**PROCESO DIRECCIÓN DE FORMACIÓN PROFESIONAL INTEGRAL**

**FORMATO GUÍA DE APRENDIZAJE**

## IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: ANALISIS Y DESARROLLO DE SISTEMAS DE INFORMACION.
- Código del Programa de Formación: 228106
- Nombre del Proyecto:
- Fase del Proyecto: ANÁLISIS
- Actividad de Proyecto: DESARROLLO LOS MÓDULOS DEL SISTEMA INTEGRAL WEB DE ACUERDO A LOS ESTÁNDARES DE CALIDAD Y NORMATIVIDAD LEGAL VIGENTE.
- Competencia: CONSTRUIR EL SISTEMA QUE CUMPLA CON LOS REQUISITOS DE LA SOLUCIÓN INFORMÁTICA.
- Resultados de Aprendizaje ALCANZAR: REALIZAR LA CODIFICACIÓN DE LOS MÓDULOS DEL SISTEMA Y EL PROGRAMA PRINCIPAL, A PARTIR DE LA UTILIZACIÓN DEL LENGUAJE DE PROGRAMACIÓN SELECCIONADO, DE ACUERDO CON LAS ESPECIFICACIONES DEL DISEÑO.
- Duración de la Guía

## 2. PRESENTACIÓN

En el desarrollo web el lenguaje más utilizado en el mundo es JavaScript.

Vamos a aprender a programar en este lenguaje, conociendo los fundamentos básicos de programación (variables, condicionales, funciones y demás). Fundamentos básicos y necesarios para el desarrollo web. Quizás al principio será un poco difícil de comprender, pero con práctica, perseverancia y buena actitud puedes lograr comprenderlo.

## 3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

- Materiales:
  git o simulador javascript o visual code
  Portatil ó Computador de Escritorio
- Ambiente Requerido
- Descripción de la(s) Actividad(es)

**ECM6**

ECMAScript es el estándar que define cómo debe de ser el lenguaje Javascript. es la especificación que los fabricantes deben seguir al crear intérpretes para Javascript.

Cabe mencionar que Java Script es un lenguaje interpretado y procesado por múltiples plataformas; todos los responsables de cada una de esas tecnologías ó plataformas se encargan de interpretar el lenguaje tal como dice el estándar ECMAScript.

JavaScript 5 (Js5), es el estándar ó la versión más usada. Así que cuando alguien dice que conoce ó maneja este lenguaje, se refiere a que maneja ó programa en la versión Js5. Desde que se le encargó a ECMA (una organización internacional cuyo objetivos son desarrollar, en cooperación con las organizaciones de estándares nacionales, europeas e internacionales, estándares y reportes técnicos con el fin de facilitar y estandarizar el uso de las tecnologías de información y comunicación y dispositivos electrónicos; promover el uso correcto de los estándares, mediante la influencia en el medio en el que se aplican; y, finalmente, hacer públicos estos estándares y reportes técnicos de forma impresa o electrónica, pudiendo ser copiados por las partes interesadas de manera libre.) el trabajo de su estandarización, han aparecido diversas versiones de Javascript con cada vez mayores funcionalidades, estructuras de control, etc. La Versión de la ECMA que nos interesa es la ES6. Para entender esta nueva versión, debemos tener los conceptos claros de JS5.

**JavaScript**

Para empezar a escribir ó programar en Js en una página web, se escribe dentro de un documento Html, y debe ir dentro de las etiquetas <script>y</script>. Los comandos pueden ir en las secciones <body> y <head>. Aunque también el código js, puede escribirse en un archivo externo al Html, el archivo tiene la extensión .js y este archivo debe ser llamado en la página Html de la siguiente manera:

<script src=”xxxx.js”>

</script>

Las ventajas de tener un archivo externo .js:

* Separa el Html y el código.

* Hace la lectura más fácil del código Html y el js, así como su mantenimiento.

* Archivos js en caché puede acelerar la carga de las páginas web.

Una buena práctica en programación es escribir comentarios (son usados para describir o explicar lo que el código está haciendo). En js un comentario de una línea se escribe con // y para múltiples líneas el comentario se escribe entre:

/* y */

**VARIABLES**

Las variables son contenedores para registrar datos, el valor puede variar a o largo del programa. La variable se declara con: var

Nota: JS ES SENSIBLE A LAS MAYÚSCULAS. Una variable puede ser declarada sin un valor, esto quiere decir que en algún momento del programa recibe un valor ya sea ingresado por el usuario ó por algún cálculo. Cuando se declara sin valor, se dice que es indefinido.

Ejemplo:

Var x=10;

Var X=10; (variable diferente a la x minúscula)

Var y=x; a la variable y se le asigna el valor de x

Para declarar variables debes tener cuidado de no utilizar palabras reservadas de este lenguaje de programación. A continuación se muestran las palabras reservadas:

| Palabras reservadas en JavaScript | | | |
|---|---|---|---|
| abstract | else | instanceof | switch |
| boolean | enum | int | synchronized |
| break | export | interface | this |
| byte | extends | long | throw |
| case | false | native | throws |
| catch | final | new | transient |
| char | finally | null | true |
| class | float | package | try |
| const | for | private | typeof |
| continue | function | protected | var |
| debugger | goto | public | void |
| default | if | return | volatile |
| delete | implements | short | while |
| do | import | static | with |
| double | in | super | |

**TIPOS DE DATOS**

Son lo tipos de valores que puede trabajar un programa. Las variables de js pueden guardar muchos tipos de datos, como: números, cadena de texto, arreglos, etc.

