Buenos Aires aprende
Agencia de Habilidades para el Futuro

Talento Tech

Front-End JS

Clase 11 - “Programación modular con funciones”

Buenos Aires Ciudad

¡Les damos la bienvenida!

 Vamos a comenzar a grabar la clase

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Talento Tech

Clase 10.

JS 2 - Condicionales y ciclos

1. Diagrama de flujo
2. Condicional: ¿Qué es?
3. Operadores lógicos y de comparación: ¿Qué son y cuál es su uso en los condicionales?
4. Bucles: ¿Qué son? Tipos y diferencias entre sí
5. Cómo combinar operadores lógicos y ciclos

Clase 11.

JS 3 - Programación modular con funciones

1. Funciones: ¿Que son? Parámetros de entrada y de salida
2. Scope global y local
3. Programación modular vs. Funciones
4. Parámetros.
5. Funciones nativas.

Clase 12.

JS 3 - DOM y eventos

1. Manipulación del DOM
2. Definición, alcance y su importancia para operar sobre elementos HTML
3. Eventos en JS
4. Eventos: ¿Que son, para qué sirven y cuáles son los más comunes?
5. Escuchar un evento sobre el DOM

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Buenos Aires Ciudad

Talento Tech

condicionales?

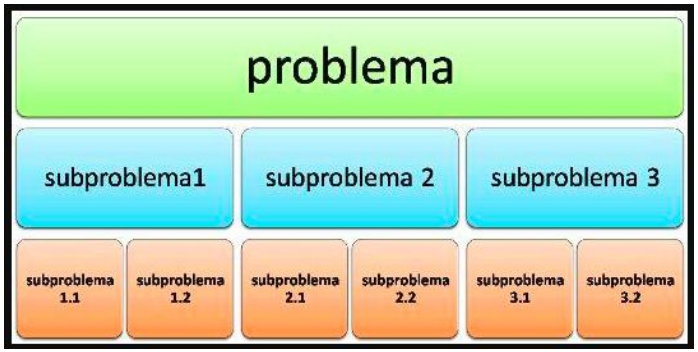
JavaScript

Funciones

Las **funciones** son estructuras esenciales dentro del código.
Una función es un grupo de instrucciones que constituyen una unidad lógica del programa y resuelven un problema muy concreto.
Presentan varias *ventajas*, entre ellas las de permitir dividir un problema complejo en partes menores y más simples, reutilizar código en el mismo o en otro programa, simplificar
la depuración, etcétera.
JavaScript proporciona al usuario una serie de funciones implementadas y
listas para utilizar. Sin embargo, no es difícil encontrar situaciones en las que necesitamos realizar alguna tarea para la cual no existe una función disponible, y debemos utilizar los mecanismos que nos proporciona JS para construir nuestras propias funciones.

Programación Modular

La metodología de división por módulos se conoce habitualmente como “divide y vencerás” y en programación se llama Desarrollo Top Down.
¿Cuál será la estrategia para resolver problemas? Pensar en el problema general e ir descomponiéndolo en sub-problemas (sub-algoritmos). A su vez, estos subproblemas se podrán seguir dividiendo hasta llegar a un subproblema lo bastante simple como para poder resolverse de forma sencilla.



Abstracción

Podemos definir la abstracción como el aislamiento de un elemento de su contexto o del resto de los elementos que lo acompañan. En programación la abstracción está relacionada con “qué hace”.
Concretamente, **la abstracción se produce cuando creamos módulos**.
Lo importante, para entender el concepto de abstracción, es comprender que cada módulo es independiente de los demás módulos (bajo acoplamiento) y que es ideal que realice una sola tarea (alta cohesión).
Los módulos son independientes entre sí, aunque algunos pueden necesitar colaborar con otros, o trabajar de forma conjunta.

Funciones

Las **funciones** nos permiten agrupar líneas de código en tareas con un nombre (subprograma), para que posteriormente podamos referenciar ese nombre para realizar dicha tarea. Algunas razones para declarar funciones:

- **Simplificación**: Cuando un conjunto de instrucciones se va a usar muchas veces, se crea una función con esas instrucciones y se llama la cantidad de veces que sea necesario, reduciendo un programa complejo en unidades más simples.
- **División**: Una función me permite modularizar, es decir, armar módulos. De esta manera un equipo puede dividir el trabajo en partes. Cada integrante realiza una función, para luego integrarlas en un programa principal más grande.
- **Claridad**: Usando funciones un programa gana claridad, aunque esa función solo se llame una vez.
- **Reusabilidad**: Una función es reutilizable, sólo es necesario cambiar los valores de entrada.

