CONCEPTOS Y PARADIGMAS DE LENGUAJES DE PROGRAMACIÓN
TRABAJO INTEGRADOR

Grupo: 2

Lenguajes asignados: PHP y JavaScript

Integrantes del grupo:

- Alvarez, Cristian Gabriel 16048/2
- Azcona, Marcos 15821/2
- Fazzano, Juan Manuel 16173/7
- Massacesi, Franco 15333/0

Bibliografía utilizada

- https://www.php.net/manual/es/
- https://www.php.net/manual/es/intro-whatis.php
- https://www.php.net/manual/es/language.variables.php
- https://www.php.net/manual/es/language.types.php
- https://www.php.net/manual/es/language.operators.php
- https://www.php.net/manual/es/language.control-structures.php
- https://www.php.net/manual/es/language.expressions.php
- https://www.php.net/manual/es/intro.apc.php
- https://www.php.net/manual/es/language.exceptions.php
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction
- https://devdocs.io/javascript/
- https://hackernoon.com/webassembly-the-journey-jit-compilers-dfa4081a6ffb
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Grammar_and_Types
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
- https://www.w3schools.com/js/js_type_conversion.asp

Introducción a los lenguajes analizados:

Javascript: Es un lenguaje de scripting que fue concebido para la creación de sitios web orientado al client-side por medio de la manipulación del DOM, una representación del documento HTML como nodos y objetos, permitiendo así cambiar su estructura, estilo y contenido.
PHP: Es un lenguaje de scripting de propósito general concebido para desarrollo web más orientado del lado del servidor (BACKEND). Para generar scripts del lado del servidor, es necesario el analizador de PHP, un servidor web y un browser. Para ejecutar el servidor se

necesita tener instalado PHP, pudiendo acceder al resultado del programa por medio del mismo. A su vez, este lenguaje permite crear aplicaciones de escritorio(aunque no es su fuerte)

Como se mencionó anteriormente, ambos son lenguajes de scripting que fueron concebidos para el desarrollo web, poniendo cada uno su fuerte en el lado del servidor, y otro en el lado del cliente. Por lo tanto, para poder realizar una comparación objetiva, se establecerán las diferencias a partir de los criterios de evaluación además de aspectos sintácticos y semánticos de los lenguajes.

Enunciado:

A. Enuncie y compare distintas características (o criterios de evaluación) de cada uno de los lenguajes asignados fundamentando cada una con ejemplos de código.
B. Defina y compare diferentes aspectos de la sintaxis que Ud. considere. Ejemplifique.
C. Enuncie y compare distintos aspectos semánticos (tanto de la semántica estática y dinámica) de cada uno de los lenguajes asignados.
D. Defina una porción de código donde pueda apreciarse las características más relevantes de las variables en cuanto a sus atributos. Elija alguno de los lenguajes asignados que presente mayores posibilidades para mostrar estas características y desarrolle el ejercicio de la misma forma que se realiza en la práctica. Luego, si es necesario realice las explicaciones que permitan una mayor comprensión del ejercicio.
E. Mencione y compare entre ambos lenguajes los diferentes tipos de parámetros y características de su implementación.
F. Analice y explique la fortaleza del sistema de tipos de cada uno de los lenguajes asignados.
G. Enuncie las características más importantes del manejo de excepciones que presentan los lenguajes asignados.
H. Conclusión sobre los lenguajes: Realice una entrevista a un usuario/s experimentado de los lenguajes asignados con el fin de obtener una opinión acerca del lenguaje respecto de los temas tratados en el trabajo. Esta conclusión no debe superar una carilla.
I. Conclusión sobre el trabajo: Realice una conclusión mencionando los aportes que le generó la realización del trabajo en comparación con sus conocimientos previos de los lenguajes asignados.

A)Evaluacion y comparacion de caracteristicas de ambos lenguajes

Simplicidad y legibilidad

Con respecto a la estructura, ambos presentan similitudes ya que tienen una etiqueta de comienzo, bloque de código y etiqueta de finalización. Dichas etiquetas son utilizadas los intérpretes para saber dónde tienen que empezar y terminar la interpretación. Además nos permite, si estamos trabajando con código incrustado en HTML, poder identificar dónde comienza y dónde termina el script escrito en cada uno de los dos lenguaje.

Con respecto a las componentes elementales
Codificación: Tanto **Javascript** como **PHP** presentan la codificación de caracteres UTF-8 donde para la representación de un carácter se utiliza entre 1 y 4 Bytes.

