Bienvenidos

Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Aplicaciones Móviles y Cloud Computing

Profesor:
Diego Polverelli

Estudiantes:

Por encuesta de Google Forms
https://forms.gle/9o9E6UodvNEACCEc9

1. Experiencia
2. Conocimientos previos
3. Expectativas para la cursada
4. y algunos otros datos, que te pido completes para conocerte mejor...

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Clase 01. Introducción a la materia
Primeros pasos con JavaScript

Aplicaciones Móviles
Y
Cloud Computing

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Temario

01

Introducción a la materia JavaScript

- ✓ Introducción a la materia
- ✓ Primeros pasos en JavaScript

02

Programación Asíncrona

- ✓ Programación Asíncrona con JavaScript
- ✓ Ajax, Fetch, Axios

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Objetivos de la clase

- Repasar los principales sucesos en la hitoria del Cloud Computer y las Aplicaciones Móviles
- Conocer los conceptos básicos de JavaScript
- Aprender a trabajar con Clases y Objetos en JavaScript

Cloud Computing

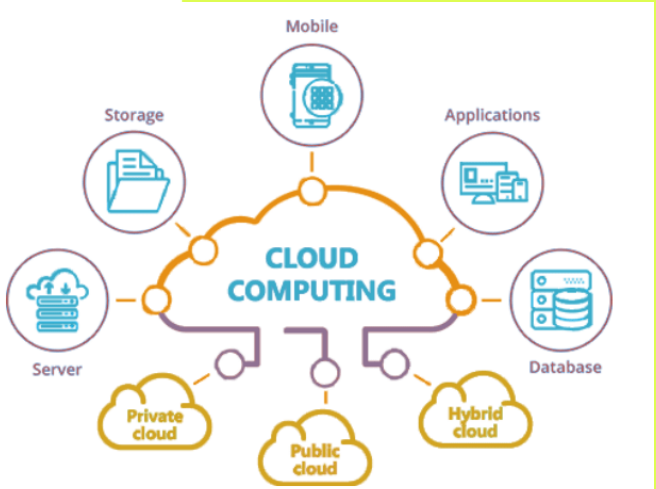
Cloud Computing

¿Qué es?

La computación en la nube, o cloud computing, es un modelo de entrega de servicios informáticos que permite el acceso a recursos de computación, almacenamiento y aplicaciones a través de Internet, sin la necesidad de tenerlos físicamente en el lugar de trabajo.

Características: escalable, flexible, rentable, y accesible desde cualquier lugar del mundo.

Servicios: infraestructura como servicio (**IaaS**), plataforma como servicio (**PaaS**) y software como servicio (**SaaS**). Adaptable a diferentes soluciones y modelos de negocio



Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Cloud Computing

Un poco de historia:

- ✓ Etapa previa: ARPANET (1969), Virtual Storage Personal Computing (VSPC) es un software de IBM que se lanzó en 1976
- ✓ 1999 SalesForce ofrece su CRM en la nube, inaugurando el concepto de SaaS
- ✓ A principios de 2000 comienza a utilizarse “masivamente” la virtualización
- ✓ En 2002 Amazon comienza a utilizar infraestructura en la nube para sus operaciones internas
- ✓ En 2006 Amazon Web Services (AWS) lanza Elastic Compute Cloud (EC2), un servicio de infraestructura de nube pública
- ✓ En 2008 Google presenta Google Clouds

Cloud Computing

Un poco de historia:

- ✓ En 2010 Microsoft pone en funcionamiento Azure
- ✓ 2013: Docker lanza su plataforma de contenedores de código abierto
- ✓ 2014: Google lanza Kubernetes, una plataforma para orquestación de contenedores de código libre
- ✓ 2014: AWS lanza Lambda, un servicio que permite ejecutar código sin preocuparse de la infraestructura subyacente: Serverless
- ✓ 2020: La pandemia acelera la adopción de soluciones basadas en la nube, como respuesta a la necesidad de las empresas de continuar con sus actividades de manera remota
- ✓ 2021: Comienza a ser tendencia la modalidad de nube híbrida, que convina recursos locales, con nube privada, y nube pública.

