MODULO

# JAVASCRIPT

# JAVASCRIPT BÁSICO

1. SINTAXIS EN JAVASCRIPT
2. VARIABLES
3. PALABRAS RESERVADAS
4. TIPOS DE VARIABLES
5. OPERADORES
6. OBJETOS NATIVOS
7. FUNCIONES
8. PROPIEDADES

# Historia

JavaScript es un lenguaje de programación interpretado cuyo uso se ha popularizado gracias a la web.

El lenguaje JavaScript se usa principalmente para el diseño y generación de páginas web de manera dinámica (pudiendo alterar la presentación de la misma mediante dicho lenguaje).

Es por lo tanto en la máquina del cliente donde se realiza la interpretación de dicho código.

Cualquier navegador actual es capaz de interpretar el lenguaje JavaScript

# Historia

JavaScript fue desarrollado por Brendan Elch (Netscape).

Originalmente se nombró Mocha, posteriormente LiveScript y finalmente JavaScript (Diciembre de 1995)

En 1997 fue adoptado por la ECMA (European Computers Manufacturers Association) como un estándar.

Existen múltiples librerías para usar con JavaScript, como Node.js (para programar con JavaScript en el lado del servidor), Ajax (para peticiones asíncronas), Angular.js (implementación de arquitectura MVC)...

# Características

- Tipado Dinámico → El tipo de una variable está asociado a su valor, es decir, una misma variable puede contener un número y luego un texto
- Objetual → Casi todo en JavaScript son arrays asociativos, es decir, el nombre de su propiedad es la clave de un mapa. var.p y var['p'] es equivalente
- Funciones de primera clase → Una función es un objeto en JS, y como tal poseen propiedades y métodos. Por lo tanto, una función puede tener una propiedad que sea otra función.
- Prototipos → JavaScript usa prototipos en vez de clases para el uso de herencia

# SINTAXIS EN JAVASCRIPT

- SIMILAR A OTROS LENGUAJES
- NO CUENTA ESPACIOS NI SALTOS DE LINEA
- DISTINGUE MAYUSCULAS CON MINUSCULAS
- NO SE DEFINE EL TIPO DE VARIABLES
- NO ES OBLIGATORIO TERMINAR CON ;
- PERMITE COMENTARIOS

```
/**
 * Este es un comentario
 */

// Este es otro comentario
```

# Ejercicios:

- Escribid el siguiente HTML, que nos servirá como ejemplo para pruebas futuras. Después hacer que en lugar de saludar, se despida.

```
<html>
<script type="text/javascript">
function saluda(){
    alert("Saluditos");
}
</script>
<body>
</body>
<button type="button" onclick="saluda()"> Prueba.</button>
<p id="demo"></p>
</html>
```

# VARIABLES

- SE DEFINEN MEDIANTE VAR

```
var mivariable;
var mivariable2;
var mivariable3=10;
var mivariable4=10,mivariable5=11;
```

- NO NOMBRAR CON $ NI _
- NO HAY OBLIGACIÓN DE INICIALIZAR LAS VARIABLES
- SI NO SE INDICA VAR, SE CREA UNA VARIABLE GLOBAL

# VARIABLES

- COMPORTAMIENTO

```
<script type="text/javascript">
    var variableLocal = "esta es una variable global en realidad";
    function probandoVariables(){
        alert(variableLocal);
        variableGlobalAlNoEstarDeclarada = "VariableGlobalNoDeclarada";
    }
    function probandoConVariablesComoParametros(variableLocal){
        alert(variableGlobalAlNoEstarDeclarada);
        alert(variableLocal);
    }
    probandoConVariablesComoParametros(variableLocal);
    probandoVariables();
    probandoConVariablesComoParametros(variableLocal);

</script>
```

# PALABRAS RESERVADAS

- RESERVADAS
  - ACTUALES
    - BREAK,ELSE, NEW,VAR, CASE, FINALLY,RETURN,VOID,CATCH,FOR,SWITCH,
    - WHILE,CONTINUE,FUNCTION,THIS,WITH,DEFAULT,IF,THROW,DELETE,IN,TRY,DO,
    - INSTANCEOF,TYPEOF
  - FUTURAS
    - ABSTRACT,ENUM,INT,SHORT,BOOLEAN,EXPORT,
    - INTERFACE,STATIC,BYTE,EXTENDS,LONG,SUPER,
    - CHAR,FINAL,NATIVE,SYNCHRONIZED,CLASS,FLOAT,PACKAGE,THROWS,CONST,GOTO,PRIVATE,
    - TRANSIENT,DEBUGGER,IMPLEMENTS,PROTECTED,
    - VOLATILE,DOUBLE,IMPORT,PUBLIC

# TIPOS DE VARIABLES

- TIPOS PRIMITIVOS
  - TYPEOF: DETERMINA EL TIPO DE DATO EN UNA VARIABLE

```
<script type="text/javascript">
function prueba(){
var estoEsUnNumero = 7;
var estoEsUnTexto = "Siete";
alert(estoEsUnNumero+" : "+typeof estoEsUnNumero);
alert(estoEsUnTexto+" : "+typeof estoEsUnTexto);
}
prueba();
```

# TIPOS DE VARIABLES

- TIPO UNDEFINED
  - Variables no definidas o variables no declaradas
    - Var variableDeclarada;
    - Typeof variableNoDeclarada;
    - Typeof variableDeclarada;
  - Los dos casos dan el mismo resultado
    - «undefined»

# TIPOS DE VARIABLES

- TIPO NULL
  - Variable similar a undefined..
  - Javascript no es capaz de diferenciarlas.
  - Representa la referencia a un objeto que todavía no existe.
  - Var objetoQueTodaviaNoExiste = null;

# TIPOS DE VARIABLES

- TIPO BOOLEAN
  - ALMACENAN TRUE O FALSE
  - NO SON NI PALABRAS NI NÚMEROS
  - LA CONVERSIÓN DE UN NÚMERO A BOOLEAN PRODUCE EL SIGUIENTE CAMBIO
    - 0 : FALSE
    - CUALQUIER OTRO : TRUE

# TIPOS DE VARIABLES

- TIPO NUMÉRICO
  - ALMACENA LOS SIGUIENTES TIPOS DE VALORES
    - ENTERO ( var entero = 10 )
    - DECIMAL ( var decimal = 3.26 )
    - OCTAL ( var octal = 033 )
    - HEXADECIMAL (var hexadecimal = 0xD3 )
  - VALORES ESPECIALES
    - INFINITY : DIVISIONES POR CERO
    - NAN : OPERACIONES CON VARIABLES NO NUMÉRICAS
      - isNaN(variableUOperacionAComprobar);
      - Devuelve true si no es numérico el resultado.

