

Estructuras de Control de Flujo

Si solamente trabajamos con operadores y variables, generamos una sucesión de pasos lineales. No podemos decidir si algo se mostrará o no según determinadas circunstancias o en todo caso repetir una instrucción tantas veces.

Para realizar estas secuencias más complejas y con múltiples posibilidades se debe trabajar con estructuras de control de flujo.

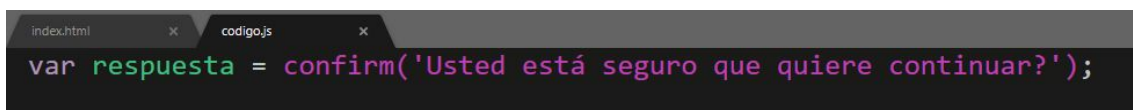
Para poder avanzar, necesitamos recordar algunos tipos de variables y operadores que vamos a utilizar en estas nuevas estructuras.

Tipo Booleano

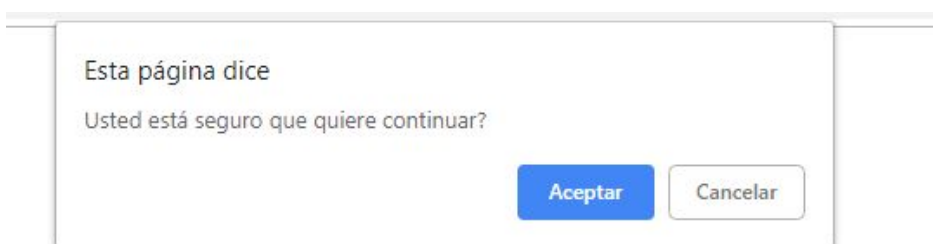
Los tipos de variables de dato booleanos son aquellos conformados únicamente por 2 posibles valores: true/false.

Un booleano puede ser definido en forma explícita, por ejemplo, asignando un "true" a una variable.

Recordemos que por ejemplo, si generamos una ventana de confirmación estamos esperando que el resultado según el usuario acepte o no , es verdadero/falso,

A screenshot of a web browser window with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing a line of JavaScript code: `var respuesta = confirm('Usted está seguro que quiere continuar?');` The code is syntax-highlighted with 'var' in green, 'respuesta' in green, '=' in white, 'confirm' in pink, and the string in quotes in pink.

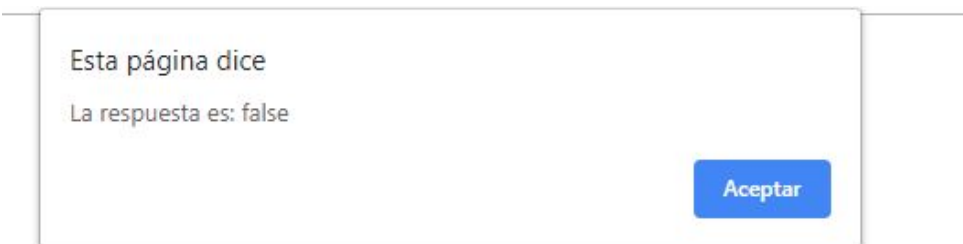
El resultado en la pantalla del navegador será el siguiente,



Si mostramos resultado en una alerta,

```
index.html x código.js x
var respuesta = confirm('Usted está seguro que quiere continuar?');
alert('La respuesta es: ' + respuesta);
```

Si el usuario cancela,



Si por caso, trabajamos con operadores relacionales,

- > (Mayor a)
- < (Menor a)
- >= (Mayor o igual a)
- <= (Menor o igual a)
- == ("Igual a" ⇒ Utilizado para verificar si 2 valores coinciden)
- != ("Distinto de" ⇒ Utilizado para verificar si 2 valores NO coinciden)
- === ("Idéntico a" ⇒ Utilizado para verificar si 2 valores coinciden y son del mismo tipo de dato)

El resultado, también nos devolverá true/false, por ejemplo,

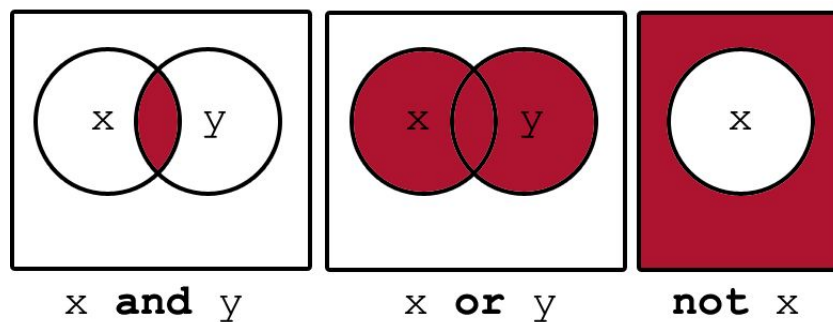
```
index.html x código.js x
var numero1 = 10;
var numero2 = 20;
var respuesta = numero1 < numero2;
alert(respuesta);
```