Cloud Computing

Ejemplos de servicios ofrecidos:

Servicio AWS	Servicio Azure	Servicio Google Cloud	Descripción
EC2 (Elastic Compute Cloud)	Azure Virtual Machine	Google Compute Engine	Servicio de cómputo escalable en la nube que permite lanzar instancias de servidores virtuales y configurar la capacidad de cómputo, el almacenamiento y la memoria según sea necesario.
S3 (Simple Storage Service)	Azure Blob Storage	Google Cloud Storage	Servicio de almacenamiento de objetos que permite a los usuarios almacenar y recuperar grandes cantidades de datos desde cualquier lugar de la web.
RDS (Relational Database Service)	Azure SQL Database	Google Cloud SQL	Servicio de base de datos relacional que permite a los usuarios configurar, operar y escalar bases de datos en la nube.
Lambda	Azure Functions	Google Cloud Functions	Servicio de computación sin servidor que permite a los usuarios ejecutar código en respuesta a eventos y escalar automáticamente la capacidad de cómputo según sea necesario.

Aplicaciones Móviles

Aplicaciones Móviles

Un poco de historia:

- ✓ 2007: Apple lanza el iPhone, que revoluciona la forma en que las personas interactúan con sus dispositivos móviles.
- ✓ 2008: Se lanza la App Store de Apple, que permite a los desarrolladores crear y distribuir aplicaciones móviles.
- ✓ 2008: Google lanza Android, un sistema operativo móvil de código abierto que se convierte en el principal competidor de iOS de Apple.
- ✓ 2014: Facebook lanza React Native, un marco de trabajo de código abierto que permite a los desarrolladores crear aplicaciones móviles nativas utilizando JavaScript y React.
- ✓ 2015: Google lanza Firebase, su plataforma de desarrollo de aplicaciones móviles.

Break

¡Unos minutos y volvemos!

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Primeros pasos con JavaScript

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

¿Cómo probamos Javascript?

Cliente web

Lo utilizamos desde la consola del navegador. No necesitamos instalar nada, ya que todo el motor vive en el navegador

Node JS

Debemos instalarlo ya que se usa fuera del navegador, por lo que podremos escribir Javascript desde nuestro computador directamente.

Cliente web

Se utiliza desde cualquier navegador. Algunos de los comandos que más usarás y verás a lo largo de un debug en cliente web son:

- ✓ **console.log(“texto”)** Mostrará un texto simple
- ✓ **console.warn(“texto”)** Mostrará una advertencia
- ✓ **console.error(“texto”)** Mostrará un error
- ✓ **console.clear()** Limpiará la consola para evitar que se acumule el código.

Node js

Cuando trabajamos en backend, no tenemos un navegador. Es por ello que se precisa de alguna tecnología que nos permita correr código de Javascript, sin necesidad de abrirlo en el navegador. Ahí es cuando utilizamos Node js. Para ello, toca considerar:

- ✓ Node js levantará un entorno completo para probar nuestras funciones.
- ✓ Debe correrse desde un CLI para poder visualizar los avances del código.
- ✓ Usualmente usaremos el CLI de Visual Studio Code.
- ✓ Node cuenta con una extensa comunidad, que aporta desarrollos muy útiles, accesibles vía npm

