COLOMBIA POTENCIA DE LA VIDA

TIC

Módulo 1

LECCIÓN 2

Instrucciones base, variables, estructuras y funciones

TALENTO TECH

AZ | PROYECTOS EDUCATIVOS

Universidad Tecnológica de Pereira

Sintaxis

La sintaxis de JavaScript es muy similar a la de otros lenguajes de programación como Java y C. Las normas básicas que definen la sintaxis de JavaScript son las siguientes:

- No se tienen en cuenta los espacios en blanco y las nuevas líneas, ya que el intérprete de JS los ignora, lo que permite formatear el código para una mejor legibilidad.
- Se distinguen las mayúsculas y minúsculas.
- No se define el tipo de las variables
- No es necesario terminar cada sentencia con el carácter de punto y coma, sin embargo es una convención y se recomienda seguirla.
- Se pueden incluir comentarios de una sola línea o de varias líneas

JavaScript

```
//Comentario de una línea

/* Este es un comentaro que
puede ocupar diferented
*/ líneas
```

Variables

Las variables son espacios de memoria en la computadora donde podemos almacenar distintos tipos de datos.
Un identificador de JavaScript debe comenzar con una letra, un guión bajo (_) o un signo de dólar ($). Los siguientes caracteres también pueden ser dígitos (0-9).
Dado que JavaScript distingue entre mayúsculas y minúsculas, las letras incluyen los caracteres "A" a "Z" (mayúsculas), así como "a" a "z" (minúsculas).

Las variables y funciones dentro de JS se suelen declarar utilizando la nomenclatura **camelCase,** que recomienda que la primera letra de la primera palabra del nombre vaya en minúscula y cada palabra de allí en adelante inicie en mayúscula.

Las clases se nombran utilizando **PascalCase,** en donde cada palabra del nombre inicia en mayúscula y, las constantes pueden ser declaradas utilizando screaming sanake_case, básicamente es mayúscula sostenida separando cada palabra del nombre por guiones bajos **SCREAMING_SNAKE_CASE**

En JavaScript existen tres formas de declarar variables:

- var Es una variable que sí puede cambiar su valor y su scope es local, Se recomienda no utilizar.
- let Es una variable que si puede cambiar su valor y su scope pertenece al bloque en el cual fue declarada
- const Es una variable la cual NO puede cambiar su valor una vez que ya fue asignado.

Las palabras reservadas como **var, let**y const solo pueden utilizarse para el propósito que fueron creadas. No pueden ser utilizadas como: nombres de variables, funciones, métodos o identificadores de objetos.

```javascript
let nombre = 'Juan';
let apellido = 'Gomez';
const valorDePi = 3.14;
var nombreCompleto = nombre + ' ' + apellido;
```

Ámbito de variables

Cuando se declara una variable fuera de cualquier función, se denomina variable global, porque está disponible para cualquier otro código en el documento actual. Cuando declaras una variable dentro de una función, se llama variable local, porque solo está disponible dentro de esa función.

Utilizando var - no recomendado

```javascript
if (true) {
var x = 5;
}
console.log(x); // x es 5
```

Utilizando let

```javascript
if (true) {
 let y = 5;
}
console.log(y); // ReferenceError: y no está definida
```

Elevación de variables (hoisting)

Otra cosa inusual acerca de las variables en JavaScript definidas con “var” es que puedes hacer referencia a una variable declarada más tarde, sin obtener una excepción.

Este concepto se conoce como elevación. Las variables en JavaScript son, en cierto sentido, "elevadas" a la parte superior de la función o declaración. Sin embargo, las variables que se elevan devuelven un valor de undefined.