Los números en js pueden ser enteros ó decimales:

Var x=10;

Var y=55.55;

Las cadenas de texto pueden ser cualquier texto que aparece entre comillas, pueden ser simples o dobles:

Var nombre=’pedro’;

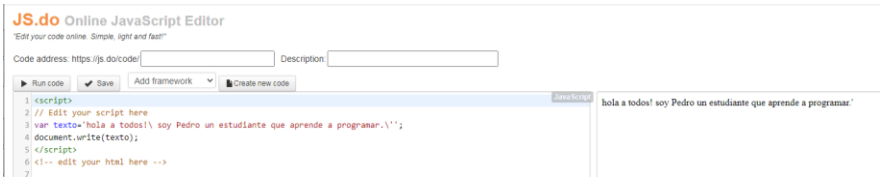
Var text=”mi nombre es pedro pérez”;

**Realice los ejercicios.**

Como puedes observar si utilizo comillas simples debo utilizarla al principio y final; lo mismo sucede con las comillas dobles. Se hace énfasis en esto, porque dentro del texto que se le asigne a una varible podemos insertar comillas o caracteres especiales que necesitamos. Lo hacemos utilizando el carácter de escape barra inversa (\) de la siguiente manera:

Var texto=’hola a todos!\’soy Pedro un estudiante que aprende a programar.\’’;

Como lo vemos a continuación:




Estos caracteres especiales pueden ser añadidos a una cadena de texto, usando \

| Código | Salida |
|---|---|
| \' | comilla simple |
| \" | comillas dobles |
| \\ | barra invertida |
| \n | nueva línea |
| \r | retorno de carro |
| \t | tabulación |
| \b | retroceso |
| \f | salto de página |

El tipo de dato booleano es un tipo de dato lógico, solo tiene dos valores true (verdadero) ó false (falso).

**OPERADORES MATEMÁTICOS**

- Operadores aritméticos:
  Realizan funciones aritméticas. A continuación en la imagen se muestran los operadores aritméticos:

| Operador | Descripción | Ejemplo |
|---|---|---|
| + | Adición | 25 + 5 = 30 |
| - | Substracción | 25 - 5 = 20 |
| * | Multiplicación | 10 * 20 = 200 |
| / | División | 20 / 2 = 10 |
| % | Módulo | 56 % 3 = 2 |
| ++ | Incremento | var a = 10; a++; Ahora a = 11 |
| -- | Decremento | var a = 10; a--; Ahora a = 9 |

  Ejemplo:
  Var x=10+5;
  Puedes colocar la cantidad de números que quieras o variables juntas:

Var x=10;
Var y=x+5+22+1000+50;
Var z=x*2;
Var y=y/4;
Var d=26%6; el operador módulo (%) retoma lo que sobre de la división. En este caso el número 2.

**Realice los ejercicios.**

++ incremento y – decremento: aumentan o disminuyen el valor en uno. Si el operador se coloca antes del operando, retorna el valor incrementado en uno ó disminuye el valor en uno. Si se coloca después del operando, retorna el valor original y luego incrementa el operando ó decrementa el operando (dependiendo del operador que utilices).
Ejemplos:

| Operador | Descripción | Ejemplo | Resultado |
|---|---|---|---|
| var++ | Post Incremento | var a = 0, b = 10;<br>var a = **b++**; | a = 10 y b = 11 |
| ++var | Pre Incremento | var a = 0, b = 10;<br>var a = **++b**; | a = 11 y b = 11 |
| var-- | Post Decremento | var a = 0, b = 10;<br>var a = **b--**; | a = 10 y b = 9 |
| --var | Pre Decremento | var a = 0, b = 10;<br>var a = **--b**; | a = 9 y b = 9 |

- Operadores de asignación
Son aquellos que asignan valores a las variables, como lo muestra la siguiente imagen:

| Operador | Ejemplo | Es equivalente a |
|---|---|---|
| = | x = y | x = y |
| += | x += y | x = x + y |
| -= | x -= y | x = x -y |
| *= | x *= y | x= x * y |
| /= | x /= y | x = x / y |
| %= | x %= y | x = x % y |

Ten en cuenta que puedes colocar varios operadores en una línea, como ejemplo:
x-=y+=9 que equivale a: x=x-(y+9)

**Realice los ejercicios.**

Operadores de comparación
Se utilizan en declaraciones lógicas para determinar igualdad o diferencia entre variables o valores. Retornan como resultado: verdadero o falso.
Por ejemplo:
Igual que (==): este operador válida si los valores son iguales.
4==10 retorna false
**Realice los ejercicios .**

A continuación la tabla de los operadores de comparación:

| Operador | Descripción | Ejemplo |
|---|---|---|
| == | Igual que | 5 == 10 falso |
| === | Idéntico (igual y del mismo tipo) | 5 === 10 falso |
| != | No igual que | 5 != 10 verdadero |
| !== | No idéntico | 10 !== 10 falso |
| > | Mayor que | 10 > 5 verdadero |
| >= | Mayor o igual que | 10 >= 5 verdadero |
| < | Menor que | 10 < 5 falso |
| <= | Menor o igual que | 10 <= 5 falso |

Nota: debes tener presente que las variables sean del mismo tipo de dato; es decir se compara números con números, cadena de texto con cadena de textos, etc.

