CONCEPTOS BÁSICOS

* JavaScript está influenciado sobre todo por la sintaxis de Java, pero también de Awk, Perl y Python.
* JavaScript es **case-sensitive** (distingue mayúsculas y minúsculas) y utiliza el conjunto de caracteres **Unicode**.
* En JavaScript, las instrucciones son llamadas [Sentencias](https://developer.mozilla.org/es/docs/Glossary/Sentencias) y son separadas por un punto y coma (;).
* Los espacios, tabulaciones y caracteres de nueva línea son llamados espacios en blanco.
* El texto original de los scripts de Javascript se obtiene escaneando de izquierda a derecha y se convierte en una secuencia de elementos de entrada que son tokens, caracteres de control, terminaciones de línea, comentarios o espacios de línea.
* ECMAScript también define ciertas keywords (palabras clave) y literales y ***tiene reglas automáticas para insertar puntos y coma*** ([ASI](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Automatic_semicolon_insertion)) al final de las sentencias. Sin embargo, es recomendable siempre añadir punto y coma al final de sus sentencias; evitará efectos secundarios.
* Para más información, ver la referencia detallada sobre [gramática léxica](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar) (en inglés) de Javascript.

COMENTARIOS

La sintaxis de comentarios es la misma que C++ y en muchos otros lenguajes:

// comentario en una sola línea

/\* este es un comentario

multilínea

\*/

/\* no puedes, sin embargo, /\* anidar comentarios \*/ SyntaxError \*/

DECLARACIONES

Hay tres tipos de declaraciones en JavaScript.

**var**

Declara una variable, **inicializándola** opcionalmente a un valor.

**let**

Declara una variable local en un bloque de ámbito, **inicializándola** opcionalmente a un valor.

**const**

Declara una constante de solo lectura en un bloque de ámbito.

VARIABLES

Las variables se usan como nombres simbólicos para valores en tu aplicación. Los nombres de las variables, llamados *identificadores*, se rigen por ciertas reglas:

* Un identificador en JavaScript tiene que empezar con una letra, un guion bajo (\_) o un símbolo de dólar ($); los valores subsiguientes pueden ser números.
* Debido a que JavaScript diferencia entre mayúsculas y minúsculas, las letras incluyen tanto desde la **"A"** hasta la **"Z" (**mayúsculas) como de la **"a"** hasta la **"z"**.
* Puedes usar la ISO 8859-1 o letras Unicode tales como å y ü en un identificador.
* Puedes también usar el [Unicode escape sequences](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#String_literals) como caracteres en identificadores.
* Algunos ejemplos de nombre permitidos son Numero\_Visitas, temp99 y \_nombre.

**Declarando variables**

Puedes declarar una variable de tres maneras:

* Con la palabra clave [var](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Sentencias/var" \o "en-US/docs/JavaScript/Reference/Statements/var). Por ejemplo, var x=42. Esta sintaxis puede ser usada para declarar tanto variables locales como globales.
* Simplemente asignándole un valor. Por ejemplo, x=42. Esto siempre declara una variable global y no puede ser cambiada a nivel local. Esto genera una advertencia strict de JavaScript. No deberías usar esta variante.
* Con la palabra clave let. Por ejemplo, let y=13. Esta variable puede ser usada para declarar una variable local en un bloque de ámbito. Ver [Ámbito de variable](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Valores,_variables_y_literales#.C3.81mbito_de_variable) más abajo.

**Evaluando variables**

Una variable declarada usando la sentencia var o let sin asignarle un valor inicial tiene el valor undefined.