Documentación Oficial: https://nodejs.org/es/docs

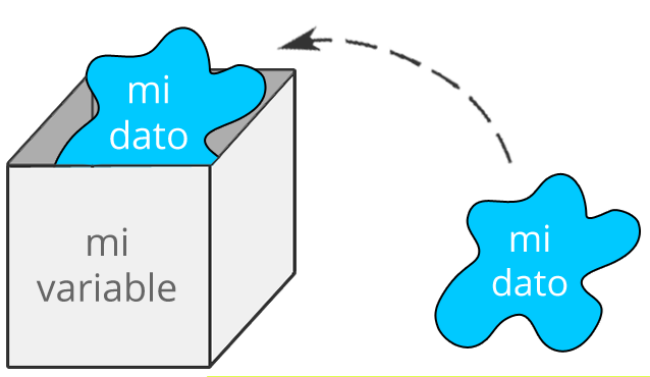
Sitio de descargas npm: https://www.npmjs.com/

Tipos de datos en Javascript

Tipo de dato

Es el atributo que especifica la clase de dato que almacena la variable.

Especifica con qué estaremos trabajando, para que la computadora reconozca qué operaciones puede hacer con él.



Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Tipos de datos

- ✓ **Tipo Primitivos:** Incluyen a las cadenas de texto (String), variables booleanas cuyo valor puede ser true o false (Boolean) y números (Number). Además hay dos tipos primitivos especiales que son Null y Undefined. La copia es por valor.
- ✓ También hay que agregar Symbol, y el BigInt (este último a partir del ES2020)
  BigInt mdn documentación

- ✓ **Tipo Objeto:** Incluyen a los objetos (Object), a los arrays (Array) y funciones. La copia es por referencia.
  https://es.javascript.info/object-copy

		Nombre	Descripción
Tipos de datos en Javascript	**Primitivos**	String	Cadenas de texto
		Number	Valores numéricos
		Booleans	True/false
		Null	Tipo especial, contiene null
		Undefined	Tipo especial, contiene undefined.
		Symbol	Es un valor único
		BigInt	Número mayor al que puede alojar un Number
	objeto	predefinidos de Javascript	Date (fecha), RegExp (expresiones regulares), Error
		Definidos por el usuario	Funciones simples, o Clases
		Arrays	Grupo de elementos. Los arrays son objetos similares a una lista, con métodos para recorrerlos y mutarlos.
		Objetos especiales	Objeto global
			Objeto prototipo
			Otros

Variables en Javascript

Variables en Javascript

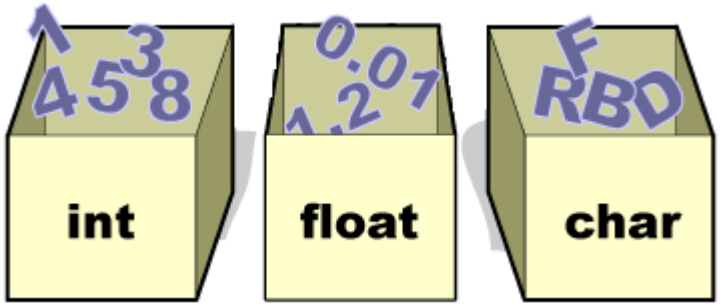
Ahora que hemos comprendido cuáles son los tipos de datos, es importante comprender dónde serán guardados dichos datos. Una variable es un **espacio de memoria** apartado por la computadora, para poder **guardar un dato.**

Tal cual lo dice su nombre, una variable puede **cambiar su valor** si así el programa lo necesita. Ello permite la reutilización de una sola variable, para los casos que se vayan requiriendo.

ACTIVIDAD EN CLASE

Datos y variables

Definir variables que almacenen datos (nombre, edad, precio, nombres de series y películas), mostrar esos valores por consola, incrementar la edad en 1, una serie a la lista y volver a mostrarlas. Compartir la definición en el Visual Studio Code.

