console.log("hello world");

console.log("second message")

//Комментарий

/\*

Многострочный

комментарий

\*/

let message;

message="Переменная";

console.log(message);

let message2="Переменная2";

let first\_name,last\_name,drink="cola",food="pizza";

const Pi=3.14;

console.log("Константа Пи="+Pi)

console.log(typeof Pi);

let b=true;

console.log(typeof b);

let s="Строка";

console.log(typeof s);

let n=123.2;

console.log(typeof n);

let undefvar;

console.log(typeof undefvar);

console.log(5%3); //Остаток от деления

console.log(5\*\*3); //Возведение в степень

/\*

a += b такой же какa = a + b

a -= b такой же какa = a - b

a \*= b такой же какa = a \* b

a /= b такой же какa = a / b

a %= b такой же какa = a % b

a \*\*= b такой же какa = a \*\* b

\*/

//Инкремент

/\*

a = a + 1;

a += 1;

a++;

\*/

//Дикремент

/\*

a = a - 1;

a -= 1;

a--;

\*/

//форматируемая строка

let score=100;

console.log("Ваш счет равен:"+score);

console.log(`Ваш счет равен: ${score}`);

//Преобразование типов

/\*

Number("5") // 5 (number)

Number("-6.5") // -6.5 (number)

Number(5) // 5 (still a number)

Number("Hello") // NaN

String(5) // "5" (string)

String(-6.5) // "-6.5" (string)

String("Hello") // "Hello" (still a string)

String(NaN) // "NaN" (string)

Boolean(0) // false (is zero)

Boolean(NaN) // false (is NaN)

Boolean(1) // true (is not zero or NaN)

Boolean(42) // true (is not zero or NaN)

Boolean(Infinity) // true (is not zero or NaN)

Boolean("") // false (is empty string)

Boolean("Hello") // true (is not empty string)

Boolean("0") // true (is not empty string)

Boolean(true) // true (no change)

Boolean(false) // false (no change)

\*/

//Сравнение

/\*

5 == 5 // true

5 == "5" // true (converts types)

5 === 5 // true

5 === "5" // false (types are different)

5 != "5" // false (converts types)

5 !== "5" // true (types are different)

\*/

//not

let bb = true;

console.log(!bb); // false

//и или

/\*

false || false // false

false || true // true

true || false // true

true || true // true

false && false // false

false && true // false

true && false // false

true && true // true

\*/

score = 120;

console.log(score >= 100 ? "Nice score!" : "Try again!");

/\*

0 ? "truthy" : "falsy" // "falsy"

1 ? "truthy" : "falsy" // "truthy"

"" ? "truthy" : "falsy" // "falsy"

"Hello!" ? "truthy" : "falsy" // "truthy"

\*/

// Имя функции опущено

let myFunc = function()

{

console.log("Hello world!");

};

myFunc();

console.log(typeof myFunc); // function

// Объявите функцию, которая вызывает функцию, переданную в качестве ее параметра

function logReturnValueOf(func)

{

console.log(`The function provided returned: ${func()}`);

}

// Вызовите вышеуказанную функцию, а также предоставьте ей функцию для вызова

logReturnValueOf(function ()

{

return "Hello world!";

});

// Объявите функцию, которая возвращает другую функцию

function getFunction()

{

return function()

{

console.log("Hello world!");

};

}

// Вызовите функцию getFunction(), которая возвращает функцию

let returnedValue = getFunction();

// Вызовите возвращаемую функцию, регистрируя сообщение "Hello world!".

returnedValue();

function factorial(n)

{

if (n === 1)

{

return 1;

}

else

{

// Recursion happens here

return n \* factorial(n - 1);

}

}

console.log(`5! = ${factorial(5)}`);

function factorial(n)

{

let product = 1;

for ( ; n > 1; n--)

{

product \*= n;

}

return product;

}

console.log(`5! = ${factorial(5)}`);

//Стрелочные функции

function (a, b)

{

return a + b;

}

(a, b) =>

{

return a + b;

}

(a) => a + 1

a => a + 1

// Equivalent to:

function (a)

{

return a + 1;

}

() => "Hello world!"

// Equivalent to:

function ()

{

return "Hello world!";

}

unusedParam => "Hello world!"

\_ => "Hello world!"

