UD1: La arquitectura de la web e introducción al lenguaje JavaScript (III)

Parte 3: Funciones y objetos

María Rodríguez Fernández mariarfer@educastur.org

EJERCICIO PROPUESTO “Piedra, papel o tijera” (Parte 1)

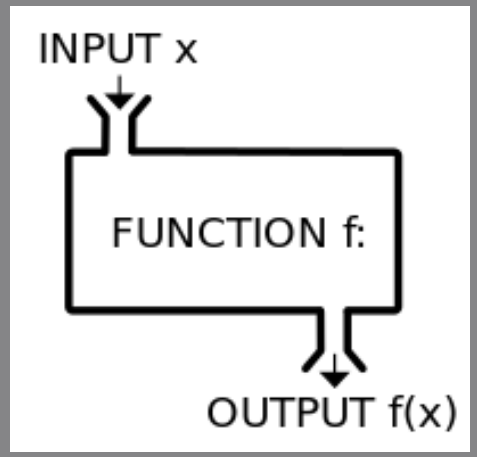
- Crea dos variables **p1** y **p2** que representen las jugadas del jugador 1 y jugador 2 (string que puede ser “*Piedra*”, “*Papel*” o “*Tijera*”).
- Muestra el resultado teniendo en cuenta que:
  - Piedra gana sobre Tijera, Tijera gana sobre Papel, Papel gana sobre Piedra
- Por lo que el programa tendrá 3 posibles salidas:
  - El ganador es p1
  - El ganador es p2
  - Es un empate

EJERCICIO ¿RESUELTO?

```
var p1 = "Piedra";
var p2 = "Papel";

if (p1==p2) console.log ("Es un empate");
else
if (p1=="Piedra" && p2 == 'Tijeras’)
    console.log ("El ganador es p1");
else
if (p1=="Piedra")
    console.log ("El ganador es p2");
else
if (p1=="Papel" && p2 == 'Tijeras’)
    console.log ("El ganador es p2");
else
if (p1=="Papel" && p2 == 'Piedra’)
    console.log ("El ganador es p1");
else
if (p1=="Tijeras" && p2 == 'Papel’)
    console.log ("El ganador es p1");
else console.log ("El ganador es p2");
```

Al terminar la clase de hoy…

- Cambiarás el chip en cuando al concepto de función en JavaScript
- Aprenderás cosas más avanzadas sobre funciones
  - Valores por defecto
  - Sobrecarga
- Conocerás y empezarás a usar los objetos literales para almacenar información



Funciones en JavaScript

Funciones

- Una **función** es un conjunto de instrucciones que se agrupan para realizar una tarea concreta y se pueden reutilizar de manera sencilla
- Facilitan la organización, mantenimiento y depuración de los programas
- Primero declaramos la función y luego la utilizamos (invocación)

```
/* Definición */
function nombreFunc(){
sentencias;
}

/* Invocación */
nombreFunc();
```

Son la base de la **programación funcional** (React)

Funciones simples: Ejemplo

```
function sumayMuestra(){
 var resultado = numero1 + numero2;
 alert("El resultado es "+ resultado);
}
```

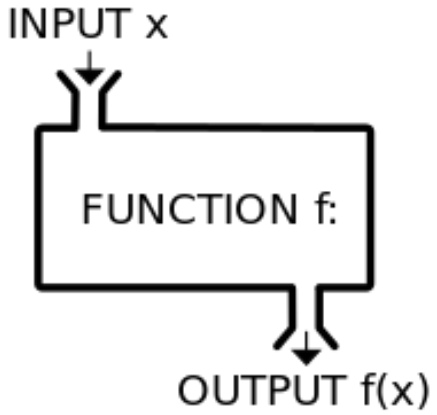
Definición

```
var numero1=3;
var numero2=5;

sumayMuestra();

numero1=5;
numero2=6;
sumayMuestra();
```

Llamada o invocación

E/S de datos en funciones

- Argumentos/parámetros
  - Permiten especificar las **entradas** de la función
- Retorno
  - Especifica el **valor que devuelve** la función.




```
/* Definición */
function nombreFuncion(argumento1, argumento2){
sentencias;
return valor;
}
```

Valor de retorno: Ejemplo

```
function suma(primernumero,segundonumero){
 var resultado = primernumero + segundonumero;
 return resultado;
}
```

Definición

```
//Declaración de las variables
var numero1=3;
var numero2=5;

//Llamada a la función
var resultado=suma(numero1,numero2);
alert(resultado);
```