# TIPOS DE VARIABLES

- Constantes predefinidas en JS
  - Math.E
  - Math.LN2
  - Math.LN10
  - Math.LOG2E
  - Math.LOG10E
  - Math.PI
  - Math.SQRT1_2
  - Math.SQRT2

# TIPOS DE VARIABLES

- TIPO STRING
  - CADENAS DE TEXTO RODEADAS POR COMILLAS SIMPLES O DOBLES
  - EL PRIMER CARÁCTER SE ALMACENA EN LA POSICIÓN CERO
  - NO PERMITE LA INSERCIÓN DE CIERTOS CARACTERES COMO EL SALTO DE CARRO.
  - SE DEFINEN LOS CARACTERES ESPECIALES COMO:
    - \n SALTO DE LINEA
    - \t TABULADOR
    - \' COMILLAS SIMPLES
    - \» COMILLAS DOBLES
    - \\ BARRA

# Control del flujo

- While / do ... while → Ejecuta un bloque mientras la condición se cumpla
- for → Ejecuta un bloque mientras se cumpla una condición
- if...else → Ejecuta un bloque u otro en función de una condición
- switch → Ejecuta un bloque u otro en función de unos valores
- try...catch → Control de errores sobre un bloque

# while / do ... while

- do ... while → Ejecuta un bloque mientras la condición se cumpla

```
function dowhile(){
    var x=0;
    var total=0;
    do{
        total+=x;
        x++;
    } while (x<5);
    alert(total);
}
```

```
function whiledo(){
    var x=0;
    var total=22;
    while (x<5){
        total+=x;
        x++;
    }
    alert(total);
}
```

# for

- for → Ejecuta un bloque mientras se cumpla una condición

```
function buclefor(){
    for (var i=0;i<3;i++){
        console.log("molesto? "+i)
    }
}
```

# if…else

- if...else → Ejecuta un bloque u otro en función de una condición

```
function ifelse(){
    var x=9;
    if (x>10){
    alert("mayor que 10");
    }
    else if (x>0){
        alert("mayor que 0");
    }
    else{
        alert("menor o igual a 0");
    }
}
```

# switch

- switch → Ejecuta un bloque u otro en función de unos valores

```
function bucleswitch(){
    switch (new Date().getDay()) {
        case 0:
            day = "Domingo";
            break;
        case 6:
            day = "Sábado";
            break;
        default:
            day = "no es finde";
    }
    console.log(day);
}
```

# try...catch

- try...catch → Control de errores sobre un bloque

```
function trycatch(){
var x=1000;
try {
   if (x>100){throw "Muy grande"}
}
catch(err) {
    alert(err);
}
}
```

# Ejercicios

- Mostrar por pantalla el factorial del número 17
- Sacar 100 mensajes por consola que digan "hola"
- Usando "Math.random()"contar todas las veces que este da <0.8, cuando arroje otro valor, parar

# TIPOS DE VARIABLES

- CONVERSIONES
  - .toString()
  - DEVUELVE EL CONTENIDO DE UNA VARIABLE EN TEXTO.
    - var miVariableNumero = 10;
    - var variableTransformada = miVariableNumero.toString();
  - parseInt(variableATransformar)
    - Devuelve un entero
    - Solo recoge los primeros digitos. El resto los ignora
  - parseFloat(variableATransformar)
    - Devuelve un decimal
    - Mismo comportamiento que parseInt()

# Ejercicio

- Usando la plantilla mostrada, hacer una "calculadora" que sume, reste, multiplique y divida (Opcional: Controlar errores)

```
function suma(){
var a= document.getElementById('op1').value;
var b= document.getElementById('op2').value;
document.getElementById('result').innerHTML= ?????
}
```

```
<input id="op1" type="text">
<input id="op2" type="text">
<p id="result">Resultado</p>
<button type="button" onclick="suma()"> Suma.</button>
```

# TIPOS DE VARIABLES

- TIPOS DE REFERENCIA
  - SEMEJANTE A CLASES EN PROGRAMACIÓN
  - CREADOS A PARTIR DE NEW
  - PARÉNTESIS NO OBLIGATORIOS SIN PARÁMETROS (PERO RECOMENDABLES!!)
  - UN STRING POSEE MÉTODOS PROPIOS.
    - var miCadenaDeTexto = «La cadena de texto almacenada»;
    - miCadenaDeTexto.length;
  - PODEMOS MODIFICAR LOS DATOS POR VALOR O POR REFERENCIA
    - POR VALOR SE LE ASIGNA UNA COPIA Y NO SE MODIFICA EL ORIGINAL
    - POR REFERENCIA SE MODIFICA EL OBJETO EN SÍ, Y LAS REFERENCIAS, AL OBSERVAR EL MISMO OBJETO, LO VEN MODIFICADO.

# TIPOS DE VARIABLES

- MODIFICACIÓN POR REFERENCIA

```
<script type="text/javascript">
function probemos(){
    var fechaDeMiCumple=new Date(1978,10,3);
    var otraFecha = fechaDeMiCumple;
    alert(fechaDeMiCumple);
    alert(otraFecha);
    otraFecha.setFullYear(1999,11,31);
    alert(fechaDeMiCumple);
    alert(otraFecha);
}
probemos();
</script>
```

# TIPOS DE VARIABLES

- TIPO OBJECT
  - BASE PARA LOS OBJETOS
  - SE TRANSFORMA EN TIPOS PRIMITIVOS
  - var valor1 = new Object(72);
  - var valor2 = new Object(true);
  - var valor3 = new Object(«esto es un texto»);