// Declare a function that calls the function passed as its parameter

function logReturnValueOf(func)

{

console.log(`The function provided returned: ${func()}`);

}

// Call the above function, and also give it a function to call

logReturnValueOf(function ()

{

return "Hello world!";

});

// Declare a function that calls the function passed as its parameter

function logReturnValueOf(func)

{

console.log(`The function provided returned: ${func()}`);

}

// Call the above function, and also give it a function to call

logReturnValueOf(() => "Hello world!");

//Замыкания

function getFunction()

{

let message = "Hello world!";

return function()

{

return message;

}

}

// Get a function returning 'message'

let returnedFunc = getFunction();

// Call it and log the result to the console

console.log(returnedFunc());

function makeAddFunction(addNumber)

{

return function(n)

{

return n + addNumber;

}

}

let add2 = makeAddFunction(2);

let add10 = makeAddFunction(10);

console.log(add2(5)); // 7

console.log(add10(5)); // 15

function makeCounterFunction()

{

let counter = 0;

return function()

{

return counter++;

};

}

let counterFuncA = makeCounterFunction();

let counterFuncB = makeCounterFunction();

console.log(counterFuncA()); // 0

console.log(counterFuncA()); // 1

console.log(counterFuncA()); // 2

console.log(counterFuncB()); // 0

//Объекты

let myObject = {

firstProp: "Hello world!",

secondProp: 123

};

console.log(typeof myObject); // object

console.log(myObject.firstProp); // Hello world!

console.log(myObject.thirdProp); // undefined

myObject.thirdProp = 456;

delete myObject.secondProp;

let myObject = {

// Use a string for a property name

"myProperty": 5

};

// This is equivalent to:

let myObject = {

myProperty: 5

};

// String properties allow spaces and special characters

let myObject = {

"my property!": 5

};

// Normal properties don't allow this

myObject = {

// SyntaxError

my property!: 5

};

let myObject = {

"my property!": 5

};

// Assign to a string property

myObject["my property!"] = 7;

// Read a string property

console.log(myObject["my property!"]); // 7

let myObject = {

myProperty: 5

};

// Read with dot property

console.log(myObject.myProperty); // 5

// Read with string property

console.log(myObject["myProperty"]); // 5

let myObject = {

myProperty: 5

};

// Access a property via a string variable

let propName = "myProperty";

console.log(myObject[propName]); // 5

let myObject = {

myProperty: undefined

};

// Reading a missing property returns undefined

console.log(myObject.otherProperty); // undefined

// But reading myProperty also returns undefined

console.log(myObject.myProperty); // undefined

let myObject = {

myProperty: undefined

};

console.log("otherProperty" in myObject); // false

console.log("myProperty" in myObject); // true

let person = {

// The "name" property is another object

name: {

first: "Joe",

last: "Bloggs"

},

age: 30

};

// Access the first name with dots

console.log(person.name.first); // Joe

// Access the first name with strings

console.log(person["name"]["first"]); // Joe

let myObject = {

// Prefer to use null for empty properties

myProperty: null

};

let myObject = {

myProperty: null

};

console.log(myObject?.myProperty?.anotherProperty); // undefined

let someFunction = null;

// Not an error even though someFunction is null,

// because optional ?.() syntax used

console.log(someFunction?.());

function logFood(obj)

{

// Log the "food" property of the object

console.log(obj.food);

}

// An object can be passed via a variable storing an object.

let myObject = {

food: "banana"

};

logFood(myObject); // logs "banana"

// An object can also be passed directly. This logs "pizza".

logFood({

food: "pizza"

});

// Add a global property named myGlobal

globalThis.myGlobal = "Initial string";

// Assign myGlobal as if it's a variable

myGlobal = "Hello world!";

// Log myGlobal as if it's a variable

console.log(myGlobal); // Hello world!

