**BASES DEL LENGUAJE**

Con **console.log()** podremos imprimir en consola lo que le indiquemos dentro de la función.

Para hacer comentarios y que no se ejecute lo que escribamos podemos utilizar **//** para una sola línea o, **/\* \*/** para hacer un comentario con múltiples líneas.

**Cadenas de texto**

Javascript te permite escribir cadenas de texto (strings) con comillas simples (‘ ’) o comillas dobles (“ ”); lo importante es que siempre se abran y se cierren con el mismo tipo de comillas.

También podemos escribir cadenas de texto con las comillas invertidas (``). Al escribir un string con las comillas invertidas podremos añadir código JS dentro del string sin tener que concatenar el código, encapsulando ese código dentro de $(código). Esto hace que el código sea mucho más legible y limpio.

Ejemplo con comillas normales:

**let base = 5;**

**let altura = 7;**

**console.log(“Resultado del área de un triángulo de base “ + base + “y altura “ + altura + “ : ” + base \* altura / 2;**

**Resultado en consola**: “Resultado del área de un triángulo de base 5 y altura 7: 17.5”

Ejemplo con comillas invertidas:

**let base = 5;**

**let altura = 7;**

**console.log(`Resultado del área de un triángulo de base $(base) y altura $(altura): $(base \* altura / 2)`);**

**Resultado en consola**: “Resultado del área de un triángulo de base 5 y altura 7: 17.5”

**Manipulando strings**

Si vamos a utilizar un string como parámetro de una función y queremos que sea modificable, lo recomendable es asignarlo dentro de una variable:

**function stringFnc (str){ let string = str; }**

Si sumamos dos cadenas de texto nos la unirá en una sola. Ej: “cadena” + “texto”: “cadenatexto”.

Cada letra de un string tiene un índice, empezando desde 0.

**Métodos utilizables sobre strings y arrays**

* **endsWith()** -> Permite consultar si una cadena de texto termina con ciertos caracteres.
* **startsWith()** -> Permite consultar si una cadena de texto empieza con ciertos caracteres.
* **toLowerCase()** -> Transforma una cadena de texto en minúsculas.
* **toUpperCase()** -> Transforma una cadena de texto en mayúsculas.
* **slice()** -> Permite cortar una cadena de caracteres. El primer parámetro a especificar es el carácter desde el que se va a empezar a cortar ese string y, el segundo parámetro, si lo indicamos cómo un número entero positivo, nos devolverá la cantidad de caracteres que vamos a seleccionar. En caso de que lo especifiquemos el segundo parámetro en negativo, nos devolverá todos los caracteres menos los indicados empezando desde el final. Por ejemplo: Si realizamos .slice(0, -2) en un string de 10 caracteres nos devolverá el string con los primeros 8 caracteres y los dos últimos los descartará. Si no especificamos segundo parámetro nos devolverá el string cortado desde el primer parámetro hasta el final de la cadena.
* **split()** -> Convierte un string en un array. Nos separa esa cadena de texto en items de un array utilizando como referencia del separador el/los caracteres que especifiquemos: Ej: **“string”.split(“t”);** nos devolverá un array tal que [“s”, “ring”]. Si no le pasamos ningún carácter como parámetro nos separará todo el string en caracteres dentro del array. Ej: **“string.split(“”);** [“s”, “t”, “r”, “i”, “n”, “g”].
* **join()** -> Convierte un array en una cadena de caracteres separando cada elemento del array con el separador que indiquemos como parámetro. Es lo mismo que el método .split() solo que a la inversa.
* **reverse()** -> Recorre un array de forma inversa.
* **chartAt()** -> Selecciona un carácter concreto especificado por el número entero que pasemos como parámetro. Ej: **“string”.charAt(2);** devuelve **“r”**.

**¿Qué es una función?**

Una función encapsula una pieza de código que podemos ejecutar dependiendo de valores que les pasemos. Para escribir nuestras funciones daremos uso a la palabra clave “function”.