# TIPOS DE VARIABLES

- TIPO BOOLEAN OBJETO
  - SE PUEDEN CREAR PRIMITIVOS U OBJETOS.
  - NUNCA SE UTILIZAN OBJETOS BOOLEANOS
  - CUALQUIER OBJETO, SEA BOOLEANO O NO, SE CONSIDERA TRUE, SEA CUAL SEA SU VALOR INTERNO.
  - PUEDE LLEGAR A CONFUNDIR EL USO DE OBJETOS BOOLEAN
    - primitivoATrue && objetoAFalse
      - El resultado es true, aunque debería ser false;

# TIPOS DE VARIABLES

- TIPO NUMBER OBJETO
  - SE CREAN OBJETOS TIPO NUMBER
    - ENTERO O DECIMAL
    - CON valueOf() OBTENEMOS EL PRIMITIVO
      - miObjetoNumerico.valueOf()¨;
    - POSEE UN MÉTODO PARA AJUSTAR DECIMALES.
      - var numeroConDecimales = new Number(2.3425234);
      - numeroConDecimales.toFixed(); //2
      - numeroConDecimales.toFixed(4);//2.3425
      - ES PELIGROSO AL SER UNA OPERACIÓN DE COMA FLOTANTE. UN DECIMAL A REDONDEAR COMO 5 PUEDE INCREMENTAR O DECREMENTAR ALEATORIAMENTE.

# Ejercicio

- Usando la plantilla para la calculadora:
  - Formatear un número decimal escrito por el usuario con la precisión solicitada por él mismo.

# TIPOS DE VARIABLES

- TIPO STRING OBJETO
  - GRAN CANTIDAD DE MÉTODOS Y UTILIDADES
- OPERADOR INSTANCEOF
  - SUSTITUTO DE TYPEOF, AL NO PODER DISTINGUIR LOS OBJETOS.
  - SOLO DEVUELVE TRUE O FALSE
  - SE DEBE CONOCER EL OBJETO EN CONCRETO PARA PODER DETECTARLO

```
var x=new Object(23)
undefined
x instanceof Number
true
```

# OPERADORES

- ASIGNACIÓN «=»
  - ASIGNA VALORES A VARIABLES
- INCREMENTO Y DECREMENTO

```
<script type="text/javascript">
    function probandoIncrementales(){
        var numero = 7;
        alert("VALOR ACTUAL : " + numero);
        alert("PREINCREMENTAL : " + ++numero);
        alert("VALOR ACTUAL : " + numero);
        alert("POSTINCREMENTAL : " + numero++);
        alert("VALOR ACTUAL : " + numero);
    }
    probandoIncrementales();
</script>
```

# OPERADORES

- NEGACIÓN
  - Es el operador **!**
  - DEVUELVE LA OPERACIÓN CONTRARIA LÓGICA
  - EN CASO DE NEGAR NÚMEROS, USA LA LÓGICA ANTERIORMENTE DEFINIDA
    - 0 -> FALSE -> NEGACION TRUE
    - DISTINTO DE 0 -> TRUE -> NEGACIÓN FALSE

# OPERADORES

- AND Y OR
  - AND: **&&**
  - OR: ||
  - OPERADOR QUE EVALUA CONDICIONES LÓGICAS
  - AND NECESITA QUE TODOS LOS VALORES SEAN TRUE PARA DEVOLVER TRUE
  - OR NECESITA QUE TODOS LOS VALORES SEAN FALSE PARA DEVOLVER FALSE

# OPERADORES

- MATEMÁTICOS
  - SUMA (+), RESTA (-), MULTIPLICACIÓN (*) Y DIVISIÓN (/)
  - MÓDULO (%) (EL RESTO DE LA DIVISIÓN)
- RELACIONALES
  - MAYOR QUE(>), MENOR QUE(<), MAYOR O IGUAL QUE (>=), MENOR O IGUAL QUE (<=), IGUAL (==) Y DISTINTO (!=)
  - RESULTADO SIEMPRE UN BOOLEANO
  - LOS COMPARADORES PUEDEN USARSE PARA CADENAS DE TEXTO.
    - CRITERIO A< -- < z
    - COMPARA LETRA A LETRA Y AL ENCONTRAR DISCORDANCIAS, USA EL CRITERIO SUPERIOR.
  - OPERADOR ===
    - SIRVE PARA COMPARAR DOS ELEMENTOS SIN HACER CONVERSIONES
      - EN CASO DE COMPARAR 7 TEXTO Y 7 NUMERICO, == DEVUELVE TRUE, SIN EMBARGO === DEVUELVE FALSE

# Ejercicios

- Crea una función que dado un número diga si es o no número primo

- Crearemos un página con dos recuadro de texto y dos botones.
  - Si se pulsa el primero, se comparan los valores de ambos campos en modo numérico
  - Si se pulsa el segundo, se comparan los valores de ambos en modo texto.

# OBJETOS NATIVOS

- EXISTEN CLASES PRECONSTRUIDAS PARA SU USO EN JAVASCRIPT
  - ARRAY
    - CONSTRUCTOR CON NÚMERO DE ELEMENTOS
      - var miArray = new Array(5);//arary de 5 posiciones
    - LOS ELEMENTOS DEL ARRAY PUEDEN SER DE DISTINTOS TIPOS A LA VEZ.
    - SE PUEDE CONSTRUIR EN LA DECLARACIÓON DEL ARRAY O A POSTERIORI
    - var miArray = new Array("Esto es un ejemplo",true,7,5.33)

# OBJETOS NATIVOS

- OTRA FORMA DE CONSTRUCCIÓN SERÍA
  - var miArray2 = new Array();
  - miArray2[0]="Esto es un ejemplo"
  - miArray2[1]=true;
  - …
- EL ARRAY COMIENZA SIEMPRE EN LA POSICIÓN CERO
- EL TAMAÑO DEL ARRAY AUMENTA DE FORMA DINÁMICA

# OBJETOS NATIVOS

- El array en Javascript (cómo se suele usar en mongoDB)

```
>x=[]
[]
>x.push(2)
1
>x
[2]
>x.push(3)
2
>x
[2, 3]
>x.pop()
3
>x
[2]
>x.push(32)
2
>x
[2, 32]
>x.pop()
32
```