Funciones

Para usar funciones es necesario hacer dos cosas:

- **Declarar la función:** crear la función es darle un nombre, definir los datos de entrada (opcional) e indicar las tareas (instrucciones) que realizará y qué valor retornará (opcional).
- **Ejecutar la función:** «Llamar» (invocar) a la función para que realice las tareas del código que aloja. Se puede invocar una misma función la cantidad de veces que se necesita desde el programa principal.

```
// Declaración de la función "saludar"
function saludar() {
 // Contenido de la función
  console.log("Hola, soy una función")
}
```

Primer paso:
Declarar la función

```
// Ejecución de la función
saludar()
```

Segundo paso:
Ejecutarla

Funciones

El nombre de la función tiene que ser significativo y describir lo que hace. Los nombres de las funciones tienen las mismas características que los de las variables. Idealmente deben ser:

- Nombres simples, claros.
- Representativos de la tarea que realiza la función.
- Verbos en infinitivo (-ar, -er, -ir).
- Si es más de una palabra, utilizar la nomenclatura camelCase.

Es necesario definir los datos de entrada (si existen) e incluir las instrucciones necesarias para que realice su tarea. Opcionalmente se puede definir qué valor retornará.

Funciones | Ejemplo

Este código muestra la tabla de multiplicar por 5.

```
for (i = 1; i <= 10; i++) {
    console.log("1 x", i, "=", 5 * i)
}
```

Este código muestra la tabla de multiplicar por 5 tres veces. Funciona, pero usa demasiado código, repetido.

```
// Primera vez
for (i = 1; i <= 10; i++) {console.log("5 x", i, "=", 5 * i)}
// Segunda vez
for (i = 1; i <= 10; i++) {console.log("5 x", i, "=", 5 * i)}
// Tercera vez
for (i = 1; i <= 10; i++) {console.log("5 x", i, "=", 5 * i)}
```

Solución con bucle y función. La función tablaDelCinco() usa un for de 10 iteraciones. El otro for ejecuta la función 3 veces.

```
//Declaración de la función tablaDelCinco()
function tablaDelCinco(){
  for (i = 1; i <= 10; i++){console.log("5 x", i, "=", 5 * i)}
}
//Bucle que ejecuta 3 veces la función tablaDelCinco()
for (let i = 1; i <= 3; i++) {tablaDelCinco()}
```

Funciones | Clasificación

Según reciban o no datos, y devuelvan o no valores, las funciones se pueden clasificar en:

Funciones sin parámetros:

- Que no devuelven valores
- Que devuelven valores

Funciones con parámetros:

- Que no devuelven valores
- Que devuelven valores

Funciones | Parámetros y Argumentos

Los **parámetros** son las variables que ponemos cuando se define una función. En la siguiente función tenemos dos parámetros “a” y “b”:

```
function sumar(a, b){
    console.log(a + b)
}
```

Los **argumentos** son los valores que se pasan a la función cuando ésta es invocada, “7” y “4” en el ejemplo:

```
var suma = sumar(7, 4) //Pedimos valores
```

Dentro de la función, los argumentos se copian en los parámetros y son usados por ésta para realizar la tarea.

Funciones | Parámetros y Argumentos

Esta función tiene un sólo parámetro que indica hasta qué valor calculará:

```
// Declaración
function tablaMultiplicar(hasta) {
      for (var i = 1; i <= hasta; i++)
        console.log("1 x", i, "=", 1 * i)
}
```

```
//Ejecución
tablaMultiplicar(4)
```

En este ejemplo la función muestra un texto concatenado a un argumento pasado por parámetro:

```
// Declaración
function saludarDos(miNombre){
    console.log("Hola " + miNombre)
}
```

```
//Ejecución
saludarDos("Codo a Codo") //Argumento fijo
var nombre= prompt("Ingrese su nombre")
saludarDos(nombre) //Argumento variable
```

Funciones | Parámetros múltiples

Cuando se utilizan parámetros múltiples hay que respetar el orden en que los declaramos y el de los argumentos usados al llamarla. Esta función tiene dos parámetros: el valor de la tabla a generar y hasta qué valor calculará.

```
// Declaración
function tablaMultiplicar(tabla, hasta) {
      for (var i = 1; i <= hasta; i++)
        console.log(tabla + " x " + i + " = ", tabla * i)
}
```

```
// Ejecución
tablaMultiplicar(1, 10) // Tabla del 1, calcula desde el 1 hasta el 10
tablaMultiplicar(5, 8) // Tabla del 5, calcula desde el 1 hasta el 8
```