//Методы

let person = {

name: "Joe",

logMessage: function()

{

console.log("Hello world!");

}

};

// Logs "Hello world!" to the console

person.logMessage();

let person = {

name: "Joe",

// Shorthand syntax for method

logMessage()

{

console.log("Hello world!");

}

};

person.logMessage();

let person = {

name: "Joe",

logName()

{

console.log(`The person's name is ${this.name}`);

}

};

// Logs "The person's name is Joe"

person.logName();

// Note the logName function uses 'this'

function logName()

{

console.log(`The person's name is ${this.name}`);

}

let person1 = {

name: "Joe"

};

let person2 = {

name: "Tom"

};

// Add the logName function as properties of both objects

person1.logName = logName;

person2.logName = logName;

// Logs: "The person's name is Joe"

person1.logName();

// Logs: "The person's name is Tom"

person2.logName();

// This function is called via 'new', so 'this'

// will be set to a new empty object

function Person(name, age, preferredFood)

{

// Add the parameters as properties on the new object

this.name = name;

this.age = age;

this.preferredFood = preferredFood;

}

// Now each person object can be created with 'new'

let person1 = new Person("Joe", 25, "pizza");

let person2 = new Person("Jenny", 30, "noodles");

let person3 = new Person("Maya", 20, "burrito");

// Show each person in the console

console.log("person1:", person1);

console.log("person2:", person2);

console.log("person3:", person3);

function Person(name, age, preferredFood)

{

this.name = name;

this.age = age;

this.preferredFood = preferredFood;

// Add a function

this.logName = function ()

{

console.log(`The person's name is ${this.name}`);

}

}

// Create a person

let person = new Person("Joe", 25, "pizza");

// Call the logName function on it

// Logs: The person's name is Joe

person.logName();

let myArray = ["pizza", "noodles", "burrito"];

console.log(`The array length is ${myArray.length}`);

let myArray = ["pizza", "noodles", "burrito"];

// Repeat once for each element of myArray

for (let elem of myArray)

{

console.log(`The array element is ${elem}`);

}

let myArray = ["😀", "👽", "👾"];

// Add a new element at the end of the array

myArray.push("🤖")

// Show the array in the console

// It now has: 😀, 👽, 👾, 🤖

console.log(myArray);

let myArray = ["😀", "👽", "👾"];

// Remove element from end of array

let removedElem = myArray.pop();

// Show the array in the console

// It now has: 😀, 👽

console.log("The array has: ", myArray);

// Show the removed element in the console (👾)

console.log(`The removed element is: ${removedElem}`);

// Create a new empty set

let mySet = new Set();

// Add a string to the set

mySet.add("😀");

// Display the set in the console

console.log("mySet is: ", mySet);

// Create a set with two strings

let mySet = new Set(["😀", "👽"]);

// Add another string to the set

mySet.add("👾");

// Display the set in the console

// It now has: 😀, 👽, 👾

console.log("mySet is: ", mySet);

// Create a set with two strings

let mySet = new Set(["😀", "👽"]);

// Add another string to the set

mySet.add("👾");

// Add the same string again - this does nothing

// as it's already in the set

mySet.add("👾");

// Display the set in the console

// It still has: 😀, 👽, 👾

console.log("mySet is: ", mySet);

console.log(`The set size is ${mySet.size}`);

let mySet = new Set(["😀", "👽", "👾"]);

mySet.delete("👽");

// Set has 😀 and 👾

console.log("mySet is: ", mySet);

let mySet = new Set(["😀", "👽"]);

console.log("mySet is: ", mySet);

// true

console.log("mySet has 😀: ", mySet.has("😀"));

// false

console.log("mySet has 👾: ", mySet.has("👾"));

let mySet = new Set(["😀", "👽", "👾"]);

for (let item of mySet)

{

console.log(`mySet has item ${item}`);

}

let mySet = new Set(["😀", "👽"]);

let asArray = [...mySet];

console.log("asArray: ", asArray);

function removeDuplicates(arr)

{

// Convert array to set to remove duplicates

let asSet = new Set(arr);

// Return set converted back to an array

return [...asSet];

}

console.log("Example 1: ", removeDuplicates(["😀", "👽", "👾", "😀"]));

console.log("Example 2: ", removeDuplicates(["👽", "👽", "👽", "👾", "🦄", "👾"]));

// Create a new empty map

let myMap = new Map();

// Add three keys for emoji and their associated description

myMap.set("😀", "smiley face");

myMap.set("👽", "alien");

myMap.set("🦄", "unicorn");

// Show the map in the console

console.log("myMap is: ", myMap);

// Retrieve the description for each emoji

console.log("Value for 😀: ", myMap.get("😀"));

console.log("Value for 👽: ", myMap.get("👽"));

console.log("Value for 🦄: ", myMap.get("🦄"));

let myMap = new Map([

["😀", "smiley face"],

["👽", "alien"],

["🦄", "unicorn"]