Operadores lógicos o booleanos
Son operadores que evalúan expresiones y retornan true(verdadero) ó false (falso). Estos operadores son: Y (&&), O (II), NO (!)

| Operadores lógicos | |
|---|---|
| && | Retorna verdadero, si ambos operandos son verdaderos |
| \|\| | Retorna verdadero, si uno de los operandos es verdadero |
| ! | Retorna verdadero, si el operando es falso, y falso, si es verdadero |

Ejemplo:
(4>2)&&(10<15) como las dos condiciones son verdaderas, el resultado es true.

**Realice los ejercicios .**

**Revise el link de code y hacer los ejercicios - Curso C**

Operadores de cadena de texto
El operador más útil es el de concatenación, que se representa con el signo (+). Se utiliza para unir o juntar cadenas de textos, ó cadena de textos con otros tipos de datos.
Ejemplo 1:
var texto1=”Mi nombre es Pedro Pérez ”
var texto2=”aprendo  JavaScript”
document.write(texto1 + texto2);

El resultado es: Mi nombre es Pedro Perez aprendo JavaScript

Ejemplo 2:
var x=”50”;
var y=”200”;
document.write(x+y);

El resultado será: 50200
Nota: los números entre comillas son tratados como texto y no como números.
**Realice los ejercicios .**

CONDICIONALES
If: se utiliza para especificar un bloque de código que será ejecutado si una condición especifica es verdadera.
La estructura del if es la siguiente:

```
if (condition) {
   statements
}
```

Ejemplo:

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 < myNum2) {
   alert("JavaScript is easy to
learn.");
}
```

En el ejemplo anterior la palabra alert(), se utiliza para crear una ventana de alerta emergente que contiene la información que está entre los parentesís.
Cuando una condición es falsa, no se ejecuta nada de lo que esta dentro de los {} del if, se salta y continúa el código que está después del corchete de cierre del if. Ejemplo de una condición falsa:

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
   alert("JavaScript is easy to
learn.");
}
```

else: se utiliza para ejecutar una sentencia de código que se ejecuta si la condición del if es falsa. La estructura es la siguiente:

```
if (expression) {
   // executed if condition is
true
}
else {
   // executed if condition is
false
}
```

Ejemplo:

```
var myNum1 = 7;
var myNum2 = 10;
if (myNum1 > myNum2) {
   alert("This is my first
condition");
}
else {
   alert("This is my second
condition");
}
```

En el ejemplo anterior, se imprime la segunda condición ya que 7 no es mayor quen10.
Else if: se utiliza esta para especificar una nueva condición si la primera condición es falsa.
Ejemplo:

```
var course = 1;
if (course == 1) {
   document.write("<h1>HTML
Tutorial</h1>");
} else if (course == 2) {
   document.write("<h1>CSS
Tutorial</h1>");
} else {
   document.write("<h1>JavaScript
Tutorial</h1>");
}
```

Cambiamos ahora el valor de la variable course a: 3
Cual crees que será el resultado?
**Realice los ejercicios .**

Switch
Se utiliza cuando necesitas evaluar multiples condiciones y con base a esto ejecutar diferentes acciones.
Estructura:

```
switch (expression) {
  case n1:
     statements
     break;
  case n2:
     statements
     break;
  default:
     statements
}
```

- En la estructura o sintaxis del switch, viene acompañado de un break, este rompe la secuencia del bloque apenas encuentre la respuesta a la condición; es decir no continúa evaluando las demás opciones. Y el default, el cual especifica el código a ejecutar si no hay ninguna coincidencia.(la palabra default se puede omitir si usted considera que el código si tendrá coincidencia).
- La condición switch es evaluada un vez. El valor de la expresión es comparado con los valores de cada caso (case). Si se encuentra una coincidencia, el bloque asociado de código es ejecutado.
- Ejemplo:

```
var day = 2;
switch (day) {
  case 1:
    document.write("Monday");
    break;
  case 2:
    document.write("Tuesday");
    break;
  case 3:
    document.write("Wednesday");
    break;
  default:
    document.write("Another
day");
}

// Outputs "Tuesday"
```

**Realice los ejercicios .**

Los Bucles
Estos se utilizan cuando se quiere ejecutar varias veces el mismo código mó sentencia.
En js se manejan el for, while y do while.
For
Estructura

```
for (statement 1; statement 2;
statement 3) {
   code block to be executed
}
```

Según la imagen anterior:
Stament1 (declaración 1): es ejecutada antes de que el bucle comience.
Stament2 (declaración 2): determina la condición para ejecutar el bucle.
Stament3 (declaración 3): es ejecutada cada vez y después que el bucle ha sido ejecutado.
Ejemplo de un for que imprime los primeros 5 numeros (los números del 1 al 5).

```
for (i=1; i<=5; i++) {
   document.write(i + "<br />");
}
```

En el ejemplo anterior lo que quiere decir es que la variable i empieza desde el numero 1, luego se pregunta si es menor ó igual a 5, después incrementa el valor de i y ejecuta el código que está dentro del bucle for. Cuando regresa nuevamente al bucle i vale 2, repite el mproceso anterior hasta que i sea mayor que 5; de esta manera el bucle se rompe o deja de repetirse.

