Unidad 2.- Manejo de la sintaxis del lenguaje:

a) Constantes y variables.

b) Tipos de datos.

c) Asignaciones.

d) Operadores.

e) Comentarios al código.

f) Sentencias.

g) Decisiones.

h) Bucles.

JavaScript es un lenguaje de programación que fue desarrollado en sus inicios para el navegador web Netscape Navigator y creado por Brendan Eich en 1995.

JavaScript fue diseñado para que su código se incluyese en documentos HTML y fuera interpretado por el navegador web. Ha cumplido con tanto éxito su propósito que el consorcio responsable de la especificación del lenguaje HTML (W3C ⇒ World Wide Web Consortium http://www.w3c.org en inglés y http://www.w3c.es en español), entre otras cosas, lo ha elegido como estándar para HTML5. Actualmente la responsable del desarrollo de JavaScript es la Mozilla Foundationhttp://www.mozilla.org/foundation/, con la documentación del lenguaje en inglés en https://developer.mozilla.org/en-US/docs/JavaScript y en castellano en https://developer.mozilla.org/es/docs/JavaScript.

Además existe una estandarización de JavaScript realizada por el ECMA (European Computer Manufacturers Association) International a través del ECMAScript 262, que se puede consultar en la página web http://www.ecma-international.org/publications/standards/Ecma-262.htm.

Deberemos tener en cuenta ciertas cuestiones a la hora de realizar programas en este lenguaje de programación.

- Una cosa muy importante a tener en cuenta cuando se programa en JavaScript es que los distintos navegadores van a poder tener diferencias en el lenguaje de programación ya que realizan diferentes interpretaciones del estándar.
- Diferencia las letras mayúsculas de las letras minúsculas, de tal forma que los siguientes nombres en un programa en JavaScript representan distintos elementos: MARIA, Maria, maria, MariA,...
- Todas o casi todas las instrucciones de JavaScript terminan en un punto y coma.
- Una instrucción puede ocupar una o varias líneas.
- En una línea se pueden incluir varias instrucciones, siempre y cuando estén separadas por un punto y coma.
- Varios espacios consecutivos son considerados como si fuese un único espacio.

a) Constantes y variables.

Una constante es un valor que no se puede modificar, permanece inalterable a lo largo de la ejecución del programa.

Para declarar constantes vamos a utilizar

const ***nombre-constante-1*=*valor-1*** **[**, ***nombre-constante-2*=*valor-2***] ...**

```
001  const CstApellidos="Muñoz Bayón";
002  const stNombre="Félix Ángel";
```

Las constantes van a ser del tipo de datos correspondiente al valor que se la asigna.

Una variable es una zona de la memoria donde vamos a alojar un dato y al cual vamos a poder hacer referencia a través de un nombre. El dato que vamos a tener almacenado puede variar a lo largo de la ejecución del programa.

JavaScript es un lenguaje débilmente tipado, esto nos indica que una variable va a ser del tipo de datos correspondiente al dato que tiene almacenado. Es más, a una variable le vamos a poder asignar valores de diferentes tipos de datos.

Para declarar una variable vamos a poner

var ***nombre-variable***;

```
001  varVstNombre;
```

o bien

let ***nombre-variable***;

La diferencia entre declarar una variable con var o con let va a ser el ámbito u alcance de la misma. Mientras que con let el ámbito es el bloque de código en el que se declara, con var su ámbito es la función dentro de la que se encuentra. Por consiguiente let se utilizará dentro de las funciones y demás cuerpos de bloques de código, y var cuando este al inicio del fichero que se encuentra en ningún bloque de código.

```
001  let VstModulo;
```

Al mismo tiempo que la variable se declara se la puede asignar un valor a través de

var ***nombre-variable*=*valor*** ;

let ***nombre-variable*=*valor*** ;

```
001  var VstApellido=”Muñoz”;
002  let VstModulo="cliente";
```

En una misma declaración vamos a poder declarar varias variables para lo cual usaremos

var ***nombre-variable-1*** **,** ***nombre-variable-2*****, … ;**

let ***nombre-variable-1*** **,** ***nombre-variable-2*****, … ;**

```
001  var VnbEdad, VstDomiclio;
```

al mismo tiempo que se declaran varias variables se pueden inicializar

var ***nombre-variable-1*** **[=***valor-1***],** **[***nombre-variable-2*** **[=***valor-2***]]… ;**

let ***nombre-variable-1*** **[=***valor-1***],** **[***nombre-variable-2*** **[=***valor-2***]]… ;**

```
001  var VnbEdad=35, VstDomicilio=”micasa”;
```

b) Tipos de datos.