No devolverá el siguiente resultado en el navegador,

Esta página dice
true

Aceptar

Esto es , porque obviamente, 10 es menor a 20. Por otro lado, si trabajamos con operadores lógicos (&& / || / !), nos presentan también situaciones donde los resultados serán true/false.



Para mayores detalles sobre este tipo de variable, y estos operadores les recomendamos ver el módulo anterior.

Estructuras de Control Condicionales

Son aquellas que nos permiten tomar decisiones acerca de qué sentencias (órdenes) debe ejecutar el programa, en base a una condición / expresión booleana, es decir si algo es true/false.

Se puede decir que a diferencia de los programas que veníamos generando, en este caso según el dato de entrada es que se tomará una serie de pasos u otra, por lo tanto el resultado o dato de salida dependerá del dato de entrada.

Estructura if

El condicional if nos permite tomar una decisión en base a una condición.

Cuando la condición sea evaluada como verdadera, se ejecutará un bloque con un conjunto de sentencias asociado al if.

Hagamos el siguiente pseudocódigo:

```
Inicio
Entero: Edad
Tomar a través de ventana de usuario la Edad
Si (Edad >= 20) entonces
    Mostrar en alerta "Tu tienes 20 o más"
Fin Si
Fin Algoritmo
```

Cómo generamos este pseudocódigo en JS, primero entendamos la sintaxis

A screenshot of a code editor with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing the following JavaScript code:

```
if(condicion){
    resultado
}
```

 The word 'condicion' is highlighted in blue, 'resultado' is highlighted in pink, and the closing curly brace '}' is highlighted in white.

En el caso anterior, se realizará de la siguiente manera,

```
index.html x código.js x
if(edad >= 20){

    alert("Tiene 20 o más años")

}
```

Dado que hemos decidido que el dato de entrada sea a través de una ventana de usuario , lo haremos de la siguiente forma,

```
index.html x código.js
var edad = prompt("Cuando años tiene el usuario?")

if(edad >= 20){

    alert("Tiene 20 o más años")

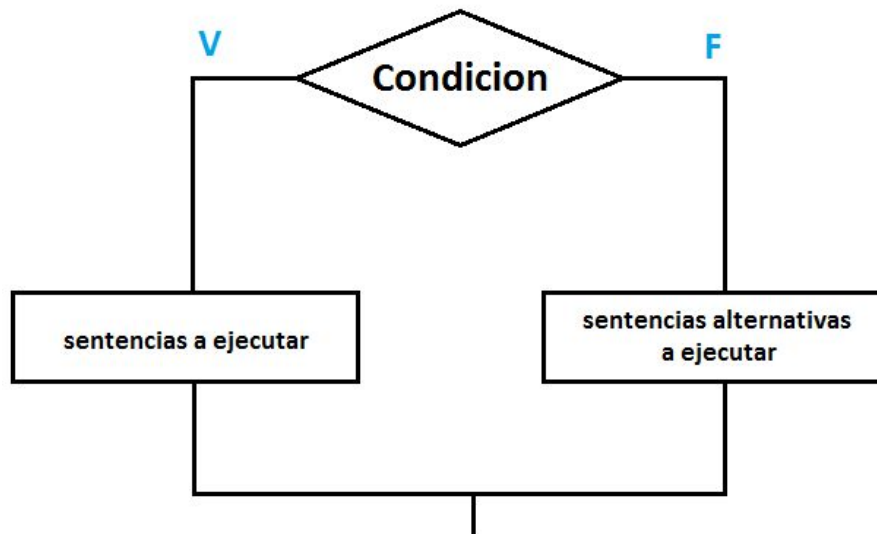
}
```

Como el usuario, efectivamente tiene 20 o más, el resultado en el navegador será el siguiente,



Estructura if...else

En este caso fijamos una alternativa, de esta forma un algoritmo que represente esta situación sería,