**Realice los ejercicios .**

While
Mientra la condición sea verdadera se repite el código.
Estructura:

```
while (condition) {
   code block
}
```

Ejemplo:

```
var i=0;
while (i<=10) {
   document.write(i + "<br />");
   i++;
}
```

El bucle se repite mientras que i sea menor ó igual que 10. Por ende el ejemplo anterior imprimirá en pantalla los números del 1 al 10. Si olvidas incrementar la variable utilizada en la condición, el bucle nunca terminará, será eterno.
**Realice los ejercicios .**

Do while
Este bucle es una variante del bucle while. Este bucle ejecuta el código una vez antes de evaluar si la condición es verdadera, y entonces el bucle se repetirá mientras esa condición sea verdadera.

Estructura:

```
do {
   code block
}
while (condition);
```

Ejemplo:

```
var i=20;
do {
  document.write(i + "<br />");
  i++;
}
while (i<=25);
```

**Realice los ejercicios .**

Diferencia entre el break y el continue

El break rompe el bucle, es decir sale ó finaliza el bucle
Ejemplo:

```
for (i = 0; i <= 10; i++) {
   if (i == 5) {
     break;
   }
   document.write(i + "<br />");
}
```

El continue detiene solo una iteración del bucle y el bucle continua.
Ejemplo:

```
for (i = 0; i <= 10; i++) {
   if (i == 5) {
      continue;
   }
   document.write(i + "<br />");
}
```

**Realice los ejercicios .**

**Revise el link de code y hacer los ejercicios - Curso D**

FUNCIONES
Una función es un bloque de código para ejecutar una tarea especifica. La ventaja de usar funciones es que se puede:
Reusar el código: definir el código una vez y usarlos muchas veces.
Para definir una función en js, se utiliza la palabra clave function, luego el nombre (pueden tener letras, dígitos, guiones bajos, y signos de dólar) y los paréntesis, el código que será ejecutado va dentro de corchetes.
Estructura:

```
function name() {
  //code to be executed
}
```

Para llamar una función simplemente colocamos el nombre, seguido de los paréntesis y el punto y coma.

```
myFunction();
//Alerts "Calling a Function!"
```

Las funciones pueden tener parámetros, van dentro de los paréntesis de las funciones y se les debe dar nombres. Si son varios parámetros, van separados por comas:

```
functionName(param1, param2,
param3) {
   // some code
}
```

Ejemplo:

```
function sayHello(name) {
   alert("Hi, " + name);
}

sayHello("David");
//Alerts "Hi, David"
```

```
function sayHello(name, age) {
  document.write( name + " is " +
age + " years old.");
}
```

Las funciones pueden tener la delaracion return. Esta se utiliza para retornar un valor desde una función. Es útil cuando hacemos cálculos que requieren un resultado. Nota: cuando js llega hasta return, la función detiene su ejecución.

Ejemplo:

```
function myFunction(a, b) {
   return a * b;
}

var x = myFunction(5, 6);
// Return value will end up in x
// x equals 30
```

**Realice los ejercicios .**

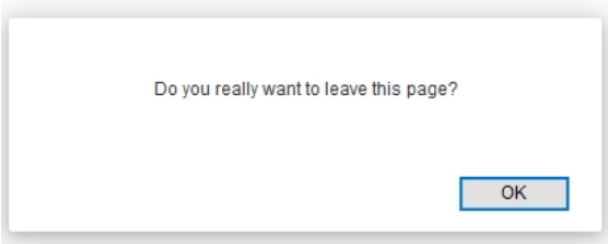
VENTANAS ALERT, PROMPT, CONFIRM

Js maneja tres tipos de ventanas:

Alert: ventana de alerta. Ventana utilizada cuando quieres que la información llegue al usuario. En esta ventana el usuario debe dar clic en “ok” para continuar.

Ejemplo:

```
alert("Do you really want to leave
this page?");
```




Prompt: ventana de solicitud. Utilizada para pedir información al ususario y por ende él ingresa un valor.

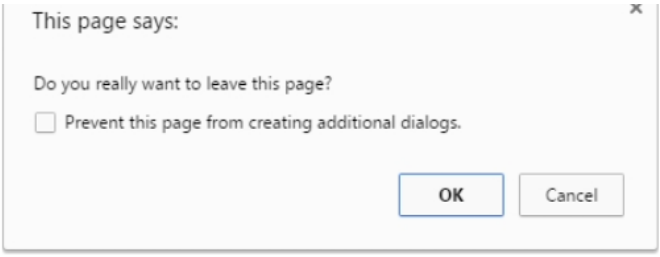
Ejemplo:

```
var user = prompt("Please enter
your name");
alert(user);
```


Confirm: ventana de confirmación. Se utiliza para permitir al usuario validar o aceptar algo. El usuario puede dar ok o cancelar para continuar.si le da ok el valor retornado es verdadero y si cancela retornará falso.

Ejemplo:

```
var result = confirm("Do you
really want to leave this
page?");
if (result == true) {
  alert("Thanks for visiting");
}
else {
  alert("Thanks for staying with
us");
}
```




**Realice los ejercicios .**

EMACAcrips6 (ES6)

Esta sexta edición añade una cantidad de sintaxis nueva, considerable para escribir aplicaciones complejas.

Variables y strings

En ES 6 se tienen 3 formas de declarar variables:

```
var a = 10;
const b = 'hello';
let c = true;
```

Nota: el tipo de declaración que se use depende del ámbito (scope). El ámbito es el que define la visibilidad de una variable.