En este lenguaje tenemos diferentes tipos de datos que son:

- Numéricos, **Number** que almacena números enteros y reales en notación decimal (136), hexadecimal (0x136) o científica (13.6e8). Los

valores que pueden tomar este tipo de datos van comprendidos entre **Number.MIN_VALUE** y **Number.MAX_VALUE**, además de estos valores, también pertenecen a este tipo de datos los valores **Infinity**, **-Infinity** y **NaN** (Not a Number). **Number(valor)** devuelve el valor numérico del valor que representa.

- Cadenas, **String** es una cadena de caracteres encerrada entre comillas o apostrofes. Los limitadores deben ser iguales, el inicial y el final, no se pueden mezclar. Incluso pueden ocupar varias líneas, basta con poner al final de cada línea el carácter '\'.
- **Plantillas de cadenas** es una cadena que va delimitada por el carácter del acento grave "`", puede ocupar más de una fila y no hace falta poner ningún carácter al final de las líneas. Dentro de una plantilla vamos a poder poner la referencia a una variable para que nos muestre su valor en ese lugar de la plantilla, el nombre de la variable debe ir encerrada entre llaves y precedida del signo de dólar, **${***nombre-variable***}**.
- Lógicos o Booleanos solamente admite los valores **true** o bien **false**.
- Undefined este tipo de datos solamente admite el valor **undefined**, valor que se asigna de forma automática a todas aquellas variables que se han declarado y que no se la ha asignado valor.
- **Object**, representa un objeto genérico.
- Null que solamente puede tomar el valor **null**, este valor se suele asignar a los objetos.

```
001  var VstQuijote="En un lugar de la mancha de cuyo nombre no quiero \
     alcordame no ha mucho que vivia un hidaldo de los \
                    antaño";
002  var VstInicio=`El inicio del quijote dicheasi: ${VstQuijote} la
                   continuación del mismo `;
```

La clase **Number** posee las siguientes propiedades:

- **MAX_VALUE**: valor máximo.
- **MIN_VALUE**: valor mínimo
- **NaN**: no es un número
- **NEGATIVE_INFINITY**: infinito negativo, para overflow.
- **POSITIVE_INTINITY**: infinito positivo, para overflow.
- **EPSILON**: valor de épsilon que es la diferencia entre 1 y el número de punto flotante más pequeño mayor que 1.
- **MAX_SAFE_INTEGER**: valor máximo de un número entero en doble precisión (2^{53} - 1).
- **MIN_SAFE_INTEGER**: valor mínimo de un número entero en doble precisión ($-(2^{53} - 1)$).

Un objeto **Number** también dispone de los siguientes métodos:

- **isNaN(*número*)**: devuelve un valor lógico que nos indica si el número es NaN.

- **isFinite(*número*)**: devuelve un valor lógico que nos indica si el número es finito.
- **isInteger(*número*)**: devuelve un valor lógico que nos indica si el número es un entero.
- **isSafeInteger(*número*)**: devuelve un valor lógico que nos indica si el número es un entero en doble precisión.
- **parseInt(*cadena* [, *base*])**: devuelve la transformación de los primeros números de la cadena, hasta encontrar el primer carácter no numérico, a un número entero, si no se indica la base es decimal. Si el número empieza por cero se pasa a octal, si empieza por 0x o bien por 0X se pasa a hexadecimal, si se indica la base se pasa a ese sistema. Si el primer no se puede transformar devuelve NaN. Es igual a la función del mismo nombre.
- **parseFloat(*cadena*)**: devuelve la transformación de los primeros números de la cadena, hasta encontrar el primer carácter no numérico, a un número real.Es igual a la función del mismo nombre.
- **toExponential([*número-decimales*]):** devuelve una cadena con el número en notación científica, exponencial. Se pueden indicar el número de decimales de la mantisa, si es menor de los que tiene realiza un redondeo.
- **toFixed([*número-decimales*]):** devuelve una cadena con el número en notación de coma fija inicialmente sin decimales, se puede indicar cuantos decimales se desean mostrar.
- **toString()**: devuelve una cadena con el valor del número.

```
001  var VnbPrimero=Number("36");
002  var VnbSegundo=78;
```

