="Yuhuuu! Estoy dentro de un bucle!!";

for  (let i=0;      i<5;          i++)
 console.log(mensaje);
```

Inicialización

Condición

Actualización

Bucle `while`

- Repite una serie de sentencias mientras se cumpla una condición
  - Habitualmente en las sentencias controlo la condición de salida del bucle

- ¿Qué hace este ejemplo?→

```
while(condición){
    sentencias;
    }
```

```
var resultado=0;
var numero=100;
var i=0;

while(i<=numero){
resultado+=i;
i++;
}

alert(resultado);
```

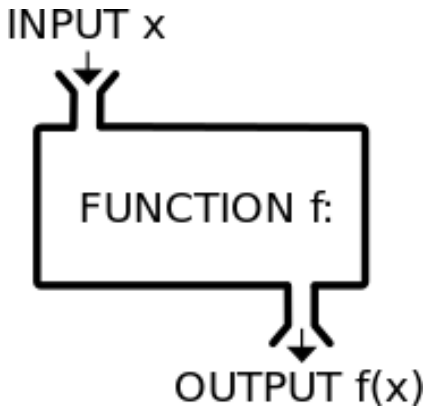
Bucle `do..while`

- Repite una serie de sentencias mientras se cumpla una condición
  - La diferencia con el **`while`** es que las sentencias se ejecutan **al menos una vez**, mientras que con el while podrían no ejecutarse nunca
- ¿Qué hace este ejemplo?→

```
do{
 sentencias;
} while(condición);
```

```
var resultado=1;
var numero=5;
do{
 resultado*=numero;
 numero--;
} while(numero>0);
alert(resultado);
```

UD1 ACT1 Ejercicio2



Funciones (introducción)

Funciones

- Una función es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y se pueden reutilizar de manera sencilla
- Facilitan mucho la organización, y por consiguiente el mantenimiento y depuración de los programas
- Son la base de la **programación funcional** (React)

Funciones simples

- Primero declaramos la función y luego la utilizamos (llamada o invocación):

```
/* Definición */
function nombreFuncion(){
sentencias;
}

/* Llamada o invocación */
nombreFuncion();
```

Funciones simples: Ejemplo

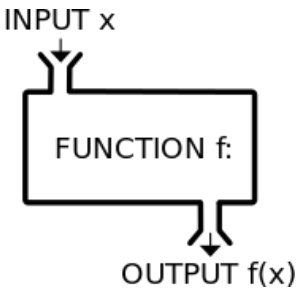
```
function sumayMuestra(){
 var resultado = numero1 + numero2;
 alert("El resultado es "+ resultado);
}
```

Definición

```
var numero1=3;
var numero2=5;

sumayMuestra();

numero1=5;
numero2=6;
sumayMuestra();
```

Llamada o invocación

E/S de datos en funciones



- **Argumentos/parámetros**
  - Permiten especificar las **entradas** de la función
- **Retorno**
  - Especifica el **valor que devuelve** la función.

```
/* Definición */
function nombreFuncion(argumento1, argumento2){
sentencias;
return valor;
}
```

Valor de retorno: Ejemplo

```
function suma(primernumero,segundonumero){
 var resultado = primernumero + segundonumero;
 return resultado;
}
```

Definición

```
//Declaración de las variables
var numero1=3;
var numero2=5;

//Llamada a la función
var resultado=suma(numero1,numero2);
alert(resultado);
```

Funciones anónimas

- Si sólo las necesito una vez, puedo crear y ejecutar la función en el momento:

```
console.log(function (){
    return "Comenzando..."
}())
```

Los () del final hacen que se ejecute

Además, en JavaScript...

- Las funciones son **un tipo de dato** más
  - Pueden guardarse en variables y constantes

```
const hello = function () {
    console.log("Hola Mundo");
}
hello();
```

  - Una función puede devolver otra función

```
function hello() {
    console.log("Hola Mundo");
    return function(){ return "Hola interno"};
}
console.log(hello()());
```

Funciones flecha

- Notación alternativa muy usada para definir funciones

```
function sumar(x,y){
    return x+y
}
const sumar = (x,y)=>x+y;
```

EQUIVALE A

```
const sumar = (x,y)=>x+y;
```

Parámetros que recibe la función

Cuerpo de la función
Si solo tiene una línea no hay que poner llaves y se asume un return

Funciones flecha

Si la función solo tiene una sentencia (que es el `return`)	• Sin function • Sin llaves • Sin return

```
() => sentencia; //función sin parámetros
unParam=> sentencia; //función con un parámetro
(param1, param2,paramN) => sentencia; //más parámetros
```

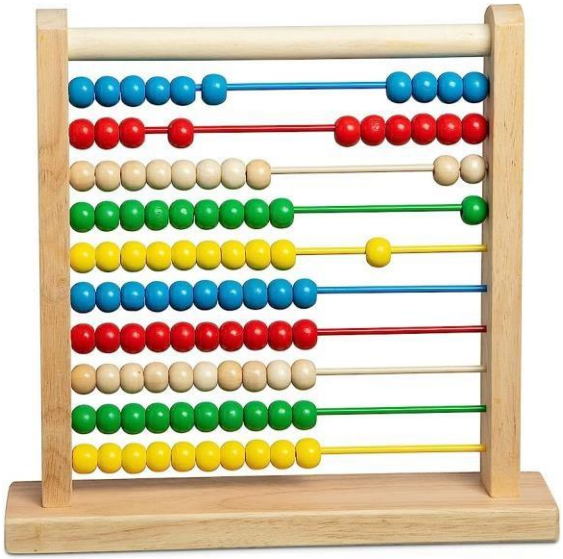
Si la función tiene varias sentencias	• Sin function • Con llaves • Con return