# FUNCIONES

- ARRAYS
  - .length: NUMERO DE ELEMENTOS DE UN ARRAY
  - .concat(): CONCATENACIÓN DE VARIOS ARRAYS
  - .join(separador): CONTRARIA A SPLIT, TRANSFORMA EL ARRAY EN UNA CADENA DE TEXTO CON EL SEPARADOR ASIGNADO.
  - .pop() ELIMINA EL ÚLTIMO ELEMENTO Y LO DEVUELVE
  - .shift() ELIMINA EL PRIMER ELEMENTO Y LO DEVUELVE

# FUNCIONES

- .unshift() AÑADE UN ELEMENTO EN EL ARRAY. SE PUEDEN AGREGAR TANTOS ELEMENTOS COMO SE QUIERA DE UNA VEZ.
- .reverse() INVIERTE EL ORDEN DE LOS ELEMENTOS

# Ejercicio

- Crear un programa que vaya almacenando datos en un array, y los muestre en pantalla si el usuario lo solicita

# OBJETOS NATIVOS

- DATE
  - REPRESENTACIÓN Y MANIPULACIÓN DE FECHAS.
  - REPRESENTACIÓN DEL NÚMERO DE MILISEGUNDOS DESDE 1970
  - POSEE UN CONSTRUCTOR DE MILISEGUNDOS
    - var miFecha = new Date(0); /1970
  - POSEE CONSTRUCTORES MAS INTUITIVOS
    - (AÑO,MES,DIA)
    - (AÑO,MES,DIA,HORA,MINUTO,SEGUNDO)
      - **LOS MESES SE CUENTAN DESDE CERO!**
        - 11 -> DICIEMBRE,0-> ENERO

# OBJETOS NATIVOS

- MÉTODOS DISPONIBLES
  - getTime()-> fecha en ms desde 1 de enero de 1970
  - getMonth()-> número de mes desde 0 a 11
  - getFullYear()->año en 4 cifras
  - getYear()->año en 2 cifras
  - getDate()->dia del mes
  - getDay()dia de la semana de 0(Domingo) a 6(Sábado)
  - getHours()->horas
  - getMinutes()->minutos
  - getSeconds()->segundos
  - getMilliseconds()->milisegundos
- POSEE TAMBIÉM SUS CORRESPONDIENTES SETTERS PARA MODIFICAR LOS DATOS.

# Ejercicio

- Hacer un programa que pida dos fechas (precisión de días) y diga cuántos segundos de diferencia hay entre ellas

- Crear un pequeño programa que nos permita llevar un histórico de incidencias. Una incidencia se inserta dando una fecha y un texto. Tendremos dos arrays, uno para fechas y otro para los textos (la posición de la fecha y la del texto deben coincidir para cada incidencia). Mostrar el histórico ordenado por fecha de incidencias usando el console.log() OPCIONAL: usar un único array, pero sin usar objetos ¿Podrías?

# OBJETOS NATIVOS

- FUNCTION
  - SE USA RARA VEZ
  - SE INDICAN MÚLTIPLES PARÁMETROS.
    - TODOS MENOS UNO SON LOS PARÁMETROS DE ENTRADA, Y EL ÚLTIMO SON LAS INSTRUCCIONES Y LA DEVOLUCIÓN DEL PARÁMETRO
    - var suma=new Function("a","b","return a+b")

```
var suma=new Function("a","b","return a+b")
undefined
suma
function anonymous(a,b
/**/) {
return a+b
}
suma(2,3)
5
```

# FUNCIONES

- NO SE SUELEN DEFINIR COMO CLASE (EJ. ANTERIOR)
- NO NECESITAN DEVOLVER UN VALOR. EN ESE CASO SE DEVUELVE POR DEFECTO UNDEFINED
- SE PUEDE INVOCAR UNA FUNCIÓN CON MENOS O MÁS PARÁMETROS.
  - CUANDO SE USAN MENOS, SE COMPLETAN CON UNDEFINED,
  - CUANDO SE USAN MÁS, SE IGNORAN.

# FUNCIONES

- UNA FUNCIÓN PERMITE OTRAS FUNCIONES EN SU INTERIOR
- function operaciones(a,b){
  - function suma(x,y){return x+y;}
  - return suma(a,a)+suma(b,b);
- }

# FUNCIONES

- SE PUEDE ACCEDER A LOS ARGUMENTOS DE UNA FUNCIÓN, AUNQUE NO HAYAN SIDO DECLARADOS PREVIAMENTE
  - arguments.length -> longitud del array
  - arguments[i] → dato en posición
- POSEE UNA PROPIEDAD LLAMADA CALLEE QUE NOS PERMITE SABER EL NÚMERO DE PARÁMETROS ESPERADO
  - Arguments.callee.length
- EL OBJETO ARGUMENTS NO ES UN ARRAY PERO PUEDE SER TRATADO COMO TAL.

# Ejercicios

- Ejercico: crea una función suma (definida en el código, no hace falta usar new) que permita agregar todos los parámetros que queramos

# FUNCIONES

- FUNCIONES BÁSICAS
  - CADENAS DE TEXTO
    - .length : NUMERO DE CARACTERES
    - + Ó .concat() : CONCATENACIÓN DE CADENAS
    - .toUpperCase(): PASO A MAYUSCULAS
    - .toLowerCase(): PASO A MINUSCULAS
    - .charAt(posicion):DEVUELVE EL CARÁCTER EN ESA POSICION
    - .indexOf(letra): PRIMERA POSICION DEL CARÁCTER INDICADO, O -1 SIN COINCIDENCIAS
    - .lastIndexOf(letra): ANÁLOGO A LA ANTERIOR
    - .substring(posicion): TROZO DE TEXTO DESDE LA POSICION INDICADA HASTA EL FINAL.
      - EN CASO DE QUE SE INDIQUE POSICIÓN NEGATIVA, DEVUELVE EL TEXTO COMPLETO

# FUNCIONES

- .substring(inicio,fin)
  - DEVUELVE EL TEXTO ENTRE LA POSICIÓN DE INICIO Y LA INMEDIATAMENTE ANTERIOR A LA DEL FINAL.
  - EN CASO DE INVERTIR LOS VALORES, JAVASCRIPT REORDENA LOS VALORES Y USA EL MENOR COMO PRIMERO Y EL MAYOR COMO SEGUNDO
- .split(separador)
  - TRANSFORMA UN TEXTO EN UN ARRAY USANDO COMO SEPARADOR EL TEXTO INDICADO.
  - SI NO SE INDICA NADA, SE SEPARAN POR LETRAS.