Un objeto o una variable **String** posee las siguientes propiedades

- **length**: longitud de la cadena.
- **prototype**: nos sirve para definir a continuación más propiedades o métodos.

Un objeto o una variable **String** posee los siguientes métodos.

- **charAt(*posición*)**: devuelve el carácter de la cadena que ocupa la posición indicada.
- **charCodeAt(*posición*)**: devuelve el número que representa el carácter que ocupa la posición indicada en unicode.
- **concat(*lista-cadenas*)**: devuelve una cadena que es la unión de la cadena y la lista de cadenas.
- **indexOf(*cadena* [, *posición*])**: devuelve la posición en que se encuentra la primera aparición de la cadena indicada, devuelve -1 si no la encuentra. La búsqueda se inicia en el principio a no ser que se indique la posición a partir de la cual se quiere realizar la búsqueda.
- **lastIndexOf(*cadena* [, *posición*])**: devuelve la posición en que se encuentra la primera aparición de la cadena indicada, devuelve -1 si no la encuentra. La búsqueda se inicia en el final a no ser que se indique

la posición a partir de la cual se quiere realizar la búsqueda. La búsqueda se realiza de final a principio.

- **includes(*cadena* [, *posición*])**: devuelve un valor lógico que nos indica si la cadena se encuentra dentro de la variable. La búsqueda se inicia en el principio a no ser que se indique la posición a partir de la cual se quiere realizar la búsqueda.
- **search(*cadena*)**: devuelve la posición que ocupa en la cadena inicial la cadena indicada.
- **substr(*posición*, *número-caracteres*)**: devuelve una cadena con tantos caracteres como se indican a partir de la posición indicada.
- **substring(*posición-1* [, *posición-2*])**: devuelve una cadena con los caracteres existentes entre las posiciones indicadas, excepto el carácter que ocupa la posición indicada por posición-2.
- **slice(*posición-1* [, *posicón-2*])**: copia de la cadena inicial todos los caracteres que hay desde la posición-1 a la posicón-2 excluido; si se omite posición-2 es hasta el final y si posición-2 es negativo se indican el número de caracteres del final que no se copian.
- **toLowerCase()**: devuelve una copia de la cadena con las letras en minúsculas.
- **toUpperCase()**: devuelve una copia de la cadena con las letras en mayúsculas.
- **toString([*base*])**: devuelve una cadena con el valor del objeto, si el valor es numérico se indica la base,por defecto 10.
- **fromCharCode(*lista-códigos-unicode*)**: devuelve una cadena con los caracteres indicados a través de su código unicode.
- **codePointAt(*posición*)**: devuelve el código Unicode correspondiente al carácter que ocupa la posición indicada en la cadena.
- **replace(*cadena-remplazar*, *cadena-nueva*)**: devuelve una nueva cadena en la cual a la cadena inicial se la ha sustituido la cadena-remplazar por la cadena-nueva.
- **repeat(*número*)**: devuelve una nueva cadena que es la cadena inicial repetida tantas veces como indica número.
- **trim()**:devuelve la cadena sin los blancos de la izquierda ni de la derecha.
- **trimLeft()**: devuelve la cadena sin los blancos iniciales de la izquierda. Se sustituye por el siguiente método.
- **trimStart()**: devuelve la cadena sin los blancos iniciales de la izquierda
- **trimRight()**: devuelve la cadena sin los blancos finales de la derecha. Se sustituye por el siguiente método
- **trimEnd()**: devuelve la cadena sin los blancos finales de la derecha.
- **valueOf()**: valor de la cadena.
- **padEnd(*longitud*)**:devuelve la cadena con al menos la longitud indicada, poniendo espacios al final de la cadena, si es necesario.
- **padStart(*longitud*)**:devuelve la cadena con al menos la longitud indicada, poniendo espacios al inicio de la cadena, si es necesario.