JavaScript

```
console.log(x === undefined); // true
var x = 3;
```

JavaScript

```
console.log(x); // ReferenceError
let x = 3;
```

Tipos de datos

Los tipos de datos le permiten a JavaScript conocer las características y funcionalidades que estarán disponibles para ese dato

Tipo de dato	Descripción	Ejemplo
Numéricos (number)	Los numéricos pueden ser números enteros o decimales	let numberOne = 1; let numberTwo = 2;
Cadenas de carácteres (strings)	Son cadenas de texto alfanuméricas	let myString = 'Hola, como va?' let myString2 = "Hola, como va?"
Lógicos o booleanos (Boolean)	Es un tipo de dato que solo tiene dos valores posibles true o false	let esVerdad = true; let esFalso = false;

Arreglos (Arrays)	Los arrays son colecciones de datos, son considerados un tipo especial de objetos	let myArray = ['valor1', 'valor2']; let comidasFavoritas = ['Milanesas', 'Ravioles'];
Objetos (object)	A diferencia de otros tipos de datos, los objetos son colecciones de datos y en su interior pueden existir todos los anteriores	let firstPerson = { nombre: 'Fernando', edad: 26, soltero: false, }

Los tipos de datos especiales le permiten a JavaScript determinar estados especiales que pueden tener los datos

Tipo de dato	Descripción	Ejemplo
NaN (Not A Number)	Indica que el valor no puede ser parseado como un número	`let malaDivision = '35' / 2;`
Null (valor nulo)	Lo asignamos nosotros para indicar un valor vacío o desconocido	`let temperatura = null;`
undefined (valor sin definir)	Indica ausencia de valor, las variables tienen este valor por defecto cuando les asignamos uno.	`let saludo;`

Operadores

Los operadores nos permiten manipular el valor de las variables, realizar operaciones y comparar sus valores.

Operadores de asignación

Operador	Explicación	Ejemplo
=	Asignan el valor de la derecha en la variable de la izquierda.	`let edad = 26;`

Aritméticos

Operador	Explicación	Ejemplo
+	Operador Suma	`10 + 15`
-	Resta	`10 - 15`
*	Multiplicación	`10 * 15`
/	División	`15 / 10`
%	Módulo o residuo	`10 % 2`
++	Incremento en 1	`15++`
--	Decremento en 1	`15--`

Los operadores “+” y “-” también cuenta con una función adicional, cuando se trabajan con cadenas, el operador “+” hace la función de concatenar; mientras que trabajando con números, el operador “-” asigna un valor negativo similar a multiplicar por -1.

JavaScript

```
let fila = 'M';
let asiento = 7;
let ubicacion = fila + asiento;
console.log(ubicacion)// Devuelve ‘M7’
```

De comparación simple

Comparar dos valores, devuelve true o false.

Operador	Explicación	Ejemplo
==	Igualdad en valor	`10 == 10`
!=	Diferente	`10 != 15`
>	Estrictamente mayor	`10 > 15`
>=	Mayor o igual que	`10 >= 10`
<	estrictamente menor	`20 < 10`
<=	Menor o igual que	`19 <= 33`

De comparación estricta

Operador	Explicación	Ejemplo
===	Operador de igualdad estricta, nos devuelve si los datos comparados tienen o no el mismo valor y el tipo de dato	`10 === '10'`
!==	Operador de desigualdad estricta, nos devuelve si los datos comparados tienen o no el mismo valor y el tipo de dato	`10 !== ‘10’`

Lógicos

Al igual que los operadores de comparación, la salida de los operadores lógicos siempre es un booleano

Operador	Explicación	Ejemplo
&&	Operador and. Todos los valores deben evaluar como true para que el resultado sea true.	`(10 > 15) && (10 != 20)` // False `(12 % 4 == 0) && (12 != 24)` // True
||	Operador or. Al menos un valor debe evaluar como true para que el resultado sea true	`(10 > 15) || (10 != 20)`// True `(12 % 5 == 0) || (12 != 12)`//False

!	Operador negación. Niega la condición. Si era true será false y viceversa.	`!false` //True
		`!(20 > 15)`
		//False

Funciones

Una función es un bloque de código que nos permite agrupar funcionalidad para usarla todas las veces que necesitemos. Normalmente realiza una tarea específica y retorna un valor como resultado.

