Utilización de los objetos predefinidos del lenguaje javascript

--

Utilización de objetos

--

Objetos Nativos

1. Utilización de objetos

a. Paradigma

La programación orientada a objetos (POO) es un modelo de programación que alcanzó una total implantación en la década de los 90. Consigue una modularización mayor que la propia programación modular. Una forma simple de explicar POO es que se basa en la construcción de objetos que se pueden comunicar entre sí. Un objeto es una estructura que aglutina datos y acciones. Los datos son conocidos como atributos o propiedades del objeto. A las acciones se las conoce como métodos. Podemos definir diferentes tipos de objetos. Los tipos son conocidos como clases.
Por ejemplo, un objeto de la clase coche podría tener propiedades como marca, color, velocidad, y métodos como arrancar, parar, repostar.
Los objetos deben contar además con un mecanismo que permita generar objetos de un tipo determinado. A esto se conoce con el término constructor.
La POO debe cumplir con las características

i. Abstracción.

Gracias a su interfaz externa podemos reutilizar un objeto con desconocimiento de su estructura interna o implementación.

ii. Encapsulamiento

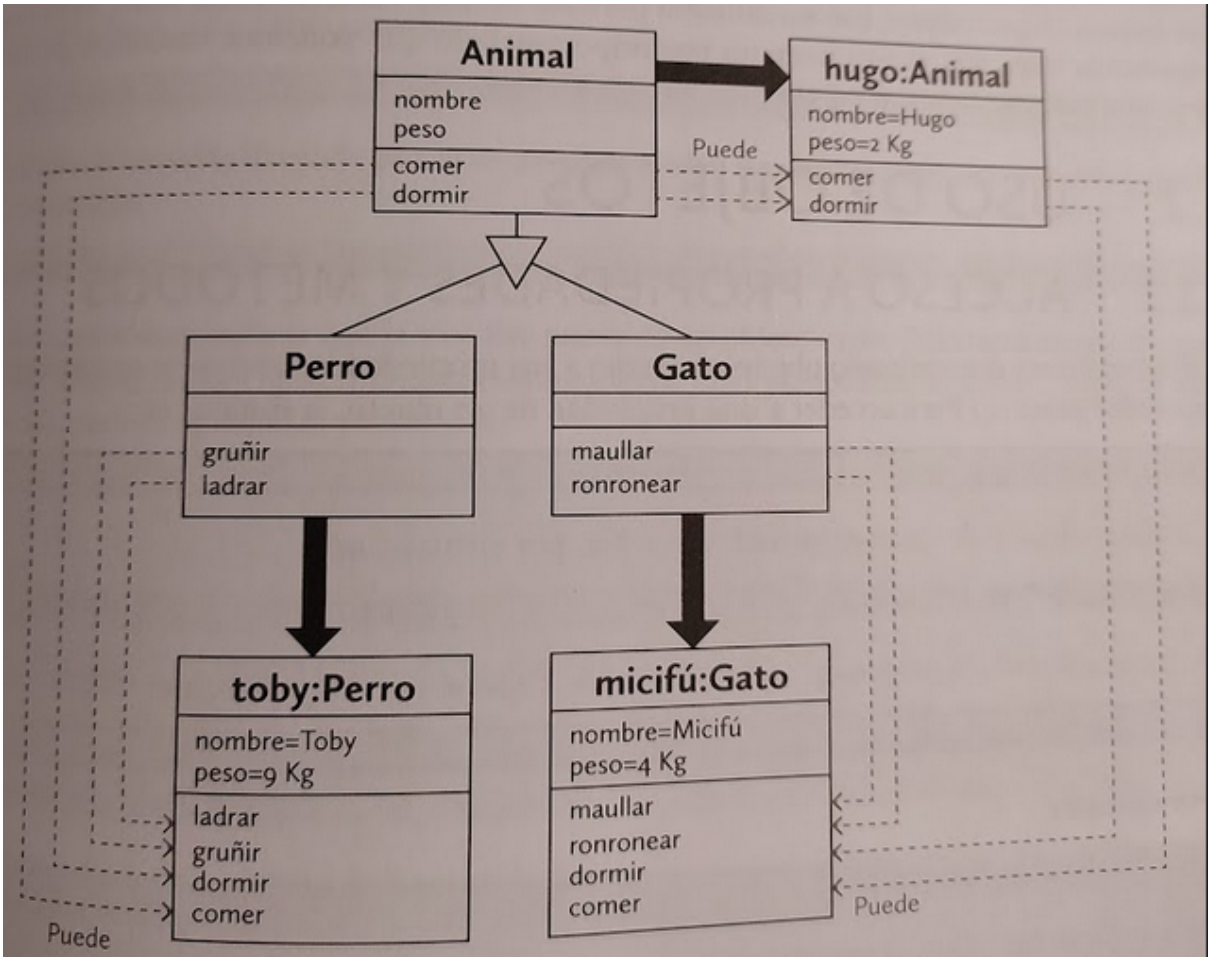
Capacidad para ocultar métodos y propiedades de un objeto y elegir cuales queremos que sean visibles.

iii. Polimorfismo

Diferentes objetos pueden tener métodos con el mismo nombre, aunque las acciones realizadas sean diferentes.

iv. Herencia

Permite que una clase herede características de otras clases. Por ejemplo, podríamos crear una clase automóvil que heredará los métodos de la clase coche. Ejemplo de clases con herencia:



Clases: Animal, Perro y Gato.
Propiedades:nombre y peso.
Métodos: comer, dormir, gruñir, ladrar, maullar y ronronear.
Objetos o instancias de clases: toby, hugo y micifú.