# Ejercicio

- Realizar un programa que, dado un texto de entrada, cuente cuántas veces aparece cada palabra (usad arrays asociativos).
  **NOTA**: para obtener las keys de un array asociativo usar "Object.keys(elArray)"

- Hacer un validador de contraseñas que dado un texto, diga si es bueno o no como contraseña. Para ser bueno ha de tener al menos 8 caracteres y empezar por mayúscula

# EXPRESIONES REGULARES

- PATRONES DE COINCIDENCIAS
- EJEMPLO WILDCARDS DE DOS
  - ? Y *
- LAS EXPRESIONES REGULARES SON UNA FORMA SENCILLA PARA
  - MANIPULAR DATOS
  - BUSQUEDA
  - REEMPLAZO DE STRINGS

# EXPRESIONES REGULARES

- SINTAXIS
  - PODEMOS CREAR LAS EXPRESIONES POR OBJETO O VARIABLE
    - Var reg = /patron/flags
    - New RegExp("patron","flags");
  - FLAGS
    - BUSQUEDAS GLOBALES (g)
      - DEVUELVE UN ARRAY CON LAS OCURRENCIAS
    - IGNORE CASE (i)
      - NO FUNCIONA CON CARACTERES EXTENDIDOS
    - ENTRADA MULTILINEA (m)
      - LOS TAGS DE INICIO Y FIN SIRVEN POR CADA LINEA

# EXPRESIONES REGULARES

- PATRONES
  - \ ESCAPE DE LITERALES
  - {N} EXACTAMENTE N VECES
  - {N,} N O MAS VECES
  - {N,M} DE N A M VECES
  - * EQUIVALENTE A {0,}
  - + EQUIVALENTE A {1,}
  - ? EQUIVALENTE A {0,1}

# EXPRESIONES REGULARES

- (PATRON)  CAPTURA LA COINCIDENCIA
- (?:PATRON) INDICA LA COINCIDENCIA PERO NO LA CAPTURA
- | OR (a|b)  CAPTURA A Y B, NO TIENEN QUE ESTAR SEGUIDOS
- [CARACTERES] INDICA LOS CARACTERES COINCIDENTES
  - [a-d] BUSCA "A B C D"
- [^CARACTERES] INDICA SI NO ES COINCIDENTE
  - [^a-d] BUSCA CUALQUIERA EXCEPTO "A B C D"

# EXPRESIONES REGULARES

- CARACTERES ESPECIALES
  - ^ INICIO DE LA ENTRADA
  - $ FIN DE LA ENTRADA
  - . CUALQUIER CARÁCTER MENOS SALTO DE LINEA

# EXPRESIONES REGULARES

- CARACTERES DE ESCAPE
  - \f FORM-FEED
  - \r RETURM
  - \n SALTO DE LINEA
  - \t TABULADOR HORIZONTAL
  - \v TABULADOR VERTICAL
  - \0 CARÁCTER NULO
  - \s ESPACIO EN BLANCO
  - \S CUALQUIERA EXCEPTO ESPACIO EN BLANCO

# EXPRESIONES REGULARES

- \w CARACTERES ALFANUMERICOS
- \W TODO MENOS CARACTERES ALFANUMERICOS
- \d DIGITOS
- \D TODO MENOS DIGITOS
- \cX captura control + carácter X
- …

# EXPRESIONES REGULARES

- USO EN JAVASCRIPT
  - PATRON.exec(STRING)
    - DEVUELVE LA INFORMACIÓN COINCIDENTE
  - PATRON.test(STRING)
    - DEVUELVE EL BOOLEANO DE LA COINCIDENCIA
  - STRING.match(PATRON)
    - CON g DEVUELVE EL ARRAY CON COINCIDENCIAS, O NULL
  - .search(PATRON)
    - DEVUELVE LA POSICIÓN DE LA PRIMERA COINCIDENCIA

# EXPRESIONES REGULARES

- STRING.replace(PATRON,STRING)
  - DEVUELVE EL STRING EDITADO
- STRING.split(PATRON)
  - STRING TRANSFORMADO EN UN ARRAY (usando el patrón como separador)

# Ejercicios

- Volver a realizar la práctica del validador de contraseñas, pero usando ahora una expresión regular.
- Crear un programa que valide la expresión regular para:
  - Un correo electrónico
  - Un código postal de UK
  - Un teléfono (tanto con como sin prefijo internacional)

# EJERCICIOS

- Crear expresiones regulares que reconozcan:
  - La cadena que contenga una b, precedida de 0 o más a ("aaaab", "ab", "b"…)
  - La cadena "ab" o "b“
  - La cadena formada por 1 o más a seguidas de 1 o más b ("aaaaab", "ab", "abbbb", "aaabbb"…)
  - La cadena que empieza por "main" y después contiene unos paréntesis ("main()", "main123(234)"…)
  - La cadena que acaba por \

# EXPRESIONES REGULARES

- Back reference:
  - En ocasiones es de suma utilidad vincular el resultado parcial de una expresión con otra zona de la misma. Esto es el denominado Back Reference.
    - Con () agrupamos una expresión
    - Con \\n indicamos que esperamos aquí el mismo resultado obtenido en el grupo indicado por n (n es el total de paréntesis abiertas existentes para llegar ahí)
    - Podemos acceder a un subgrupo con exec(), que da un array con los grupos (0 es el total)

# EXPRESIONES REGULARES

- Ejemplo
  - Vamos a buscar las expresiones que empiecen y acaban por el mismo número de as

```
var re=new RegExp("(a+)[bcd]+\\1","g")
"aabcaa".match(re)
    ["aabcaa"]
```

# Ejercicios

- Busca líneas que tengan al menos 6 letras iguales 2 a 2 (la 1-2, la 3-4 y la 5-6). (bookkeeper)
- Busca palíndromos de 6 letras (abccba)
- Validar una ip