- **endsWith(*cadena* [, *longitud*])**:devuelve un valor lógico que nos indica si la cadena inicial termina por la cadena dada, o bien por los caracteres indicados de la cadena dada.
- **startsWith(*cadena* [, *longitud*])**:devuelve un valor lógico que nos indica si la cadena inicial empieza por la cadena dada, o bien por los caracteres indicados de la cadena dada.

```
001 var VstQuijote="En un lugar de la mancha de cuyo nombre no quiero alcordarme no ha mucho que vivia";
002 document.write(VstQuijote.length+"<br />");
003 // Resultado -> 82
004 document.write(VstQuijote.charAt(12)+"<br />");
005 // Resultado -> d
006 document.write(VstQuijote[12]+"<br />");
007 // Resultado -> d
008 document.write(VstQuijote.charCodeAt(12)+"<br />");
009 // Resultado -> 100
010 document.write(VstQuijote+"un hidalgo"+"<br />");
011 // Resultado -> En un lugar de la mancha de cuyo nombre no quiero alcordarme no ha mucho que viviaun hidalgo
012 var VnbPosicion1=VstQuijote.indexOf("no");
013 document.write("primero aparición de 'no', en posición "+VnbPosicion1.toString()+"<br />");
014 // Resultado -> primero aparición de 'no', en posición 33
015 var VnbPos2=VstQuijote.indexOf("no",VnbPosicion1+1);
016 document.write("segunda aparición de 'no', en posición "+VnbPos2.toString()+"<br />");
017 // Resultado -> segunda aparición de 'no', en posición 40
018 var VnbPosi=VstQuijote.lastIndexOf("no");
019 document.write("última aparición de 'no', en posición "+VnbPosi.toString()+"<br />");
020 // Resultado -> última aparición de 'no', en posición 61
021 var VnbPosic03=VstQuijote.lastIndexOf("no",VnbPosicion1);
022 document.write("no encuentra más 'no' "+VnbPosic03+"<br />");
023 // Resultado -> no encuentra más 'no' 33
024 var VnbLugar=VstQuijote.search("mancha");
025 document.write("posicion de mancha "+VnbLugar+"<br />");
026 // Resultado ->posicion de mancha 18
027 document.write(VstQuijote.substr(12,8)+"<br />");
028 // Resultado -> de la ma
029 document.write(VstQuijote.substring(26)+"<br />");
030 // Resultado -> e cuyo nombre no quiero alcordarme no ha mucho que vivia
031 document.write(VstQuijote.substring(10,20)+"<br />");
032 // Resultado -> r de la ma
033 document.write(VstQuijote.slice(25)+"<br />");
034 // Resultado -> de cuyo nombre no quiero alcordarme no ha mucho que vivia
035 document.write(VstQuijote.slice(15,30)+"<br />");
036 // Resultado -> la mancha de cu
037 document.write(VstQuijote.slice(20,-3)+"<br />");
038 // Resultado ->ncha de cuyo nombre no quiero alcordarme no ha mucho que vi
039 document.write(VstQuijote.toLowerCase()+"<br />");
040 // Resultado -> en un lugar de la mancha de cuyo nombre no quiero alcordarme no ha mucho que vivia
041 document.write(VstQuijote.toUpperCase()+"<br />");
042 // Resultado -> EN UN LUGAR DE LA MANCHA DE CUYO NOMBRE NO QUIERO ALCORDARME NO HA MUCHO QUE VIVIA
043 document.write(VstQuijote.replace("mancha","alcarria")+"<br />");
044 // Resultado -> En un lugar de la alcarria de cuyo nombre no quiero alcordarme no ha mucho que vivia
```

Métodos de la clase **String** para HTML

- **anchor(*nombre*)**: devuelve una cadena con el resultado de crear un enlace, etiqueta <a> con el literal de la cadena inicial y cuyo atributo name es el nombre indicado.