]);

console.log("Value for 😀: ", myMap.get("😀"));

console.log("Value for 👽: ", myMap.get("👽"));

console.log("Value for 🦄: ", myMap.get("🦄"));

let myMap = new Map([

["😀", "smiley face"],

["👽", "alien"],

["🦄", "unicorn"]

]);

// Delete a key

myMap.delete("👽");

// The size is 2, as one of the keys was deleted

console.log(`myMap size is ${myMap.size}`);

// true

console.log("myMap has 😀: ", myMap.has("😀"));

// false (it was deleted)

console.log("myMap has 👽: ", myMap.has("👽"));

let myMap = new Map([

["😀", "smiley face"],

["👽", "alien"],

["🦄", "unicorn"]

]);

// Iterate map keys

for (let key of myMap.keys())

{

console.log(`Map key: ${key}`);

}

// Iterate map values

for (let value of myMap.values())

{

console.log(`Map value: ${value}`);

}

let myMap = new Map([

["😀", "smiley face"],

["👽", "alien"],

["🦄", "unicorn"]

]);

// Iterate both keys and values

for (let [key, value] of myMap.entries())

{

console.log(`Map key: ${key}, value: ${value}`);

}

let myMap = new Map([

["😀", "smiley face"],

["👽", "alien"],

["🦄", "unicorn"]

]);

// Convert map back to array

let asArray = [...myMap];

// Log array to console.

// This is the same format as the map was

// initialised with, i.e.:

// [

// ["😀", "smiley face"],

// ["👽", "alien"],

// ["🦄", "unicorn"]

// ]

console.log("Map converted to array: ", asArray);

// Try entering in to the browser console:

// Math.PI: convenient constant for PI

Math.PI // 3.141592653589793

// Math.round(): round number to nearest whole number

Math.round(5.2) // 5

Math.round(5.6) // 6

Math.round(-1.2) // -1

// Math.floor(): round down to nearest whole number

// (towards negative infinity)

Math.floor(5.2) // 5

Math.floor(5.6) // 5

Math.floor(-1.2) // -2

// Math.ceil(): round up to nearest whole number

// (towards infinity)

Math.ceil(5.2) // 6

Math.ceil(5.6) // 6

Math.ceil(-1.2) // -1

// Math.max(): get highest of a range of numbers

// (you can pass as many parameters as you like)

Math.max(3, 5) // 5

Math.max(3, 7, 1, 4) // 7

// Math.min(): get the lowest of a range of numbers

Math.min(3, 5) // 3

Math.min(3, 7, 1, 4) // 1

// Math.abs(): get absolute value (turns a number positive)

Math.abs(-3) // 3

Math.abs(3) // 3

// Math.sqrt(): calculate square root

Math.sqrt(25) // 5

// Math.pow(): raise to power, like \*\* operator

Math.pow(5, 2) // 25

// Math.random(): generate random number in range 0-1

// (excluding 1) - each call returns a different number

Math.random()

// Trigonometric calculations. Note these work with angles

// in radians - where a full rotation is 2 \* pi -

// instead of using degrees. Remember fractional calculations

// are not always perfectly precise, so answers may be very

// close to but not exactly equal to the true answer.

Math.sin(Math.PI / 2) // 1

Math.cos(0) // 1

Math.tan(Math.PI / 4) // 0.9999999999999999

Math.asin(1) // 1.5707963267948966

Math.acos(1) // 0

Math.atan(1) // 0.7853981633974483

// Math.atan2(y, x) is another special method which does some extra

// work to make it useful for calculating angles. For example a position

// at (10, 10) is 45 degrees from the origin, so Math.atan2(10, 10)

// returns Math.PI / 4 (which is 45 degrees in radians).

Math.atan2(10, 10) // 0.7853981633974483

// Get a random whole number up to, but not including, the limit

function getRandomNumber(limit)

{

return Math.floor(Math.random() \* limit);

}

// Generate 20 random numbers from 0-9 in an array

let arr = [];

for (let i = 0; i < 20; i++)

{

arr.push(getRandomNumber(10));

}

// Display array of random numbers in console

console.log("Random numbers: ", arr);

// Try entering in to the browser console:

// toLowerCase() returns a new string with lower case

"Hello World".toLowerCase() // "hello world"

// toUpperCase() returns a new string with upper case

"Hello World".toUpperCase(); // "HELLO WORLD"