En el caso anterior, nuestro pseudocódigo sería,

Inicio

Entero: Edad

Tomar a través de ventana de usuario la Edad

Si(Edad >=20) entonces

Mostrar en alerta "Tu tienes 20 o más"

Sino

Mostrar en alert "Usted tiene menos de 20 años"

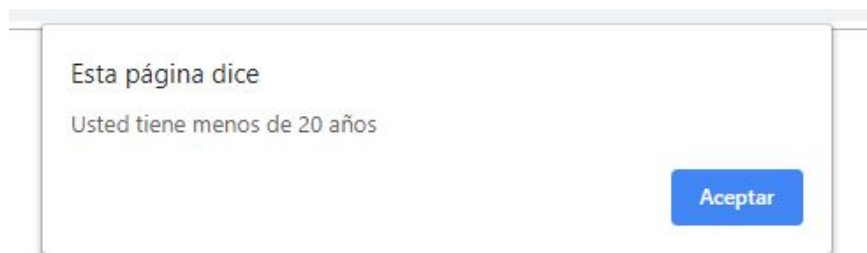
Fin Si

Fin Algoritmo

Si llevamos este pseudocódigo a JS,

```
index.html x  codigo.js x
var edad = prompt("Cuando años tiene el usuario?")
if(edad >= 20)
    {alert("Tiene 20 o más años")}
else
    { alert("Usted tiene menos de 20 años")}
```

El resultado en el caso de no cumplirse la condición será,



Estructura if...elseif...else

La estructura elseif, nos brinda una posibilidad de qué se ejecute una serie de pasos si una opción (no la primera) es verdadera.

Por ejemplo, nuestro pseudocódigo podría ser,

Inicio

Entero: Edad

Tomar a través de ventana de usuario la Edad

Si(Edad >20) entonces

Mostrar en alerta "Usted tiene más de 20 años"

Sino

Si(Edad=20 o Edad > 15)

Mostrar en una alerta "Usted tiene entre 20 y 15 años"

Sino

Mostrar en una alerta "Usted tiene menos de 15 años, adios!"

FinSi

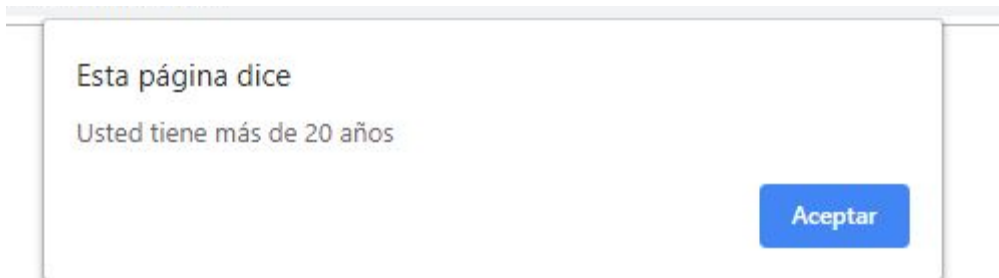
Fin Algoritmo

En nuestro JS sería de la siguiente forma,

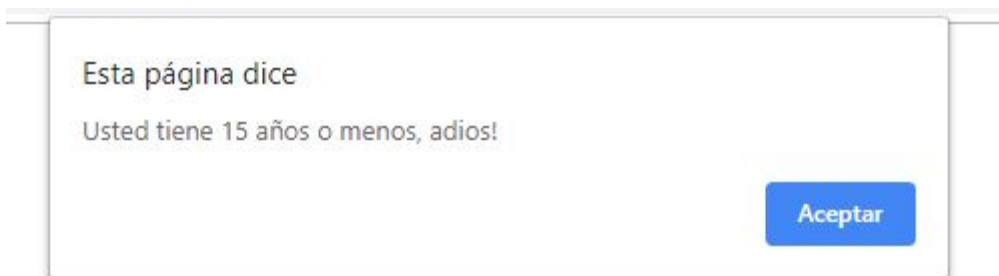
```
codigo.js x index.html x
var edad = prompt("La edad del usuario es?")

if (edad > 20) {
    alert("Usted tiene más de 20 años")
} else if (edad == 20 || edad > 15) {
    alert("Usted tiene entre 20 y 15 años")
} else {
    alert("Usted tiene 15 años o menos, adios!")
}
```