Existen dos tipos de funciones declaradas y expresadas.

- **Funciones declaradas:** son aquellas que se declaran usando la estructura básica. Pueden recibir un nombre, escrito a continuación de la palabra reservada function, a través del cual podemos invocar. Las funciones con nombre son funciones nombradas.

JavaScript

```
function hacerHelado(cantidad) {
  return '🍦'.repeat(cantidad);
};
```

- **Funciones expresadas:** son aquellas que se asignan como valor de una variable. En este caso, la función en sí no tiene nombre, es una función anónima. Para invocar podremos usar el nombre de la variable que declaremos.

```javascript
let hacerSushi = function (cantidad) {
  return '🍣'.repeat(cantidad);
};
```

Invocación de funciones

Es posible imaginar las funciones como si fueran carros. Durante la declaración se ocupa de construir el carro y durante la invocación se pone a funcionar.

Antes de poder invocar una función se necesita que haya sido declarada. La forma de invocar, llamar o ejecutar una función es escribiendo su nombre seguido de apertura y cierre de paréntesis.

Ahora, si una función cuenta con parámetros deben ser pasados dentro de los paréntesis, teniendo cuidado de respetar el orden.

Los parámetros son datos o información que necesita la función para poder ejecutarse correctamente, por ejemplo, una función que saluda con el nombre de un usuario, va a necesitar que de alguna forma se le envíe tal información.

```javascript
function saludar(nombre,apellido) {
  return 'Hola ' + nombre + ' ' +
apellido;
};

saludar('Juan', 'Peréz');
//Retornará ‘Hola Juan Peréz’***
```

En caso de no haber recibido los parámetros que esperaba y si no, se le define valores por defecto, automáticamente JavaScript le asignará el valor de **undefined.**
Y en caso de querer **guardar** lo que nos **retorna** esa función debemos **almacenarlo en una variable.**

```
JavaScript
let unSaludo = saludar('Juan', 'Peréz');
```

Condicionales

Permiten evaluar condiciones y realizar diferentes acciones según el resultado de esas evaluaciones.

if, else, else if

Versión más básica del if. Establece una condición y un bloque de código a ejecutar en caso de que sea verdadera

```
JavaScript
if (condición) {
  // código a ejecutar si la condición es verdadera.*
};
```

Igual al ejemplo anterior, pero el else agrega un bloque de código a ejecutar en caso de que la condición no se cumpla.

```
JavaScript
if (condición) {
    //código a ejecutar si la condición es verdadera
}else {
  //código a ejecutar si la condición es falsa
};
```

El último bloque es el else if, aquí se agregan tantos bloques como sean necesarios, el else if funciona igual que el if, indicando una condición que se debe cumplir dentro de los paréntesis, si queremos que se ejecute el bloque puntualmente.

JavaScript

```
if (condición) {
  //código a ejecutar si la condición es
verdadera
}else if (condición) {
  //código a ejecutar si la otra condición es
verdadera
}else {
 //código a ejecutar si todas las condiciones
son falsas
};
```

Tanto el **else if** como el **else** son opcionales dentro del bloque condicional.

Switch

El bloque switch toma una sola expresión / valor como una entrada, y luego pasan a través de una serie de opciones hasta que encuentran una que coincida con ese valor, ejecutando el código correspondiente que va junto con ella.

JavaScript

```
switch (expresion) {
 case choice1:
  ejecuta este código
  break;

 case choice2:
  ejecuta este código
  break;

 // Se pueden incluir todos los casos que quieras

 default:
  por si acaso, corre este código
}
```

Una declaración llamada **break**, seguida de un punto y coma. Si la elección previa coincide con la expresión, el bloque deja de ejecutarse y pasa a cualquier código que aparece debajo de la declaración de **switch.** En caso de no agregar el **break,** el bloque se sigue ejecutando hasta finalizar.

