



Bootcamp

Programando en JavaScript

Yesid Oswaldo Quintero
Martinez
Ingeniero de Sistemas
Fecha 13/02/2024

UT TALENTOTECH

Tabla de contenidos

1

Programación Básica

2

Funciones

3

Promesas

Programación en JavaScript

JavaScript (JS) es un lenguaje de programación ligero, interpretado, o compilado justo-a-tiempo (just-in-time) con funciones de primera clase. Si bien es más conocido como un lenguaje de scripting (secuencias de comandos) para páginas web, y es usado en muchos entornos fuera del navegador, tal como Node.js, Apache CouchDB y Adobe Acrobat JavaScript es un lenguaje de programación basada en prototipos, multiparadigma, de un solo hilo, dinámico, con soporte para programación orientada a objetos, imperativa y declarativa

Programación en JavaScript

JavaScript

Orientado a objetos. No hay distinción entre tipos de objetos. La herencia se realiza a través del mecanismo de prototipo, y las propiedades y métodos se pueden agregar a cualquier objeto de forma dinámica.

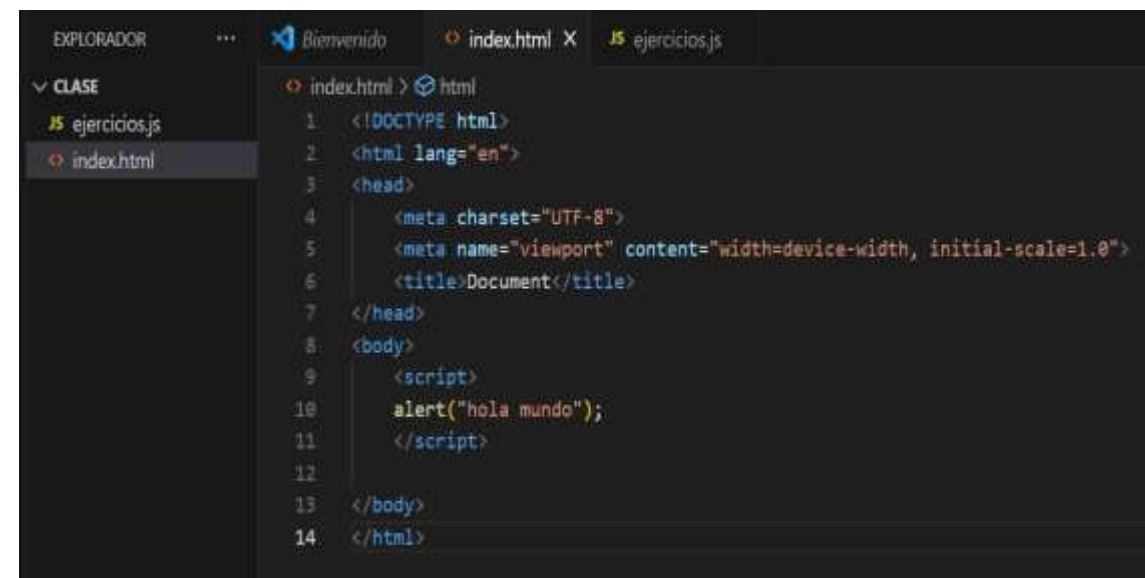
Los tipos de datos de las variables no se declaran (tipado dinámico, tipado flexible).

Java

Basado en clases. Los objetos se dividen en clases e instancias con toda la herencia a través de la jerarquía de clases. Las clases y las instancias no pueden tener propiedades o métodos agregados dinámicamente.

Los tipos de datos de las variables se deben declarar (tipado estático, fuertemente tipado).