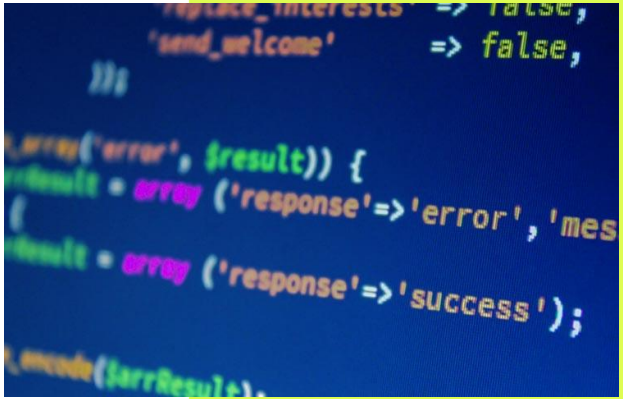


Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Funciones en Javascript

¿Qué es una función?

Es un bloque de código que puede ser llamado en diferentes momentos de la ejecución de nuestro programa. Este conjunto o bloque de instrucciones, trabaja sobre un scope interno (conocido como scope o ámbito local). Las funciones pueden encontrarse en su sintaxis básica o en su notación flecha.



Ejemplos de los diferentes tipos de funciones

1.js

```
function nombreDeLaFuncion(parametros){
    /*Cuerpo de la función, todas las instrucciones
    * internas que necesitamos que la función
    * realice.
    */
   let variableParaMiFuncion=2;
   return variableParaMiFuncion;
   /* Con la palabra "return" podemos MANDAR FUERA
   *  DEL SCOPE alguna variable que necesite en
   *  otra parte del código.
   */
}
/*EJEMPLO COMPLETO*/
function sumarDosNumeros(numero1,numero2){
    //resultado sólo existe dentro de la función
    let resultado;
    resultado = numero1+numero2;
    /*Cuando la función acabe, "resultado" muere
    * así que hay que mandarla afuera
    */
   return resultado;
}
//Mandamos llamar a la función con valores reales
let total = sumarDosNumeros(2,3);
console.log(total) //5
```

2.js

```
/*Una función flecha es ANÓNIMA, quiere decir que
* No tiene nombre, pero puede asignarse a una
* variable para poder identificarse.
*/
const identificadorDeFuncion = (parametros) =>{
    /*Cuerpo de la función, todas las instrucciones
    * internas que necesitamos que la función
    * realice.
    */
    let variableParaMiFuncion=2;
    return variableParaMiFuncion;
    /* Con la palabra "return" podemos MANDAR FUERA
    *  DEL SCOPE alguna variable que necesite en
    *  otra parte del código.
    *  La función flecha cuenta con un return
    *  IMPLÍCITO
    */
}
/*EJEMPLO COMPLETO */
const sumarDosNumeros = (numero1,numero2) =>{
    let resultado;
    resultado = numero1+numero2;
    return resultado;
}
const sumarReturnImplicito = (num1,num2)=>num1+num2;
```

PARA RECORDAR

Contenido destacado

La función flecha permite un return implícito, lo cual permite utilizar instrucciones sin llaves. Esto sólo es posible si la función tiene **una instrucción**. Verás muchas de estas en un ambiente laboral.
Si la función flecha sólo tiene un argumento, no es necesario encerrar el parámetro en un paréntesis. Esto sólo es necesario al utilizar dos argumentos o más.

Scopes

Scopes

El scope define el alcance de una variable o constante en un cierto contexto. Esto permite utilizar el mismo nombre para diferentes variables, sin confundir al computador.

El **scope global** afectará a todo el nivel del archivo en el que trabajemos, mientras que el **scope local** afectará a la función o bloque en el que esté declarado.

Ejemplo de scope no válido

```
function exampleFunction() {
   // x solo se puede utilizar en exampleFunction
   const x = 'declarada en el scope local'
   console.log(x)
}

console.log(x) // ReferenceError: x is not defined
```

Si la variable está definida exclusivamente **dentro de la función**, no será accesible desde fuera de la misma o desde otras funciones.

Ejemplo de scope válido

```
const x = 'declarada en el scope global'

function example() {
   console.log(x) // x existe acá adentro
}

example() // esto no lanza error

console.log(x) // x existe acá afuera también
```

El siguiente código es válido debido a que la variable se declara **fuera de la función**, lo que la hace global.