La declaración **default** se ejecuta si no coincide con ningún otro caso o si un caso en el que haya coincidido ha omitido el **break,** el similar al else.

Operador Ternario

es una pequeña sintaxis que prueba una condición y ejecuta un bloque si es **true**, y otro si es **false**

JavaScript

```
//( condición ) ? ejecuta este código : ejecuta este código en su lugar


let greeting = isBirthday
  ? "Feliz cumpleaños Sra. Smith. ¡Esperamos que tenga un gran día!"
 :"Buenos días señora Smith.";
```

Arreglos o Arrays

Son la estructura que permite almacenar datos de manera ordenada, esto quiere decir que cada elemento cuenta con una posición fija dentro del arreglo al que se puede acceder con un índice; el orden de los elementos no cambia a no ser que se manipulen por parte del programa. Dentro de un arreglo se pueden modificar, eliminar o añadir elementos.

Cuentan con una gran cantidad de propiedades y métodos que la hacen una de las estructuras de datos más poderosas y versátiles dentro del lenguaje.

Cada elemento dentro del array puede ser de cualquier tipo, números, objetos literales, cadenas, booleanos e inclusive otros arrays. Los arreglos pueden almacenarse en variables y tratarse de la misma manera que cualquier otro tipo de valor, la diferencia es que podemos acceder individualmente a cada valor dentro de la lista y hacer cosas útiles y eficientes con la lista, como recorrerlo con un bucle y hacer una misma cosa a cada valor. Tal vez tenemos una serie de productos y sus precios almacenados en un arreglo, y queremos recorrerlos e imprimirlos en una factura, sumando todos los precios e imprimiendo el total en la parte inferior.

Ahora imaginen que no existen los arrays, tener una lista de 1000 productos, se tendrían que crear igual cantidad de variables para guardar los productos. Ahora qué tal si cada producto tiene una cantidad determinada de atributos, Cuántas variables necesitamos o cómo podemos almacenar toda esta información para que mantengan relación entre sí?

Para crear un array es suficiente con asignar los elementos dentro de corchetes, ningún elemento indica un array vacío. También pueden ser vinculados con una variable como se muestra a continuación

```javascript
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019]];
```

Dos de las operaciones más básicas a la hora de trabajar con arrays son conocer su longitud y acceder a un elemento. Comenzando por el segundo caso, los arrays son 0-index, es decir que el primer elemento, en caso de existir, está ubicado en la posición 0, el segundo en la 1 y así sucesivamente hasta su longitud -1. Los índices de un array también funcionan de forma negativa, siendo su último elemento el que se ubica en la posición -1.

Para conocer la longitud del array podemos acceder a su propiedad **length** (de forma nemotécnica para no equivocarse con leng**ht** y leng**th** utilizar la frase Gallina Tiene Huevos, hoy en día los editores de código ayudan bastante para evitar errores de este tipo, pero nunca sobra estar al pendiente)

```JavaScript
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019]];

myArray[0]; //'Stars Wars'

myArray.length; // 4

myArray[4]; //undefined

myArray[3].length; // 9
```

Ejercitación

¿Cómo se puede utilizar el método split para convertir la cadena en un array? ¿Qué ocurre si al resultado se le aplica el método join?

```JavaScript

let myData = "Manchester,London,Liverpool,Birmingham,Leeds,Carlisle";
```

Las siguientes operaciones a aplicar sobre un array es la opción de añadir o eliminar elementos y finalmente recorrerlo.

Push

Para añadir elementos al final de array, puede añadirse varios elementos de una sola vez y dado que el array crece, su longitud es modificada

JavaScript

```
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019]];
```

```
myArray.push('George Walton Lucas Jr.', 1944)

//['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019], 'George Walton Lucas Jr.', 1944)]
```

Pop

Para eliminar un elemento al final, se utiliza pop, funciona igual, pero elimina solo un elemento a la vez.