Al intentar acceder a una variable no declarada dará como resultado la excepción ReferenceError:

var a;

console.log('El valor de a es ' + a); // El valor de a es undefined

console.log('El valor de b es ' + b); // El valor de b es undefined

var b;

console.log('El valor de c es ' + c); // Uncaught ReferenceError: c no está definida

let x;

console.log('El valor de x es ' + x); // El valor de x es undefined

console.log('El valor de y es ' + y); // Uncaught ReferenceError: y no está definida

let y;

Se puede usar undefined  para determinar si una variable tiene un valor. En el siguiente código a la variable **input** no se le asigna ningún valor y la sentencia de control if la evalúa como true.

var input;

if(input === undefined){

hazEsto();

} else {

hazEso();

}

El valor undefined se comporta como un false cuando se utiliza *en un contexto booleano*. Por ejemplo, el siguiente código ejecuta la función myFunction porque el elemento **myArray** no ha sido definido:

var myArray = new Array();

if (!myArray[0]) myFunction();

El valor undefined se convierte en NaN, no numérico, cuando se usa en una operación aritmética.

var a;

a + 2; // Se evalua a NaN

Cuando se evalúa una variable nula, el valor null se comporta como el 0 en operaciones aritméticas y como false en operaciones lógicas. Por ejemplo:

var n = null;

console.log(n \* 32); // Va a lanzar 0 a la consola

**Ámbito de variable**

Cuando declaras una variable fuera de una función, ésta es llamada *variable global*, porque *está disponible para cualquier otro código en el documento actual*. Cuando declaras una variable dentro de una función, ésta es llamada *variable local*, porque está disponible solo dentro de esa función.

Antes de ECMAScript 6 Javascript no tenía ámbito de  [sentencias de bloque](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Control_de_flujo_y_manejo_de_errores); más bien, una variable declarada dentro de un bloque es local para la *función (o ámbito global)*en la que reside el bloque.Por ejemplo, el siguiente código registrará 5, porque el ámbito de x es la función (o contexto global) dentro del cual se declara x, no el bloque, que en este caso es la sentencia if.

if (true) {

var x = 5;

}

console.log(x); // x vale 5

Este comportamiento cambia, cuando usamos la declaración let introducida en ECMAScript 2015.

if (true) {

let y = 5;

}

console.log(y); // ReferenceError: y no está definida

**Variable hoisting**

Otra cosa inusual acerca de las variables en JavaScript es que pueden hacer referencia a una variable declarada más tarde, sin obtener una excepción. Este concepto se conoce como **hoisting**; Las variables en JavaScript son en cierto sentido "elevadas" o elevadas a la parte superior de la función o declaración. **Sin embargo**, las variables que no se han inicializado todavía devolverán un valor undefined.

/\*\*

\* Ejemplo 1

\*/

console.log(x === undefined); // true

var x = 3;

/\*\*

\* Ejemplo 2

\*/

// devolverá un valor undefined

var myvar = "my value";

(function() {

console.log(myvar); // undefined

var myvar = "local value";

})();

Los mismos ejemplos anteriores serán interpretados como:

/\*\*

\* Ejemplo 1

\*/

var x;

console.log(x === undefined); // true

x = 3;

/\*\*

\* Ejemplo 2

\*/

var myvar = "my value";

(function() {

var myvar;

console.log(myvar); // undefined

myvar = "local value";

})();

Debido al hoisting, todas las declaraciones var en una función deben colocarse lo más cerca posible de la parte superior de la función. Esta buena práctica mejora la claridad del código.

En ECMAScript 2015, let (const) no elevará la variable a la parte superior del bloque. Sin embargo, hacer referencia a la variable en el bloque antes de la declaración de la variable resulta en un ReferenceError. La variable se encuentra en una "zona muerta temporal" desde el inicio del bloque hasta que se procesa la declaración.

console.log(x); // ReferenceError

let x = 3;

**Función hoisting**

Para las funciones, solo la declaración de la función se eleva a la parte superior y no a la expresión de la función.

/\* Declaración de la función \*/

foo(); // "bar"

function foo() {

console.log("bar");

}

/\* Expresión de la función \*/

baz(); // TypeError: baz no es una función

var baz = function() {

console.log("bar2");

};

**Variables Globales**

Las variables globales son, de hecho, propiedades del *objeto global*. En las página web el objeto global es window, puede establecer y acceder a las variables globales usando la sintaxis window.variable.

