# Funciones (Declaración)

La forma de definir la función determinará como se comportará.

- **Declaración de función** *(Function Statement)*
- **Expresión de función** *(Function Expression)*
- **Función IIFE** *Inmediatamente Invocada (IIFE: Immediately Invoked Function Expression)*
- **Función Flecha** *(Arrow Function)*
- **Declaración abreviada** *Shorthand*
- **Generador de función** *Function Generator*
- **Constructor de función** *Function Constructor*

# Funciones (Declaración)

## Declaración de función

- Se crea con la palabra reservada function
- El nombre de la función es obligatorio
- Seguida de una lista de parámetros opcionales entre paréntesis ()
- Después los corchetes {} delimitarán el conjunto de sentencias de la función
- A las declaraciones de funciones se les hace hoisting

```
function saludar(nombre) {
    console.log(`Hola ${nombre}.`)
}

saludar('Gerardo');
```

## Funciones (Declaración)

## Expresión de función

- No comienza con la palabra reservada function
- El nombre de la función es opcional
- Se puede asignar a una variable
- Al estar asignada a una variable la podemos pasar como parámetro a otra función
- Se le puede dar un nombre y así puede ser usada dentro de la función para referirse a sí misma (Recursividad)
- A las expresiones de funciones no se les hace hoisting

```
var saludar = function (nombre) {
    console.log(`Hola ${nombre}.`);
}

saludar('Gerardo');
```

## Funciones (Declaración)

## IIFE *Immediately Invoked Function Expression*

- También se les conoce como funciones de auto-invocación
- Se ejecutan inmediatamente y no pueden ser accesibles después
- Se debe crear la función entre paréntesis seguida nuevamente de paréntesis

```
(function saludar(nombre) {
    console.log(`Hola ${nombre}.`);
})("Gerardo");
```

## Funciones (Declaración)

## Función flecha

- Son anónimas y su sintaxis es mas corta
- Se define una lista de parámetros entre paréntesis (opcionales) y a continuación el símbolo =>
- Posteriormente corchetes para delimitar el código de la función (opcionales si es solo una instrucción)
- No crean su propio contexto (this) de ejecución. (Declaración y Expressión de Función si lo crean)
- El objeto arguments no se encuentra en el contexto de la función.
- También son conocidas como función flecha gruesa o fat arrow function.

```
var saludar = (nombre) => "Hola " + nombre;
console.log(saludar("Gerardo"));
```

# Funciones (Declaración)

# Función flecha

**El por qué de las funciones flecha**

Dos factores influenciaron la introdución de las funciones flecha: tener funciones más cortas y el léxico this.

En patrones funcionales, es mejor usar funciones más cortas, por ejemplo:

```
var a = ["Manzana","Pera","Naranja","Melon"];
var a2 = a.map(function(s){ return s.length });
var a3 = a.map( s => s.length );
```

**Funciones flecha no tienen su propio this**

Hasta antes de las funciones flecha, cada nueva función definía su propio valor this:

En el caso de un constructor...un nuevo objecto

En llamada a funciones en modo estricto...no definido

En llamada como un "método de objeto"....el objeto de contexto

Esto probó ser molesto en un estilo de programación orientada a objetos.

```
function Persona() {
  this.edad = 0;                    // El constructor Person() define this como el mismo

  setInterval(function crece() {    // En modo no estricto, la funcion growUp() define `this` como el objeto global
    this.edad++;                    // el cual es diferente del this definido por el constructor Person()
  }, 1000);
}

var p = new Persona();
```

# Funciones (Declaración)

## Declaración abreviada

- Se usa como método en la declaración de un objeto o clase
- Debe llevar un nombre de función
- Entre paréntesis la lista de parámetros
- Entre corchetes el código de la función

```
// Declaración "normal"
const saludos = {
    persona: [],
    agregar: function(persona) {
             this.persona.push(persona);
            },
    saludar: function(index) {
             return `Hola ${this.persona[index]}`;
             }
  };
```

```
// Declaración "shorthand"

const saludos = {
    persona: [],
    agregar(persona){ this.persona.push(persona); },
    saludar(index)  { return `Hola ${this.persona[index]}`;}
  };
```

# Funciones (Declaración)

# Generador de función

Misma sintaxis que declaración de función y expresión de función.
Pero se usa * al inicio de la definción de la función.

Estas funciones nos permiten parar la ejecución función en un punto dentro de ella.
Para retornar posteriormente la ejecución desde ese punto en el que paramos.

```
function *generador() {                    // Se declara la función con *
         console.log('Inicio Función');
         yield 'Hola ';                    // yield = Salir // value = 'Hola'
         console.log('Continua Función');
         yield 'Mundo!';                   // yield = Salir // value = 'Mundo'
}

const gen = generador();                   // Se asigna la función a una constante
console.log(gen.next().value);             // La invocamos por primera vez y comienza su ejecución
console.log('Función en pausa');           // Ejecutamos código fuera de la función
console.log(gen.next().value);             // Regresamos a la ejecución de la función
```

# Funciones (Declaración)

## Constructor de función

En Javascript las funciones son objetos y se pueden crear en tiempo de ejecución usando el constructor Function() para crear nuevas funciones.

Podemos crear una variable que invoque al objeto Function.
Los n argumentos que le mandemos seran los parámetros
El último argumento será el código de la función.

```
const sumar = new Function ( ' x ', ' y ', ' return x + y ' );
sumar ( 5, 5 );
```

# Funciones (Declaración)

### Declaración de función

```
function saludar(nombre) {
    console.log(`Hola ${nombre}.`)
}

saludar('Gerardo');
```

### Expresión de función

```
var saludar = function (nombre) {
    console.log(`Hola ${nombre}.`)
}

saludar('Gerardo');
```

### Declaración abreviada

```
const saludos = {
    persona: [],
    agregar(persona){ this.persona.push(persona); },
    saludar(index)  { return `Hola ${this.persona[index]}`;}
  };
```

### IIFE

```
(function saludar(nombre) {
    console.log(`Hola ${nombre}.`)
})("Gerardo");
```

### Función flecha

```
var saludar = (nombre) => "Hola " + nombre;
console.log(saludar("Gerardo"));
```

### Generador de función

```
function* generador() {
    var contador = 1;
    yield contador++;
    yield contador++;
}

var g = generador() ;
console.log ( g.next().value ); // 1
console.log ( g.next().value ); // 2
```

### Constructor de función

```
const saludar = new Function (`nombre`,"return `Hola ${nombre}.`");
console.log(saludar('Gerardo'));
```