Programación en JavaScript



The screenshot shows a code editor with a dark theme. On the left, the 'EXPLORADOR' (Explorer) sidebar shows a file structure with 'ejercicios.js' and 'index.html'. The main editor area displays the content of 'index.html', which is an HTML document. The code includes a DOCTYPE declaration, a lang attribute set to 'en', a head section with meta tags for charset (UTF-8) and viewport (width=device-width, initial-scale=1.0), and a title 'Document'. The body section contains a script tag with an alert function that displays 'hola mundo'.

```
index.html > html
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <script>
10     alert("hola mundo");
11   </script>
12 </body>
13 </html>
```


Que es una variable

Una variable es un elemento de datos con nombre cuyo valor, puede cambiar durante el curso de la ejecución de un programa.

Las variables pueden ser globales o locales. Una variable es global a no ser que esté declarada dentro de una definición de función. Las variables globales resultan visibles y disponibles para todas las sentencias de un script. Las variables locales sólo resultan visibles y disponibles dentro de la función en la que están definidas.

Tipos de Datos

```
// Tipos de Datos primitivos
```

```
let numeros = 5; //Number  
let decimal = 7.5 // Number  
let texto = "hola mundo"; //String  
let verdadero = true; // Boolean  
let falso = false; // Boolean  
let sin_valor;  
let undef = undefined; // Undefined  
let nulo = null; //Null
```

```
// Tipos de Datos de referencia  
Array , Object, functions, Clases
```


Operador typeof

Es el operador que nos permite retornar el tipo de dato que se encuentra almacenado en la variable ejemplo

```
numeroDecimal = 10.7;  
console.log(typeof(numeroDecimal))  
numero = 655;  
console.log(typeof(numero))  
mensaje = "hola mundo" ;  
console.log(typeof(mensaje))
```


Métodos mas comunes para una variable tipo String en el tabla ejemplo str

Método/Atributo	Acción
<u>str.length</u>	Retorna la longitud en caracteres del string
str. [n]	Retorna el carácter en la n posición
str.toLowerCase()	Retorna el string en minúscula
str.toUpperCase()	Retorna el string en mayúscula
str.split(separador)	Retorna un array de strings separados
str.substr(inicio, fin)	Retorna la posición del string definida por los valores de inicio y fin
str.includes(str2)	Define si str2 está incluido en str1; por lo que retorna true o false

Operadores Aritméticos

```
console.log (5+7.5)  
console.log (5-7.5)  
console.log (5*7.5)  
console.log (5/2)  
console.log (5%7.5)  
console.log (5**7.5)
```


Operaciones de comparación y sentencias condicionales

```
console.log (5!= 7) // true  
console.log (5>7.5) // false  
console.log (5<7.5) // true  
console.log (5 == 7.5) // false  
console.log (5 == 5) // true
```

Operadores AND y OR

AND(&&) se deben cumplir las condiciones para activar el flujo

OR(||) se puede cumplir solo una condición

```
let y = 5;  
  
✓ if(y>2 || y== 0){  
    console.log("y es mayor que 2 ")  
✓ }else{  
    console.log(" y es menor a 2 ")  
}
```


Switch-case

switch-case, que se trata de una estructura de flujo condicional en diferentes casos específicos. Su estructura base se puede apreciar a continuación:

```
switch (variable){  
  case 1:  
    console.log("variable es igual a uno");  
    break;  
  
  case 2:  
    console.log("variable es igual a dos");  
    break;  
  
  case 3:  
    console.log("variable es igual a 3");  
    break;  
  default:  
  
}
```

Bucles

Como se puede apreciar, la instrucción del ciclo está compuesta por tres zonas, cada una separada por punto y coma.

- a. let i=0: inicializa la variable i dentro del ciclo. Esta variable es dinámica, cambia su valor durante cada iteración y sólo es reconocida dentro del ciclo for.
- b. i < 10: condición que permite la repetición del ciclo siempre que ésta sea cierta.
- c. i++: incrementa en 1 el valor de i durante cada iteración.

```
for (let i = 0; i<10; i++ ){  
  console.log(i);  
}
```


Estructuras for in y for of

Un ciclo for of recorre la estructura de datos elemento a elemento, este es parecido a forEach de otros lenguajes de programación. se recorren todos los valores de la clave nombre y id en la lista de personas.