Por lo tanto, puedes acceder a variables globales declaradas en una ventana o frame de otra ventana o frame especificando el nombre de la ventana o frame. Por ejemplo, si una variable denominada **phoneNumber** es declarada en un documento, puede consultar a esta variable desde un iframe como parent.phoneNumber.

CONSTANTES

Puede crear una de solo lectura, llamada constante con la palabra clave const. La sintaxis del identificador de la constante es el mismo como para un indentificador de variable: debe de empezar con una letra, guión bajo(\_) o símbolo de dollar($) y puede contener alfabéticos, numéricos o guiones bajos.

const PI = 3.14;

Una constante no puede cambiar de valor mediante la asignación o volver a declararse mientras se ejecuta el script.

Las reglas de ámbito para las constantes son las mismas que las de las variables let en un ámbito de bloque. Si la palabra clave const es omitida, el identificador se asume que representa una variable.

No puedes declarar una constante con el mismo nombre que una función o una variable en el mismo ámbito. Por ejemplo:

// ESTO CAUSARÁ UN ERROR

function f() {};

const f = 5;

// ESTO TAMBIÉN CAUSARÁ UN ERROR

function f() {

const g = 5;

var g;

//sentencias

}

**ESTRUCTURAS DE DATOS Y TIPOS**

**Tipos de datos**

El último estándar ECMAScript define siete tipos de datos:

* Seis tipos de datos que son [primitivos](https://developer.mozilla.org/es/docs/Glossary/Primitivo) (Un primitivo (valor primitivo, tipo de datos primitivo) es un dato que no es un objeto y no tiene métodos:
* [Boolean](https://developer.mozilla.org/es/docs/Glossary/Boolean). true y false.
* [null](https://developer.mozilla.org/es/docs/Glossary/Null). Una palabra clave especial que denota un valor null. Como JavaScript es case sensitive, null no es lo mismo que null, NULL, o cualquier otra variante.
* [undefined](https://developer.mozilla.org/es/docs/Glossary/undefined). Una propiedad de alto nivel cuyo valor no es definido.
* [Number](https://developer.mozilla.org/es/docs/Glossary/Numero). 42 ó 3.141516.
* [String](https://developer.mozilla.org/es/docs/Glossary/String). "Hola"
* [Symbol](https://developer.mozilla.org/es/docs/Glossary/Symbol) (nuevo en ECMAScript 6).
* y [Object](https://developer.mozilla.org/es/docs/Glossary/Objecto).

Aunque estos tipos de datos son una cantidad relativamente pequeña, permiten realizar funciones útiles con tus aplicaciones. Los otros elementos fundamentales en el lenguaje son los Objects y las funciones. Puedes pensar en objetos como contenedores con nombre para los valores, y las funciones como procedimientos que puede realizar tu aplicación.

**Conversión de tipos de datos**

JavaScript es un lenguaje de tipo dinámico. Esto significa que no tienes que especificar el tipo de dato de una variable cuando la declaras, y los tipos de datos son convertidos automáticamente de acuerdo a lo que se necesite en la ejecución del script. Así, por ejemplo, puedes definir una variable de la siguiente manera:

var respuesta = 42;

Y luego, puedes asignarle una cadena a esa misma variable, por ejemplo:

respuesta = "Gracias por todo el pescado...";

Debido a que es un lenguaje de tipos dinámicos, esta asignación no causa un mensaje de error.

En expresiones que involucran valores numéricos y de string con el operador +, JavaScript convierte valores numéricos a string. Por ejemplo, considera las siguientes declaraciones:

x = "La respuesta es " + 42 // "La respuesta es 42"

y = 42 + " es la respuesta" // "42 es la respuesta"

En declaraciones que involucran otros operadores, JavaScript no convierte los valores numéricos a string. Por ejemplo:

"37" - 7 // 30

"37" + 7 // "377"

**Convertir string a números**

En el caso que un valor, representando un número, está en memoria como string, hay métodos para la conversión.