Si por ejemplo el usuario ingresa el valor 25 el resultado será el siguiente,



En el caso de que el usuario ingrese 15 o menos,

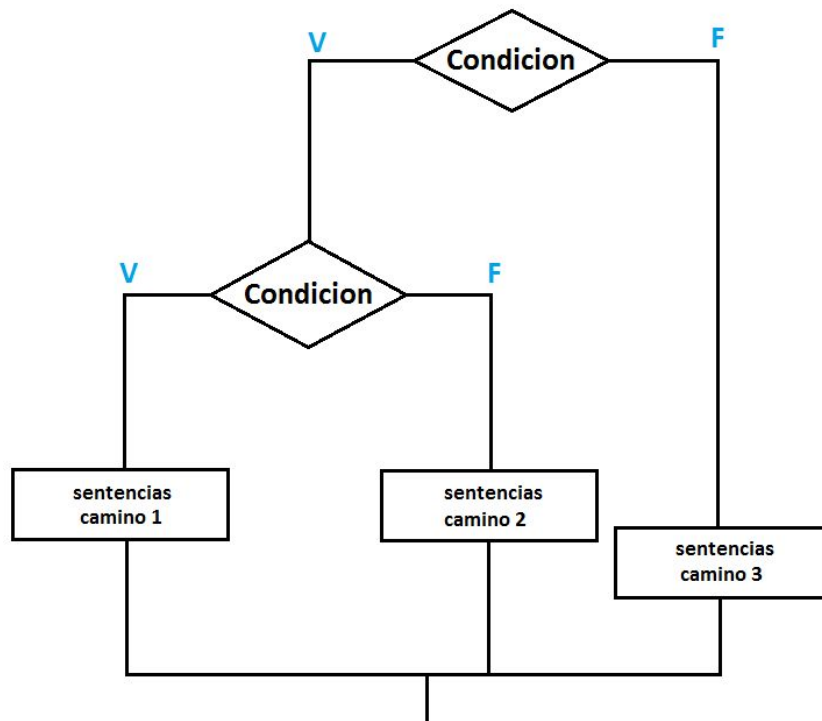


Anidamiento de condicionales if

Un condicional if puede escribirse dentro del bloque de otro if (sea en su camino principal o en su camino alternativo), de manera de generar múltiples caminos de ejecución. Mediante el anidamiento de condicionales el código comienza a “ramificarse”: Se generarán varias bifurcaciones.

Un simple if con su respectivo else nos proveen 2 posibles caminos de ejecución; en cambio, anidando condicionales if podemos generar 3 o más bifurcaciones.

Ejemplo de algoritmo genérico,



En nuestro caso nuestro pseudocódigo,

Tomar a través de ventana de usuario la Edad

Si(Edad >=20) entonces

 Preguntar Nombre a través de ventana de usuario

 Si(Nombre == Pedro)entonces

 Saludar en un alerta a Pedro

 Sino

 Mostrar en alerta "Usted tiene 20 años o más pero no es Pedro"

 FinSi

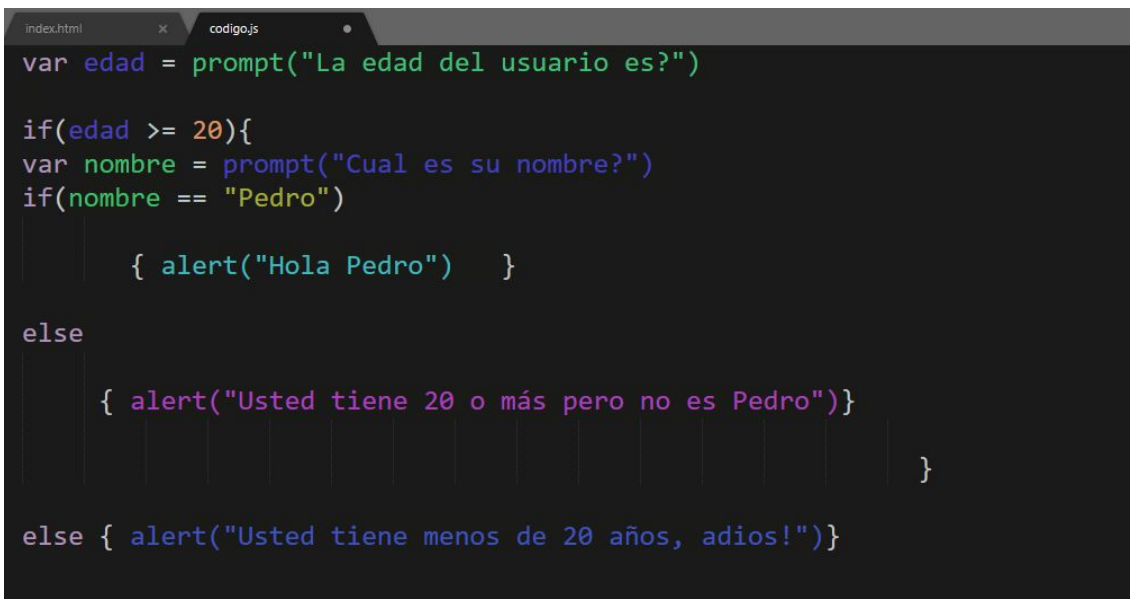
Sino

 Mostrar en una alerta "Usted tiene menos de 20 años, adios!"