JavaScript genera debate en cuanto a si es realmente un lenguaje de POO. Esto es debido a que no es purista en cuanto al uso de los objetos. Por ejemplo, JavaScript permite el uso de prototipos en lugar de clases. (Se han incluido las clases en javascript para comodidad de los programadores “clásicos”). El uso de prototipos es más flexible ya que los prototipos son modificables (tanto en lo que a atributos como a métodos se refiere). Todos los objetos creados en base a ese prototipo quedarán automáticamente modificados.

b. Acceso a propiedades y métodos

Se consigue acceder a las propiedades e invocar los métodos de los objetos gracias al operador punto (.). Por ejemplo:

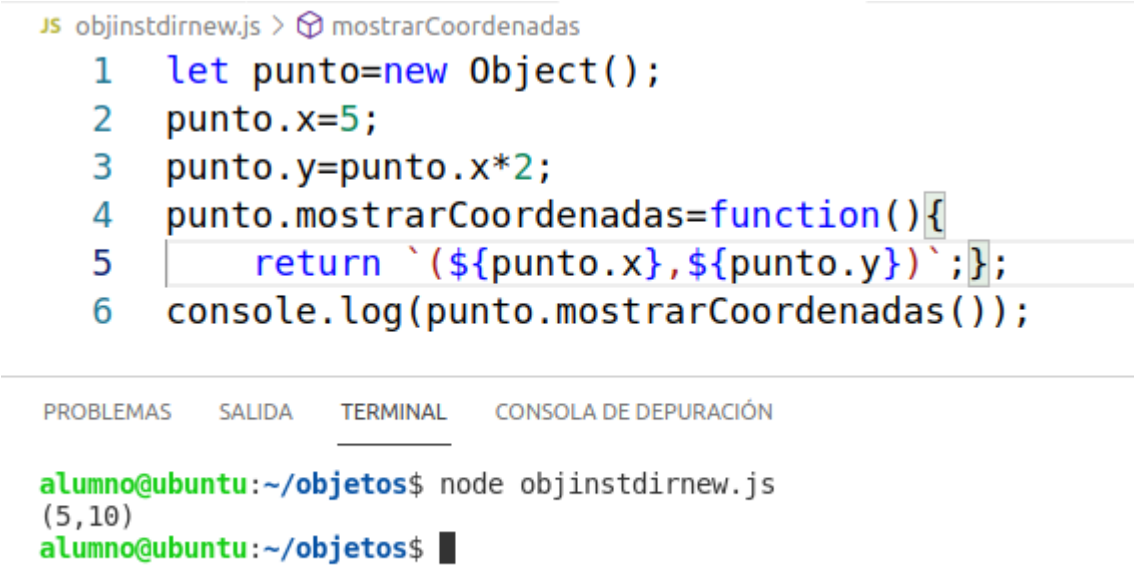
```
coche.color=”rojo”;
coche.acelerar(25);
```

Aunque también disponemos de la notación:

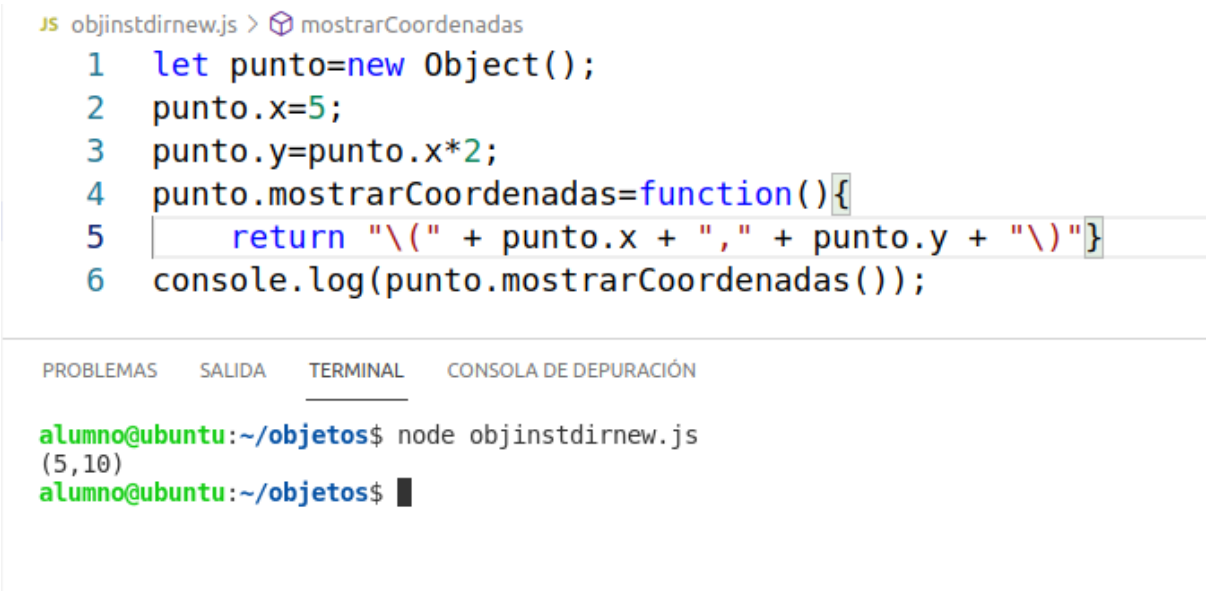
```
coche[“color”]=”rojo”;
coche[“acelerar”](25);
```