**parseInt()** y **parseFloat()**

* [parseInt()](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/parseInt)
* [parseFloat()](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/parseFloat)

parseInt sólo retornará números enteros, por lo que su uso es disminuido por los decimales. Adicionalmente, una buena práctica para parseInt es incluir siempre el parámetro **radix**. El parámetro **radix** es usado para especificar qué sistema numérico será usado.

Un método alternativo para recibir un número de un string es con el operador + (más unario):

"1.1" + "1.1" = "1.11.1"

(+"1.1") + (+"1.1") = 2.2

// Nota: los paréntesis son para fines demostrativos, no son obligatorios.

**LITERALES**

**Los literales** se utilizan para representar valores en JavaScript. Estos son valores fijos, no variables, que literalmente proporciona en su script. Esta sección describe los siguientes tipos de literales:

* [Literales Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Literales_Array)
* [Literales Boolean](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Literales Boolean)
* [Literales  de punto Flotante](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Literales _de_punto_Flotante)
* [Literales Enteros](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Literales_Enteros)
* [Literales  Tipo Objeto](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Literales _Tipo_Objeto)
* [Literales String](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Literales_String_)

**Literales Array**

Un literal array es un lista de cero o más expresiones, cada uno representa un elemento array, entre corchetes ([]). Cuando crea un array usando un literal array, se inicializa con los valores especificados como sus elementos, y su longitud se establece por el número de argumentos especificados.

El siguiente ejemplo crea el array *cafes* con tres elementos y una longitud de tres:

var cafes = ["Tostado Frances", "Colombiano", "Kona"];

**Nota:** Un literal array es un tipo de inicializador de object. Consulte [Uso de iniciadores de objeto](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects#Using_object_initializers).

Si un array se crea usando un literal en un script de nivel superior, JavaScript interpreta el array cada vez que se evalúa la expresión que contiene el literal array. Además, un literal usado en una función se crea cada vez que es llamada la función.

Los literales array son también objetos array. Consulte [Array](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/Array) y  [colecciones indexadas](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Indexed_collections) para los detalles sobre objetos Array.

**Comas adicionales en literales array**

No tienes que especificar todos los elementos en un literal array. Si pones dos comas en una fila, se crea el array con undefined para los elementos no especificados. El siguiente ejemplo crea el array peces:

var peces = ["Leon", , "Angel"];

Este array tiene dos elementos con valores y un elemento vacío (peces[0] es "Leon", peces[1] es undefined, y peces[2] es "Angel").

Si incluyes una coma al final de la lista de los elementos, la coma es ignorada. En el siguiente ejemplo, la longitud del array es tres. No hay  miLista[3]. Todas las demás comas en la lista indican un nuevo elemento. (**Nota:** Las comas finales pueden crear errores en las versiones anteriores del navegador y es una buena práctica eliminarlos.)

var miLista = ['casa', , 'escuela', ];

En el siguiente ejemplo, la longitud del array es cuatro, y miLista[0] y miLista[2]  faltan.

var miLista = [ , 'casa', , 'escuela'];

En la siguiente ejemplo, la longitud del array es cuatro, y miLista[1] y miLista[3] faltan. Solo la última coma es ignorada.

var miLista = ['casa', , 'escuela', , ];

Comprender el comportamiento de las comas adicionales es importante para comprender JavaScript como un lenguaje, sin embargo, cuando escribimos nuestro propio código: declaramos explícitamente los elementos que faltan como undefined esto aumentará la claridad y el mantenimiento de su código.

**Literales Booleanos**

Los literales de tipo Booleanos tienen 2 valores posibles: true y false.

NO confundir los valores primitivos Booleanos true y false con los valores true y false del Objeto Booleano. El objeto Booleano es un contenedor alrededor del tipo de dato Primitivo Booleano. Para más información revisa [Booleano](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Booleano) .

**Literales Enteros**

Los Enteros pueden ser expresados en decimal (base 10), hexadecimal (base 16), octal (base 8) y binario (base 2).