# JAVASCRIPT

- OBJETOS
- CLASES
- OTROS CONCEPTOS

# OBJETOS

- DEFINIDOS PARA ORGANIZAR EL CÓDIGO Y ENCAPSULAR LOS MÉTODOS Y FUNCIONES
- ARRAY ASOCIATIVO FORMADO POR MÉTODOS Y PROPIEDADES DEL OBJETO
- UN ARRAY ASOCIATIVO ES UN ARRAY QUE NO POSEE ASOCIACIÓN NUMÉRICA, SINO ASOCIADO A TEXTOS.

```
var arrayAsociativo = new Array();
arrayAsociativo['uno']=1;
arrayAsociativo['dos']=2;
arrayAsociativo['tres']=3;
```

# OBJETOS

- SE PUEDE ACCEDER VIA NOTACIÓN DE PUNTOS
  - arrayAsociativo.uno
- COMO UN OBJETO ES UN ARRAY ASOCIATIVO, EL ACCESO MÁS RÁPIDO ES LA NOTACIÓN DE PUNTOS
- PARA ASIGNAR UN VALOR A UNA PROPIEDAD, NO ES NECESARIO QUE LA PROPIEDAD HAYA SIDO DEFINIDA.

# OBJETOS

- PODEMOS ASIGNAR MÉTODOS MEDIANTE LA NOTACIÓN DE PUNTOS
  - arrayAsociativo.getUno = function() {
    - this.uno;
  - }
- SE PUEDE USAR LA PALABRA RESERVADA THIS PARA ACCEDER A PROPIEDADES DEL OBJETO.
- SE PUEDE ASOCIAR A UNA FUNCIÓN DECLARADA CON ANTERIORIDAD
  - arrayAsociativo.nuevaFuncion = otraFuncionYaCreada;

# OBJETOS

- PODEMOS ALMACENAR OBJETOS DENTRO DE OBJETOS

```
var trabajador = new Objetc();
trabajador.nombre = "Ruben";
trabajador.apellido1 = "Gomez";
trabajador.apellido2 = "Garcia";
trabajador.getNombreCompleto = new function(){
    return this.nombre+" "+this.apellido1+ " "+ this.apellido2;
};
trabajador.direccion = new Object();
trabajador.direccion.nombre = "Mayor";
trabajador.direccion.tipo = "Plaza";
trabajador.direccion.numero = 1;
```

# OBJETOS

- call();
  - EJECUTA UNA FUNCIÓN COMO SI FUERA UN MÉTODO DE OTRO OBJETO.

```
function multiplica(x){
    return this.valor*x
}
var unObjetoNuevo = new Object();
unObjetoNuevo.valor=7;
multiplica.call(unObjetoNuevo,2);
```

  - Es como si multiplica fuera un método de unObjetoNuevo y se llamara con el parámetro 2
- apply() COMO CALL, PERO ADMITE UN ARRAY COMO LISTA DE PARÁMETROS

# OBJETOS

- NOTACIÓN JSON
  - JAVASCRIPT OBJECT NOTATION
  - FORMATO SENCILLO PARA INTERCAMBIO DE INFORMACIÓN
  - ALTERNATIVA AL FORMATO XML
  - XML SIGUE SIENDO MUY SUPERIOR
  - TIPO MIME application/json

# OBJETOS

- CON JSON, LA CREACIÓN DE ARRAYS ES MAS SENCILLA
  - var prueba = [“uno”,”dos”,”tres”];
- LA CREACIÓN DE ARRAYS ASOCIATIVOS TAMBIÉN ES MAS SENCILLA
  - var pruebaAsociativa = {uno:1,dos:2,tres:3}
    - CONTENIDOS DE OBJETOS CON { Y }
    - SEPARACIÓN DE ELEMENTOS CON ,
    - CLAVE Y VALOR PARA ARRAYS ASOCIATIVOS CON :
    - CREACIÓN DE ARRAYS CON [ Y ]

# OBJETOS

- var array = [valor1,valor2,…,valorn];
- Var objeto = {clave1:valor1,clave2:valor2,…};
- ^Permite generar estructuras complejas anidadas de una sola vez
- Se puede usar notación tradicional y notación JSON a la vez.
- Podemos usar JSON.stringify(objeto) para mostrar el JSON
- Podmeos obtener un objeto con

  var ob=JSON.parse('{"x":1}')

# Práctica

- Vamos a crear una página que permite añadir usuarios (formados por nombre, edad, y preferencias). Las preferencias del usuario se añaden una a una a un array.
- Tendremos también un listado de "canciones" formadas por un tipo y un título.
- La finalidad es poder mostrar un listado de sugerencias de canciones para un usuario.

# CLASES

- CONCEPTO BASE DE LA ORIENTACIÓN A OBJETOS.
- DEFINICIÓN DE PLANTILLAS DE LAS CUALES SE CREARÁN LOS OBJETOS
- JAVASCRIPT NO PERMITE EL USO DE CLASS, SOLO PARA FUTURAS VERSIONES

# CLASES

- FUNCIONES CONSTRUCTORAS
  - EN JAVASCRIPT NO EXISTEN LOS CONSTRUCTORES
  - SE EMULAN POR MEDIO DE FUNCIONES
  - AL CREAR EL NUEVO OBJETO, REALMENTE SE LLAMA A LA FUNCIÓN

```
function Trabajador(idTrabajador, nif){
    this.idTrabajador = idTrabajador;
    this.idNif = nif;
}
new Trabajador("pepe","123123");
```

# CLASES

- PROTOTYPE
  - LAS FUNCIONES CONSTRUCTORAS PUEDEN DEFINIR PROPIEDADES, PERO TAMBIÉN PUEDEN DEFINIR LOS MÉTODOS

# CLASES

```
function Trabajador(id,nombre,nif){
    this.id = id;
    this.nombre = nombre;
    this.nif = nif;

    this.getId = function(){
        alert(this.id);
    }
    this.getNombre = function(){
        alert(this.nombre);
    }
    this.getNif = function(){
        alert(this.nif);
    }
}
function muestraDatos(){
    var trabajador = new Trabajador(1,"Ruben","771281230B");
    trabajador.getNombre();
    trabajador.getNif();
    var trabajador2 = new Trabajador(2,"Pedro","774881230B");
    trabajador2.getNombre();
    trabajador2.getNif();
}
muestraDatos();
```

# CLASES

- Cada vez que se instancia un objeto, el constructor crea nuevas funciones, produciendo un consumo excesivo de recursos.
- Para ello, se puede crear un objeto prototipo, del cual heredarán los demás objetos.
- prototype es el molde con el que los objetos se crearán de forma efectiva.
- si se modifica el prototipo, todos los objetos son modificados.

