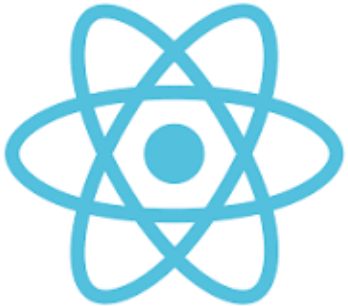
<codoa codo/>

Agencia de Aprendizaje a lo largo de la vida

ESPECIALIZACIÓN REACT



BA

<codoa codo/>

Clase 02. REACT JS

REPASO JS

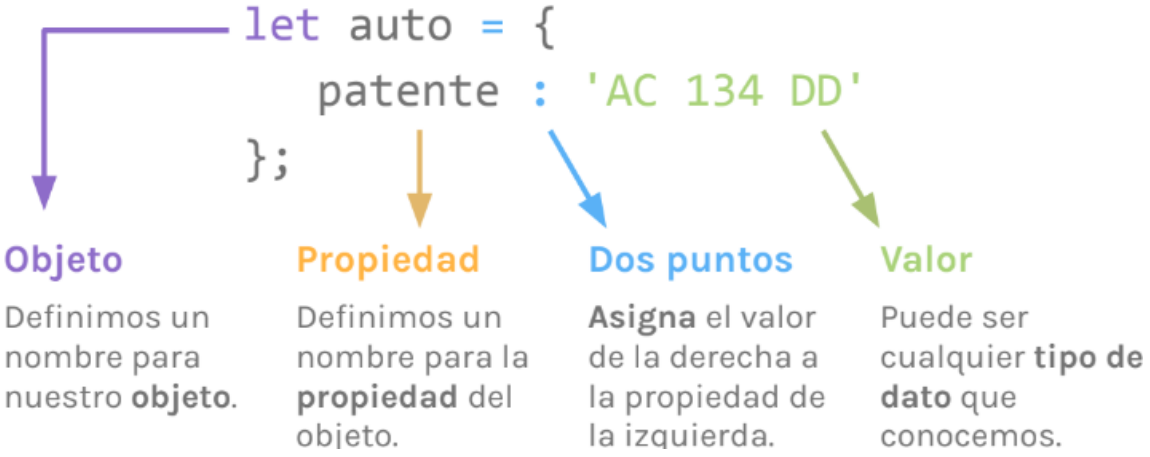
BA

OBJETOS

OBJETOS / ESTRUCTURA

Podemos decir que son la representación en código de un elemento de la vida real.

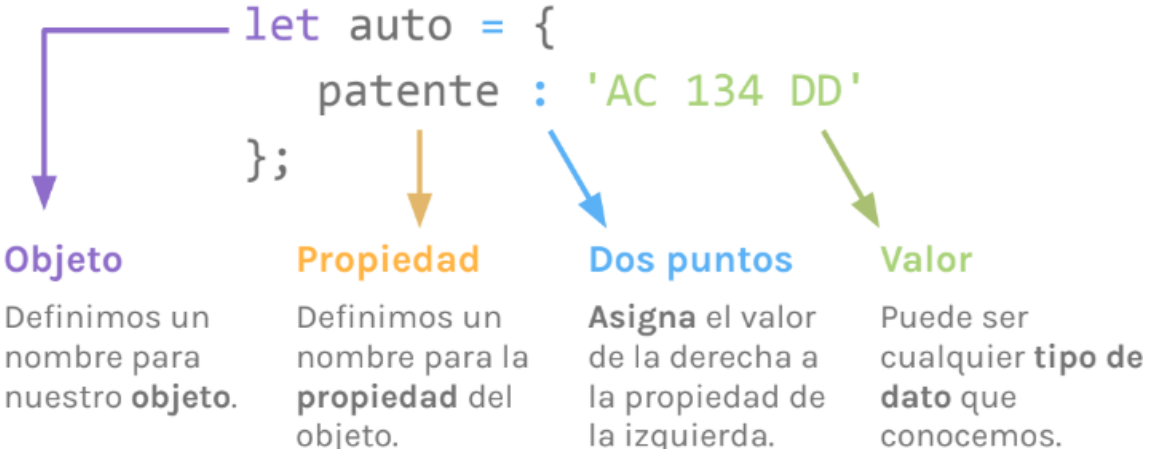
Un **objeto** es una estructura de datos que puede contener **propiedades** y **métodos.**
Para crearlo usamos llave de apertura y de cierre { }.



OBJETOS / ESTRUCTURA

Podemos decir que son la representación en código de un elemento de la vida real.