Funciones anónimas

- Son funciones que se definen sin utilizar un identificador
  - Facilitan la programación, pero pueden complicar la lectura y depuración del código

```
console.log(function (){
    return "Comenzando..."
}())
```

Las () del final hace que se ejecute

Es una práctica muy extendida en JS (se usa mucho en eventos - ya lo veremos), pero requiere una cierta experiencia para depurar

Además, en JavaScript…

- Las funciones son **un tipo de dato** más
  - Pueden guardarse en variables y constantes

```
const hello = function () {console.log("HolaMundo"); }
hello();
```

  - Una función puede devolver otra función

```
function hello() {
    console.log("Hola Mundo");
    return function(){ return "Hola interno"};
}
console.log(hello()());
```

Bucle **forEach**


```
var diasLaborables = ["Lunes", "Martes", "Miércoles", "Jueves",
"Viernes", "Sábado", "Domingo"];
```

- **forEach**(función Callback)
  - Ejecuta la función indicada una vez por cada elemento del array

```
diasLaborables.forEach( function(valor, indice) {
    console.log("En el índice " + indice + " hay este valor: " + valor);
});
```

Funciones flecha o funciones arrow

- Forma compacta de definir una función:
  - Si la función sólo contiene una sentencia - que es el return:
    - Sin function
    - Sin llaves
    - Sin return

```
() => sentencia //función sin parámetros
unParam => sentencia //función con un parámetro
(param1, param2,…, paramN) => sentencia //más parámetros
```

  - Si es una función multilínea:
    - Sin function
    - Con llaves
    - Con return

```
() => { varias sentencias}
unParam => { varias sentencias }
(p1, p2,…, pN) => { sentencias }
```

Ejemplos

- Ejemplo típico:

```
function sumar(x,y){
    return x+y
}
const sumar = (x,y)=>x+y;
```

- Ejemplos con distintos tipos de datos:

```
const showText = ()=>"hola mundo";
const showNumber = ()=> 22;
const showBoolean = () => true;
const showObject = () => ({nombre:"Manolito"})
```

  - OJO al devolver el objeto hay que poner paréntesis para que no interprete las {} como cuerpo de la función:

Ejemplo: función “normal” a función “arrow”

```
//Sintaxis convencional 1 ("estilo Java")
function arraysConcatenados (array1, array2) {
  return array1.concat(array2);
}

//Sintaxis convencional 2 ("estilo JavaScript")
var arraysConcatenados = function (array1, array2) {
  return array1.concat(array2);
}

//Sintaxis con función flecha ("estilo JavaScript Moderno")
var arraysConcatenados = (array1, array2) => array1.concat(array2);
```

```
//A las tres se les llama de la misma forma
console.log(arraysConcatenados([1,2],[3,4,5]));
```

EJERCICIO PROPUESTO “Piedra, papel o tijera” (Parte 2)

- “Rescata” el código de la Parte 1 y crea una función que reciba como parámetros las jugadas de los dos jugadores y devuelva el resultado del combate

```
function ppt(p1,p2){
    //algoritmo
    return resultado;
}
ppt ("Piedra", "Papel");
```

Si estás pensando en algo así, es correcto, ¡pero dale una vuelta!

Estableciendo parámetros por defecto

ES5

```
function saludar(nombre, titulo, saludo){
var t=titulo || 'D.';
var s=saludo || 'Hola' + t
console.log(s + ' ' + nombre);
};
saludar('Jordi'); // Hola D. Jordi
```

ES6

```
function saludar(nombre, titulo = 'D.', saludo = 'Hola' + titulo){
console.log(saludo + ' ' + nombre);
};
saludar('Jordi'); // Hola D. Jordi
```

Sobrecarga de funciones

- En JS no existe la **sobrecarga** (mismo nombre, distinto comportamiento), **pero** podemos llamar a una función con cualquier número de parámetros
  - En caso de no coincidir los parámetros no se considera un error del lenguaje, sino que el intérprete se intentará adaptar:
    - Si faltan parámetros, su valor será “undefined”
    - Si sobran parámetros, podemos acceder a través de la variable **arguments**
      - Es un array que siempre está disponible dentro de una función y contiene todos los parámetros que se le han pasado a la función

Sobrecarga - Ejemplo

```
function concatena(p1,p2,p3){
    alert(p1+" "+p2+" "+p3);
}

concatena("Felipe","Juan");
```

Felipe Juan undefined

p3 → undefined

```
function concatena (){
    var salida="";
    for (var i=0;i<arguments.length;i++)
        salida+=arguments[i]+" ";
    alert(salida);
}
concatena("Felipe","Juan","Froilán");
```