```
() => {  sentencias;}
unParam=> { sentencias; }
(p1, p2,pN) => { sentencias; }
```

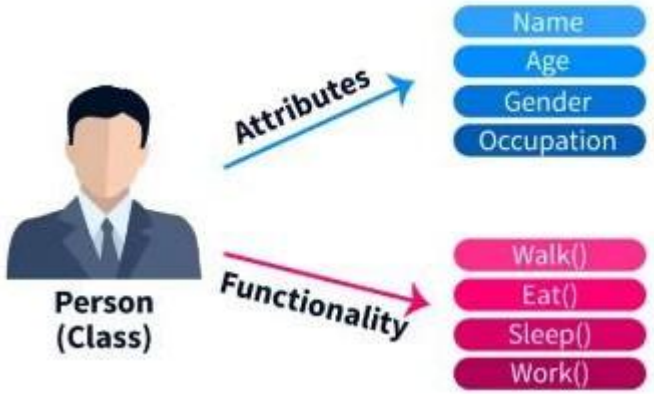
Ejemplo funciones flecha

```
//Sintaxis convencional 1 ("estilo Java")
function arraysConcatenados(array1, array2) {
  return array1.concat(array2);
}

//Sintaxis convencional 2 ("estilo JavaScript")
const arraysConcatenados= function(array1, array2) {
  return array1.concat(array2);
}

// Sintaxis con función flecha ("estilo JavaScript Moderno")
const arraysConcatenados= (array1, array2) => array1.concat(array2);

//A las tres se les invoca de la misma forma
console.log(arraysConcatenados([1,2],[3,4,5]));
```

UD1 ACT1 Ejercicio 3



UD1 ACT1 Ejercicio 4



Objetos (introducción)

Indice

Objetos en JavaScript

- En la mayoría de lenguajes de programación los objetos se crean con **new**
  - JavaScript también lo permite desde 2015
  - Se usa cuando hacemos **programación orientada a objetos**
- JavaScript permite además utilizar la **notación literal** para crear objetos
  - Más abreviada
  - “Línea directa” con JSON

Objetos literales

- Se crean con las llaves {}

```
// Esto es un objeto vacío
const objeto = {};
```

- Se entienden como un conjunto de variables de cualquier tipo declaradas como **clave: valor** sin necesidad de crear una clase

```
const jugador = {
    nombre: "Pepelu",
    vidas: 99,
    potencia: 10,
};
```

  - Acceso a propiedad es con punto o corchete

```
// Notación con puntos (preferida)
console.log(jugador.nombre); // Muestra "Manz"
// Notación con corchetes
console.log(jugador["vidas"]);// Muestra 99
```

Métodos en objetos literales

```
const usuario = {
    nombre: "Manolito García",
    edad: 30,
    nacimiento: {
        pais: "España",
        ciudad: "Oviedo"
    },
    amigos: ["Menganito", "Antoñito"],
    activo: true,
    sendMail: function (){
        return "Enviando email..."
    }
}
console.log(usuario);
console.log(usuario.nombre);
console.log(usuario.nacimiento.ciudad);
console.log(usuario.amigos)
console.log(usuario.sendMail); //devuelve el código
console.log(usuario.sendMail()); //ejecuta la función
```

EQUIVALENTE:

```
sendMail (){
  return "Enviando email..."
}
```

UD1 ACT1 Ejercicio 5

UD1 ACT1 Ejercicio 6

Shorthand property names

- Podemos crear un objeto a partir de otras constantes/variables
  - Existe una notación abreviada donde sólo es necesario poner su nombre y se crea la propiedad

```
const nombre="portatil"
const precio =3000;

const nuevoProducto = {
    nombre: nombre,
    precio: precio
}
```

```
const
nombre="portatil"
const precio =3000;

const nuevoProducto =
{
    nombre
    precio
}
```

Desestructurar un objeto

- Las **llaves** me permiten usar sólo una parte del objeto:

```
function imprimirInfo({nombre}){
    return "<h1>Hola "+nombre+"</h1>";
}
document.body.innerHTML=imprimirInfo(usuario);
```

La función recibe el objeto completo pero solo usa una propiedad

- También se puede pasar el objeto completo y luego desestructurarlo antes de usarlo:

```
function imprimirInfo(usuario){
    const {nombre, edad} = usuario;
    return "<h1>Hola "+nombre+"</h1>";
}
document.body.innerHTML=imprimirInfo(usuario);
```

UD1 ACT1 Ejercicio 7

- Batalla Pokemon!

¿Cómo te fue?