Template strings

Template strings

- ✓ Funcionan como un **superset** de una string.
- ✓ Permite **incrustar** información dentro de ella, evitando la concatenación.
- ✓ Reconoce los saltos de línea, para mantener el formato.
- ✓ Pero más importante aún es que no presenta el límite de caracteres de una string normal (Permitiendo hacer estructuras más complejas, como plantillas)

```
`texto de cadena de caracteres`

`línea 1 de la cadena de caracteres
 línea 2 de la cadena de caracteres`

`texto de cadena de caracteres ${expresión} texto adicional`
```

Ejercicio:

Funciones

Consigna: Definir una función “mostrarLista”, la cual recibirá un arreglo con elementos como parámetro.

- ✓ Si la lista está vacía, devolver un mensaje indicando “Lista vacía”.
- ✓ Si la lista cuenta con elementos, mostrarlos 1 por 1 en consola. Finalizar el proceso devolviendo la longitud de la lista (Utilizar template strings)
- ✓ Invocar la función con los casos de prueba.

Se espera una duración de 10 minutos.

Break

¡Unos minutos y volvemos!

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

¿Cuál dirías que es el resultado del siguiente trozo de código?

```
264  let numero1=104;
265
266  const suma5 = (numero)=>{
267      let agrega=5;
268      return numero+agrega;
269  }
270
271  suma5(numero1);
272  console.log(numero1);
273
```

- 5
- 104
- undefined
- 109

¿Cuál dirías que es el resultado del siguiente trozo de código?

```
250  let nombre1="Diego";
251  let cantLetras=0;
252
253  const cuentaCaracteres = (palabra) => {
254      let cantLetras=palabra.lenght;
255      return cantLetras;
256  }
257
258  cantLetras=cuentaCaracteres(nombre1);
259  console.log(cantLetras);
```

- **5**
- **0**
- **undefined**
- **Uncaught ReferenceError: nombre1 is not defined**

Universidad Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Closures en Javascript

Closures

Una clausura o closure es una función que puede mantener variables declaradas de manera interna, además de contar con una función que pueda acceder a ambos scopes, tanto el suyo como el de su función padre, logrando así un efecto de **variable privada.**

Este efecto de encapsulamiento se utilizaba con anterioridad debido a la falta de la implementación de **clases** en Javascript. Sin embargo, al introducirse las clases, el cambio fue muy notorio, dejando a las closures en desuso, o bien en casos de uso muy limitados e insuficientes para códigos enterprise.

Uso de clases en Javascript

Clases

Una clase es una representación de una entidad. Nos sirve para modelar múltiples cosas como: un auto, una persona, o bien cosas más abstractas como: un administrador de archivos, un conector a base de datos, etc.

Las clases funcionan como **moldes**, por lo que una vez definida una de éstas, podemos crear múltiples objetos con la misma forma y con las mismas funcionalidades. A éstos objetos se les llama **instancias.**

Declaración de clases

Ejemplo de una estructura general de clases, con métodos y variables

clases.js

clases.js > nombreDeMiClase

```
/*Comenzamos utilizando la palabra reservada "class" */
class nombreDeMiClase{
    /*Una clase cuenta con un método(función) constructor, éste se ejecutará CADA VEZ QUE INSTANCÍE UN OBJETO */
    constructor(parametrosDeCreacion){
        console.log("Nuevo objeto creado");
        this.variableInterna=2;
        /*Cada instancia de la clase contará con variables internas, para poder declararlas y utilizarlas
        * necesitamos colocar un "this." antes de la variable.
        */
    }

    static variableEstatica = 4;
    /*La palabra "static" es una variable que puede utilizarse SIN NECESITAR UNA INSTANCIA, además, todas sus
    * instancias pueden acceder a ella de igual manera. SI ALGUNA INSTANCIA CAMBIA LA VARIABLE ESTÁTICA, todas las
    * instancias se enterarán.
    */
    metodo1(){
        /*Los métodos son funciones que sólo puede utilizar una instancia de la clase.*/
        console.log("¡Soy un método de la clase!")
    }
    metodo2 = () =>{
        console.log(`Soy una función flecha, puedo incrustar variables: ${this.variableInterna}, todo dentro
        de una clase. ¡Una locura!
        `)
    }
}
```