Felipe Juan Froilán

Parámetros REST: ES5 vs ES6

ES5

```
function concatena (nombre){
    var salida=nombre;
    for (var i=1;i<arguments.length;i++)
        salida+=arguments[i]+" ";
    alert(salida);
}
concatena("Felipe","Juan","Froilán");
```

ES6

```
function concatena (nombre, ...guays){
    var salida=nombre;
    guays.forEach(function(valor){salida += ' ' + valor});
    alert(salida);
}
concatena("Felipe","Juan","Froilán");
```

FUNCIÓN ANÓNIMA

EJERCICIO PROPUESTO “Piedra, papel o tijera” (Parte 3)

- “Rescata” el código de la Parte 2 y haz que, por defecto, de no pasarse jugada, se juegue “Piedra” contra “Piedra”.
- *RETO: La función podría recibir un número indefinido de jugadas*

Objetos literales

Objetos en JavaScript

- En la mayoría de lenguajes de programación los objetos se crean con **new**
  - JavaScript también lo permite desde 2015
  - Se usa cuando hacemos **programación orientada a objetos**
- JavaScript permite además utilizar la **notación literal** para crear objetos
  - Más abreviada
  - "Línea directa" con JSON

Objetos literales

- Se crean con las llaves {}

```
// Esto es un objeto vacío
const objeto = {};
```

- Se entienden como un conjunto de variables de cualquier tipo declaradas como **clave: valor** sin necesidad de crear una clase

```
const jugador = {
    nombre: "Manz",
    vidas: 99,
    potencia: 10,
};
```

  - Acceso a propiedades con punto o corchete

```
// Notación con puntos (preferida)
console.log(jugador.nombre); // Muestra "Manz"
// Notación con corchetes
console.log(jugador["vidas"]);// Muestra 99
```

¿y qué pasa con los métodos?

```
const usuario = {
    nombre: "Manolito García",
    edad: 30,
    nacimiento: {
        pais: "España",
        ciudad: "Oviedo"
    },
    amigos: ["Menganito", "Antoñito"],
    activo: true,
    sendMail: function (){
        return "Enviando email..."
    }
}
console.log(usuario);
console.log(usuario.nombre);
console.log(usuario.nacimiento.ciudad);
console.log(usuario.amigos)
console.log(usuario.sendMail); //devuelve el código
console.log(usuario.sendMail()); //ejecuta la función
```

EQUIVALENTE:

```
sendMail (){
  return "Enviando email..."
}
```

¿Te suena?

String literals

- Ofrecen una forma limpia de insertar variables (sustituyéndolas por su valor) en strings con ${} entre comillas invertidas:

```
const background= "red";
const color= "white"

boton.style=`background: ${background}; color: ${color}`;
```

- Introduciendo **condicionales** con **operador ternario**:

```
boton.style = `background: ${estaAutorizado? 'red' :
'white'}; color: ${color}`;
```

EJERCICIO PROPUESTO “Piedra, papel o tijera” (Parte 4)

- “Rescata” de nuevo el código de la Parte 3 y usa un objeto literal para almacenar información sobre qué gana a qué
- Usa string literals para componer la cadena resultado

Shorthand property names

- Podemos crear un objeto a partir de otras constantes/variables
  - Existe una notación abreviada donde sólo es necesario poner su nombre y se crea la propiedad

```
const nombre="portatil"
const precio =3000;

const nuevoProducto = {
    nombre: nombre,
    precio: precio
}
```

```
const nombre="portatil"
const precio =3000;

const nuevoProducto = {
    nombre
    precio
}
```

- Lo mismo se aplica a los Arrays

Desestructurar un objeto

- Las **llaves** me permiten usar sólo una parte del objeto:

```
function imprimirInfo({nombre}){
    return "<h1>Hola "+nombre+"</h1>";
}
document.body.innerHTML=imprimirInfo(usuario);
```

- También se puede pasar el objeto completo y luego desestructurarlo antes de usarlo:

```
function imprimirInfo(usuario){
    const {nombre, edad} = usuario;
    return "<h1>Hola "+nombre+"</h1>";
}
document.body.innerHTML=imprimirInfo(usuario);
```

EJERCICIO PROPUESTO “Piedra, papel o tijera” (Parte 5)

- A partir del código generado en la parte 4, crea un objeto “partida” para almacenar la jugada de los dos jugadores.
  - Usa desestructuración de objetos en la función para acceder a las jugadas

¿Cómo te fue?