- **big()**: devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <big>.
- **blink()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <blink>. opera y firefox.
- **bold()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <b>.
- **fixed()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <tt>.
- **fontcolor(*color*)**: devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <font color="color">.
- **fontsize(*tamaño*)**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <fontsize="tamaño">.
- **italics()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <i>.
- **link(*url*)**:devuelve una cadena con el resultado de crear un enlace, etiqueta <a> con el literal de la cadena inicial y cuyo atributo href es la url indicada.
- **small()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <small>.
- **strike()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <strike>.
- **sub()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <sub>.
- **sup()**:devuelve una cadena en la cual se ha incluido el contenido de la cadena inicial dentro de la etiqueta <big>.

Una variable de tipo string puede ser manejada como un array.

```
001 var VstUno=new String("dato");//crea objeto String
002 var VstDos=String("primero");// crea una cadena
003 var VstTres="prueba";//crea una cadena
```

c) Asignaciones.

Las asignaciones nos permiten introducir un valor en una variable, este valor puede venir de una expresión.

Operadores de asignación

=	Asignación	Asigna el valor de la expresión a la variable
+=	Suma/concatenación y asignación	Suma el valor de la expresión al valor de la variable y luego asigna el valor a la misma. A+=3. Incrementa el valor de A en 3.
-=	Resta y asignación	Resta el valor de la expresión al valor de la variable y luego asigna el valor a la misma. A-= 3 * B. equivale a A=A - (3 * B)
***=**	Multiplicación y asignación	Multiplica el valor de la expresión con el valor de la variable y luego asigna el valor a la misma.

/=	División y asignación	Divide el valor de la variable entre el valor de la expresión y luego asigna el cociente a la variable.
%=	Módulo y asignación	Divide el valor de la variable entre el valor de la expresión y luego asigna el resto a la variable.
****=**	Potencia y asignación	Realiza la potencia del valor de la variable elevado al valor de la expresión

d) Operadores.

Los operadores nos permiten realizar ciertas operaciones con los operandos, vamos a tener varios tipos de operadores, que son:

Operadores aritméticos:

++	Incremento	incrementan en uno al operando, puede ir delante del operando (preincremento, primero se incrementa el operando y luego se realiza la expresión donde se encuentra ++a) o bien detrás del operando (postincremento, primero se realiza la expresión donde se encuentra y luego se incrementa el operando a++).
		A = 5;
		B = ++A * 3; // B = 6 * 3 = 18, A=6.
		C = A++ * 5; // C = 6 * 5 = 30. A=7.

--	Decremento	Decrementa en uno al operando, puede ir delante del operando (predecremento, primero se decrementa el operando y luego se realiza la expresión donde se encuentra --a) o bien detrás del operando (postdecremento, primero se realiza la expresión donde se encuentra y luego se decrementa el operando a--).
		A = 5;
		B = --A * 3; // B = 4 * 3 = 12. A=4.
		C = A-- * 5; // C = 4 * 5 = 20. A=3.

+	Suma	suma los operandos situados a su lado.
-	Resta	resta al operando de izquierda el valor del operando de su derecha.
*****	Multiplicación o producto	multiplica los dos operandos
/	División o cociente división.	divide el operando de su izquierda entre el operando de su derecha.
%	Resto división entera o módulo	realiza la división entera y devuelve el resto.
******	Potencia	Eleva la base, situada a la izquierda del operador, al exponente, situado a la parte derecha del operador

Operadores booleanos o lógicos.

!	Negación	niega el operando de su derecha.
&&	Y lógico o AND	sólo devuelve verdadero si los dos son verdaderos y falso en caso contrario.
||	O lógico u OR.	devuelve verdadero si uno de sus operandos es verdadero y falso en caso contrario.
??		