# CLASES

- Los prototipos deben ser de constantes y funciones, no de datos, ya que sino, provocamos el acceso concurrente desde distintos objetos.

# CLASES

```
function Trabajador(id,nombre,nif){
    this.id = id;
    this.nombre = nombre;
    this.nif = nif;
}
Trabajador.prototype.getId = function(){
    alert(this.id);
}
Trabajador.prototype.getNombre = function(){
    alert(this.nombre);
}
Trabajador.prototype.getNif = function(){
    alert(this.nif);
}
function muestraDatos(){
    var trabajador = new Trabajador(1,"Ruben","771281230B");
    trabajador.getNombre();
    trabajador.getNif();
    var trabajador2 = new Trabajador(2,"Pedro","774881230B");
    trabajador2.getNombre();
    trabajador2.getNif();
}
muestraDatos();
```

# Ejercicios

- Crea una función prototype para la clase Array, llamada "indexOf(Object)" que recibe un objeto, lo busca en el array, y devuelve su posición (o -1 si no existe)

# CLASES

- UNA DE LAS POSIBILIDADES MAS INTERESANTES ES LA DE MODIFICAR PROPIEDADES Y MÉTODOS DE OBJETOS PREDEFINIDOS
- CREEMOS UN ARRAY CON LA FUNCIÓN DE DEVOLVER LA POSICIÓN DE UN OBJETO
  - Array.prototype.indexOf = function(…){
    - …
  - }

# CLASES

- EXISTE UNA LIBRERÍA LLAMADA Prototype QUE USA LA CAPACIDAD DE prototype PARA AUMENTAR LAS CAPACIDADES DE LOS OBJETOS DE JAVASCRIPT.
- PODEMOS GENERAR MÉTODOS COMO TRIM O AGREGAR ELEMENTOS SIN REPETICIÓN EN UN ARRAY.

# CLASES

- HERENCIA Y AMBITO
  - JAVASCRIPT NO POSEE AMBITOS COMO OTROS LENGUAJES DE PROGRAMACIÓN
  - (PUBLIC, PRIVATE, PROTECTED)
  - UNA TÉCNICA QUE SIMULA PROPIEDADES PRIVADAS SE BASE N DECLARAR LAS REFERENCIAS CON _ DELANTE DEL NOMBRE DE LA VARIABLE.

# OTROS CONCEPTOS

- EXCEPCIONES
  - DEFINIDO POR LAS PALABRAS RESERVADAS
    - TRY{
      - DEBE CONTENER EL CODIGO A PROBAR
    - }
    - CATCH(EXCEPTION){
      - CODIGO A EJECUTAR EN CASO DE FALLO
    - }
    - FINALLY{
      - SE EJECUTA SIEMPRE, PASE LO QUE PASE EN TRY O CATCH
    - }

# OTROS CONCEPTOS

- DESPUÉS DE UN TRY DEBE HABER OBLIGATORIAMENTE UN CATCH O UN FINALLY
- AUNQUE SE EJECUTE UN RETURN DENTRO DEL TRY, SE SIGUE EJECUTANDO EL FINALLY.
- SE PERMITE EL LANZAMIENTO DE EXCEPCIONES MANUALMENTE
  - throw new Error('Esto es un error!');

# OTROS CONCEPTOS

- REFLEXION
  - CAPACIDAD DE OBTENER INFORMACIÓN DE SI MISMO Y DE AUTOMODIFICARSE EN TIEMPO DE EJECUCIÓN.
  - DESCUBRIMIENTO DE MÉTODOS Y PROPIEDADES DE OBJETOS JAVASCRIPT
- if(objeto.propiedad) {
  - *True si el objeto tiene la propiedad*
- }

# OTROS CONCEPTOS

- SI LA PROPIEDAD DEVUELVE UN FALSE, NULL O 0, NO SE EJECUTA CORRECTAMENTE!
  - if(typeof(objeto.propiedad) != 'undefined') {
    - *// ahora sí!*
  - }

# Práctica

Vamos a simular un pequeño juego de combate.

Para ello vamos a crear 2 objetos que deben tener:

- nombre (texto)
- hp (número)
- stamina (número)
- weapon (objeto con alguna o varias propiedades hp y stamina)

El juego consiste en llamar varias veces a una función

atacar(o1,o2) y atacar(o2,o1). Esta función permite aplicar los efectos que posea el arma del primera parámetro al segundo parámetro. Si el primer parámetro tiene vida 0 se muestra que ha muerto, si tiene stamina 0 dice "muy cansado para atacar". Cada vez que un objeto ataca reduce su stamina en 1 punto. Si no ataca por tener stamina 0, recarga un random de stamina de entre 1 y 3 puntos.

# Práctica

- Crear un interfaz para una biblioteca que permita:
  - Gestionar libros (darlos de alta y de baja)
  - Gestionar usuarios (darlos de alta y baja)
  - Gestionar préstamos (usuario y libro, ambos deben existir)

Opcional: un botón para mostrar todos los datos como JSON. Permitir también la carga a través de un JSON de estos datos.

# Phaser

- Para poner en práctica estos conceptos de JavaScript vamos a intentar desarrollar un pequeño juego en JavaScript usando Phaser, un motor de creación de videojuegos.