Funciones | Parámetros múltiples

Ejemplo con tres parámetros. Se evalúa la mayoría de edad de una persona:

```
// Declaración
function mayoriaEdad(miApellido, miNombre, miEdad){
    console.log("Apellido y nombre: " + miApellido + ", " + miNombre)
    if (miEdad >= 18) {
        console.log("Es mayor de edad " + "(" + miEdad + ")")}
        else{
        console.log("No es mayor de edad " + "(" + miEdad + ")")}
    }
```

```
//Ejecución
var ape= prompt("Ingrese su apellido")
var nom= prompt("Ingrese su nombre")
var edad= prompt("Ingrese su edad")
mayoriaEdad(ape, nom, edad)
```

Esta función recibe tres parámetros y en función del valor de uno de ellos (miEdad) determina si la persona es mayor de edad (>=18)

Parámetros predeterminados

Los parámetros predeterminados de función permiten que los parámetros con nombre se inicien con valores predeterminados si no se pasa ningún valor o undefined. En JavaScript, los parámetros de función están predeterminados en undefined. Sin embargo, a menudo es útil establecer un valor predeterminado diferente.

```
function multiplicar(a, b = 1) {
    return a * b;
  }

console.log(multiplicar(5, 2));  // salida: 10
console.log(multiplicar(5));  // salida: 5
```

Funciones | retorno de valores

Una función puede devolver información, para ser utilizada o almacenada en una variable. Se utiliza la palabra clave return, que regresa un valor y finaliza la ejecución de la función. Si existe código después del return, nunca será ejecutado. Puede haber más de un return por función.

```
// Declaración
function sumar(a, b) {
  return a + b // Devolvemos la suma de a y b al exterior de la función
}
```

```
// Ejecución
var a = 5, b = 5
var resultado = sumar(a, b) // Se guarda 10 en la variable resultado
console.log("La suma entre "+ a +" y "+ b +" es: "+ resultado)
```

Funciones | retorno de valores

Veamos dos funciones que hacen lo mismo, una retorna valores y otra no:

```
function sumar(num1, num2){
    var suma = num1 + num2
    console.log("La suma es " + suma)
}
sumar(2,5)
```

Esta función muestra “La suma es …” en la consola, pero no retorna ningún valor al programa.

```
function sumarDos(num1, num2){
    var suma = num1 + num2
    return suma
}
n1 = 2
n2 = 3
var resultado = sumarDos(n1, n2)
console.log("El resultado es: " + resultado)
```

En este caso la función devuelve un valor, y se almacena en una variable llamada resultado que contiene la suma de dos valores realizada por la función sumarDos.

Funciones | retorno de valores

Otra alternativa es hacer que la función guarde directamente el resultado que devuelve en una variable:

```
var suma = function sumarTres(numero1, numero2) {
    return numero1 + numero2
}
console.log(suma(40, 15))
```

Al retornar un valor, éste se guarda en la variable suma.

```
var numeroMaximo = function (valor1, valor2) {
    if (valor1 > valor2) {  return valor1 }
    return valor2
}
var v1 = parseInt(prompt("Ingrese un número entero"))
var v2 = parseInt(prompt("Ingrese otro número entero"))
console.log("El número máximo es:", numeroMaximo(v1,v2))
```

En este caso se piden dos valores y si la condición no se cumple se asume que el valor2 es el máximo (no es necesario un else)

Scope (Alcance)

El **scope** (alcance) determina la accesibilidad (visibilidad) de las variables. Define ¿en qué contexto las variables son visibles y cuándo no lo son?. Una variable que no está “al alcance actual” no está disponible para su uso.

En JavaScript hay dos tipos de alcance:

- Alcance local (por ejemplo, una función)
- Alcance global (entorno completo de JavaScript)

Las variables definidas dentro de una función no son accesibles (visibles) desde fuera. La función “crea un ámbito cerrado” que impide el acceso a una variable de su interior desde fuera de ella o desde otras funciones.

Scope (Alcance)

Variables Locales

En el siguiente ejemplo creamos una variable llamada carName a la cual le asignamos un valor:

```
// aca no puedo usar la variable carName
function myFunction() {
    var carName = "Volvo"
    // aca si puedo usar la variable carName
}
// aca no puedo usar la variable carName
```

Podremos acceder al contenido de la variable **carName** solamente dentro de la función.

Este tipo de variables son de alcance local, porque solamente valen en el ámbito de la función, y no en el ámbito a nivel de programa. Los parámetros de la función funcionan como **variables locales** dentro de las mismas.