FinSi

Fin Algoritmo

Vamos a construir este pseudocódigo en nuestro archivo.js.



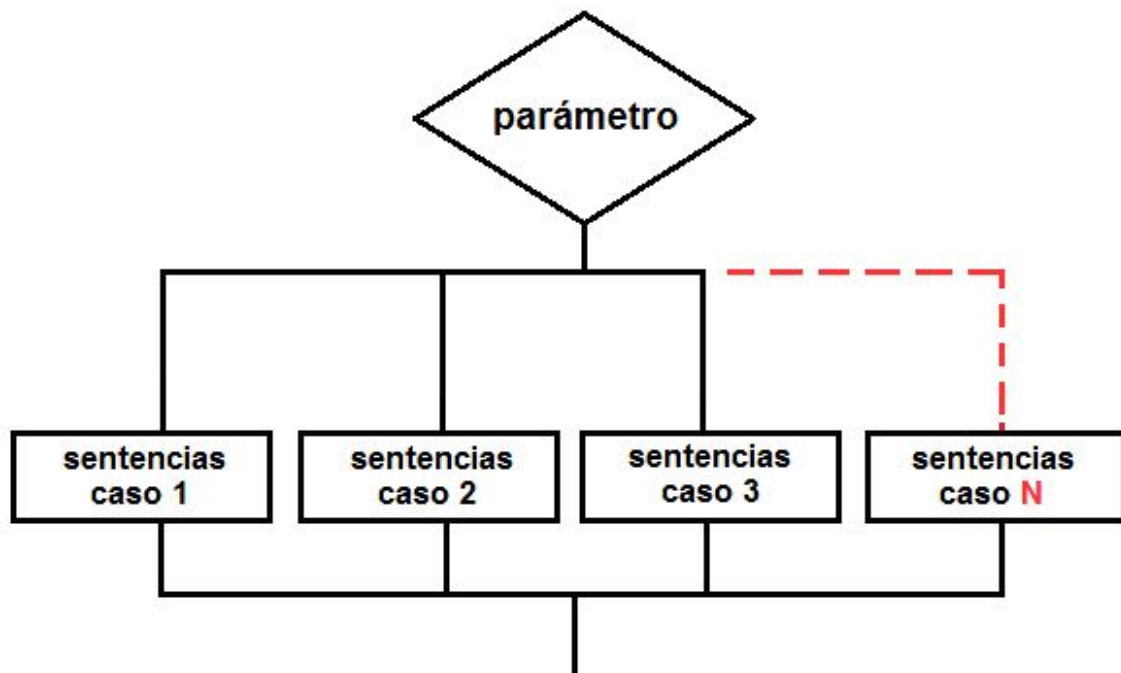
```
index.html x  codigo.js
var edad = prompt("La edad del usuario es?")

if(edad >= 20){
  var nombre = prompt("Cual es su nombre?")
  if(nombre == "Pedro")
    { alert("Hola Pedro")  }
  else
    { alert("Usted tiene 20 o más pero no es Pedro")}
  }
else { alert("Usted tiene menos de 20 años, adios!")}
```

Tanto este ejercicio, como el anterior podrán ser descargados desde la plataforma Alumni

Condicional múltiple o Switch

El condicional múltiple es muy útil en aquellos casos en que haya muchos caminos de ejecución posibles, y no queramos escribir un conjunto de ifs anidados. En casi todo lenguaje de programación el condicional múltiple es conocido como: switch – case.