Un **objeto** es una estructura de datos que puede contener **propiedades** y **métodos.**
Para crearlo usamos llave de apertura y de cierre { }.



OBJETOS / ESTRUCTURA

Un **objeto** puede tener la cantidad de propiedades que queramos, si hay más de una las separamos con comas , .
Con la notación **objeto.propiedad** accedemos al **valor** de cada una de ellas.

```
let persona = {
    nombre: 'Bruce',
    apellido: 'Wayne'
};

console.log(persona.nombre) // Bruce
console.log(persona.apellido) // Wayne
```

MÉTODOS DE UN OBJETO

Una propiedad puede **almacenar** cualquier **tipo de dato**.
Si una propiedad almacena una función, diremos que es un **método** del objeto.

```
let persona = {
    nombre: 'Bruce',
    apellido: 'Wayne',
    edad: 38,
    saludar: function() {
        return '¡Hola! Me llamo Bruce';
    }
};
```

MÉTODOS DE UN OBJETO

Una propiedad puede **almacenar** cualquier **tipo de dato**.
Si una propiedad almacena una función, diremos que es un **método** del objeto.

```
let persona = {
    nombre: 'Bruce',
    apellido: 'Wayne',
    edad: 38,
    saludar: function() {
        return '¡Hola! Me llamo Bruce';
    }
};
```

CONSTRUIR UN OBJETO

Clase

Funcionan como las plantillas y son utilizadas para instanciar (crear) objetos.
Una clase encapsula (contiene) todas las propiedades y métodos que después almacenarán los objetosn instanciados.
Una clase Representa a un tipo de objeto; ejemplo: Libro, Automóvil, Perro.

Constructor

Es un método que se llama en el momento de la creación de instancias (objetos).
Los constructores son útiles para (valga la redundancia) "construir" o inicializar las propiedades de los objetos.

```
class Curso {
  constructor(instructor, tecnologia) {
    this.instructor = instructor;
    this.tecnologia = tecnologia;
  }
}

let react = new Curso("Gabriel Muñoz", "React");
let php = new Curso("Alejandro Zapata", "php");

console.log(react);  Curso { instructor: 'Gabriel Muñoz', tecnologia: 'React' }
console.log(php);  Curso { instructor: 'Alejandro Zapata', tecnologia: 'php' }
```

La clase Curso , tiene un constructor que espera 2 parámetros : instructor y tecnología
Para crear un objeto curso debemos usar la palabra reservada new y llamar a la clase pasándoles los parámetros que espera

FUNCIONES

INTRODUCCION A LAS FUNCIONES

Las funciones son una parte muy importante de todo lenguaje de programacion y sobre todo en JavaScript. Són tipos particulares de Objetos, llamados `callable objects` u objetos invocables, por lo que tienen las mismas propiedades que cualquier objeto.

Ahora que tenemos un conjunto de variables, necesitamos funciones para calcularlas, cambiarlas, hacer algo con ellas. Hay tres formas en que podemos construir una función.

```
function miFuncion() {}
let otraFuncion = function () {};
let yOtra = () => {};
```

Una función comenzará con la palabra clave `function`, esto le dice a lo que sea que esté ejecutando tu programa que lo que sigue es una función y que debe tratarse como tal.
Después de eso viene el nombre de la función, nos gusta dar nombres de funciones que describan lo que hacen. Luego viene un paréntesis abierto y uno cercano.
Y finalmente, abra y cierre los corchetes. Entre estos corchetes es donde irá todo nuestro código a ejecutar.

ARGUMENTOS

Ahora que podemos ejecutar una función básica, vamos a comenzar a pasarle argumentos.

```
function logHola(nombre) {
  console.log("Hola, " + nombre);
}

logHola("Martin");
```

Si agregamos una variable a los paréntesis cuando declaramos la función, podemos usar esta variable dentro de nuestra función. Iniciamos el valor de esta variable pasándola a la función cuando la llamamos. Entonces en este caso `nombre = 'Martin'`. También podemos pasar otras variables a esto:

```
function logHola(nombre) {
  console.log(`Hola, ${nombre}`);
}

let miNombre = "Antonio";
logHola(miNombre);
```

Podemos agregar múltiples argumentos colocando una coma entre ellos:

```
function sumarDosNumeros(a, b) {
  var suma = a + b;
  return suma;
}

sumarDosNumeros(1, 5); // 6
```

En el ejemplo anterior presentamos la declaración `return`.
No vamos a usar `console.log` con todo lo que salga de una función.
Lo más probable es que queramos devolver algo. En este caso es la suma de los dos números. Piense en la declaración de retorno ("return") como la única forma en que los datos escapan de una función.
No se puede acceder a nada más que a lo que se devuelve fuera de la función.
También tenga en cuenta que cuando una función golpea una declaración de retorno, la función detiene inmediatamente lo que está haciendo y "devuelve" lo especificado.