- Para crear una función: **function nombreDeLaFuncion(parametros){código}**

Las funciones pueden tener un nombre, pero si no definimos este nombre son **funciones anónimas**. Los valores que recibe se llaman parámetros y, devuelven un valor usando la palabra clave “return”.

Ej: Calcular el área de un triángulo con una función normal

**function triangleArea(base, height) {**

**return base \* height /2;**

**}**

**triangleArea(5,7);**

En Javascript tenemos la opción de utilizar las **Arrow Function**. Son funciones mucho más compactas que se asignan dentro de una variable. Hacen el código más legible y compacto.

Si las escribimos en una sola línea podemos eliminar los **corchetes {}** y la palabra clave **return**.

Ej: Calcular el área de un triángulo con una Arrow Function

**let triangleArea = (base, height) => base \* height / 2;**

**EJERCICIO: Escribe una función que te calcule el área de un círculo, de un rectángulo o de un cuadrado.**

Área de un círculo:

Para calcular el área de un círculo vamos a necesitar de Math. Math es un objeto del core de Javascript que te permite usar funciones matemáticas básicas. Por ejemplo: Math.pow(a, b) nos permite calcular potencias de una base.

**let circleArea = (radius) => Math.PI \* Math.pow(radius, 2);**

**circleArea(3);**

**RESULTADO:** 28.274333882308138

Área de un rectángulo:

**let rectangleArea = (side1, side2) => side1 \* side2;**

**rectangleArea(2, 4);**

**RESULTADO:** 8

Área de un cuadrado:

**let squareArea = (side) => side \* side;**

**squareArea(4);**

**RESULTADO:** 16

**Tipos de variables**

En Javascript tenemos tres tipos de variables que se comportan de distinta forma: var, let y const.

**var** -> Es una variable modificable que no respeta el scope donde han sido declaradas (variables globales).

**let** -> Es una variable modificable que respeta el scope donde han sido declaradas. Son mucho más seguras y más recomendables.

**const** -> Son variables inmutables (no se pueden modificar), también conocidas como constantes. Una buena práctica es declarar este tipo de variables en mayúsculas dejando en claro que son constantes y no se van a modificar.

**Condicionales, expresiones y booleanos**

Las condiciones nos permiten decidir el código a ejecutar dependiendo de que condición suceda. Las condicionales se definen con la palabra clave **if** y, la estructura es:

**if (condición){**

**//código a ejecutar si se cumple la condición.**

**} else {**

**//código a ejecutar si no se cumple la condición.**

**}**

Podemos concatenar varias condiciones con la palabra clave **else if**. Ej:

**if (condición){**

**//código a ejecutar si se cumple la condición.**

**} else if (condición){**

**//código a ejecutar si se cumple esta otra condición**

**} else {**

**//código a ejecutar si no se cumple la condición.**

**}**

Si añades en la condición el nombre de la variable de un boleano ( **if (variable)** ) será lo mismo que hacer esto: **if (variable == true)**.

Operadores booleanos para condicionales:

* **<** menor que
* **>** mayor que
* **<=** menor o igual que
* **>=** mayor o igual que
* **==** igual
* **!=** diferente
* **===** igual valor y tipo de variable
* **&&** y
* **||** o

Más operadores:

* **+=** nos permite sumar un valor al valor actual. Equivale a **[n = n + valor]**
* **!** operador diferente
* **?** operador ternario. Sirve para evaluar una condición (es un if simplificado)

Ej: Verificación de edad para ver una película

**const starWars7 = 'Star Wars: El despertar de la Fuerza';**

**const pgStarWars7 = 13;**

**const nameDani = 'Daniel';**

**let ageDani = 22;**

**const nameSanti = 'Santi';**

**let ageSanti = 12;**

**// Función Verificar Edad**

**const canWatchStarWars7 = (name, age, isWithAdult = false) => {**

**if (age >= pgStarWars7) {**

**alert(`${name} puede pasar a ver ${starWars7}`);**

**} else if(isWithAdult && age < pgStarWars7){**

**alert(`${name} puede pasar a ver ${starWars7}.**

**Aunque su edad es ${age}, se encuentra acompañada/o por un adulto.`);**

**}**

**else{**

**alert(`${name} no puede pasar a ver ${starWars7}.**

**Tiene ${age} años y necesita tener ${pgStarWars7}`);**

**}**

**}**

**canWatchStarWars7(nameDani, ageDani);**

**canWatchStarWars7(nameSanti, ageSanti, true);**

**Ciclos For**

Como en la gran mayoría de lenguajes de programación tenemos disponible el **ciclo for**, que nos permitirá ejecutar el código dentro del bloque todas las veces que definamos.