c. Creación de objetos mediante el método de instancias directas (objetos literales)

Admite dos sintaxis. La primera con la palabra clave new. Una vez creado el objeto, se van definiendo las propiedades y métodos.



Fíjate en cómo se ha construido el literal devuelto por la función anónima. También sería válido construirlo de este otro modo:



La segunda sintaxis no utiliza la palabra clave new y el objeto Object() sino que se describe directamente la estructura del nuevo objeto.

JS coordenadas1.js > ...

```
let punto={
    x:5,
    y:32,
    mostrarCoordenadas:function(){
        return `(${punto.x},${punto.y})`;
    }
};
console.log(punto.mostrarCoordenadas());
```

```
alumno@ubuntu:~/objetos$ node coordenadas1.js
(5,32)
alumno@ubuntu:~/objetos$
```

Vamos a practicar la creación de objetos por instancia directa diseñando un programa javascript (ejecutar desde la terminal con node) que define un **objeto *animal*** con las **propiedades *nombre* (Toby)** y ***peso* (20)**, y los **métodos *dormir*** (devuelve el literal “todo el día”) y ***comer*** (devuelve el literal “pienso para perros”). Visualiza a continuación en la terminal, qué debe comer y cuánto debe dormir Toby utilizando, por supuesto los métodos del objeto animal.

Versión con new Object():

```
let animal=new Object();
animal.nombre="Toby",
animal.peso=20,
animal.comer=function(){
        return "pienso de perros";
    },
animal.dormir=function(){
        return "todo el día";
    }
console.log(animal.nombre + " debe comer " + animal.comer());
console.log(animal.nombre + " debe dormir " + animal.dormir());
```

Versión sin new Object()

```
let animal={
    nombre:"Toby",
    peso:20,
    comer:function(){
        return "pienso de perros";
    },
    dormir:function(){
        return "todo el día";
    }
};
console.log(animal.nombre + " debe comer " + animal.comer());
console.log(animal.nombre + " debe dormir " + animal.dormir());
```

Crea una página web que pida el nombre y el peso de un perro mediante prompt. Con esos datos se crea un objeto animal con la mismas características que en el ejercicio anterior. La página web devuelve mediante alert los mensajes de lo que debe comer y dormir el animal, utilizando claro está los métodos del objeto animal.

crearperro.html > html > body > script

```
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
</head>
<body>

    <script src=crearanimal.js></script>
</body>
</html>
```

A continuación se muestra el contenido del fichero crearanimal.js.

```
var nombreInt=prompt("Introduce nombre ");
var pesoInt=prompt("Introduce peso ");
let animal=new Object();
animal.nombre=nombreInt,
animal.peso=pesoInt,
animal.comer=function(){
        return "pienso de perros";
    },
animal.dormir=function(){
        return "todo el día";
    }
alert(animal.nombre + " debe comer " + animal.comer());
alert(animal.nombre + " debe dormir " + animal.dormir());
```

Practica por tu cuenta realizando el siguiente ejercicio.
Se desea diseñar un ***objeto*** denominado ***cuenta.*** Su estructura es la siguiente:

Propiedades:
- numeroCuenta
- nombreTitular
- Nacionalidad
- DNI
- banco
- swift
- sucursal
- saldo (inicializado a cero)

Métodos:
- datosTitular Su función es mostrar el literal:

  Titular: XXXXXXXXXX
  DNI: XXXXXXXXXX
  Nacionalidad: XXXXXXXXXXXXX

- datosBanco Su función es mostrar el literal:

  Banco: XXXXXXXXXX
  Códigp SWIFT: XXXXXXXXXX

Se creará una página web que solicitará los valores de las propiedades (excepto el saldo) mediante prompt(), a continuación se visualiza los datos del titular y los del banco utilizando los métodos datosTitular y DatosBanco mediante document.write. Haz una primera versión con el método que te encuentres más cómodo (con o sin new). Finalmente, haz una segunda versión con el otro método.

d. La palabra clave this

Se trata de una referencia al objeto actual, al que está siendo usado en el preciso instante de la ejecución de la sentencia donde aparece.

```
function doblarx(){
  return this.x*=2;
   };
let punto={
   x:15,
   y:7,
   doble:doblarx
};

let incognita={
   x:5,
   dbl:doblarx
};
punto.doble();
incognita.dbl();
console.log(punto.x); //30
console.log(incognita.x);//10
```

Vemos como gracias al empleo de this podemos utilizar la función doblarx, invocada desde dos métodos pertenecientes a diferentes objetos.