Con un for in se puede recorrer la colección de propiedades (nombre, tipo, id) de un objeto, por ejemplo:

```
let lista_personas = [ {"nombre" : "carlos", "tipo": "persona", "id" : 435},  
                        {"nombre" : "hugo", "tipo": "persona", "id" : 876},  
                        {"nombre" : "daniel", "tipo": "persona", "id" : 673}  
]  
for(let personas of lista_personas){  
  console.log(personas.nombre);  
  console.log(personas.id);  
}
```

While

la sentencia while se usa para repetir un conjunto de operaciones hasta que se cumpla la condición especificada.

```
const n = 6;  
var fact = 1 ;  
  
for (let i = 1; i<= n; i++){  
    fact *= i;  
}  
console.log(" el factorial del numero " + n + " es " + fact);
```


Estructuras de datos Arrays

Los arrays son objetos similares a una lista cuyo prototipo proporciona métodos para efectuar operaciones de recorrido y de mutación. Tanto la longitud como el tipo de los elementos de un array son variables. Dado que la longitud de un array puede cambiar en cualquier momento, y los datos se pueden almacenar en ubicaciones no contiguas, no hay garantía de que los arrays de JavaScript sean densos; esto depende de cómo el programador elija usarlos.

Metodos para manipular arrays en este caso personas

Método/Atributo	Acción
<code>personas.length</code>	Retorna la longitud del array
<code>personas.push(el)</code>	Adicional el elemento el al final del array
<code>personas.pop()</code>	Elimina el último elemento del array
<code>personas.unshift(el)</code>	Adicional el elemento el al inicio del array
<code>personas.shift()</code>	Eliminar el primer elemento del array
<code>personas.indexOf(el)</code>	Retorna el índice del elemento el
<code>personas.splice(pos, numEl)</code>	Retorna un nuevo array con la cantidad de elementos numEl , sobre la posición pos especificada sobre el array original. Y modifica el array original removiendo los elementos especificados
<code>personas.slice()</code>	Retorna una copia del array original

Funciones

Las funciones son uno de los bloques de construcción fundamentales en JavaScript. Una función en JavaScript es similar a un procedimiento — un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función

Una definición de función (también denominada declaración de función o expresión de función) consta de la palabra clave function, seguida de:

El nombre de la función.

Una lista de parámetros de la función, entre paréntesis y separados por comas.

Las declaraciones de JavaScript que definen la función, encerradas entre llaves, { ... }.

Ejercicios funciones

```
pruebas.js > ...  
  ▾ // Funciones  
    // Ejercicio 1  
  ▾ function imprimir () {  
      console.log("esto es una funcion");  
      console.log(57+90);  
      return 80/2;  
  }  
  const impi = imprimir();  
  console.log(impi);  
  
  //Ejercicio 2  
  ▾ function sumar(){  
      let n1= 25;  
      let n2= 50;  
      let suma= n1+n2;  
      return suma;  
  }  
  
  console.log(sumar());
```


Ejercicios funciones

```
//Ejercicio 3
const sum = function(){
  console.log(76+90);
}

sum();

// Ejercicio 4
let sumas = (a,b) => a+b;
console.log(sumas(8,10));

// Ejercicio 5
let mensaje = () => ` hola Bienvenidos`;
console.log(mensaje());

//Ejercicio 6

let sumando = function (num1, num2=5){
  return num1 + num2;
}

console.log(sumando(7));
console.log(sumando(237));
```

Igualdad Estricta (===)

El operador de estricta igualdad (===) revisa si dos operandos son iguales y produce un resultado Booleano. A diferencia del operador de igualdad regular (==), el operador de estricta igualdad siempre considera que los operandos de distinto tipo de valor son diferentes y nunca similares.

