FUNDACION UNIVERSITARIA SAN MATEO

LUIS FERNANDO AHUMADA AHUMADA

PROGRAMACION EN ENTORNOS WEB

PROFESOR HECTOR FABIO SOTO DURAN

11 de septiembre de 2019

INVESTIGACIÓN DE OPERADORES MATEMÁTICOS Y FUNCIONES BASICAS DE JAVA

OPERADORES MATEMÁTICOS DE JAVA SCRIPT

1. Suma o adición (+)

La operación suma se produce mediante la suma de número o strings concatenados.

Operador: x + y

* 1. // Número + Número = Adición
  2. 1 + 2 // 3
  3. // Bolean + Número = Adición
  4. true + 1 // 2
  5. // Bolean + Bolean // Adición
  6. false + false // 0
  7. // Número + String = Concatenación
  8. 5 + 'foo' // "5foo"
  9. // String + Bolean = Concatenación
  10. 'foo' + true // "footrue"
  11. // String + String = Concatenación
  12. 'foo' + 'bar' // "foobar"

1. Resta o sustracción (-)

La resta se produce cuando se restan el resultado de los operadores, produciendo su diferencia.

Operador: x - y

* 1. 5 - 3 // 2
  2. 3 - 5 // -2
  3. 'foo' - 3 // NaN

1. División (/)

El operador división se produce el cociente de la operación donde el operando izquierdo es el dividendo y el operando derecho es el divisor.

Operador: x / y

* 1. 1 / 2 // devuelve 0.5 en JavaScript
  2. 1 / 2 // devuelve 0 en Java
  3. 2.0 // devuelve 0.5 en JavaScript y Java
  4. 2.0 / 0 // devuelve Infinito en JavaScript
  5. 0.0 // devuelve Infinito
  6. -0.0 // devuelve -Infinito en JavaScript

1. Multiplicación (\*)

El operador multiplicación produce el producto de la multiplicación de los operandos.

Operador: x \* y

* 1. 2 \* 2 // 4
  2. -2 \* 2 // -4
  3. Infinito \* 0 // NaN
  4. Infinito \* Infinito // Infinito
  5. 'foo' \* 2 // NaN

1. Resto o Residuo  (%)

El operador resto devuelve el resto que queda cuando un operando se divide por un segundo operando. Siempre toma el signo del dividendo, no el divisor. Utiliza una función de modulo incorporada para producir el resultado, que es el resto entero de dividir var1 por var2 - por ejemplo - var1 modulo var2. Existe una propuesta para obtener un operador de módulo real en una versión futura de ECMAScript, con la diferencia de que el resultado del operador de módulo tomaría el signo del divisor, no el dividendo.

Operador: var1 % var2

* 1. 12 % 5 // 2
  2. -1 % 2 // -1
  3. 1 % -2 // 1
  4. NaN % 2 // NaN
  5. 1 % 2 // 1
  6. 2 % 3 // 2
  7. -4 % 2 // -0
  8. 5.5 % 2 // 1.5

1. Exponenciación (\*\*)

El operador de exponenciación devuelve el resultado de elevar el primer operando al segundo operando de potencia. es decir, var1var2, en la declaración anterior, donde var1 y var2 son variables. El operador de exponenciación es asociativo a la derecha. a \*\* b \*\* ces igual a a \*\* (b \*\* c).

Ejemplo

Operador: var1 \*\* var2

Ejemplo

* 1. -2 \*\* 2;
  2. // 4 en Bash, -4 en otros idiomas.
  3. // Esto no es válido en JavaScript, ya que la operación es ambigua.
  4. -(2 \*\* 2);
  5. // -4 en JavaScript y la intención del autor no es ambigua.
  6. Ejemplos
  7. 2 \*\* 3 // 8
  8. 3 \*\* 2 // 9
  9. 3 \*\* 2.5 // 15.588457268119896
  10. 10 \*\* -1 // 0.1
  11. NaN \*\* 2 // NaN
  12. 2 \*\* 3 \*\* 2 // 512
  13. 2 \*\* (3 \*\* 2) // 512
  14. (2 \*\* 3) \*\* 2 // 64

Para invertir el signo del resultado de una expresión de exponenciación:

* 1. -(2 \*\* 2) // -4

Para forzar la base de una expresión de exponenciación a ser un número negativo:

* 1. (-2) \*\* 2 // 4

1. Incremento (++)
   1. El operador de incremento incrementa (agrega uno a) su operando y devuelve un valor.
   2. Si se usa postfijo, con el operador después del operando (por ejemplo, x++), devuelve el valor antes de incrementar.
   3. Si se usa prefijo, con el operador antes del operando (por ejemplo, ++x), devuelve el valor después de incrementar.