Practiquemos el uso de estas funciones para crear métodos añadiendo al objeto animal, la propiedad ***ingesta***. En principio vale 0. Se trata de incluir un método que, mediante una función externa al objeto, sea capaz de calcular cuántos gramos al día debe ingerir el animal. Se calcula multiplicando por 50 el peso del animal. Visualizar el resultado por console.

```
function calcular(){
   return this.ingesta+=this.peso*50;
    }
 var animal={
    nombre:"Toby",
    peso:7,
    ingesta:0,
    comida:calcular
 };
  animal.comida();
 console.log(animal.ingesta);
```

Modifica la página web que crea el objeto animal con new object() añadiendo la nueva propiedad (ingesta) y el nuevo método (calcular la cantidad de pienso diaria que debe comer el animal). También se añadirá la visualización del cálculo mediante un alert.

Añade el método ingresar al objeto de la cuenta bancaria. Solicita una cantidad y la suma al código.

Añade otro método que visualiza el saldo con alert().

e. Recorrer las propiedades de un objeto

Para recorrer las propiedades de un objeto está indicado el uso de for … in.

```
for(let propiedad in punto){
   console.log(propiedad + " tiene el valor " +
`${punto[propiedad]}`);};
```

Ejercicio: Añade al ejercicio de las coordenadas con la función doblarx un bucle que permita mostrar las propiedades del objeto punto junto a su contenido.

```
1  function doblarx(){
2     return this.x*=2;
3     };
4  let punto={
5     x:15,
6     y:7,
7     doble:doblarx
8  };
9
10 let incognita={
11    x:5,
12    dbl:doblarx
13 };
14 punto.doble();
15 incognita.dbl();
16 console.log(punto.x); //30
17 console.log(incognita.x);//10
18 for(let propiedad in punto){
19    console.log(propiedad + " tiene el valor " + `${punto[propiedad]}`);};
20
21
```

```
PROBLEMAS   SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN

alumno@ubuntu:~/objetos$ node pruebathis.js
30
10
x tiene el valor 30
y tiene el valor 7
doble tiene el valor function doblarx(){
   return this.x*=2;
    }
alumno@ubuntu:~/objetos$
```

Ejercicio: Recorre y muestra las propiedades y métodos del último objeto animal.

f. Borrar propiedades de un objeto

Ejercicio: Eliminar el atributo x del objeto punto y recorrer las propiedades del objeto punto visualizándolas para comprobar que ya no existe punto.x.

```
delete punto.x;
for(let propiedad in punto){
   console.log(propiedad + " tiene el valor " +
`${punto[propiedad]}`);//propiedad en sí
};
```

Ejercicio: Elimina la propiedad ingesta y el método que la calcula del último objeto animal. Comprueba lo ocurrido con un bucle for … in.

g. Añadir propiedades a un objeto

```
Object.defineProperty(punto, 'z', {value:40, writable:true});
console.log(punto.z);
punto.z=10;
console.log(punto.z);
```

Ejercicio: Añade la propiedad sexo al último objeto animal. A continuación, se solicitará su valor mediante prompt. Comprobar que el usuario solamente pueda introducir ‘M’ o ‘H’.

h. Crear objetos a través de constructores

Esta es la forma “clásica” de crear objetos en lenguajes POO como java o C++. Javascript lo permite también aunque, si conoces otros lenguajes, notarás un matiz diferente.

Javascript aprovecha el hecho de que sus funciones tienen la capacidad de devolver objetos y de que se puede utilizar this para referenciar el objeto actual que está manejando la función. Si a esto se le une el operador new, que permite crear objetos a partir de funciones, aquí tenemos la estratagema utilizada en javascript para trabajar con constructores de objetos.

```
//ejemplo de una función constructora
function Punto(coordx,coordy){
   this.x=coordx;
   this.y=coordy;
```

```
this.mostrarxy=function(){return "\(" + this.x + "," + this.y + "\)";};
//this.mostrarxy=()=>`(${this.x},${this.y})`;
}
let a=new Punto(10,20);
let b=new Punto(-4,3);

console.log(a.mostrarxy());
console.log(b.mostrarxy());
```

De está forma hemos creado dos instancias (a y b) de la clase Punto. En javascript, es más apropiado decir que hemos creado dos objetos (a y b) del tipo de objeto Punto.

Ejercicio:
Crea dos animales con la primera estructura inicial utilizando constructor. Primero, haremos una versión en la que el nombre y el peso se pasarán como parámetros, mediante literales.