Var y let

Var: define una variable global o local a toda una función independientemente del ámbito ó scope del bloque.

Let: te permite declarar variables limitadas al bloque, a la declaración o a la expresión en la que se utilicen.

- Ejemplo:
- Diferencia entre var y let

```
function varTest() {
  var x = 1;
  if (true) {
    var x = 2;  // same variable
    console.log(x);  // 2
  }
  console.log(x);  // 2
}

function letTest() {
  let x = 1;
  if (true) {
    let x = 2;  // different
variable
    console.log(x);  // 2
  }
  console.log(x);  // 1
}
```

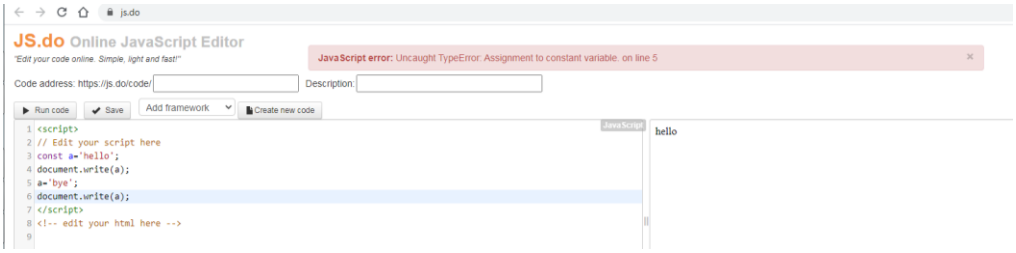
- Ejemplo del uso de let en bucles:

```
for (let i = 0; i < 3; i++) {
  document.write(i);
}
```

**Realice los ejercicios .**

Const

Tiene el mismo concepto de let, solo se diferencia es que son inmutables; es decir, no perimiten que sean reasignadas.




Template literals

Se utilizan para devolver variables en la cadena. Los template literals van encerrados por el acento inverso(``), y no por comillas simples o dobles. La ${expresion} puede incluir cualquier expresión, que será evaluada e insertada en la template literals.

Ejemplos:

```
let name = 'David';
let msg = `Welcome ${name}!`;
console.log(msg);
```

```
let a = 8;
let b = 34;
let msg = `The sum is ${a+b}`;
console.log(msg);
```

**Realice los ejercicios .**

- Bucles y funciones

  For…of

  Ejemplo:

  En el siguiente ejemplo a la variable val se le asigna el elemento correspondiente de la lista.

```
let list = ["x", "y", "z"];
for (let val of list) {
  console.log(val);
}
```

  Funciones

  Puedes no tener que escribir function y return, asi como algunos paréntesis y llaves.

  Antes:

```
function add(x, y) {
  var sum = x+y;
  console.log(sum);
}
```

Ahora con ES6:

```
const add = (x, y) => {
  let sum = x + y;
  console.log(sum);
}
```

Ahora también se pueden colocar parámetros predeterminados en las funciones.
Ejemplo:

```
const test = (a, b = 3, c = 42) =>
{
  return a + b + c;
}
console.log(test(5)); //50
```

**Realice los ejercicios**

- Objetos

```
let tree = {
  height: 10,
  color: 'green',
  grow() {
    this.height += 2;
  }
};
tree.grow();
console.log(tree.height); // 12
```

```
let height = 5;
let health = 100;

let athlete = {
  height,
  health
};
```

Nombres de propiedad calculados: usando [], podemos usar una expresión para un nombre de propiedad, incluyendo cadenas que se concatenan.
Ejemplos:

```
let prop = 'name';
let id = '1234';
let mobile = '08923';

let user = {
  [prop]: 'Jack',
  [`user_${id}`]: `${mobile}`
};
```

```
var i = 0;
var a = {
  ['foo' + ++i]: i,
  ['foo' + ++i]: i,
  ['foo' + ++i]: i
};
```

```
var param = 'size';
var config = {
  [param]: 12,
  ['mobile' +
param.charAt(0).toUpperCase() +
param.slice(1)]: 4
};
console.log(config);
```

**Realice los ejercicios .**

Object.assign()
Permite crear un duplicado de un objeto existente. Además permite combinar multiples fuentes en un destino para crear un nuevo objeto.

Ejemplo:
En el ejemplo el primer parámetro del object.assign es el objeto destino y los otros dos son las fuentes.

```
let person = {
  name: 'Jack',
  age: 18,
  sex: 'male'
};
let student = {
  name: 'Bob',
  age: 20,
  xp: '2'
};
let newStudent = Object.assign({},
person, student);
```

Ejemplo2:
En el siguiente ejemplo se crea un duplicado de un objeto

```
let person = {
  name: 'Jack',
  age: 18
};

let newPerson = Object.assign({},
person);
newPerson.name = 'Bob';

console.log(person.name); // Jack
console.log(newPerson.name); //
Bob
```

Ejemplo3:
En el ejemplo se muestra como asignar un valor a una propiedad de objeto en la declaración.

```
let person = {
  name: 'Jack',
  age: 18
};

let newPerson = Object.assign({},
person, {name: 'Bob'});
```

**Realice los ejercicios .**

- Desestructuración

Se utiliza para descomponer un array en diferentes variables.
Ejemplo:

```
let arr = ['1', '2', '3'];
let [one, two, three] = arr;

console.log(one); // 1
console.log(two); // 2
console.log(three); // 3
```