// includes() returns a boolean indicating if a given

// string appears in the string it is called on

"Hello world".includes("world") // true

"Hello world".includes("pizza") // false

// startsWith() returns a boolean indicating if the

// string it is called on starts with a given string

"Hello world".startsWith("Hello") // true

"Hello world".startsWith("Pizza") // false

// endsWith() does the same but for the end of the string

"Hello world".endsWith("world") // true

"Hello world".endsWith("pizza") // false

// repeat() returns a new string which repeats the

// string it is called on a number of times

"😀".repeat(5) // 😀😀😀😀😀

// split() returns an array of strings separated

// by a given character

"pizza,chocolate,burrito".split(",")

// returns an array:

// ["pizza", "chocolate", "burrito"]

// A string with an emoji

let str = "Hi😀";

// Convert the string to an array

let arr = [...str];

// The array now has:

// [ "H", "i", "😀" ]

// Log details about the string from the array

console.log(`The array length is ${arr.length}`);

console.log(`The 3rd character is ${arr[2]}`);

// A string with an emoji

let str = "Hi😀";

// Use a for-of loop to repeat for each character

for (let ch of str)

{

console.log(`String character: ${ch}`);

}

// Try entering in to the browser console:

["😀", "👽", "🤖"].join(",") // "😀,👽,🤖"

// Try entering in to the browser console:

["😀", "👽", "🤖"].at(-1) // 🤖

// Try entering in to the browser console:

["😀", "👽", "🤖"].includes("👽") // true

["😀", "👽", "🤖"].includes("👻") // false

// Try entering in to the browser console:

["😀", "👽", "🤖"].indexOf("👽") // 1

["😀", "👽", "🤖"].indexOf("👻") // -1

let arr = ["😀", "👽"];

console.log(`Array at start: ${arr.join(",")}`);

// Add at start of array

arr.unshift("🤖");

// Array is now:

// ["🤖", "😀", "👽"]

console.log(`Array after unshift: ${arr.join(",")}`);

// Remove from start of array

let removedElem = arr.shift();

// Array is now:

// ["😀", "👽"]

console.log(`Array after shift: ${arr.join(",")}`);

console.log(`Removed element: ${removedElem}`);

let arr = ["😀", "👽", "🤖"];

arr.fill("👾");

// Array is now ["👾", "👾", "👾"]

console.log(`Array after fill: ${arr.join(",")}`);

let arr = ["😀", "👽", "🤖"];

arr.reverse();

// Array is now ["🤖", "👽", "😀"]

console.log(`Array after reverse: ${arr.join(",")}`);

let arr1 = ["😀", "👽", "🤖", "👻"];

// Copy from index 1 to the end

// This returns ["👽", "🤖", "👻"]

let arr2 = arr1.slice(1);

// Copy from index 1 to index 3 (not including 3)

// This returns ["👽", "🤖"]

let arr3 = arr1.slice(1, 3);

let arr = ["😀", "👽", "🤖", "👻"];

console.log(`Starting array: ${arr.join(",")}`);

// splice(1, 2) means deletes 2 elements from index 1

arr.splice(1, 2);

// Array is now: ["😀", "👻"]

console.log(`After deleting 2 elements: ${arr.join(",")}`);

// splice(1, 0, "👽", "🤖") means from index 1, don't delete

// anything (as we provide 0 for the number of elements to

// delete), and insert elements "👽" and "🤖".

// So this adds back the elements that were deleted.

arr.splice(1, 0, "👽", "🤖");

// Array is now: ["😀", "👽", "🤖", "👻"]

console.log(`After adding 2 elements: ${arr.join(",")}`);

// We can also simultaneously delete and insert elements.

// This time we delete 2 elements and insert 2 elements,

// replacing the middle two elements with different emoji.

arr.splice(1, 2, "👾", "🎉");

// Array is now: ["😀", "👾", "🎉", "👻"]

console.log(`After replacing 2 elements: ${arr.join(",")}`);

let arr = ["beta", "alpha", "gamma"];

arr.sort();

// Array is now: ["alpha", "beta", "gamma"]

console.log(`Sorted array: ${arr.join(",")}`);

let counter = 0;

setInterval(() =>

{

console.log(`setInterval callback, counter = ${counter++}`);

}, 1000);

document.addEventListener("click", () =>

{

console.log("Click event!");

setTimeout(() =>

{

console.log("Callback 1 second after click")

}, 1000);

});