Operadores de bits

&&	Y lógico o AND	
||	O lógico u OR.	
^	XOR	
~	Complementación o negación	
<<	Desplazamiento a la izquierda	Desplaza los bits a la izquierda.16 << 2. Al número 16 le desplaza dos bits a la izquierda.
>>	Desplazamiento a la derecha	Desplaza los bits a la derecha insertando cero por la izquierda. 24 >> 3. Al número 24 le desplaza tres bits a la derecha.
>>>	Desplazamiento a la derecha, insertando ceros por la izquierda	Si el número es positivo es igual al anterior, si es negativo no.

Operadores relacionales

==	Igual que	¡¡¡Cuidado al comparar expresiones de números reales!!! 0.3 == 0.2 + 0.1 es falso
!=	Distinto que	
===	Idéntico, incluso en tipo	
!==	Diferente a (tipo y/o valor)	
<	Menor que	
<=	Menor o igual que	
>	Mayor que	
>=	Mayor o igual que	

Operador de cadenas

+	Concatenación	

Otros operadores

Condición*?*valor1*:*valor2	Operador condicional	Si le condición es verdadera devuelve el valor1 y si es falsa devuelve el valor2.

typeof *expresión*	Indica el tipo de la expresión

e) Comentarios al código.

Existen dos formas de poner comentarios en Javascript, que son:

Comentarios de una línea, para lo cual utilizaremos:

//*comentario*

todo lo que va detrás de las dos barras y hasta el final de línea es un comentario.

Comentarios de más de una línea.

/*

comentario

***/**

todo lo que hay entre los caracteres de inicio (**/***) y los de final (***/**) es un comentario, puede ocupar una o varias líneas.

f) Sentencias.

Vamos a ver algunas sentencias que vamos a ir utilizando como son:

```
alert(cadena);
window.alert(cadena );
```

Nos muestra un cuadro de mensaje con la cadena indicada.

```
confirm(cadena);
window.confirm(cadena );
```

Nos muestra un cuadro de mensaje con la cadena y con los botones de Aceptar o Cancelar, devuelve true o 1 para Aceptar y false o 0 para Cancelar.

```
prompt( cadena);
window.prompt(cadena );
```

Nos muestra un cuadro de mensaje con la cadena y nos pide un valor, nos muestra también los botones de Aceptar o Cancelar. Devuelve un valor string con el valor introducido si se pulsa Aceptar y con el valor nulo, null, si se pulsa el botón Cancelar.

```
console.log(cadena);
window.console.log(cadena);
```

Nos muestra la cadena indicada en la consola del navegador web.

```
document.write(cadena);
document.writeln(cadena);
```

Escribe en el documento el contenido de la cadena, si la cadena incluye código HTML, las etiquetas son interpretadas por el navegador.

La instrucción para detección de errores es:

```
try {
        instrucciones-1}
catch (variable-excepción if condición){   ...
        instrucciones-2}

catch( variable-excepción ) {   ...
        instrucciones-3}

finally {
        instrucciones-n }
```

Si al ejecutar las instrucciones1 se produce un error o una excepción se ejecutan las instrucciones que acompañan al **catch** correspondiente a ese error o excepción, sino se encuentra se ejecutan las instrucciones-3. Además, siempre que se incluyan, al final se van a ejecutar las instrucciones que hay a continuación de **finally**.

```
001  try {
002      let VobPepe=new Object();
003          VobPepe=VobJuan;
004  }
005  catch(err){
006      console.log("fallo");
007  }
008  finally{
009      console.log("Siempre");
010  }
```

Para generar excepciones tenemos la instrucción **throw** cuya forma es:

```
throw excepción ;
```

Genera una excepción representada por el valor de la expresión, que puede ser una cadena, un número o un valor lógico.

```
001  var VstCadena;
002  try {
003      VstCadena=window.prompt("Introduce una cadena");
004      if(VstCadena.length==0) throw"Cadena vacía"
005      else if(VstCadena.length%2==0) throw"Longitud par";
006
007  }catch(err) {
008      console.log(err);
009  }finally {
010      console.log("valor introducido "+VstCadena);
011  }
```

g) Decisiones.