Descomponer un array devuelto por una función:

```
let a = () => {
  return [1, 3, 2];
};

let [one, , two] = a();
```

Ahora ejemplos descomponiendo objetos

```
let obj = {h:100, s: true};
let {h, s} = obj;

console.log(h); // 100
console.log(s); // true
```

```
let a, b;
({a, b} = {a: 'Hello ', b:
'Jack'});

console.log(a + b); // Hello Jack
```

Ahora sin utilizar los paréntesis:

```
let {a, b} = {a: 'Hello ', b:
'Jack'};
console.log(a + b);
```

También puedes asignar el objeto a nuevos nombres de variables.
**Por ejemplo:**

```
var o = {h: 42, s: true};
var {h: foo, s: bar} = o;

//console.log(h); // Error
console.log(foo); // 42
```

**Realice los ejercicios .**

Por último, puedes asignar valores predeterminados a variables, en caso de que el valor extraído del objeto sea indefinido.
**Por ejemplo:**

```
var obj = {id: 42, name: "Jack"};

let {id = 10, age = 20} = obj;

console.log(id); // 42
console.log(age); // 20
```

**Realice los ejercicios .**

- Parámetros Rest y Spread
  Rest
  Usando un parámetro rest se puede pasar una cantidad de parámetros variables en una función.
  Ejemplo:

```
function containsAll(arr, ...nums)
{
  for (let num of nums) {
    if (arr.indexOf(num) === -1)
{
      return false;
    }
  }
  return true;
}
```

En el ejemplo anterior la variable nums es el parámetro rest, el cogerá todos los argumentos adicionales pasados en la función.
Nota: tener en cuenta que solo el ultimo parámetro de la función puede ser marcado como un parámetro rest.

**Realice los ejercicios .**

Spread
es parecido al operador rest, pero tiene otra finalidad utilizándolo en objetos, arrays ó llamadas a funciones (argumetos).

Spread en funciones:
Antes

```
function myFunction(w, x, y, z) {
  console.log(w + x + y + z);
}
var args = [1, 2, 3];
myFunction.apply(null,
args.concat(4));
```

Después

```
const myFunction = (w, x, y, z) =>
{
  console.log(w + x + y + z);
};
let args = [1, 2, 3];
myFunction(...args, 4);
```

Ejemplo:

```
var dateFields = [1970, 0, 1];  //
1 Jan 1970
var date = new
Date(...dateFields);
console.log(date);
```

**Realice los ejercicios .**

Spread en literals de arrays:
Antes:

```
var arr = ["One", "Two", "Five"];

arr.splice(2, 0, "Three");
arr.splice(3, 0, "Four");
console.log(arr);
```

Después:

```
let newArr = ['Three', 'Four'];
let arr = ['One',
'Two', ...newArr, 'Five'];
console.log(arr);
```

Spread en literals de objetos
Copia las propiedades del objeto proporcionado y los traslada a un objeto nuevo.
Ejemplo:

```
const obj1 = { foo: 'bar', x:
42 };
const obj2 = { foo: 'baz', y:
5 };

const clonedObj = {...obj1}; //
{ foo: "bar", x: 42 }
const mergedObj =
{...obj1, ...obj2}; // { foo:
"baz", x: 42, y: 5 }
```

No puedes combinarlos porque no obtendrás el resultado esperado.
Ejemplo:

```
const obj1 = { foo: 'bar', x:
42 };
const obj2 = { foo: 'baz', y:
5 };
const merge = (...objects) =>
({...objects});

let mergedObj = merge(obj1,
obj2);
// { 0: { foo: 'bar', x: 42 }, 1:
{ foo: 'baz', y: 5 } }

let mergedObj2 = merge({}, obj1,
obj2);
// { 0: {}, 1: { foo: 'bar', x:
42 }, 2: { foo: 'baz', y: 5 } }
```

Nota: para combinar o clonar objetos, usa el Objectassign().

**Realice los ejercicios .**

- Clases

  Una clase es utilizada para crear multiples objetos. Se utiliza la palabra class y se caracteriza por tener un método, llamado constructor para que se pueda inicializar la clase. Se debe tener presente que sólo puede haber un constructor en cada clase.

  Ejemplo:

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
}
```

  Ahora la clase del ejemplo anterior se utilizará para crear otros objetos, para ello utilizamos la palabra new:

```
const square = new Rectangle(5,
5);
const poster = new Rectangle(2,
3);
```

  Nota: puedes definir una clase con una expresión de clase. Puede estar o no nombrada.

  Ejemplo cuando esta nombrada:

```
var Square = class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
```

Ejemplo cuando no está nombrada:

```
var Square = class {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
};
```

**Realice los ejercicios .**

Métodos de clase
Ejemplo:
En el siguiente ejemplo el área es un getter y calcArea es un método.