clases.js

```
/*Una vez que terminé de definir mi clase, es hora de instanciar*/
/**
 * Usaremos el operador "new" El cual crea una instancia de la clase.
 */
let instancia = new nombreDeMiClase(); // Se ejecutará el constructor diciendo "Nuevo objeto creado" (constructor)

/*Nota cómo ahora la instancia cuenta con las variables y métodos definidos en la clase previamente */
console.log(instancia.variableInterna);
instancia.metodo1();
instancia.metodo2();
/*Para usar la variable estática, no es necesaria la instancia, simplemente lo llamamos desde la clase */

nombreDeMiClase.variableEstatica;

/*La magia está en que, como la variable es un molde, se pueden crear múltiples instancias de ésta*/

let instancia_2 = new nombreDeMiClase(); // Se ejecutará el constructor diciendo "Nuevo objeto creado" (constructor)
let instancia_3 = new nombreDeMiClase(); // Se ejecutará el constructor diciendo "Nuevo objeto creado" (constructor)

/*¡Cada instancia será diferente de las otras en cuanto a sus variables y métodos (excepto los static) */
```

Ejemplo en vivo

- ✓ Se declarará una clase **Persona,** la cual debe crearse con un nombre que identifique la instancia.
- ✓ Además, habrá una variable estática utilizable para todos.
- ✓ Se comprobará la individualidad entre las dos instancias.

clases.js

clases.js > Persona > getEspecie

```
48
49  class Persona{
50      constructor(nombre){
51          this.nombre = nombre;
52      }
53      static especie = "humano";
54      saludar = () =>{
55          console.log(`¡Hola, soy ${this.nombre}, mucho gusto!`)
56      }
57      getEspecie = () =>{
58          console.log(`Aunque no lo creas, soy un ${Persona.especie}`)
59      }
60  }
61  let persona1 = new Persona("Jorge");
62  let persona2 = new Persona("Catalina");
63  persona1.saludar();
64  persona2.saludar();
65  persona1.getEspecie();
66  persona2.getEspecie();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL powershell

```
¡Hola, soy Jorge, mucho gusto!
¡Hola, soy Catalina, mucho gusto!
Aunque no lo creas, soy un humano
Aunque no lo creas, soy un humano
```

ACTIVIDAD EN CLASE

Creación de una clase contador

Consigna: **Se creará una clase que permitirá llevar cuentas individuales según cada responsable.**

- ✓ Definir clase Contador
- ✓ La clase se creará con un nombre, representando al responsable del contador.
- ✓ El contador debe inicializarse en 0
- ✓ Debe existir una variable estática que funcione como **contador global** de todas las instancias de contador creadas.

Se espera una duración de 10 minutos.

HANDS ON LAB

Creación de una clase contador

- ✓ Definir el método getResponsable, el cual debe devolver el responsable de dicho contador.
- ✓ Definir el método contar, el cual debe incrementar, tanto su cuenta individual, como la cuenta global.
- ✓ Definir el método getCuentaIndividual, el cual debe devolver sólo la cuenta individual del contador
- ✓ Definir el método getCuentaGlobal, el cual debe devolver la variable estática con el conteo global.
- ✓ Realizar prueba de individualidad entre las instancias.

¿Preguntas?

Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región

Resumen
de la clase hoy

- ✓ Introducción a la materia
- ✓ Primeros pasos con JavaScript

Muchas gracias.

Universidad
Provincial del Sudoeste
Promoviendo el Desarrollo Armónico de la Región