

3. Funciones

Programación estructurada

Cuando un programa crece:

- Es importante mantenerlo ordenado
- No repetir código
- Agrupar el código según su función
- Dar nombre a las operaciones comunes

```
var cantidad = prompt("¿Cuántas entradas?");  
cantidad = parseInt(cantidad);
```

```
if (isNaN(cantidad) || cantidad <= 0) {  
    cantidad = 1;  
} else if (cantidad > 100) {  
    cantidad = 100;  
}
```

```
var precio = prompt("¿El precio de una entrada?");  
precio = parseInt(precio);
```

```
if (isNaN(precio) || precio <= 0) {  
    precio = 1;  
} else if (precio > 100) {  
    precio = 100;  
}
```

```
alert("Vas a recaudar: " + (precio*cantidad) + " €");
```

```
var cantidad = prompt("¿Cuántas entradas?");  
cantidad = parseInt(cantidad);
```

```
if (isNaN(cantidad) || cantidad <= 0) {  
    cantidad = 1;  
} else if (cantidad > 100) {  
    cantidad = 100;  
}
```

```
var precio = prompt("¿El precio de una entrada?");  
precio = parseInt(precio);
```

```
if (isNaN(precio) || precio <= 0) {  
    precio = 1;  
} else if (precio > 100) {  
    precio = 100;  
}
```

```
alert("Vas a recaudar: " + (precio*cantidad) + " €");
```

Programación estructurada

¿No sería más fácil hacerlo así?

```
var cantidad = promptNumero("¿Cuántas entradas?", 1, 100);  
var precio = promptNumero("¿El precio de una entrada?", 1, 100);  
  
alert("Vas a recaudar: " + (precio*cantidad) + " €");
```

Programación estructurada

¿No sería más fácil hacerlo así?

```
var cantidad = promptNumero("¿Cuántas entradas?", 1, 100);  
var precio = promptNumero("¿El precio de una entrada?", 1, 100);  
  
alert("Vas a recaudar: " + (precio*cantidad) + " €");
```

Funciones

```
function promptNumero (param1, param2, param3) {  
  // Cuerpo de la función  
  // ...  
  return undefined;  
}
```

Tres elementos importantes:

- palabra clave **function**
- lista de parámetros
- valor de retorno

Funciones

Ejercicio:

Escribe una función que devuelva siempre el número 7

Funciones

Así definimos la función:

```
function siete () {  
    return 7;  
}
```

La podemos invocar:

```
siete();
```

Funciones

Ejercicio:

Escribe una función que devuelva un número aleatorio entre 1 y 6

Funciones

```
function tirada () {  
  return 1 + Math.floor(Math.random() * 6);  
}
```

```
> tirada()  
2
```

Funciones

¿Qué devuelve esta función?

```
function nada () {  
}
```

Funciones

¿Qué devuelve esta función?

```
function seis_o_siete () {  
    return 6;  
    return 7;  
}
```

Funciones

Parámetros:

- Se nombran al definir la función
- Dentro de la función se comportan como variables
- Los valores se especifican en la invocación
- Se asignan por posición

Funciones

Ejercicio:

Escribe una función que devuelva el valor que se le pasa como parámetro

```
> a(1)
1
> a("Beast of Burden")
"Beast of Burden"
```

Funciones

```
function a (parametro) {  
    return parametro;  
}
```


Funciones

¿Qué devuelve esto?

$a(a)$

Funciones

```
function a (parametro) {  
    return parametro;  
}
```

```
var que = a;
```

```
// Qué devuelve?
```

```
que(1986);
```

```
var b = que(a);
```

```
// Qué devuelve?
```

```
b(b);
```

Funciones

¿Qué devuelve esto?

a()

Funciones

Ejercicio:

Escribe una función que reciba dos parámetros, a y b, y devuelva la suma $a + b$



Funciones

```
function suma (a, b) {  
  return a + b;  
}
```

```
> suma(10, 5)  
15  
-----  
> suma(1, 2)  
3
```

Funciones

```
function suma (a, b) {  
    return a + b;  
}
```



```
> suma(10, 5)  
15  
> suma(1, 2)  
3
```

Funciones

¿Qué devuelve?

```
suma(10, (5 - 3));
```

Funciones

¿Qué devuelve?

```
suma(10, suma(5, 5))
```


Funciones

Ejercicio:

Escribe una función que:

- Si recibe dos parámetros: devuelva el producto de multiplicar el primero por el segundo
- Si recibe solo un parámetro: devuelva el primer parámetro multiplicado por si mismo

Funciones

Una función se puede llamar a sí misma

```
function suma2 (a, b) {  
  if (b == 0) {  
    return a;  
  } else {  
    return suma2(a + 1, b - 1);  
  }  
}
```

Funciones

Una función puede llamar a otra función

```
function par (n) {  
  if (n == 0) {  
    return true;  
  } else {  
    return impar(n - 1);  
  }  
}
```

```
function impar (n) {  
  if (n == 0) {  
    return false;  
  } else {  
    return par(n - 1);  
  }  
}
```

Funciones

Ejercicio:

Escribe una función recursiva que:

Reciba un número natural como parámetro

Devuelva la suma de todos los números naturales desde 0 hasta el número pasado como parámetro

sumaRango(10)

$10 + 9 + 8 + 7 + 6 + 5 + 4 + 3 + 2 + 1$

-> **55**

Funciones

Ejercicio:

Escribe una función que:

- Reciba una cadena de caracteres como parámetro
- Le añada al final “, por favor”

```
> conEducacion("pásame la sal")  
"pásame la sal, por favor"
```

Funciones

Ejercicio:

Escribe una función que:

- Reciba una cadena de caracteres como parámetro
- Le añada al final “, date prisa!!!”

```
> conPrisas("pásame la sal")  
"pásame la sal, date prisa!!!"
```

Funciones

Ejercicio:

Escribe una función **di** que puede recibir uno o dos parámetros

- El primer parámetro siempre es una cadena de caracteres
- El segundo parámetro puede ser una función o nada
- Si el segundo parámetro está vacío, simplemente muestra el primer parámetro por la consola
- Si no, muestra en la consola el resultado de llamar a la función que hay en el segundo parámetro con el primer parámetro

Funciones

```
> di("Hola")
```

```
Hola
```

```
< undefined
```

```
> di("cierra la puerta", conEducacion)
```

```
cierra la puerta, por favor
```

```
< undefined
```

```
> di("cierra la puerta", conPrisas)
```

```
cierra la puerta, date prisa!!!
```

```
< undefined
```


Funciones

```
function misterio (mensaje, filtro) {  
  if (filtro) {  
    mensaje = filtro(mensaje);  
  }  
  console.log(mensaje);  
}
```

Funciones

Las funciones pueden ser anónimas

```
> di("tengo sed", function (mensaje) {  
  return "El personaje se levantó y dijo: " + mensaje;  
})
```

```
El personaje se levantó y dijo: tengo sed
```

```
← undefined
```

Funciones

¿Qué muestran las llamadas a **experimento**?

```
var color = "Azul";
```

```
function experimento () {  
  console.log(color);  
}
```

```
experimento();
```

```
var color = "Verde";
```

```
experimento();
```

Funciones

¿Y ahora?

```
var color = "Azul";
```

```
function experimento () {  
  var color = "Amarillo";  
  console.log(color);  
}
```



```
experimento();
```

```
var color = "Verde";
```

```
experimento();
```

Funciones

¿Qué muestra la última línea?

```
var color = "Azul";
```

```
function experimento () {  
  var color = "Amarillo";  
  console.log(color);  
}
```

```
experimento();
```

```
var color = "Verde";  
experimento();
```

```
console.log(color);
```



Funciones

Las variables creadas en el interior de funciones

- Se comportan de manera especial
- Solo existen mientras se está ejecutando la función
- Pueden “tapar” otras variables con el mismo nombre
- Se llaman: variables locales

Funciones

Ejercicio:

Escribe la función **promptNumero** que nombramos al principio del tema

Funciones

Ejercicio (grande):

- Crea una nueva página apuestas.html y un fichero javascript apuestas.js
- Vamos a escribir un juego de apuestas con dados

Funciones

```
iBienvenido!  
  
Ahora mismo tienes 100 €  
Tu apuesta está en 10 €  
< undefined  
> apostar(5)  
Ok, has apostado: 5 €  
< undefined  
> jugar()  
  
Tiras los dados...  
  
[ 3 ] [ 2 ]  
  
Has perdido...  
Ahora mismo tienes 95 €  
Tu apuesta está en 5 €  
< undefined  
>
```

```
> estado()  
Ahora mismo tienes 95 €  
Tu apuesta está en 5 €  
< undefined  
> jugar()  
  
Tiras los dados...  
  
[ 4 ] [ 5 ]  
  
Has perdido...  
Ahora mismo tienes 90 €  
Tu apuesta está en 5 €  
< undefined  
> jugar()  
  
Tiras los dados...  
  
[ 2 ] [ 1 ]  
  
Has perdido...  
Ahora mismo tienes 85 €  
Tu apuesta está en 5 €
```

Funciones

Los comandos son:

estado(): ver el estado del juego (dinero y apuesta actual)

apostar(#): fijar la apuesta a # € (número). Tiene que ser igual o menor al dinero que tiene el jugador.

jugar(): se tiran 2 dados:

- Si entre los dos suman 7, el jugador gana 10 veces lo que ha apostado. Se suma (apuesta * 10) a su dinero
- Si salen dobles, se tira otra vez
- En cualquier otro caso, el jugador pierde. Se resta apuesta de su dinero total.

Funciones

`reset()`: Se resetea el juego. Dinero vuelve a 100€ y apuesta a 10€.

Funciones

Si salen dobles

```
> jugar()  
  
Tiras los dados...  
  
[ 3 ] [ 3 ]  
  
-- Dobles! Vuelves a jugar  
  
Tiras los dados...  
  
[ 4 ] [ 2 ]  
  
Has perdido...  
Ahora mismo tienes 40 €  
Tu apuesta está en 10 €
```

Funciones

Si el jugador llega a tener más de 100.00€, ha ganado

```
iSiete!  
iiGanas 50 € !!  
Ahora mismo tienes 10000010 €  
Tu apuesta está en 5 €  
  
***** HAS GANADO *****  
  
Con 10000010€ en el bolsillo, decides retirarte de la mesa  
y empezar a planear unas vacaciones en el Caribe.
```

Si llega a 0 o menos, ha perdido

```
Has perdido...  
Ahora mismo tienes -6 €  
Tu apuesta está en -6 €  
  
***** HAS PERDIDO *****  
  
Te han desplumado. Hoy no es tu día.
```