* El Literal entero decimal consiste en una secuencia de dígitos sin un líder 0 (cero).
* 0 (cero) en un literal entero indica que está en octal. Los enteros octales pueden incluir sólo los dígitos 0-7.
* El tipo 0x (or 0X) indica hexadecimal. Los enteros Hexadecimal pueden incluir dígitos del  (0-9) y letras a-f y A-F.

Algunos ejemplos de enteros Literales son:

0, 117 and -345 (decimal, base 10)

015, 0001 and -077 (octal, base 8)

0x1123, 0x00111 and -0xF1A7 (hexadecimal, "hex" o base 16)

Para más Información, revisa la [Referencia de literales Numéricos en la Léxica Gramática](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Lexical_grammar#Numeric_literals).

**Literales de Punto Flotante**

Un Literal de punto flotante, puede tener las siguientes partes:

* Un entero decimal que puede tener signo (precedido por  "+" o "-"),
* Un Punto decimal ("."),
* Una Fracción (otro número decimal),
* Un Exponente.

La parte exponente es una  "e" o "E" seguida por un entero, la cual puede tener un signo (precedido por "+" o "-"). Un literal de punto flotante debe tener un dígito y bien sea un punto decimal  o el exponente "e" (or "E").

Algunos ejemplos de literales de punto FLotante Son:  3.1415, -3.1E12, .1e12, y 2E-12.

Específicamente, la sintaxis es:

[(+|-)][digitos][.digitos][(E|e)[(+|-)]digitos]

Por ejemplo:

3.14

2345.789

.3333333333333333333

-.283185307179586

**Literales Tipo Objeto**

Un literal Objeto es una lista de cero o más parejas de nombres de propiedades asociadas con el respectivo valor, encerradas entre corchetes ({}).

* No debes usar un objetos literal al comienzo de una sentencia.
* Esto dará lugar a un error o no se comprobará como se espera, porque los corchetes ‘{‘ serán interpretados como el inicio de un bloque.

El siguiente es un ejemplo de un objeto literal. El primer elemento del objeto carro define una propiedad, miCarro, y asigna a éste el String , "Saturn";  el segundo elemento, a la propiedad getCarro, se le asigna inmediatamente el resultado de invocar a la función  (TiposCarro("Honda")); el tercer elemento, la propiedad especial, usa una variable existente (Ventas).

var Ventas = "Toyota";

function TiposCarro(nombre) {

if (nombre== "Honda") {

return nombre;

} else {

return "Lo siento, nosotros no vendemos " + nombre + ".";

}

}

var carro = { miCarro: "Saturn", getCarro: TiposCarro("Honda"), especial: Ventas};

console.log(carro.miCarro); // Saturn

console.log(carro.getCarro); // Honda

console.log(carro.especial); // Toyota

Adicionalmente el nombre de una propiedad puede ser un literal entero o puede corresponder a otro objeto, como se muestra a continuación.

var auto = { algunosAutos: {a: "Saab", "b": "Jeep"}, 7: "Mazda" };

console.log(auto.algunosAutos.b); // Jeep

console.log(auto[7]); // Mazda

Los nombres de las propiedades de un objeto pueden ser cualquier string, incluyendo un  string vacío. Si el nombre de la propiedad no es un [identificador](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Values,_variables,_and_literals#Variables) JavaScript válido, éste debe ser encerrado en barras cuadradas. Los nombres de propiedad que no son identificadores válidos, no pueden ser accedidos con la propiedad punto (.), pero pueden ser accedidos y seteados con la notación de un arreglo ("[]").

var propiedadesDeNombreInusual = {

"": "Un string vacio",

"!": "Bang!"

}

console.log(propiedadesDeNombreInusual .""); // SyntaxError: Unexpected string

console.log(propiedadesDeNombreInusual [""]); // "Un string vacio"

console.log(propiedadesDeNombreInusual .!); // SyntaxError: Unexpected token !

console.log(propiedadesDeNombreInusual ["!"]); // "Bang!"