Luego una segunda versión donde se pedirán mediante prompts.

i. Operador instanceof

Llegados a este punto, podemos entender mejor porqué es útil el uso de instanceof, ya que typeof(), simplemente indicaría que a es un objeto y no que es un objeto de tipo Punto. También podemos usar constructor.name de la clase Object, puesto que todos los objetos heredan de la clase básica, Object.

```
function Punto(coordx,coordy){
  this.x=coordx;
  this.y=coordy;
  this.mostrarxy=()=>`(${this.x},${this.y})`;
}
let a=new Punto(10,20);
let b=new Punto(-4,3);
console.log(a.mostrarxy());
console.log(b.mostrarxy());
console.log("Mostrando comprobciones");
console.log(typeof b);//object
console.log(b instanceof Object);//true
console.log(b instanceof Punto);//true
console.log(b.constructor.name);//Punto
```

j. Prototipos

Como se indicó en el primer apartado de este documento, javascript utiliza un concepto muy interesante, el prototipo. Todos los objetos tienen prototipo, lo heredan del objeto Object. En la POO clásica, al crear un objeto se copian los atributos y métodos. En Javascript, lo que

ocurre es que los objetos quedan enlazados al prototipo de la clase. Si tenemos en cuenta que los prototipos son modificables, si lo hacemos, estaremos modificando de forma instantánea los objetos de ese tipo.

```
//prototipos
//Visualizar el prototipo. Son equivalentes las dos sentencias
//sale un conjunto vacío ya que no se ha modificado aún
console.log(a.__proto__);
console.log(Punto.prototype);
//Moficación del prototipo
Punto.prototype.sumaxy=function(){
 return this.x+this.y;
};
Punto.prototype.z=0;
//ahora al visualizar el prototipo salen las modificaciones realizadas
console.log(a.__proto__);
console.log(Punto.prototype);
//Tambíen se puede comprobar recorriendo las propiedade del objeto a
for(let propiedad in a){
   console.log(propiedad + " tiene el valor " +
`${a[propiedad]}`);//propiedad en sí
};
```

Captura de la salida:

```
{}
{}
{ sumaxy: [Function (anonymous)], z: 0 }
{ sumaxy: [Function (anonymous)], z: 0 }
x tiene el valor 10
y tiene el valor 20
mostrarxy tiene el valor ()=>`(${this.x},${this.y})`
sumaxy tiene el valor function(){
  return this.x+this.y;
}
z tiene el valor 0
```

Se puede comprobar que el objeto b también ha sido modificado.

Ejercicio: Crea dos instancias del objeto animal que está vez habrás creado mediante constructor. Visualiza el prototipo. Debe aparecer un conjunto vacío. Modifica el prototipo añadiendo nombre y teléfono del veterinario asignado al animal. Visualiza de nuevo el prototipo. Debe aparecer las nuevas propiedades. Asigna un nombre de veterinario y un

teléfono a cada uno de los animales y visualiza un mensaje del estilo “El veterinario de Lani es Juan con teléfono 565777777”, para cada uno de los animales.

Ejercicio:
Crea un constructor vacío para crear objetos “animal”. Mediante el uso del prototipo, asígnale las propiedades nombre y peso. Crea dos instancias, asígnale valores y visualiza el prototipo y el contenido de las dos instancias con unos mensajes del estilo el peso de xxxxx es xx.

```
function Animal(){

   };
let ani1=new Animal();
Animal.prototype.nombre="";
Animal.prototype.peso=0;
Animal.prototype.comer=function(){return "pienso para perros"};
console.log(Animal.prototype);
ani1.nombre="Lani";
ani1.peso=7;
console.log(ani1.nombre + " pesa " + ani1.peso);
let ani2=new Animal;
ani2.nombre="Tobi";
ani2.peso=9;
console.log(ani2.nombre + " pesa " + ani2.peso);
```

k. Notación json

JSON (acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de una notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera (año 2019) un formato independiente del lenguaje.

```
//obtener el formato json de un objeto usando el objeto JSON
console.log("Formato json de a:")
console.log(JSON.stringify(a));
```

Produce la salida:
Formato json de a:
{"x":10,"y":20}

Vemos que solo se convierten a json las propiedades y debemos saber que solamente se pueden emplear las comillas dobles para delimitar el nombre de las propiedades.

El objeto JSON también proporciona el método parse que evalúa si un texto JSON es correcto y proporciona un objeto.