Tipos de datos: **JavaScript**, a diferencia de **PHP** maneja muchos tipos de datos, compartiendo ambos los tipos boolean, object, string y null, pero agregando nuevas categorías como: **Numbers, Undefined, Symbol;** mientras que en **PHP** se manejan las representaciones tradicionales de los números: **Integer,Float/double**.

Operadores: Ambos lenguajes presentan los operadores de asignación comunes como =, +=, -=, *=, /=, operadores de comparación como **==** , **!=** , **>** , **>=**, **<=** , **<** y **===** (este último compara que los valores y su tipo sean iguales)
Javascript por su parte agrega el operador de comparación **!===** , que verifica que los operandos sean del mismo tipo y no iguales, o de diferente tipo.
Ambos presentan operadores lógicos como AND (&), OR (|) entre otros.
Como podemos ver ambos lenguajes manejan muchos componentes elementales lo que puede volver al lenguaje muy potente pero afecta su simplicidad.

Con respecto a factores como *misma semántica y distinta sintaxis* tenemos, por ejemplo, la declaración de un array, donde php y js presentan dos formas sintácticamente distintas que tienen el mismo concepto semántico

- JS: array = Array() y Array = [ ]
- PHP: $array = array() y $array = [ ]

Claridad en los bindings

Tanto en PHP como JS se presenta **binding dinámico** con **declaración inferida**, ya que la ligadura se hace en tiempo de ejecución y el tipo es definido de acuerdo al valor que se asigna, lo que nos permite cambiar el tipo de una variable a lo largo del programa
Ejemplos:

- JS: var x = 2; x = ‘hola’ ; x = Array(1,2,3)
- PHP: $x = 2; $x = ‘hola’; $x = array(1,2,3);

Tiempo de Vida: el l-valor de una variable está ligado al contexto en el que es declarada. Es decir, si una variable es declarada en el contexto de una función, su l-valor estará ligado a la misma, por lo tanto su tiempo de vida será mientras exista la función.
Alcance: el scope de una variable está definido por el contexto en que fue declarada. Si fue declarado en el contexto de una función, su scope es local a la función, por contraparte, si fue declarado en el contexto global, su scope será para todo el script.
Ejemplos:

- **JS:** let x = 2; function a() { let x = 4 } foo(); console.log(x);
- **PHP:** $x = 2; function a(){ $x = 4}; echo $x

Ambos imprimen 2 porque el alcance de x es local a la función y su tiempo de vida será mientras la función exista. En el caso de JS es particular, pues la variable x está siendo "enmascarada", por tanto el x de la función no es el mismo que el definido fuera de la función

Una diferencia a tener en cuenta es que en PHP cuando se quiere usar una variable global dentro de una función se debe anteponer la palabra clave *global* antes del nombre de la variable, cosa que no hace falta en JS.
Ejemplo:

- **JS:** let x = 2; function a() { x = 4; } foo(); console.log(x);
- **PHP:** $x = 2; function a() { global $x; $x = 4; } a(); echo $x;

Ambos imprimen 4 porque el alcance de x refiere a la variable global definida fuera de la función. En el caso de PHP se hace de forma explícita mediante la palabra reservada global, mientras que JS lo hace implícitamente creando a 'x' como una variable "undeclared global" al no encontrar ninguna de las palabras claves como *let o var* que redefinen la variable como local a la función.
Basándonos en los ejemplos podemos llegar a la conclusión de que JAVASCRIPT puede llegar a producir ambigüedades en el uso de las etiquetas *var* y *let* al momento de definir variables en un bloque y su alcance.

Confiabilidad

Con respecto al ***chequeo de tipos*** podemos afirmar que ambos son poco confiables debido a que JS es débilmente tipado, permitiendo así operaciones aritméticas entre distintos tipos con restricciones. PHP es fuertemente tipado pero permite realizar algunas operaciones entre diferentes tipos aplicando una auto conversión de tipos.

PHP: permite operaciones aritméticas entre un string y un valor numérico, al utilizar una operador aritmético como +,-,/,* lo que se hace es una auto conversión de tipos, donde se intenta convertir los strings a operadores numéricos, para estos los strings deben contener en su interior una cadena numérica válida como: '12' o '1.52', en el caso del '12' se convierte a 12 integer, en el caso de '1.52' se convierte a 1.52 double, nótese que si la cadena fuese 'hola' o '12abc', no se podría realizar una conversión.