JavaScript Demo: Expressions - Strict equality operator

```
1 console.log(1 === 1);  
2 // Expected output: true  
3  
4 console.log('hello' === 'hello');  
5 // Expected output: true  
6  
7 console.log('1' === 1);  
8 // Expected output: false  
9  
10 console.log(0 === false);  
11 // Expected output: false  
12
```


Promesa

es un objeto que representa la terminación o el fracaso de una operación asíncrona. Dado que la mayoría de las personas consumen promises ya creadas

```
const peliculasAccion = [
  {
    nombre: "Duro de matar",
    año: 2000
  },
  {
    nombre: "Ninja",
    año: 2003
  },
  {
    nombre: "la Isla",
    año: 2005
  }
];
//const peliculasAccion = [];
function getPeliculas(){
  return new Promise((resolve, reject) => {
    if(peliculasAccion.length === 0){
      reject(new Error("No se encuentran Datos error"));
    }
    setTimeout(() => { resolve(peliculasAccion); }, 2000 );
  });
}

getPeliculas()
  .then((peliculasAccion) => console.log(peliculasAccion))
  .catch((err) => console.log(err.message));
```

Función global setTimeout()

El método global setTimeout() establece un temporizador que ejecuta una función o un fragmento de código específico una vez que expira el temporizador.

setTimeout() Es una función asíncrona, lo que significa que la función del temporizador no detendrá la ejecución de otras funciones en la pila de funciones. En otras palabras, no puede utilizarlo setTimeout() para crear una "pausa" antes de que se active la siguiente función en la pila de funciones.

```
setTimeout(() => {  
  console.log("this is the first message");  
}, 5000);  
setTimeout(() => {  
  console.log("this is the second message");  
}, 3000);  
setTimeout(() => {  
  console.log("this is the third message");  
}, 1000);
```


Callbacks

Son funciones que se pasan como un argumento en otras funciones, y se ejecutan después de que se complete alguna operación, no necesariamente debe ser asíncrona, se invocan en la función principal para completar alguna tarea, se pueden llamar en otras funciones

```
function crearCita(cita, callback){  
  var miCita = "Como yo siempre digo, " + cita;  
  callback(miCita); // 2  
}  
  
function logCita(cita){  
  console.log(cita);  
}  
  
crearCita("come tus vegetales!", logCita); // 1  
  
// Resultado en la consola:  
// Como yo siempre digo, come tus vegetales!
```

Async/await

La finalidad de las funciones `async/await` es simplificar el comportamiento del uso síncrono de promesas y realizar algún comportamiento específico en un grupo de Promises. Del mismo modo que las Promises son semejantes a las devoluciones de llamadas estructuradas, `async/await` se asemejan a una combinación de generadores y promesas.

Ejercicio Async/await

```
function resolveAfter2Seconds() {  
  return new Promise((resolve) => {  
    setTimeout(() => {  
      resolve('resolved');  
    }, 2000);  
  });  
}  
  
async function asyncCall() {  
  console.log('calling');  
  const result = await resolveAfter2Seconds();  
  console.log(result);  
  // Expected output: "resolved"  
}  
  
asyncCall();
```

Imports y Exports

La declaración export se utiliza al crear módulos de JavaScript para exportar funciones, objetos o tipos de dato primitivos del módulo para que puedan ser utilizados por otros programas con la sentencia import

BIBLIOGRAFÍA:

<https://developer.mozilla.org/es/docs/Web/JavaScript>
<https://www.ibm.com/docs/es/tcamfma/6.3.0?topic=test-constants-variables-4>
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/Strict_equality
https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Using_promises
<https://www.freecodecamp.org/espanol/news/que-es-una-funcion-callback-javascript/>
<https://developer.mozilla.org/en-US/docs/Web/API/setTimeout>
https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/async_function
<https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/export>
<https://es.javascript.info/import-export>

¡Gracias!