Operador: x++ o ++x

// Postfijo

* 1. var x = 3;
  2. y = x++; // y = 3, x = 4

// Prefijo

* 1. var a = 2;
  2. b = ++a; // a = 3, b = 3

1. Decremento (--)

El operador de decremento disminuye (resta uno de) su operando y devuelve un valor.

Si se usa postfijo (por ejemplo, x--), devuelve el valor antes de decrementar.

Si usa el prefijo (por ejemplo, --x), entonces devuelve el valor después de decrementar.

Operador: x-- o --x

// Postfijo

* 1. var x = 3;
  2. y = x--; // y = 3, x = 2

// Prefijo

* 1. var a = 2;
  2. b = --a; // a = 1, b = 1

1. Negación unaria (-)

El operador de negación unaria precede su operando y lo niega.

Operador: -x

* 1. var x = 3;
  2. y = -x; // y = -3, x = 3

// el operador de negación unario puede convertir no-números en un número

* 1. var x = "4";
  2. y = -x; // y = -4

1. Unario más (+)

El operador unario más precede su operando y evalúa su operando, pero intenta convertirlo en un número, si no lo está. Aunque la negación unaria (-) también puede convertir no números, unario plus es la manera más rápida y preferida de convertir algo en un número, porque no realiza ninguna otra operación en el número.

Operador: +x

* 1. +3 // 3
  2. +'3' // 3
  3. +true // 1
  4. +false // 0
  5. +null // 0
  6. +function(val){ return val } // NaN

# Funciones y propiedades básicas de JavaScript

**Cadenas de texto**

A continuación se muestran algunas de las funciones más útiles para el manejo de cadenas de texto:

length, calcula la longitud de una cadena de texto (el número de caracteres que la forman)

var mensaje = "Hola Mundo";

var numeroLetras = mensaje. length; *// numeroLetras = 10*

+, se emplea para concatenar varias cadenas de texto.

var mensaje1 = "Hola";

var mensaje2 = " Mundo";

var mensaje = mensaje1 + mensaje2; *// mensaje = "Hola Mundo"*

Además del operador +, también se puede utilizar la función concat ()

var mensaje1 = "Hola";

var mensaje2 = mensaje1.concat(" Mundo"); *// mensaje2 = "Hola Mundo"*

Las cadenas también se pueden unir con variables numéricas:

var variable1 = "Hola ";

var variable2 = 3;

var mensaje = variable1 + variable2; *// mensaje = "Hola 3"*

Cuando se unen varias cadenas de texto es habitual olvidar añadir un espacio de separación entre las palabras:

var mensaje1 = "Hola";

var mensaje2 = "Mundo";

var mensaje = mensaje1 + mensaje2; *// mensaje = "HolaMundo"*

Los espacios en blanco se pueden añadir al final o al principio de las cadenas o indicarlos de forma explícita:

var mensaje1 = "Hola";

var mensaje2 = "Mundo";

var mensaje = mensaje1 + " " + mensaje2; *// mensaje = "Hola Mundo"*

toUpperCase (), transforma todos los caracteres de la cadena a sus correspondientes caracteres en mayúsculas:

var mensaje1 = "Hola";

var mensaje2 = mensaje1.toUpperCase(); *// mensaje2 = "HOLA"*

toLowerCase (), transforma todos los caracteres de la cadena a sus correspondientes caracteres en minúsculas:

var mensaje1 = "HolA";

var mensaje2 = mensaje1.toLowerCase(); *// mensaje2 = "hola"*

charAt(posicion), obtiene el carácter que se encuentra en la posición indicada:

var mensaje = "Hola";

var letra = mensaje. charAt (0); *// letra = ‘H’*

letra = mensaje. charAt (2); *// letra = ‘l’*

indexOf(letra), calcula la primera posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

var mensaje = "Hola";

var posicion = mensaje. indexOf('a'); *// posicion = 3*

posicion = mensaje. indexOf('b'); *// posicion = -1*

La función indexOf () comienza su búsqueda desde el principio de la palabra y solo devuelve la primera posición de todas las existentes. Su función análoga es lastIndexOf ().