JS: Con la operación "*suma*", se realiza la concatenación de los operandos, siempre que los involucrados no sean ambos de tipo "*number*". Sin embargo, en el resto de operaciones (resta, multiplicación y división) en donde se ven involucrados strings y su contenido sea un número (sean reales o enteros), y se realiza las operaciones mencionadas entre un string y un number, number y number, o string y string, se devolverá el resultado de la operación aritmética entre los mismos y el tipo de salida será de tipo "*number*".
Ejemplos:

- ('12' + 3.2) → '123.2' // Devuelve la concatenación

- (‘12’ - ‘12.5’) → -0.5 de tipo number
- (‘12a’ - 3) → NaN, pues para realizar la operación el contenido del string no es un número

En ambos lenguajes se da el caso particular de que, a la hora de sumar, restar, multiplicar y/o dividir un valor numérico o string (que pueda ser convertido a número) con un boolean, se efectuará la operación, ya que true se toma como 1 y false como 0, teniendo en cuenta que en JS la operación de suma, en caso de que ambos operandos no sean numbers, hará la concatenación.
Ejemplos:

- (‘12’ -) → 12 - 0 → 12
- (33 * true) → 33 * 1 → 33

Sin embargo, en ambos lenguajes (ante las mismas situaciones), en algunos casos tienen el mismo comportamiento, o tienen un comportamiento distinto hasta el punto de levantar una excepción.
Ejemplo: *Sumar un string con un número,sea entero o real (‘12’ + 34)*. Por lo mencionado anteriormente, JS concatena, mientras que PHP realizará la operación devolviendo un **integer.**

Excepciones
Tanto JS como PHP tienen manejo de excepciones, las cuales permiten que, ante un error, el programa no finalice su ejecución y pueda alterarse el flujo del mismo.

Soporte

JS no requiere instalación formal, sino que simplemente, con insertar código JS en un documento con extensión .js o .html, será ejecutado.
A su vez, los scripts JS pueden ser ejecutados, no sólo en un browser, sino también en servidores y cualquier dispositivo que tenga un programa especial llamado “*Javascript Engine*”, un motor que utiliza la compilación JIT (Just in Time) para su ejecución. Existen muchos motores como V8 (para chrome y opera), SpiderMonkey (en Firefox), ChakraCore (en Microsoft Edge), etc. Muchos de estos pueden ser usados para el lado del cliente como del servidor (por ejemplo V8 se usa del lado del servidor para el framework *node.js*) como se mencionó en la introducción.

PHP puede utilizarse en todos los sistemas operativos principales como: linux, windows, mac OS X, Risc OS entre otros. Su código fuente y distribuciones binarias para windows pueden descargarse en https://www.php.net/downloads.php
En cuanto a servidores, PHP admite la mayoría de los disponibles de hoy en dia. Algunos de ellos son: Apache, IIS, esto incluye servidores que puedan utilizar el binario de PHP FastCGI, como lighttpd y nginx.
Además de tener la libertad utilizarlo en varios sistemas operativos y servidores, otra característica que presenta este lenguaje es el soporte a un gran abanico de bases de datos.

También cuenta con soporte para comunicarse con otros servicios usando protocolos tales como LDAP, IMAP, SNMP, NNTP, POP3, HTTP, COM (en Windows) y muchos otros.

Abstracción

Ambos lenguajes presentan el paradigma orientado a objetos, permitiendo abstraer funcionalidades y atributos generales por medio de clases.
Presentan abstracción con respecto a ciertas estructuras de datos, como los HashMap y listas como arrays, sin necesidad de saber cómo fueron implementados.
Por otra parte, tanto JS como PHP abstraen al desarrollador de saber cómo comunicar sus scripts con un browser o un servidor respectivamente. Por tanto, cuando se escribe código en ambos lenguajes, no es necesario configurar los scripts para comunicarse con el browser o el servidor, sino que con insertar el código, estos pueden interpretar y ejecutar el script.

Ortogonalidad

Ambos lenguajes permiten combinar varias de sus componentes y características, generando resultados con significado.
Ejemplos:

- Javascript al ser débilmente tipado, las funciones pueden recibir y retornar parámetros de cualquier tipo, los Arrays permiten guardar todo tipo de datos en la misma estructura como: Array(1, Clase_Persona, “String”, true) y , como se mencionó en la simplicidad, operaciones entre tipos distintos como A = ‘2’ - 2 y A = 2 - 2, ambos guardando 0 en A de tipo number.
- PHP al igual que JS permite pasar como parámetro y devolver cualquier tipo de dato. Además de poder formar arreglos heterogéneos que nos permitirían declarar un arreglo de la siguiente forma: $*A= Array(1 => “trabajo” , ”hola” => 2)* como si fuese un HashMap.