Ejercicio: Visualiza el objeto animal en formato json. Crea un objeto utilizando el método parse.

```
function Animal(){

   };
let ani1=new Animal();
Animal.prototype.nombre="";
Animal.prototype.peso=0;
Animal.prototype.comer=function(){return "pienso para perros"};
console.log(Animal.prototype);
ani1.nombre="Lani";
ani1.peso=7;
console.log(ani1.nombre + " pesa " + ani1.peso);
let ani2=new Animal;
ani2.nombre="Tobi";
ani2.peso=9;
console.log(ani2.nombre + " pesa " + ani2.peso);
console.log(JSON.stringify(ani2));

let ani3 = JSON.parse(JSON.stringify(ani2));
console.log(ani3.nombre);
console.log(typeof ani3);//Object
console.log(ani3 instanceof(Animal));//false
```

2. Objetos nativos

a. Date

Es un tipo de objeto preparado para manejar fechas. Son muy numerosos sus métodos. Puedes consultarlos en internet, por ejemplo en el siguiente enlace:

https://desarrolloweb.com/articulos/clase-date-javascript.html

De esa página se muestra el siguiente ejemplo:

En este ejemplo vamos a crear dos fechas, una con el instante actual y otra con fecha del pasado. Luego imprimimos las dos y extraemos su año para imprimirlo también. Luego actualizaremos el año de una de las fechas y la volveremos a escribir con un formato más legible.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <script>

      //en estas líneas creamos las fechas
miFechaActual = new Date()
miFechaPasada = new Date(1998,4,23)

//en estas líneas imprimimos las fechas.
document.write (miFechaActual)
document.write ("<br>")
document.write (miFechaPasada)

//extraemos el año de las dos fechas
anoActual = miFechaActual.getFullYear()
anoPasado = miFechaPasada.getFullYear()

//Escribimos en año en la página
document.write("<br>El año actual es: " + anoActual)
document.write("<br>El año pasado es: " + anoPasado)
```

```
//cambiamos el año en la fecha actual
miFechaActual.setFullYear(2005)

//extraemos el día mes y año
dia = miFechaActual.getDate()
mes = parseInt(miFechaActual.getMonth()) + 1
ano = miFechaActual.getFullYear()

//escribimos la fecha en un formato legible
document.write ("<br>")
document.write (dia + "/" + mes + "/" + ano)
```

b. Math

El objeto global Math facilita la ejecución de operaciones matemáticas avanzadas. También pone a nuestra disposición constantes matemáticas. Son muy numerosos los métodos y puedes consultarlos en internet si necesitas emplear alguno.

Ejemplos

Ejemplo Math.PI y Math.pow:

```
//Calcular el área de un círculo.
var radio = prompt("Introduce el radio del círculo");
var area = Math.PI * Math.pow(radio, 2);
alert("El area del circulo es de : " + area);
```

Ejemplo de Math.floor y Math.random:

```
//Obtener un número aleatorio entre 1 y 10
var numero = Math.floor((Math.random() * (10-1)) + 1);
//Math.random devuelve un número aleatorio entre 0 y 1
//Math.floor redondea al número entero.
```

area=pi*radio al cuadrado

c. Number

El objeto Number representa el tipo de dato numérico en JavaScript.
Al crear un objeto Number, el valor que se crea depende de lo que le pasemos por parámetros al constructor.
Si el constructor recibe un número inicializa el objeto con el número que recibe.
Si recibe un número entre comillas lo convierte a numérico. Devuelve dicho número.
Devuelve 0 en caso de que no reciba nada.
En caso de que no le pasemos un número devuelve NaN, que significa «Not a Number» (No es un número).
Si recibe false se inicializa a 0 y si recibe true se inicializa a 1.
Podemos asignar a una variable un número o podemos darle valor, mediante el constructor Number, de esta forma:
var numero = Number(35);

PROPIEDAD	QUÉ HACE
MAX_VALUE	Devuelve el mayor número posible en JavaScript
MIN_VALUE	Devuelve el menor número posible en JavaScript
NaN	Representa el valor especial Not a Number
NEGATIVE_INFINITY	Representa el infinito negativo
POSITIVE_INFINITY	Representa el infinito positivo

MÉTODO	QUÉ HACE
toExporential()	Convierte el número en una notación exporencial
toFixed()	Formatea el número con la cantidad de dígitos decimales que pasemos como parámetro. Redondea
toPrecision()	Formatea el número con la longitud que pasemos como parámetro. Devuelve un String

Ejemplo toPrecision()

```
var num = 13.3714;
var a = num.toPrecision(); //a vale 13.3714
var b = num.toPrecision(2); //b vale 13
var c = num.toPrecision(3); //c vale 13.4
var d = num.toPrecision(10); //d vale 13.37140000
```

Ejemplo toFixed()