Su sintaxis en todo lenguaje es similar a,

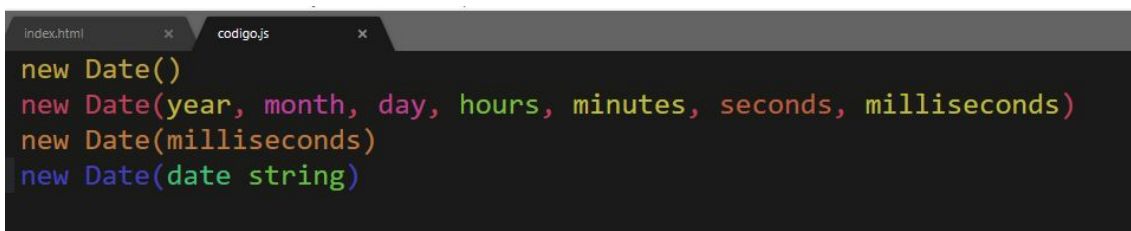
```
switch(parametro){  
    case 1:  
        //sentencias caso 1  
        break;  
    case 2:  
        //sentencias caso 2  
        break  
    case 3:  
        //sentencias caso 3  
        break  
    default:  
        //sentencias caso default  
        break  
}
```

Ejemplos de parámetros para un switch - case:

- El estado civil de una persona (soltero,divorciado,viudo,casado)
- El sexo de una persona (M,F)
- El resultado de un partido (victoria,empate,derrota)
- Los colores del arco iris (rojo, azul, violeta, etc)
- Puntos ganados en un partido (3,1 ó 0)
- Grupos sanguíneos.
- Provincias de Argentina.
- Etc

Objeto Fecha

El objeto fecha se crea a partir del constructor `new Date()`. Si por ejemplo



```

index.html x  codigo.js x
new Date()
new Date(year, month, day, hours, minutes, seconds, milliseconds)
new Date(milliseconds)
new Date(date string)
  
```

Lo más interesante es que podemos obtener datos del objeto fecha de la siguiente forma,

<code>getFullYear()</code>	Obtiene el año (yyyy)
<code>getMonth()</code>	Obtiene el mes del año (0-11)
<code>getDate()</code>	Obtiene el día del año (1-31)
<code>getHours()</code>	Obtiene las horas (0-23)
<code>getMinutes()</code>	Obtiene los minutos (0-59)
<code>getSeconds()</code>	Obtiene los segundos (0-59)
<code>getDay()</code>	Obtiene el día de la semana (0-6)

Por ejemplo si quisiera saber el día de la semana, pero de una forma que el usuario entienda de qué se trata y no meramente con números podría utilizar un switch qué es una estructura ideal para estas múltiples condiciones.

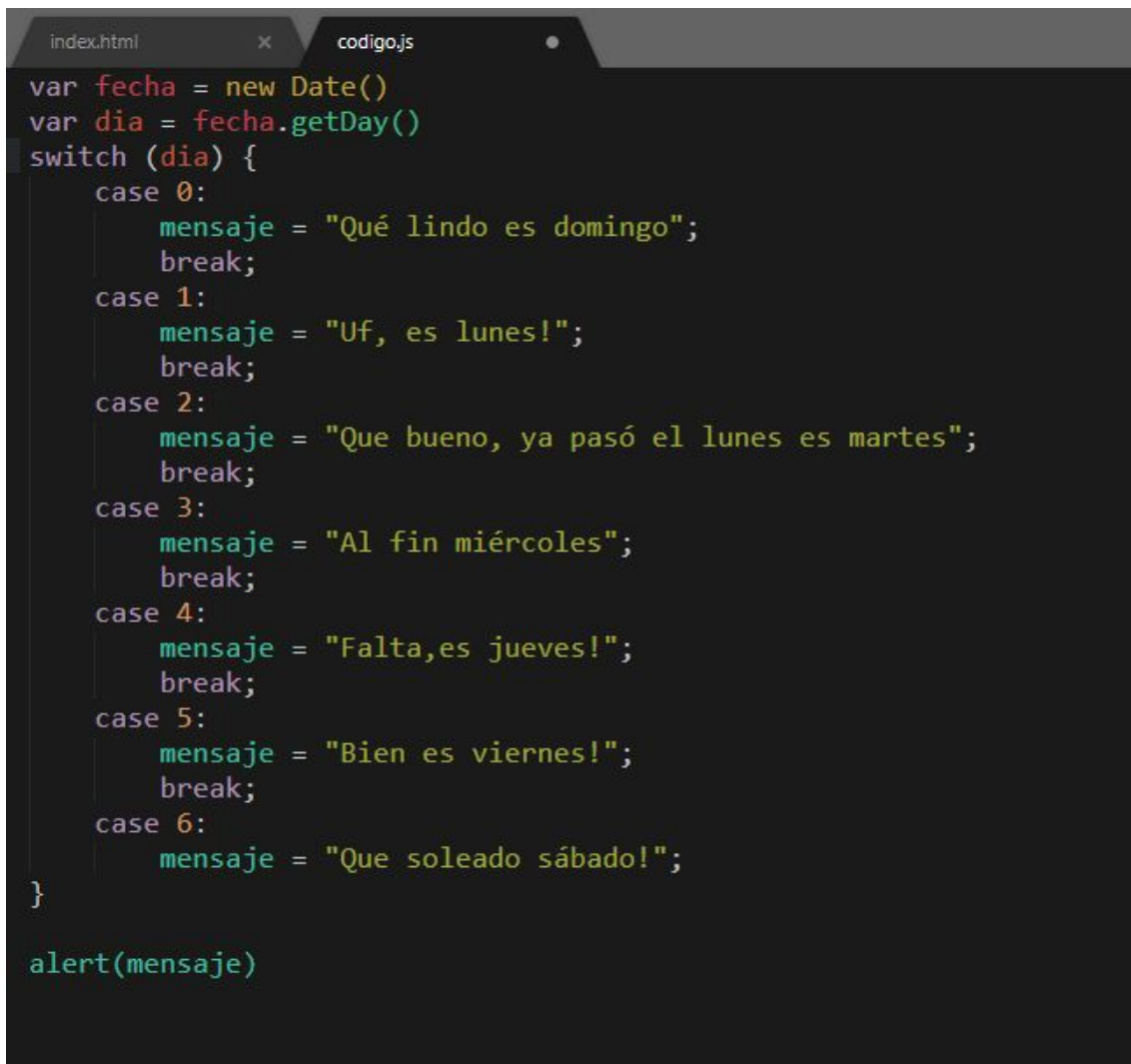
Primero con el objeto fecha obtendré el día con el método `getDay()`



```

index.html x  codigo.js x
var fecha = new Date()
var dia = fecha.getDay()
  