Eficiencia

Para hablar de “*eficiencia”* en los lenguajes, debemos indagar en cómo se realiza su ejecución. Por un lado, JS se ejecuta por medio de un intérprete, pero por medio de otra metodología subsana las deficiencias que puede mostrar el intérprete (como reinterpretar por cada iteración de un for). A esta se la conoce como JIT (Just in Time), la cual agregar al Javascript engine lo que se conoce como “profiler”, una entidad que se encarga de llevar registro de la ejecución y los tipos usados. Entonces, el script comienza a ser ejecutado por el intérprete. Cuando una pieza de código es ejecutada múltiples veces, el “profiler” marca esta pieza de código como “warm” y entra en acción el “baseline compiler”, encargado de compilar esa pieza de código para que, cuando sea ejecutada, sólo deba ejecutarse el código ya compilado. Si este código compilado se ejecuta varias veces, el “profiler” marca esa pieza de código como “hot”, dejándole la tarea al “*optimizer compiler*”, el cual genera código más eficiente.
PHP al igual que JS es interpretado, lo que degrada la performance de tiempo de ejecución, pero nos puede facilitar la comprensión de errores, aunque PHP nos provee una forma de incrementar la performance utilizando caches donde se guarda código intermedio de PHP.

B)

Asignación de variables

JS	*PHP*
etiqueta *nombre_de_variable* ***;*** **etiqueta** *nombre_de_variable* **=** *valor* ***;***	**$***nombre_de_variable;* **$***nombre_de_variable* **=** *valor* **;** **$***nombre_de_variable* **= &***variable***;** //asigna por referencia
etiqueta : 'var' - **'let'** *Puede no llevar etiqueta*	
Ejemplo: **var** numeroUno**;** **let** numeroDos **=** 42; numeroTres **=** 42**;**	*Ejemplo*: **$**var1 **=** 12**;** **$**var2 **= &$**var1**;**

En ambos casos se puede apreciar la utilización de una palabra reservada para la declaración de una variable, las cuales son **var**, **let** o **''** en JS ,y el carácter **$** en PHP seguido del nombre de la variable. En ambos casos se utiliza el carácter **=,** al que le sigue el valor correspondiente, y también existe la opción de crearlas sin un valor inicial. En el caso de PHP, si se le asigna por referencia una variable, esta comenzará con el símbolo **&** y finalizará con el nombre de la variable (teniendo en cuenta que las variables comienzan con **$).**
Para finalizar ambas expresiones se termina colocando un **;**
En Javascript, como ya se mencionó, tenemos dos palabras clave para la declaración de una variable, ***let*** y ***var***. Las variables declaradas con la palabra reservada **let** tienen un scope local, es decir, su alcance se limita al bloque en el que se encuentran declaradas (local a una función, o local al programa principal). Por contraparte, **var** se acopla al contexto, esto quiere decir, que dentro de una función permite a la variable tener scope local a ella y a su vez, fuera de la función, permite ser accedida; como también permite que su declaración en el bloque "main" tenga un scope global.

Condicional

JS

```
if (  condición ) {
   bloque
}else {
   bloque
}
```

Ejemplo:

```
if ( variableComparado == True ){
  DioComoResultadoVerdadero();
} else {
  DioComoResultadoFalso();
}
```

PHP

```
if (  condición ) {
   bloque
}else {
   bloque
}
```

Ejemplo:

```
if ($a > $b) {
   echo "a es mayor que b";
} elseif ($a == $b) {
   echo "a es igual que b";
} else {
   echo "a es menor que b";
}
```

En este caso ambas son iguales tanto en la sintaxis concreta, cómo en la abstracta; ya que comparten reglas léxicas y tienen la misma estructura. Ambos esperan las palabras reservadas **if** más una condición que tendrá que devolver un resultado booleano, esta estará entre 'paréntesis' **()**. El bloque que se ejecutará a partir de la veracidad de la condición estará encerrado entre 'llaves' **{}**.

PHP también presenta una sintaxis alternativa para algunas estructuras de control (if, while, for ,etc) donde se reemplaza el símbolo que abre la 'llave' **{** por los 'dos puntos' **:**, y la llave de cierre se reemplaza por la palabra reservada **endif;** (para el caso de los if), **endwhile;** (para los while ():) y **endfor;** (para los for ():).

Para concatenar varios **if**, PHP cuenta con la palabra clave **elseif,** mientras que en JS se tendrán que separar las words resultando como: **else if**.