La estructura de un for está compuesta por tres valores: el iterador, la condición y como cambia el iterador en cada iteración.

**for ([iterador]; [condición]; [cambio iterador]){ [CÓDIGO] }**

Ej: Vamos a imprimir 6 veces la palabra “hola”.

**for (let i = 0; i <= 5; i++){**

**console.log(‘hola’);**

**}**

La descripción exacta de la orden es esta: “Inicia una variable i con el valor 0, ejecuta el código mientras i sea menor o igual a 5 y, cada vez que completes el código dentro del bloque, suma 1 a i. Cuando i sea igual o mayor a 5 para de ejecutar la iteración.”

Ej: Vamos a recorrer e imprimir los elementos dentro de un array con los días de la semana.

**const dias = ["lunes", "martes", "miercoles", "jueves", "viernes"];**

**for (let i = 0; i < dias.length; i++){**

**console.log('dia: ', dias[i]);**

**}**

**Ciclos While**

Otro de los bucles que nos proporciona la gran mayoría de lenguajes de programación, Javascript incluido por supuesto, es el ciclo While. El ciclo while simplifica el bucle haciendo que mientras la condición se cumpla, el ciclo se siga repitiendo. Dependiendo de que requeramos daremos uso a un tipo u otro de ciclo.

La estructura es esta: **while (condición) {código}** .

Como es evidente, la construcción es mucho más simple pero el funcionamiento sigue siendo igual de complejo que un ciclo for.

**Utilizando el objeto Math para cálculos y manipular números.**

Javascript contiene un objeto que nos permite ejecutar todas las operaciones matemáticas básicas; este es Math. Con Math podemos redondear, calcular con PI, elevar una base con un exponente, logaritmos, cálculos trigonométricos, etc…

**Math.random()** -> Nos permite generar un número aleatorio entre 0 y 1 (Ej: 0.6739625186950882)

**Math.pow()** -> Eleva un número base al exponente indicado. Math.pow(base, exponente).

**Math.floor()** -> Redondea un número decimal hacia abajo. Ej: Math.floor(3.9) = 3.

**Math.ceil()** -> Redondea un número decimal hacia arriba. Ej: Math.ceil(3.1) = 4.

**Math.round()** -> Redondea al entero más cercano. Ej: Math.round(3.1) = 3 | Math.round(3.5) = 4.

**.toFixed()** -> esta función nos permite definir la cantidad de decimales a mostrar sobre un número, pasándole como parámetro el número de decimales que deseamos. (no forma parte de la librería Math sino que es un método propio del objeto tipo Number, al igual que **.sqrt()** que nos sirve para hacer una raiz cuadrada de un número).

Hay muchos más métodos disponibles en el objeto Math y en el objeto Number que podemos utilizar para hacer todo tipo de operaciones.

**Objeto Date()**

Para utilizar fechas y tiempos, Javascript tiene un objeto nativo llamado Date() que nos permitirá calcular y gestionar fechas.

La estructura para generar una fecha es: **new Date(Año[0000], Més[00], Día[00]);**

Cuidado a la hora de especificar el mes. Javascript recoge los meses en un array, por lo tanto el Enero és el més 0, y el Diciembre és el més 11. Hay que restar un número siempre donde el més para tener la fecha correcta.

Si escribimos **new Date()** sin introducir ningún parámetro nos generará la fecha actual.

Con estas fechas podemos hacer múltiples cosas con los métodos propios del objeto, y, podemos hacer cálculos entre ellas como si de un número entero se tratase.