Las instrucciones de decisión nos permiten ejecutar unas instrucciones en función de que se cumpla una determinada condición. Para ello tenemos la instrucción **if** con sus diferentes formatos.

```
if (condición) {
        instrucciones-1
} else {
        instrucciones-2
}
```

si la condición es verdadera, se ejecutan las instrucciones-1, mientras que si la condición es falsa se ejecutan las instrucciones-2, siempre y cuando se incluyan las mismas.

```
001 var VstCadena=window.prompt("Introduce una palabra");
002 if (VstCadena=="felix"){
003     window.alert('Nombre Correcto');
004 }else{
005     window.alert("Nombre Erroneo");
006 }
```

o bien

```
001 var VstCadena=window.prompt("Introduce una palabra");
002 if(VstCadena=="felix")
003     window.alert('Nombre Correcto')
004 else
005     window.alert("Nombre Erroneo");
```

Siguiente formato

```
if (condición-1) {
        instrucciones-1
} else if ( condición-2) {
        instrucciones-2
} else if (condición-3) {
        instrucciones-3
    }                          ...

    else  {
        instrucciones-n
     }
```

Se ejecutarán las instrucciones existentes a continuación de la primera condición que sea verdadera, si todas las condiciones son falsas se ejecutarán las instrucciones existentes a continuación del último **else**.

```
001 var VstCadena=window.prompt("Introduce una letra");
002 if(VstCadena=="a"){
003     window.console.log('Primera vocal "a"');
004 }else if(VstCadena=="e"){
005     window.console.log("Segunda vocal 'e'");
006 }elsei f(VstCadena=="i"){
007     window.console.log("Tercera vocal 'i'");
008 }else if(VstCadena=="o"){
```

```
009     window.console.log("Cuarta vocal 'o'");
010 }else if(VstCadena=="u"){
011     window.console.log("Quinta vocal 'u'");
012 }else{
013     window.console.log("No es una vocal");
014 }
```

o bien

```
001 var VstCadena=window.prompt("Introduce una letra");
002 if(VstCadena=="a")
003     window.console.log('Primera vocal "a"')
004 else if(VstCadena=="e")
005     window.console.log("Segunda vocal 'e'")
006 else if(VstCadena=="i")
007     window.console.log("Tercera vocal 'i'")
008 else if(VstCadena=="o")
009     window.console.log("Cuarta vocal 'o'")
010 else if(VstCadena=="u")
011     window.console.log("Quinta vocal 'u'")
012 else
013     window.console.log("No es una vocal ");
```

La alternativa múltiple la tenemos a través de sentencia **switch** cuyo formato es:

```
switch (expresión ) {
case valor-1:
        instrucciones-1
        break;
    [   case valor-2:
        instrucciones-2 ...
        break;  ]

    [   default:
        instrucciones-n ]
}
```

Se van a ejecutar las instrucciones existentes a continuación del primer valor que se encuentra y que coincide con el valor de la expresión y dejará de ejecutar las instrucciones cuando encuentre la instrucción **break**. Si ninguno de los valores que se muestran coincide con el valor de la expresión se ejecutan las instrucciones existentes a continuación de **default**, si se incluye.

```
001 var VstDigito=window.prompt("Introduce un dígito");
002 switch(VstDigito){
003     case "1":
004     case "3":
005     case "5":
006     case "7":
007     case "9":
008         window.console.log("dítitoimpar");
009         break;
010     case "0":
011         window.console.log("dígito cero");
012         break;
013     case "2":
014     case "4":
015     case "6":
016     case "8":
017         window.console.log("dítito par");
```

```
018             break;
019         default:
020             window.console.log("No es un digito");
021     }
```

```
001 var VstModulo=window.prompt("Introduce el nombre de un módulo");
002 switch(VstModulo){
003     case "Diseño interfaces web":
004         document.write(`El profesor de ${VstModulo} es Félix Ángel Muñoz`);
005         break
006     case "Desarrollo Web entorno servidor":
007         document.write(`El profesor de ${VstModulo} es Alfonso Rebolleda`);
008         break
009     case "Inglés teécnico":
010         document.write(`El profesor de ${VstModulo} es María Guerra`);
011         break
012     case "Despliegue de aplicaciones":
013         document.write(`El profesor de ${VstModulo} es Alfonso Rebolleda`);
014         break
015     case "Empresa":
016         document.write(`El profesor de ${VstModulo} es Carmen`);
017         break
018     case "Desarrollo web rntorno cliente":
019         document.write(`El profesor de ${VstModulo} es Félix Ángel Muñoz`);
020         break
021     default:
022         document.write(`No existe el módulo ${VstModulo}`);
023 }
```