lastIndexOf(letra), calcula la última posición en la que se encuentra el carácter indicado dentro de la cadena de texto. Si la cadena no contiene el carácter, la función devuelve el valor -1:

var mensaje = "Hola";

var posicion = mensaje. lastIndexOf(‘a’); *// posicion = 3*

posicion = mensaje. lastIndexOf(‘b’); *// posicion = -1*

La función lastIndexOf () comienza su búsqueda desde el final de la cadena hacia el principio, aunque la posición devuelta es la correcta empezando a contar desde el principio de la palabra.

substring (inicio, final), extrae una porción de una cadena de texto. El segundo parámetro es opcional. Si solo se indica el parámetro inicio, la función devuelve la parte de la cadena original correspondiente desde esa posición hasta el final:

var mensaje = "Hola Mundo";

var porcion = mensaje. substring (2); *// porcion = "la Mundo"*

porcion = mensaje. substring (5); *// porcion = "Mundo"*

porcion = mensaje. substring (7); *// porcion = "ndo"*

Si se indica un inicio negativo, se devuelve la misma cadena original:

var mensaje = "Hola Mundo";

var porcion = mensaje. substring (-2); *// porcion = "Hola Mundo"*

Si se indica el inicio y el final, se devuelve la parte de la cadena original comprendida entre la posición inicial y la inmediatamente anterior a la posición final (es decir, la posición inicio está incluida y la posición final no):

var mensaje = "Hola Mundo";

var porcion = mensaje. substring (1, 8); *// porcion = "ola Mun"*

porcion = mensaje. substring (3, 4); *// porcion = "a"*

Si se indica un final más pequeño que un inicio, JavaScript los considera de forma inversa, ya que automáticamente asigna el valor más pequeño al inicio y el más grande al final:

var mensaje = "Hola Mundo";

var porcion = mensaje. substring (5, 0); *// porcion = "Hola "*

porcion = mensaje. substring (0, 5); *// porcion = "Hola "*

split(separador), convierte una cadena de texto en un array de cadenas de texto. La función parte una cadena de texto dividiendo sus trozos a partir del carácter delimitador indicado:

var mensaje = "¡Hola Mundo, soy una cadena de texto!";

var palabras = mensaje. split (" ");

*// palabras = ["Hola”, “Mundo,”, “soy”, “una”, “cadena”, “de”, ¡” texto!"];*

Con esta función se pueden extraer fácilmente las letras que forman una palabra:

var palabra = "Hola";

var letras = palabra. split (""); *// letras = ["H", "o", "l", "a"]*

**Arrays**

A continuación se muestran algunas de las funciones más útiles para el manejo de arrays:

length, calcula el número de elementos de un array:

var vocales = ["a", "e", "i", "o", "u"];

var numeroVocales = vocales. length; *// numeroVocales = 5*

concat (), se emplea para concatenar los elementos de varios arrays:

var array1 = [1, 2, 3];

array2 = array1.concat(4, 5, 6); *// array2 = [1, 2, 3, 4, 5, 6]*

array3 = array1.concat([4, 5, 6]); *// array3 = [1, 2, 3, 4, 5, 6]*

join(separador), es la función contraria a split (). Une todos los elementos de un array para formar una cadena de texto. Para unir los elementos se utiliza el carácter separador:

var array = ["hola", "mundo"];

var mensaje = array. join (""); *// mensaje = "holamundo"*

mensaje = array. join (" "); *// mensaje = "hola mundo"*

pop (), elimina el último elemento del array y lo devuelve. El array original se modifica y su longitud disminuye una unidad.

var array = [1, 2, 3];

var ultimo = array.pop ();

*// ahora array = [1, 2]*

push (), añade un elemento al final del array. El array original se modifica y aumenta su longitud una unidad. También es posible añadir más de un elemento a la vez.

var array = [1, 2, 3];

array. push (4);

*// ahora array = [1, 2, 3, 4]*

shift (), elimina el primer elemento del array y lo devuelve. El array original se modifica y su longitud disminuye una unidad.

var array = [1, 2, 3];

var primero = array. shift ();

*// ahora array = [2, 3]*

unshift (), añade un elemento al principio del array. El array original se modifica y aumenta su longitud en una unidad. También es posible añadir más de un elemento a la vez.

var array = [1, 2, 3];

array. unshift (0);

*// ahora array = [0, 1, 2, 3]*

reverse (), modifica un array colocando sus elementos en el orden inverso a su posición original:

var array = [1, 2, 3];

array. reverse ();

*// ahora array = [3, 2, 1]*