```
class Rectangle {
  constructor(height, width) {
    this.height = height;
    this.width = width;
  }
  get area() {
    return this.calcArea();
  }
  calcArea() {
    return this.height *
this.width;
  }
}
const square = new Rectangle(5,
5);
console.log(square.area); // 25
```

También esta el método static, estos se utilizan para crear funciones de utilidad para una aplicación.
Ejemplo:

```
class Point {
  constructor(x, y) {
    this.x = x;
    this.y = y;
  }
  static distance(a, b) {
    const dx = a.x - b.x;
    const dy = a.y - b.y;
    return Math.hypot(dx, dy);
  }
}
const p1 = new Point(7, 2);
const p2 = new Point(3, 8);

console.log(Point.distance(p1,
p2));
```

**Realice los ejercicios .**

**Herencia**

Utilizamos la palabra extends para crear un hijo de una clase o clase secundaria. Este hijo, hereda todas las propiedades y métodos del padre.

Ejemplo:

En el siguiente ejemplo la clase Dog es hijo de  la clase Animal, por ende hereda sus propiedades y métodos.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(this.name + '
makes a noise.');
  }
}
class Dog extends Animal {
  speak() {
    console.log(this.name + '
barks.');
  }
}

let dog = new Dog('Rex');
dog.speak();
```

Para utilizar los métodos del padre se utiliza la palabra super.

```
class Animal {
  constructor(name) {
    this.name = name;
  }
  speak() {
    console.log(this.name + '
makes a noise.');
  }
}
}

class Dog extends Animal {
  speak() {
    super.speak(); // Super
    console.log(this.name + '
barks.');
  }
}

let dog = new Dog('Rex');
dog.speak();
```

**Realice los ejercicios .**

- Map y Set
  Objeto map
  Es utilizado para contener pares key/value (clave/valor), la cual puede ser objetos y valores primitivos.

  new Map([iterable]): crea un objeto Map y en el iterable es una array o cualquier otro objeto cuyo elementos son arrays (cada uno de ellos con un par clave/valor).
  un objeto es similar a un Map, pero existen casos en que es mejor utilizar Map, como:
  1. Las key pueden ser de cualquier tipo, incluye funciones, objetos, etc.
  2. Se puede obtener el tamaño de un Map (la propiedad size, devuelve la cantidad de pares clave/valor).
  3. Se puede iterar directamente sobre él.
  4. El rendimiento de un Map es mejor cuando frecuentemente se adiciona o elimina pares (clave/valor)

  Ejemplo:

```
let map = new Map([['k1', 'v1'],
['k2', 'v2']]);

console.log(map.size); // 2
```

**Realice los ejercicios .**

Métodos que se utilizan, definición y ejemplo:

**set(key, value)** Añade un par clave/valor especificado al map. Si la clave especificada ya existe, su valor correspondiente es reemplazado por el valor especificado.
**get(key)** Obtene el valor correspondiente de una clave determinada del map. Si la clave especificada no existe, se devuelve undefined.
**has(key)** Devuelve true (verdadero) si existe una clave especificada en el map o false en caso conrario.
**delete(key)** Elimina el paor clave/valor con una clave especificada del map y devuelve true. Devuelve false si el elemento no existe.
**clear()** Elimina todos los pares clave/valor de un map.
**keys()** Devuelve un iterador de claves en el map para cada elemento.
**values()** Devuelve un iterador de valores en el map para cada elemento.
**entries()** Deuele un iterador de array[clave, valor] el map para cada elemento.

```
let map = new Map();

map.set('k1', 'v1').set('k2',
'v2');

console.log(map.get('k1')); // v1

console.log(map.has('k2')); //
true

for (let kv of map.entries())
  console.log(kv[0] + " : " +
kv[1]);
```

**Realice los ejercicios .**

Objeto set
Se utiliza para tener valores únicos (no se permiten repeticiones), las claves en el objeto Set pueden ser objetos y valores primitivos.

Sintaxis:
new Set([iterable])
con la anterior sintaxis, se crea el objeto set, en el iterable va un array o cualquier otro objeto con valores.
La propiedad size devuelve la cantidad de valores diferentes de un set.
Ejemplo:

```
let set = new Set([1, 2, 4, 2, 59,
9, 4, 9, 1]);

console.log(set.size); // 5
```

**Realice los ejercicios .**

Métodos, definición y ejemplo:

**add(value)** Añade al set un nuevo elemento con el valor dado.
**delete(value)** Borra un valor especificado del set.
**has(value)** Devuelve true (verdadero) si existe un valor especificado en el set o false en caso conrario.
**clear()** Borra el set.
**values()** Devuelve un iterador de valores en el set.

```
let set = new Set();

set.add(5).add(9).add(59).add(9);

console.log(set.has(9));

for (let v of set.values())
  console.log(v);
```

**Realice los ejercicios .**

- Promises

Es una mejora para la programación asíncrona.
Ejemplo:

```
new Promise(function(resolve,
reject) {
    // Work
    if (success)
        resolve(result);
    else
        reject(Error("failure"));
});
```

Si un método devuelve una promise, sus llamadas debe usar el método then, el cual toma dos métodos como entrada: uno para el éxito y otro para el fracaso.

resolve: método para el éxito
reject: método para el fracaso

```
function asyncFunc(work) {
  return new
Promise(function(resolve, reject)
{
    if (work === "")
      reject(Error("Nothing"));
    setTimeout(function() {
      resolve(work);
    }, 1000);
  });
}

asyncFunc("Work 1") // Task 1
.then(function(result) {
  console.log(result);
  return asyncFunc("Work 2"); //
Task 2
}, function(error) {
  console.log(error);
})
.then(function(result) {
  console.log(result);
}, function(error) {
  console.log(error);
});
console.log("End");
```

**Realice los ejercicios .**

- Iteradores y generadores
  Symbol.iterator: es el iterador predeterminado para un objeto.
  Ejemplo:

```
let myIterableObj = {
  [Symbol.iterator] : function* ()
{
    yield 1; yield 2; yield 3;
...
console.log([...myIterableObj]);
```

En el ejemplo anterior se crea un objeto y se usa Symbol.iterator junto con la función generadora para rellenarlo con valores. El gen function ó función generadora es la palabra function junto el *.