JavaScript

```
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019], 'George Walton Lucas Jr.', 1944];

myArray.pop

//['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019], 'George Walton Lucas Jr.')]
```

Ejercitación

- ¿Qué ocurre si no se utiliza push() sino unshift() y no pop() sino shift()?
- Revisando la documentación, es hora de experimentar con otros métodos y propiedades Array - JavaScript | MDN, Revisar con los estudiantes métodos interesantes como flat, indexOf, join, includes…

Recorriendo un array

Existen diversos métodos que nos permiten recorrer y operar un array, entre los más comunes se puede encontrar el for...of y el forEach.

for...of

Funciona para todos los elementos iterables, al igual que forEach, que se presenta a continuación, permite ejecutar una serie de instrucciones para cada uno de los elementos dentro del iterable.

JavaScript

```
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019], 'George Walton Lucas Jr.', 1944];

for(item of myArray){
console.log(item)
};

//'Stars Wars'
//true
//23
//[1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019]
//'George Walton Lucas Jr.'
//1944
```

forEach

Este método llama a una función específica, una vez por cada elemento sobre el que itera dentro del array. No cambia el array original.

JavaScript

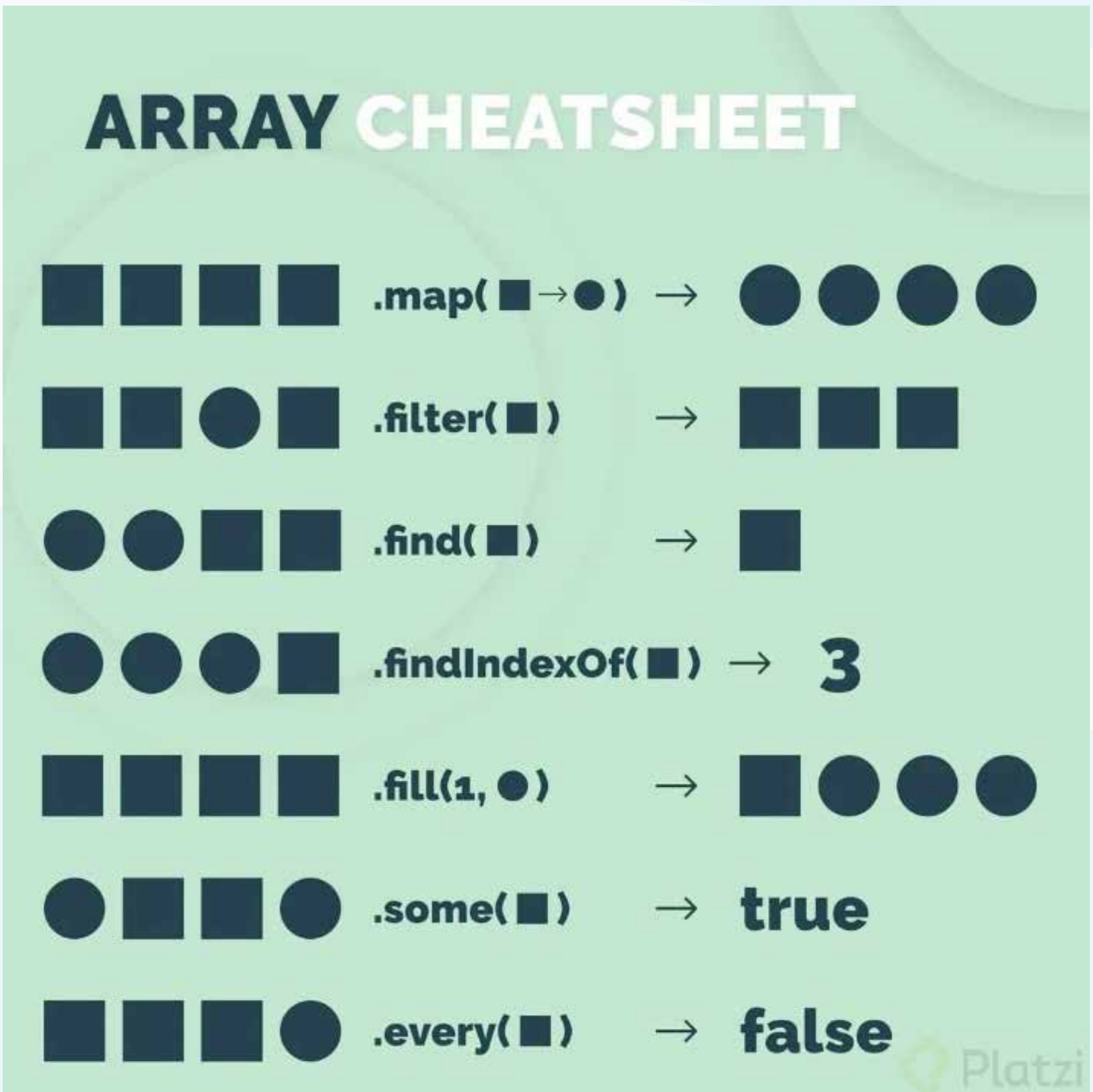
```
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019], 'George Walton Lucas Jr.', 1944];

myArray.forEach((item) => {
console.log(item)
});

//'Stars Wars'
//true
//23
//[1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019]
//'George Walton Lucas Jr.'
//1944
```