```

Luego, trabajaré con la estructura **Switch**,

A screenshot of a code editor with two tabs: 'index.html' and 'codigo.js'. The 'codigo.js' tab is active, showing a JavaScript script. The script uses a switch statement to determine the day of the week from the current date and display a message. The messages are in Spanish. The code is as follows:

```
var fecha = new Date()
var dia = fecha.getDay()
switch (dia) {
  case 0:
    mensaje = "Qué lindo es domingo";
    break;
  case 1:
    mensaje = "Uf, es lunes!";
    break;
  case 2:
    mensaje = "Que bueno, ya pasó el lunes es martes";
    break;
  case 3:
    mensaje = "Al fin miércoles";
    break;
  case 4:
    mensaje = "Falta, es jueves!";
    break;
  case 5:
    mensaje = "Bien es viernes!";
    break;
  case 6:
    mensaje = "Que soleado sábado!";
    break;
}

alert(mensaje)
```

Claro que quizás estamos trabajando demasiadas alternativas, por lo tanto, podríamos resumirlas de la siguiente forma,

```
index.html x código.js
var fecha = new Date()
var dia = fecha.getDay()
switch (dia) {
  case 0:
    mensaje = "Qué lindo es domingo";
    break;
  case 1:
  case 2:
  case 3:
    mensaje = "Aún falta para el viernes!";
    break;

  case 4:
    mensaje = "Falta, es jueves!";
    break;
  case 5:
    mensaje = "Bien es viernes!";
    break;
  case 6:
    mensaje = "Que soleado sábado!";
}

alert(mensaje)
```

HTML DOM

Para poder hacer más interesante nuestra aplicación es necesario acceder a elementos de nuestro HTML para así mostrar la información. Para poder hacerlo necesitamos acceder a ellos, por ejemplo

document.getElementById()

Con JS podemos acceder a los elementos de nuestro HTML. Estos son definidos como objetos, y poseemos métodos y propiedades de cada uno.