Ejemplo de cómo es utilizado un una function:

```
function* idMaker() {
  let index = 0;
  while (index < 5)
    yield index++;
}
var gen = idMaker();
console.log(gen.next().value);
```

Se pueden jerarquizar funciones generadoras unas dentro de otras para crear estructuras mas complejas y pasarles argumentos mientras las estamos llamando.

Ejemplo:
En el siguiente ejemplo creamos un objeto con 7 elementos usando Symbol.iterator y funciones generadoras. Luego asignamos el objeto a la constante all; y al final se imprime el valor.

```
const arr = ['0', '1', '4', 'a',
'9', 'c', '16'];
const my_obj = {
  [Symbol.iterator]: function*()
{
    for(let index of arr) {
      yield `${index}`;
    }
  }
};
const all = [...my_obj]
  .map(i => parseInt(i, 10))
  .map(Math.sqrt)
  .filter((i) => i < 5)
  .reduce((i, d) => i + d);
console.log(all);
```

Nota: se pueden salir y volver a entrar a las funciones generadoras. Son una herramienta poderosa en la programación asíncrona y se puede combinar con promise.

**Realice los ejercicios .**

- Módulos
  En js se trabajaba los modulos a través de unas bibliotecas como RequireJS, CommonJS. En ES6 soporta los módulos de forma nativa.

  Los módulos son independientes, por ende es posible las mejoras y la ampliación sin dependencia de código en otros módulos.
  Son reutilizables, al poderse utilizar en otro proyecto sin tener que reescribirlo.
  Ejemplo:

```
// lib/math.js
export let sum = (x, y) =>
{ return x + y; }
export let pi = 3.14;

// app.js
import * as math from "lib/math"
console.log(`2p = + $
{math.sum(math.pi, math.pi)}`)
```

En HTML se agrega la siguiente propiedad al script
<script type="module" src="./js/app.js"></script>

- Métodos integrados
  ES6 introdujo nuevos métodos integrados:
  1. Búsqueda de elemeto de arrays:
     Antes

```
[4, 5, 1, 8, 2, 0].filter(function
(x) {
  return x > 3;
})[0];
```

     Ahora

```
[4, 5, 1, 8, 2, 0].find(x => x >
3);
```

     Para obtener el índice del elemento:

```
[4, 5, 1, 8, 2, 0].findIndex(x =>
x > 3);
```

  2. Repeticiones de strings
     Antes

```
console.log(Array(3 +
1).join("foo")); // foofoofoo
```

     Ahora

```
console.log("foo".repeat(3)); //
foofoofoo
```

  3. Búsqueda de strings
     Antes

```
"SoloLearn".indexOf("Solo") ===
0; // true
"SoloLearn".indexOf("Solo") === (4
- "Solo".length); // true
"SoloLearn".indexOf("loLe") !==
-1; // true
"SoloLearn".indexOf("olo", 1) !==
-1; // true
"SoloLearn".indexOf("olo", 2) !==
-1; // false
```

Ahora

```
"SoloLearn".startsWith("Solo",
0); // true
"SoloLearn".endsWith("Solo",
4); // true
"SoloLearn".includes("loLe"); //
true
"SoloLearn".includes("olo", 1); //
true
"SoloLearn".includes("olo", 2); //
false
```

**4. ACTIVIDADES DE EVALUACIÓN**

**Evidencia de Conocimiento:** Realizar los ejercicios en visual studio de cada ejercicio. De esta forma afianzar los conocimientos adquiridos.

**Evidencia de Desempeño:** Evaluación

**Evidencia de Producto:** Entrega de los pantallazos de las simulaciones en un archivo de word y la entrega de cada ejercicio realizado en code.

| Evidencias de Aprendizaje | Criterios de Evaluación | Técnicas e Instrumentos de Evaluación |
|---|---|---|
| **Evidencias de Conocimiento:**<br><br>**Evidencias de Desempeño:** | **Reconoce comandos de javascript y hace uso de estos diferenciando Js y ECMA6.**<br><br>**Realiza uso de los comandos de js y ECMA6** | Taller y evaluación de términos<br><br>Evaluación práctica |

| | | |
|---|---|---|
| **Evidencias de Producto:** | **Realiza el entregable con la documentación completa** | Entrega del taller, evaluación y prueba. |

## 1. GLOSARIO DE TÉRMINOS

De acuerdo a la práctica realizar su propio glosario de términos.

## 6. REFERENTES BIBLIOGRÁFICOS

Aprender a desarrollar con JavaScript (3ª edición) - Christian VIGOUROUX

Construya o cite documentos de apoyo para el desarrollo de la guía, según lo establecido en la guía de desarrollo curricular

## 7. CONTROL DEL DOCUMENTO

| | Nombre | Cargo | Dependencia | Fecha |
|---|---|---|---|---|
| **Autor (es)** | **Néstor Rodríguez** | **Instructor** | **Teleinformática** | **NOVIEMBRE-2020** |

## 8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

| | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
|---|---|---|---|---|---|
| **Autor (es)** | | | | | |