En este caso solo se ha llamado a la función console.log() para imprimir cada elemento en consola, pero se puede aplicar cualquier transformación.

Existen otros métodos muy utilizados en los arrays que permiten transformar su contenido, como el map, filter y find.



fuente: Platzi, métodos para el uso de arrays, consultado en enero de 2024, Metodos para el uso de arrays .

find

El método find() devuelve el valor del primer elemento del array que cumple la función de prueba proporcionada.

JavaScript

```
const myArray = ['Stars Wars', true, 23, [1977,
1980, 1983, 1999, 2002, 2005, 2015, 2017,
2019], 'George Walton Lucas Jr.', 1944];

#Utilizando return explícito
let found = myArray.find((item) => {
 return item>20
});

//23


#Utilizando return implícito
let found = myArray.find((item) => item >
20);

//23
```

Ejercitación

¿Cómo se podría consultar si un valor existe utilizando el método includes()?

map

El método map() crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.

JavaScript

```
const numbers = [1, 5, 10, 15];

//return explícito + function
const doubles = numbers.map(function (x) {
  return x * 2;
});

//return implícito + arrow function
const doubles = numbers.map((x) => return x * 2)

// doubles is now [2, 10, 20, 30]
// numbers is still [1, 5, 10, 15]
```

Nótese que se aplica el bloque de instrucciones, este caso multiplica por 2 al conjunto de elementos pero no se modifica el array actual, map retorna un nuevo array con la operación aplicada.

JavaScript

```
const kvArray = [
 {clave: 1, valor: 10},
 {clave: 2, valor: 20},
 {clave: 3, valor: 30},
];

const reformattedArray = kvArray.map(function (obj) {
 const rObj = {};
 rObj[obj.clave] = obj.valor;
 return rObj;
});

//  reformattedArray  es  ahora  [{1:10},  {2:20},
{3:30}],

// kvArray sigue siendo:
// [{clave:1, valor:10},
//  {clave:2, valor:20},
//  {clave:3, valor: 30}]
```

Ejercitación

- Cree un array con 10 números (arrayNum), ahora utilizando map, va a obtener un nuevo array (arrayPrim) que indique en cada posición si el número evaluado es primo o no, puede agregar la cadena “es primo” y “no es primo”.

- Cree un array (arrayTextos) con 10 palabras, ahora utilizando map, va a obtener un nuevo array (arrayNum) que indique en cada posición, el número de carácteres o longitud de cada palabra del array inicial. Extra, ordene el array resultante.

filter

El método filter() crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.