# Servidor Web

- Por temas de seguridad, debemos ejecutar nuestro juego tras un servidor Web, para ello vamos a usar Moongose, por su simplicidad.

https://www.cesanta.com/products/binary

# Phaser.js

- Para descargar Phaser, tan sólo debemos ir a la web y descargar la librería js de phaser

https://phaser.io/

# Frontal HTML

- Como punto de entrada a nuestro juego, crearemos una página web HTML que cargará tanto la librería phaser como nuestro juego

# Frontal HTML

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Phaser Mega Game</title>
    <script src="lib/phaser.min.js"></script>
    <script src="game2.js"></script>
</head>
<body>
</body>
</html>
```

# El motor Phaser

Mediante el constructor Phaser.Game() podemos crear una instancia de un juego, este recibe (al menos) los siguientes parámetros:

- Ancho: del juego
- Alto: del juego
- Renderer: con Phaser.AUTO se autodetecta
- Parent: el elemento del DOM donde se ejecutará el juego
- Documento con las funciónes preload, créate y update.

# El motor Phaser

- Phaser divide la lógica en tres métodos distintos:
  - preload: Dedicado a la carga de recursos (imágenes, mápas de sprites…)
  - create: Dedicado a la incialización del juego
  - update: Se ejecutará en cada ciclo de nuestro juego, es la encargada de vigilar las colisiones, cambiar animaciones, etc…

# Creando un juego

```
var game = new Phaser.Game(800, 600, Phaser.AUTO, '',
    { preload: preload, create: create, update: update });

function preload() {
}

function create() {
}

function update() {
}
```

# Creando un juego

- Además de crear el objeto Game, declaramos las variables que vayamos a usar:

```
var width = 480;
var height = 320;
var game = new Phaser.Game(wid
var player;
var food;
var cursors;
var speed = 175;
var score = 0;
var scoreText;
```

# Preload

- Este método se invocará una vez, al crear el juego. En él vamos a cargar los recursos que van a ser usados durante el juego.

```
function preload() {
    game.stage.backgroundColor = '#eee';
    game.load.image('player', 'asset/blue-square.png');
    game.load.image('food', 'asset/Pollo.png');
}
```

# Preload

- En el ejemplo anterior cambiamos la propiedad "stage.backgroundColor" del objeto Game, para cambiar el color del fondo del juego.
- Tambien usamos la función load.image() para cargar una imagen, siendo el primer parámetro el nombre con el que la referenciaremos y el segundo la ruta relativa de la misma.

# Preload

- En el ejemplo anterior cambiamos la propiedad "stage.backgroundColor" del objeto Game, para cambiar el color del fondo del juego.
- Tambien usamos la función load.image() para cargar una imagen, siendo el primer parámetro el nombre con el que la referenciaremos y el segundo la ruta relativa de la misma.

# create

- En esta función inicializamos el juego, nos encargamos de definir las entidades, sus comportamientos, capturamos los controles, etc.

# create

```
function create() {
    game.physics.startSystem(Phaser.Physics.ARCADE);
    cursors = game.input.keyboard.createCursorKeys();
    player = game.add.sprite(width*0.5, height*0.5, 'player')
    player.anchor.set(0.5);
    game.physics.enable(player, Phaser.Physics.ARCADE);
    player.body.collideWorldBounds = true;
    food = game.add.group();
    food.create(width*0.1, height*0.1, 'food');
    food.create(width*0.9, height*0.1, 'food');
    food.create(width*0.1, height*0.9, 'food');
    food.create(width*0.9, height*0.9, 'food');
    for (var i in food.children) {
        food.children[i].anchor.set(0.5);
    }
    game.physics.enable(food, Phaser.Physics.ARCADE);
    scoreText = game.add.text(5, 3, score);
}
```

# Physics

- Physics.startSystem(): Con este método indicamos al motor qué sistema de físicas vamos a usar.  Existen varios tipos distintos en phaser, nosotros vamos a usar ARCADE

# Controls

- Los controles los tenemos que capturar, para ello tenemos que
  - Crear una variable que hará referencia a la tecla que queremos capturar
  - Capturar las teclas que queramos

# Controls

```
spaceKey =
game.input.keyboard.addKey(Phaser.Keyboard.SPACEBAR);
izquierda =
game.input.keyboard.addKey(Phaser.Keyboard.LEFT);
derecha =
game.input.keyboard.addKey(Phaser.Keyboard.RIGHT);
game.input.keyboard.addKeyCapture([
    Phaser.Keyboard.SPACEBAR,
    Phaser.Keyboard.LEFT,
    Phaser.Keyboard.RIGHT ]);
```

# Sprites

- Los sprites son las entidades del juego, Phaser está preparado para poder gestionarlas y detectar colisiones, overlappings…
- Para añadir un sprite hay que usar la función add.sprite(), indicando sus coordenadas x,y y la imagen asociada

# Sprites

```
player = game.add.sprite(0,0, 'player');
```

El ejemplo anterior creará un sprite que almacenará en la variable "player", en las posiciones x=0, y=0, usando la imagen cargada asociada a "player"

# Sprites

Si queremos que la física afecte a nuestros sprites, tenemos que indicarlo explícitamente

```
game.physics.enable(player);
```

# Sprites

También podemos especificar que queremos que el sprite choque con los bordes del juego

```
player.body.collideWorldBounds = true;
```

# Sprites

En ocasiones es mucho más cómodo tratar un grupo de sprites como una única entidad, para ello podemos crear un grupo e ir añadiendo sprites

```
food = game.add.group();
food.create(width*0.1, height*0.1, 'food');
food.create(width*0.9, height*0.1, 'food');
food.create(width*0.1, height*0.9, 'food');
food.create(width*0.9, height*0.9, 'food');
game.physics.enable(food, Phaser.Physics.ARCADE);
```

# Update

La función update se invoca cada ciclo, y es la encargada de llevar la lógica del juego.

En ella podemos ver por ejemplo si un sprite choca con otro o está por encima de él, y llamar a funciones para actuar en consecuencia

# Update

```
function update() {
    if (cursors.up.isDown) {
        player.body.velocity.y = -speed;
    }
    else if (cursors.down.isDown) {
        player.body.velocity.y = speed;
    }
    else {
        player.body.velocity.y = 0;
    }
    if (cursors.left.isDown) {
        player.body.velocity.x = -speed;
    }
    else if (cursors.right.isDown) {
        player.body.velocity.x = speed;
    }
    else {
        player.body.velocity.x = 0;
    }
    game.physics.arcade.overlap(player, food, eatFood);
}
```

# Update

- En el ejemplo anterior, cada vez que el sprite "player" toque un objeto del grupo "food", se llama a la función "eatFood". Esta recibirá ambos objetos como parámetros.

# Update

```
function eatFood(player, food) {
    food.kill();
    score++;
    scoreText.text = score;
}
```

# Y ya está