FUNCIONES FLECHA

funcion flecha, se elimina la palabra function y se coloca la flecha luego de los parentesis

ejemplos

si solamente usamos un parámetro no lleva paréntesis, y si tiene una sola expresión o línea de códigos NO LLEVA LLAVES

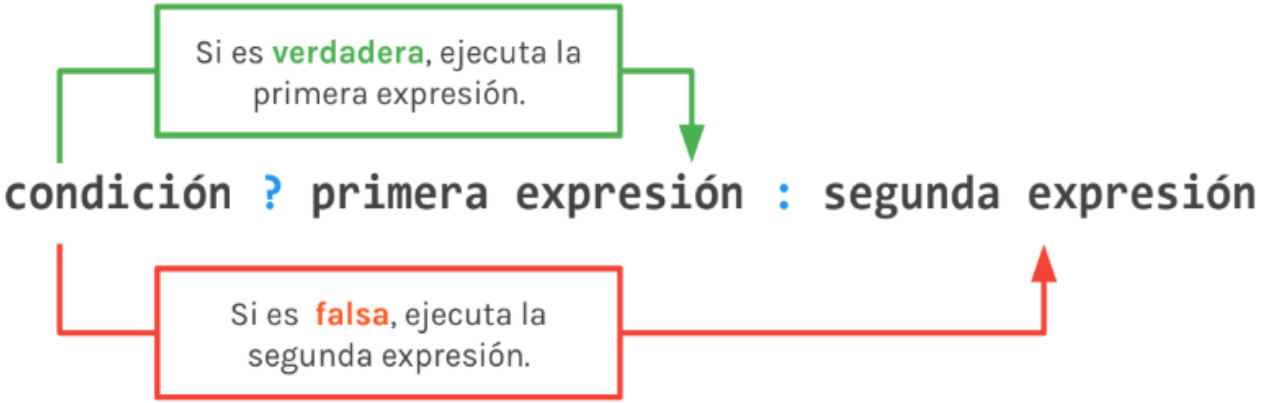
```
const saludarOK = nombre => document.write(`Hola ${nombre} como estas`);

saludarOK("Codo a Codo")

let suma = (num1, num2) => {
let resultado = num1+ num2
document.write(resultado);
document.write("<br>")
}
suma (32,32) ; //64
```

IF
IF TERNARIO

DEFINIENDO UN IF TERNARIO



Nos permiten **evaluar condiciones** y realizar diferentes acciones **según el resultado** de esas evaluaciones.

A diferencia de un if tradicional, el **if ternario** se escribe de forma **horizontal**. Ambas estructuras tienen el mismo flujo interno (si esta condición es verdadera hacé esto, si no, hacé ésto otro) pero en este caso no hace falta escribir la palabra **if** ni la palabra **else**.

IF TERNARIO

La sintaxis de un operador ternario es la siguiente: condición ? valor verdadero : valor falso;

Para entenderlo bien, vamos a reescribir el ejemplo de los temas anteriores utilizando este operador ternario. Primero, recordemos el ejemplo utilizando estructuras if/else:

```
let nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");
 if (nota < 5) {
// Acción A: nota es menor que 5
calificacion = "suspendido";
 } else {
 // Acción B: Cualquier otro caso diferente a A (nota es mayor o igual que 5)
calificacion = "aprobado"; }
console.log("Estoy", calificacion);
```

IF TERNARIO

Ahora, vamos a reescribirlo utilizando un **operador ternario**:

```
let nota = 7;
console.log("He realizado mi examen. Mi resultado es el siguiente:");
// Operador ternario: (condición ? verdadero : falso)
let calificacion = nota < 5 ? "suspendido" : "aprobado";
console.log("Estoy", calificacion);
```

Repasemos el ejemplo:

Observa que guardamos en calificacion el resultado del operador ternario.
La condición es nota < 5, se escribe al principio, previo al ?.
Si la condición es cierta, el ternario devuelve "suspendido".
Si la condición es falsa, el ternario devuelve "aprobado".
Este ejemplo hace exactamente lo mismo que el ejemplo anterior del if/else.
La idea del operador ternario es que podemos condensar mucho código y tener un if en una sola línea. Es muy práctico, legible e ideal para ejemplos pequeños donde almacenamos la información en una variable para luego utilizarla.