Scope (Alcance)

Una variable declarada fuera de una función se convierte en **global**. Esto quiere decir que tiene alcance global: todos los scripts y funciones de una página web pueden acceder a ella.

```
var carName2 = "Fiat"
    // aqui si puedo usar carName2
function myFunction() {
    // aqui tambien puedo usar la variable carName2
}
```

En este caso podremos acceder al contenido la variable carName tanto desde fuera como desde adentro de la función

El alcance determina la accesibilidad de variables, objetos y funciones de diferentes partes del código.

Scope (Alcance)

Variable automáticamente global

Si asignamos un valor a una variable que no ha sido declarada, se convertirá en una variable global. Este ejemplo declara la variable global carName, aún cuando su valor se asigna dentro de una función.

```
myFunction();
// aquí puede se puede usar carName
function myFunction() {
    carName = "Volvo"  // variable no declarada
}
```

En este caso podremos acceder al contenido la variable carName tanto desde fuera como desde adentro de la función por ser automáticamente global.

La vida útil de una variable comienza cuando se declara. Las variables locales se eliminan cuando se completa la función.

Let y Var

let: declara una variable de alcance local, limitando su alcance (scope) al bloque, declaración, o expresión donde se está usando.

var: define una variable global o local en una función sin importar el ámbito del bloque.

```
var a = 5
var b = 10
if (a === 5) {
  let a = 4 // El alcance es dentro del bloque if
  var b = 15 // El alcance es global, sobreescribe a 10
  console.log(a)  // 4, por alcance a nivel de bloque
  console.log(b)  // 15, por alcance global
}
console.log(a) // 5, por alcance global
console.log(b) // 15, por alcance global
```

Funciones nativas

¿Qué son las Funciones Nativas?

Las funciones nativas en JavaScript son aquellas que ya vienen predefinidas en el lenguaje. Nos permiten realizar operaciones comunes sin necesidad de escribir código adicional. Son herramientas listas para usar y optimizan el desarrollo, porque evitan que tengamos que "reinventar la rueda".

```
alert("¡Hola, Mundo!");
parseInt(): Convierte una cadena de texto en
un número entero.

let numero = parseInt("10");
Math.random(): Genera un número aleatorio
entre 0 y 1.


let aleatorio = Math.random();
Date(): Devuelve la fecha y hora actual.

let hoy = new Date();
console.log(hoy);
```

Más funciones nativas

```
console.log(): Imprime un mensaje en la consola del
navegador. Ideal para depuración.

console.log("Mensaje de depuración");
toUpperCase() y toLowerCase(): Convierte una cadena
de texto a mayúsculas o minúsculas.

let texto = "Hola";
console.log(texto.toUpperCase()); // Imprime "HOLA"
console.log(texto.toLowerCase()); // Imprime "hola"
slice(): Extrae una parte de un string o array.


let palabra = "JavaScript";
console.log(palabra.slice(0, 4)); // Imprime "Java"
```

¿Por qué usar Funciones Nativas?

- Ahorrás tiempo y código.
- Son más eficientes porque están optimizadas.
- Facilitan la legibilidad y mantenimiento del código.

¡Vamos a la práctica! 

Ejercicios Prácticos

Optativos | No entregables

Función para Validar la Edad de una Persona

Crear una función que reciba como parámetros el nombre y la edad de una persona. La función debe realizar lo siguiente:

1. Verificar si la persona es mayor de edad (18 años o más).
2. Mostrar un mensaje en la consola que diga si la persona es mayor o menor de edad.
3. Si es menor de edad, mostrar también cuántos años le faltan para cumplir 18.

Tips:

- **Uso de parámetros:** Pasar el nombre y la edad como argumentos a la función.
- **Condicionales:** Usar un if...else para determinar si es mayor o menor de edad.
- **Consola del navegador:** Utilizá console.log() para mostrar los resultados en la consola.

Ejercicios Prácticos

Optativos | No entregables

Función para Calcular el Precio Total de un Producto con IVA

Crear una función que reciba el precio de un producto y el porcentaje de IVA (Impuesto al Valor Agregado). La función debe:

1. Calcular el precio total del producto, sumando el IVA.
2. Mostrar el precio total en la consola.
3. Hacer que el IVA sea un parámetro opcional, con un valor predeterminado del 21% (típico en Argentina).

Tips:

- **Parámetros opcionales:** Definir un valor por defecto para el IVA si no se proporciona uno.
- **Operaciones matemáticas:** Calcular el IVA y sumarlo al precio del producto.
- **Consola del navegador:** Mostrar el precio final con console.log().