 Ten en cuenta:

var foo = {a: "alpha", 2: "two"};

console.log(foo.a); // alpha

console.log(foo[2]); // two

//console.log(foo.2); // Error: missing ) after argument list

//console.log(foo[a]); // Error: a is not defined

console.log(foo["a"]); // alpha

console.log(foo["2"]); // two

**Literales String**

Un literal String corresponde a Cero o más caracteres, encerrados dentro de comillas dobles (") o sencilla (') . Un string debe estar delimitado por comillas del mismo tipo; esto quiere decir que, siempre ambas son dobles o sencillas en cada uno de los casos. Los siguientes son ejemplos de literales String:

* "foo"
* 'bar'
* "1234"
* "Una linea \n otra linea"
* "El gato de Jhon"

Puedes utilizar cualquiera de los métodos del objeto  String en un literal de tipo string, JavaScript automáticamente convierte el literal string en un objeto String de manera temporal, llama al método, y finalmente destruye el objeto temporal de tipo String. También puedes usar la propiedad String.length con un literal string:

console.log("El gato de Jhon".length)

// Imprimira el numero de caracteres en el literal, incluyendo espacios en blanco.

// En este caso, 15.

Se recomienda que uses un literal string a menos que sea específicamente necesario el uso de un objeto de tipo String. Mira [String](https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Objetos_globales/String" \o "Un objeto que representa una serie de caracteres dentro de una cadena.) para detalles acerca de objetos String .

**Uso de caracteres especiales en strings**

Además de los caracteres normales, también puede incluir caracteres especiales en los strings, como se muestra en el siguiente ejemplo:

"una linea \n otra linea"

La siguiente tabla lista los caracteres especiales que se pueden usar en un string JavaScript.

| **Tabla 2.1 caracteres especiales JavaScript** | |
| --- | --- |
| **Character** | **Meaning** |
| \b | Retroceso |
| \f | Avance de Página |
| \n | Nueva Línea |
| \r | Retorno de Línea |
| \t | Tabulador |
| \v | Tabulador Vertical |
| \' | Apóstrofe o comilla sencilla |
| \" | Comilla doble |
| \\ | Carácter Backslash |
| \*XXX* | Los caracteres con la codificación Latin-1 especificada por tres dígitos octales *XXX* entre 0 y 377. Por ejemplo, \251 es la secuencia octal para el símbolo copyright. |
|  |  |
| \x*XX* | Los caracteres con la codificación Latin-1 especificada por dos dígitos hexadecimales *XX* entre 00 y FF. Por ejemplo, \xA9 es la secuencia hexadecimal para el símbolo copyright. |
|  |  |
| \u*XXXX* | Los caracteres Unicode especificados por la secuencia de cuatro dígitos Hexadecimales  *XXXX*. Por ejemplo, \u00A9 es la secuencia Unicode para el símbolo copyright . Ver [Secuencias de escape Unicode](https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#Secuencias_de_escape_Unicode_). |

**Caracteres de Escape**

Para caracteres no listados en la Tabla 2.1, la precedencia del  backslash es ignorada, pero su uso está obsoleto y debe ser evitado.

Puedes insertar comillas dobles dentro de un string anteponiendo un carácter backslash, significa esto como un escape de las comillas. Por ejemplo:

var quote = "El lee \"La cremación de Sam McGee\" de R.W. Service.";

console.log(quote);

El resultado de esto sería:

El lee "La cremación de Sam McGee" de R.W. Service.

Para incluir un literal backslash en un string, debes usar el caracter de escape backslash. Por ejemplo, para asignar la ruta c:\temp a un string, use lo siguiente:

var home = "c:\\temp";

También puedes insertar saltos de línea. El backslash y el salto de línea son removidos del valor del string.

var str = "this string \

is broken \

across multiple\

lines."

console.log(str); // this string is broken across multiplelines.

Aunque JavaScript no tiene sintaxis "heredoc" puede acercarse insertando un backslash y un salto de línea al final de cada línea:

var poem =

"Roses are red,\n\

Violets are blue.\n\

I'm schizophrenic,\n\

And so am I."