```
var numero = 3.14159;
var a = numero.toFixed(2);
//a vale 3.14
```

d. String

PROPIEDAD	QUÉ HACE
length	Corresponde a la longitud de la cadena. Ver ejemplo

MÉTODO	QUÉ HACE
charAt(num)	Permite acceder a un carácter en concreto de una cadena. Ver ejemplo
indexOf(string)	Devuelve la posición de la primera ocurrencia del carácter pasado como parámetro. Ver ejemplo
lastIndexOf(string)	Devuelve la posición de la última ocurrencia del carácter pasado como parámetro
match()	Busca una coincidencia en una cadena y devuelve todas las coincidencias encontradas

replace(cadena, sustituto)	Busca una coincidencia en una cadena y si existe, la remplaza por otra cadena pasada como parámetro
search()	Busca una coincidencia en una cadena y devuelve la posición de la coincidencia
slice()	Extrae una parte de una cadena en base a los parámetros que indiquemos como índices de inicio y final.
split()	Corta una cadena en base a un separador que pasamos como parámetro
substr(inicio, longitud)	Devuelve una subcadena en base a un índice y longitud pasados como parámetros

substring(inicio, fin)	Devuelve una subcadena en base a un índice de inicio y de final pasados como parámetros
toLowerCase()	Devuelve la cadena en minúsculas. No la cambia.
toUpperCase()	Devuelve la cadena en mayúsculas. No la cambia
trim()	Elimina los espacios del principio y el final del String
fromCharCode()	Convierte valores unicode en caracteres
concat()	Une dos o más Strings y los devuelve concatenados en un nuevo String
endsWith(cadena)	Comprueba si el String termina con los caracteres pasados por parámetro

charCodeAt()	Devuelve el unicode del caracter en el índice especificado
includes(cadena)	Comprueba si el String contiene la cadena pasada por parámetro
localeCompare()	Comprueba si dos cadenas son equivalentes en la configuración regional actual. Ver ejemplo
repeat()	Devuelve un String con el número de copias de la cadena especificado por parámetro. Ver ejemplo

Cuando creas una cadena, por ejemplo, usando let string = 'This is my string'; estás creando una instancia del objeto string y por lo tanto tiene una gran cantidad de propiedades y métodos disponibles.

Encontrar la longitud de un cadena

Usar la propiedad length. Ejemplos:

```
let browserType = 'mozilla';
browserType.length;
```

Extrayendo un carácter de la cadena

```
browserType[0];  //primer carácter
browserType[browserType.length-1]; //último carácter
```

Encontrar una subcadena dentro de una cadena y extraerla

browserType.indexOf('zilla');

Esto nos da un resultado de 2, porque la subcadena "zilla" comienza en la posición 2 (la cuenta empieza en 0) dentro de "mozilla".

Esto se puede hacer de otra manera, que posiblemente sea aún más efectiva.
Intenta lo siguiente:
browserType.indexOf('vanilla');//-1 indica que no existe tal subcadena.

```
if(browserType.indexOf('mozilla') !== -1) {
  // órdenes….
}
```

Cuando sabes donde comienza una subcadena dentro de una cadena, y sabes hasta cuál caracter deseas que termine, puede usarse slice() para extraerla.

browserType.slice(0,3);

Cambiando todo a mayúscula o minúscula

Los métodos de cadena toLowerCase() y toUpperCase() toman una cadena y convierten todos sus caracteres a mayúscula o minúscula, respectivamente. Esto puede ser útil, por ejemplo, si deseas normalizar todos los datos introducidos por el usuario antes de almacenarlos en una base de datos.

```
let radData = 'My NaMe Is MuD';
radData.toLowerCase();
radData.toUpperCase();
```

Actualizando partes de una cadena

En una cadena puedes reemplazar una subcadena por otra usando el método replace().

browserType = browserType.replace('moz','van');

Ejemplos tomados de la web:
https://developer.mozilla.org/es/docs/Learn/JavaScript/First_steps/Useful_string_methods

e. Expresiones regulares

Las expresiones regulares son muy útiles para la comprobación de datos. Sus posibilidades son inmensas, y no se pueden enumerar aquí. Hay mucha información disponible en internet. Te recomiendo este enlace si necesitas profundizar:
https://lenguajejs.com/javascript/caracteristicas/expresiones-regulares/

Como ejemplo ilustrativo tanto de estas expresiones como del uso del objeto String, es interesante el estudio del siguiente ejemplo:

```
<!DOCTYPE html>
<html lang="es">
<head>   <meta charset="UTF-8">   <title>DNI</title>
</head>
<body>
<script>
function nif(dni) {
 let numero
 let letr
 let letra
```

```
 let expresion_regular_dni
  //expresion_regular_dni = /^\d{8}[a-zA-Z]$/;
 expresion_regular_dni = new RegExp("[0-9]{8}[a-zA-Z]");
   if(expresion_regular_dni.test (dni) == true){
    numero = dni.substr(0,dni.length-1);
    letr = dni.substr(dni.length-1,1);
    numero = numero % 23;
    letra='TRWAGMYFPDXBNJZSQVHLCKET';
    letra=letra.substring(numero,numero+1);
   if (letra!=letr.toUpperCase()) {
      alert('Dni erróneo, la letra del NIF no se corresponde');
    }else{
      alert('Dni correcto');
    }
 }else{
    alert('Dni erróneo, formato no válido');
  }
};
let n=prompt("Escribe un número","99999999X");
nif(n);
</script>
</body>
</html>
```

Otro ejemplo:

Validar un número entero:

```
// const esEntero = numeroEntero =>
/^[-+0-9][0-9]+$/ig.test(numeroEntero);
var esEntero = function esEntero(numeroEntero) {
   return /^[-+0-9][0-9]+$/gi.test(numeroEntero);
   };

   let resultado = esEntero("+88762");
   console.log (resultado);
```

Y otro más:

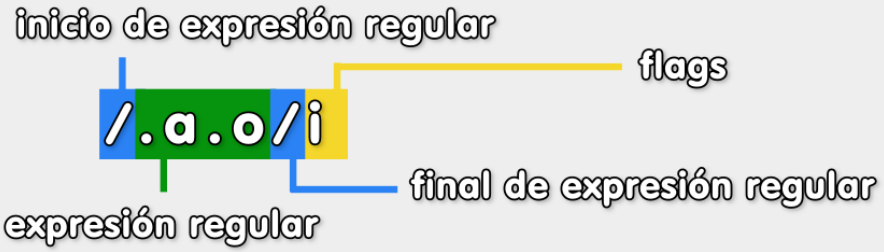
```
const names = ["Pedro", "Sara", "Miriam", "Nestor", "Adrián", "Sandro"];

// Comprobación sin usar expresiones regulares
names.forEach(function(name) {
  const firstLetter = name.at(0).toLowerCase();
  const lastLetter = name.at(-1).toLowerCase();

  if ((firstLetter === "p" || firstLetter === "s") && (lastLetter === "o" || lastLetter === "a")) {
    console.log(`El nombre ${name} cumple las restricciones.`);
  }
});

// Comprobación usando expresiones regulares
names.forEach(function(name) {
  const regex = /^(p|s).+(o|a)$/i;

  if (regex.test(name)) {
    console.log(`El nombre ${name} cumple las restricciones.`);
  }
});
```



- Los slash / (*azul*) son los **delimitadores** de las partes de una expresión regular.
- La **definición de la expresión regular** (*verde*) es un texto con símbolos especiales que indica que textos va a incluir.
- Los **flags** (*amarillo*), son una serie de carácteres que indican como funcionará la expresión regular.

Caracteres especiales:

Caracter especial	Descripción
.	Comodín, significa **cualquier caracter** (letra, número, símbolo...), pero que ocupe sólo **1 carácter**.
\	Precedido de un carácter especial, lo invalida (se llama «escapar»).

Caracter especial	Descripción
`[]`	Rango de carácteres. Cualquiera de los caracteres del interior de los corchetes.
`[^]`	Que no exista cualquiera de los caracteres del interior de los corchetes.
`|`	Establece una alternativa: lo que está a la izquierda o lo que está a la derecha.

Caracter especial	Alternativa	Descripción
`[0-9]`	`\d`	Un dígito del `0` al `9`.
`[^0-9]`	`\D`	No exista un dígito del `0` al `9`.
`[A-Z]`		Letra mayúscula de la `A` a la `Z`. Excluye `ñ` o letras acentuadas.
`[a-z]`		Letra minúscula de la `a` a la `z`. Excluye `ñ` o letras acentuadas.
`[A-Za-z0-9]`	`\w`	Carácter alfanumérico (letra mayúscula, minúscula o dígito).
`[^A-Za-z0-9]`	`\W`	No exista carácter alfanumérico (letra mayúscula, minúscula o dígito).
`[ \t\r\n\f]`	`\s`	Carácter de espacio en blanco (espacio, `TAB`, `CR`, `LF` o `FF`).
`[^ \t\r\n\f]`	`\S`	No exista carácter de espacio en blanco (espacio, `TAB`, `CR`, `LF` o `FF`).

Anclas:

Caracter especial	Descripción
`^`	Ancla. Delimita el inicio del patrón. Significa **empieza por**.
`$`	Ancla. Delimita el final del patrón. Significa **acaba en**.
`\b`	Límite de una palabra separada por espacios, puntuación o inicio/final.

Cuantificadores:

Caracter especial	Descripción
`*`	El carácter anterior puede aparecer `0` o más veces.
`+`	El carácter anterior puede aparecer `1` o más veces.
`?`	El carácter anterior puede aparecer o no (es opcional).
`{n}`	El carácter anterior aparece `n` veces.
`{n,}`	El carácter anterior aparece `n` o más veces.
`{n,m}`	El carácter anterior aparece de `n` a `m` veces.

```
const regexp = /[0-9]{2}/;    // Un número formado por 2 dígitos (del 0 al 9)

regexp.test(42);              // true
regexp.test(88);              // true
regexp.test(1);               // false (no son 2 dígitos)
regexp.test(100);             // true (tiene al menos 2 dígitos)
```

```
const regexp = /^[0-9]{2}$/;    // Un número de exactamente 2 dígitos (del 0 al 9)

regexp.test(4);                 // false (no llega a 2 dígitos)
regexp.test(55);                // true
regexp.test(100);               // false (no tiene exáctamente 2 dígitos)

const regexp = /^[0-9]{3,}$/;

regexp.test(33);                // false (debe tener al menos 3 dígitos)
regexp.test(4923);              // true

const regexp = /^[0-9]{2,5}$/;

regexp.test(2);                 // false (no tiene de 2 a 5 dígitos)
regexp.test(444);               // true
regexp.test(543213);            // false (no tiene de 2 a 5 dígitos)
```

Flags:

Existen varios pero nosotros nos centraremos en i (insensible a mayúsculas) y en g (global).