Como su nombre lo indica, el método realiza un filtro de los elementos según un criterio definido, por ejemplo se requiere obtener los números del array que sean mayor a 2000.

Dado que el array tiene otro array dentro de sus elementos, lo primero es aplanar el array y luego aplicar el filtro

JavaScript

```
const myArray = ['Stars Wars', true, 23, [1977, 1980, 1983, 1999, 2002, 2005, 2015, 2017, 2019]];

const newArray = myArray.flat().filter(item => item > 2000);

//[ 2002, 2005, 2015, 2017, 2019 ]

//o sin apalanar el array, haciendo el filtro dentro de la posición específica

const newArray2 = myArray[3].filter(item => item > 2000);
```

Objetos

JavaScript está diseñado en un paradigma simple basado en objetos. Un objeto es una colección de propiedades, y una propiedad es una asociación entre un nombre (o clave) y un valor. El valor de una propiedad puede ser una función, en cuyo caso la propiedad es conocida como un método. Además de los objetos que están predefinidos en el navegador, puedes definir tus propios objetos.

En JavaScript, un objeto es una entidad independiente con propiedades y tipos. Compáralo con una taza, por ejemplo. Una taza es un objeto con propiedades. Una taza tiene un color, un diseño, un peso, un material del que está hecha, etc. Del mismo modo, los objetos de JavaScript pueden tener propiedades que definan sus características.

A diferencia de los arrays, las propiedades dentro de los objetos no tienen un orden fijo y no son ordenadas, no es posible acceder a una propiedad utilizando un índice

```JavaScript
let tenista = {
nombre : 'Roger',
 apellido : 'Federer'
};

console.log(tenista.nombre) // Roger
console.log(tenista.apellido) // Federer
```

Cada elemento del objeto es separado del siguiente por coma (,), y cada elemento puede ser cualquier tipo de dato o estructura o como se menciona arriba, una función.

Si quiere acceder a una propiedad del objeto debo hacerlo con la notación punto(.) o con el nombre de la llave entre corchetes, mientras que para ejecutar un método, debo invocar a la llave con paréntesis

```JavaScript
let tenista = {
        nombre: 'Roger',
        edad: 42,
        activo: true,
        saludar: function() {
        return '¡Hola! Me llamo' + this.nombre;
        }, };

tenista.nombre //'Roger'
tenista["edad"] //42
tenista.saludar() //"¡Hola! Me llamo Roger"
```

De un objeto se puede tener información como las llaves y los valores o recorrerlo, similar a los arrays.

for...in

```JavaScript
let tenista = {
        nombre: 'Roger',
        edad: 42,
        activo: true,
        saludar: function() {
        return '¡Hola! Me llamo' + this.nombre;
        }, };

for (const prop in tenista){
   console.log(`${prop}: ${tenista[prop]}`)
}

//nombre: Roger
//edad: 38
//activo: true
//saludar: function() {return '¡Hola! Me llamo ' + this.nombre;}
```

Ejercitación:

- Deben revisar que ocurre con los métodos de los objetos Object.entries(), Object.values(), Object.keys(), qué retornan? en qué estructura lo hacen? qué métodos podría aplicar a la estructura que se retorna?
- ¿En qué se parecen los objetos literales de JS al formato JSON?
- ¿Cómo puede añadir propiedades a un objeto?
- Qué tal si se crea, combinando estructuras conocidas, un sistema de notas para los estudiantes de un colegio, el curso tiene 10 estudiantes, cada estudiante tiene un nombre, apellido, edad, fecha de nacimiento, curso actual y notas de los 3 cursos que están viendo actualmente. Las notas tienen nombre del docente, nombre de la asignatura y calificación.
- Su tarea es crear una estructura con toda la información de todos los estudiantes, de tal manera que puedan ser añadidos nuevos estudiantes o modificar datos de un estudiante en particular.