h) Bucles.

Los bucles nos van a permitir repetir un conjunto de instrucciones un número determinado de veces.

La primera instrucción que nos permite realizar bucles es la instrucción while cuyo formato es:

```
while( condición) {
        instrucciones
}
```

Si la condición es verdadera se van a ejecutar las instrucciones, una vez ejecutadas se vuelve a evaluar la condición y mientras que la condición sea verdadera se van a ejecutar las instrucciones. Si la primera vez que se evalúa la condición es falsa, las instrucciones nunca se llegarán a ejecutar.

```
001 var VnbNumero=1;
002 window.console.log("Primer bucle");
003 while(VnbNumero>10){
004     VnbNumero+=1;
005 }
006 window.console.log(VnbNumero);
007 window.console.log("Segundo bucle");
008 while(VnbNumero<10){
009     VnbNumero+=1;
010 }
011 window.console.log(VnbNumero);
```

Otra instrucción que tenemos para bucles es la sentencia **do..while**, cuyo formato es.

```
do {
        instrucciones
}while ( condición);
```

Las instrucciones se van a ejecutar y luego se va a evaluar la condición, si la misma es verdadera, las instrucciones se seguirán ejecutando mientras que la condición sea verdadera. En este caso las instrucciones se ejecutarán al menos una vez.

```
001 window.console.log("Tercerbucle");
002 var VnbSegundo=1;
003 do{
004     VnbSegundo+=1;
005 }while(VnbSegundo>10);
006 window.console.log(VnbSegundo);
007 window.console.log("Cuarto Bucle");
008 do{
009     VnbSegundo+=1;
010 }while(VnbSegundo<10);
011 window.console.log(VnbSegundo);
```

La última instrucción que tenemos para la realización de iteraciones es la instrucción **for**, cuyo formato es:

```
for( inicialización;condición-fin;incremento) {
        instrucciones
}
```

En primer lugar, se realiza la inicialización a continuación, si la condición es verdadera, se ejecutan las instrucciones, a continuación, se realiza la incrementación y si la condición es verdadera se vuelve a ejecutar las instrucciones y así repetidamente mientras que la condición sea verdadera.

```
001 window.console.log("QuintoBucle");
002 for(let VnbIndice=1;VnbIndice<10;VnbIndice++){
003     window.console.log(VnbIndice);
004 }
005 window.console.log("SextoBucle");
006 for(let VnbIndice=15;VnbIndice>0;VnbIndice--)
007     window.console.log(VnbIndice);
008     window.console.log("SeptimoBucle");
009 for(let VnbIndice=0;VnbIndice<41;VnbIndice+=2){
010    window.console.log(VnbIndice);
011 }
012 window.console.log("Octavo Bucle");
013 for(let VnbIndice=37;VnbIndice>1;VnbIndice-=3)
014     window.console.log(VnbIndice);
```

Dentro de los bucles tenemos dos instrucciones que son:

break;

Nos permite salir en ese instante del bucle dentro del cual se encuentra.

continue;

Salta a comprobar la condición del bucle para determinar si se va a seguir ejecutando o bien va a finalizar.

Estas dos instrucciones **no se deben utilizar a no ser casos extremos**, ya que deseamos aplicar técnicas de programación estructurada.

Tenemos otros dos bucles for relativos a objetos y miraremos con más detenimiento cuando veamos objetos.

```
for( variable in objeto) {
        instrucciones
}
```

```
for(variable of objeto) {
        instrucciones
}
```