En este caso accedemos al elemento a través del id, qué es un atributo que trabajamos desde el HTML, por ejemplo, generamos en nuestro HTML un enunciado

```
<h1> Mi enunciado </h1>
```

Luego, si queremos acceder a este, debe trabajar con un ID

```
index.html x
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Desarrollo</title>
</head>
<body>
<h1 id="titulo"> Mi enunciado </h1>

</body>
</html>
```

Luego, para poder mostrarlo este texto en una alerta, debemos hacer en nuestro código lo siguiente,

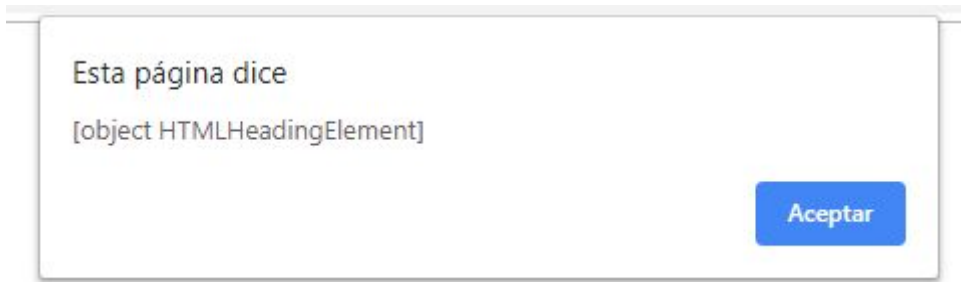
```
index.html x  codigo.js x
var enunciado = document.getElementById('titulo');
alert(enunciado)
```

Para que esto funcione correctamente, el código debe estar vinculado debajo del enunciado, ya que necesito que este se cargue para luego acceder a él, de otra forma estaremos accediendo a un elemento que no está en nuestro HTML aún,

```
index.html x  codigo.js x
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Desarrollo</title>

</head>
<body>
<h1 id="titulo"> Mi enunciado </h1>
<script src="js/codigo.js"> </script>
</body>
</html>
```

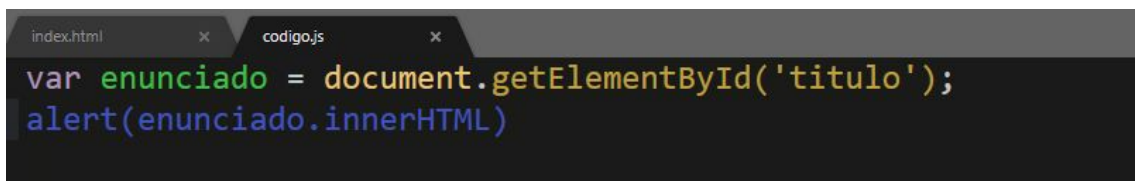
El resultado en nuestro navegador será el siguiente,



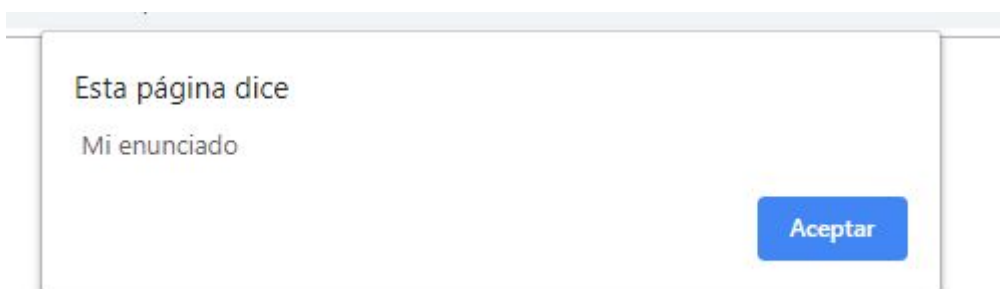
Como hemos anticipado, accedemos al objeto, pero necesitamos un método para poder mostrar su valor de texto, por ejemplo

innerHTML()

A través de este método podemos acceder a la información de un elemento o modificar su contenido. Si continuamos el caso anterior,



Ahora sí el resultado será el siguiente,



Por otro lado, si quisiéramos modificar el valor de texto lo haríamos de la siguiente forma,


```
index.html x código.js
var enunciado = document.getElementById('titulo');
enunciado.innerHTML = "Aca hay un nuevo contenido"
```

De esta forma podemos acceder a un elemento en nuestro HTML DOM, modificar su contenido u obtenerlo. Otra forma también muy simple de hacerlo es a través de **document.querySelector()**, esta manera nos permite acceder tanto a un elemento por ser un tipo de elemento o por su class o id.

```
index.html x código.js
var enunciado = document.querySelector('h1');
enunciado.innerHTML = "Aca hay un nuevo contenido"
```

En el caso anterior entonces , tenemos estas tres posibilidades ciertas,

```
index.html x código.js x
var enunciado = document.querySelector('h1');
var enunciado = document.querySelector('#titulo');
var enunciado = document.getElementById('titulo');
```