Repetición: for

JS

```
for (expr1; expr2; expr3){
   bloque
}
```

Ejemplo:

```
for (var i = 0; i < 9; i++){
  console.log(i);
}
```

PHP

```
for (expr1; expr2; expr3){
   bloque
}
```

Ejemplo:

```
for ($i = 1; $i <= 10; $i++){
   echo $i;
}
```

expr1: una expresión (incluidas las expresiones de asignación) o declaración de variable.
expr2: una expresión que se evaluará antes de cada iteración de bucle.
expr3: una expresión que se evaluará al final de cada iteración de bucle.

En el caso de la estructura de control **for** tanto para JS y PHP se comparte la palabra reservada **for**, seguida de un bloque condicional encerrado entre paréntesis. En este bloque nos encontramos 3 expresiones separadas por **;** donde cada expresión representa cosas distintas.

Funciones

JS

```
function nombre_de_la_funcion ( parámetros) {
   bloque;
   Opcionalmente puede ir el return
}

const nombre_de_la_funcion = (parámetros) => { bloque }

Ejemplo:

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

PHP

```
function nombre_de_la_funcion ( parámetros) {
   bloque;   Opcionalmente puede ir el return
}

Ejemplo:

function foo ($arg_1, $arg_2, /* ..., */ $arg_n)
{
   echo "Función de ejemplo.\n";
   return $valor_devuelto;
}
```

Ambos lenguajes son iguales respecto a la sintaxis abstracta, ya que ambas tienen la misma estructura: **function** *nombre_de_la_funcion* **(** *parámetros*) **{**

Bloque;

}

Clases

JS

```
var Rectangulo = class {
  bloque
  }

class ClaseSencilla{
    constructor(){ }
    constructor(parametros){ }//métodos de la clase
}
```

PHP

```
class ClaseSencilla
{
   // Declaración de una propiedad
   public $var = 'un valor predeterminado';

  // Declaración de un método
   public function mostrarVar() {
      echo $this->var;
   }}
```

Tanto JS como PHP usan la palabra reservada **class**, sin embargo JS cuenta con una forma particular para crear clases por medio de una asignación directamente a una variable.

Nombre de variables, clases

En ambos lenguajes, un identificador no puede comenzar con un valor numérico, pero sí pueden comenzar con una letra o signo $ o símbolo ‘_’.

C.

Semántica Estática:

- Asignación a variables inexistentes o existentes
- Parámetros de función

En ambos lenguajes, si se realizan operaciones con variables y estas no están declaradas,se producirá un error debido a que fue detectado por la semántica estática.

Como mencionamos previamente, ambos lenguajes permiten realizar operaciones aritméticas entre diferentes tipos de datos. Respecto a PHP se podrá siempre y cuando los operandos que no son de tipo integer o double puedan convertirse a estos.

Con respecto a los parámetros obligatorios, la semántica estática está encargada de verificar que la cantidad de parámetros obligatorios definidos para la función sea la misma que se pasa al momento de la definición de la función. En JS no importa si la cantidad es mayor o menor, no se produce error y se procede a ejecutar la función. En PHP se produce error si la cantidad de argumentos pasados a la función es menor a la que se esperaba.
Ejemplo del caso particular JS:

const function = (parametro1) => { console.log(parametro1) }
funcion() → En este caso, se imprimirá undefined, ya que las variables no asignadas tienen por defecto el valor *undefined*

Ejemplo del caso particular PHP:

function funcion1($x){}
funcion1() → Levantará una excepción

Semántica Dinámica:

- Creación de variables (valores por defecto y asignación de punteros)
- Operaciones aritméticas y conversión de tipos

Creación de variables:

- En PHP la asignación de una variable (ya sea de un valor u otra variable), siempre se realiza por valor, a menos que se especifique lo contrario anteponiendo el carácter “&” antes del nombre de la variable a la cual queremos copiar la referencia. Se debe tener en cuenta que esta última opción solo está disponible para asignar variables definidas.
- En JS la asignación de un valor a una variable se realiza por valor, sin embargo, si la asignación a una variable es otra variable, entonces será por referencia.

Nótese el siguiente caso particular:

JS	PHP

a = ‘hola’ //Se crea una referencia nueva
b = a //Se asigna la referencia de ‘***a***’ en ***‘b’***
a = 10 //Se crea una referencia nueva
console.log(b) //Imprime hola
	<?php
	$a= 'hola';//asignación por valor
	$b=&$a;//asignación por referencia
	$a=10;//asignación por valor
	echo($b);//imprime 10
	?>

En este caso, como en JS, a diferencia de PHP, cada asignación o declaración genera una nueva referencia, la variable *a inicializada con ‘hola’* **no es la misma** que *la variable a*

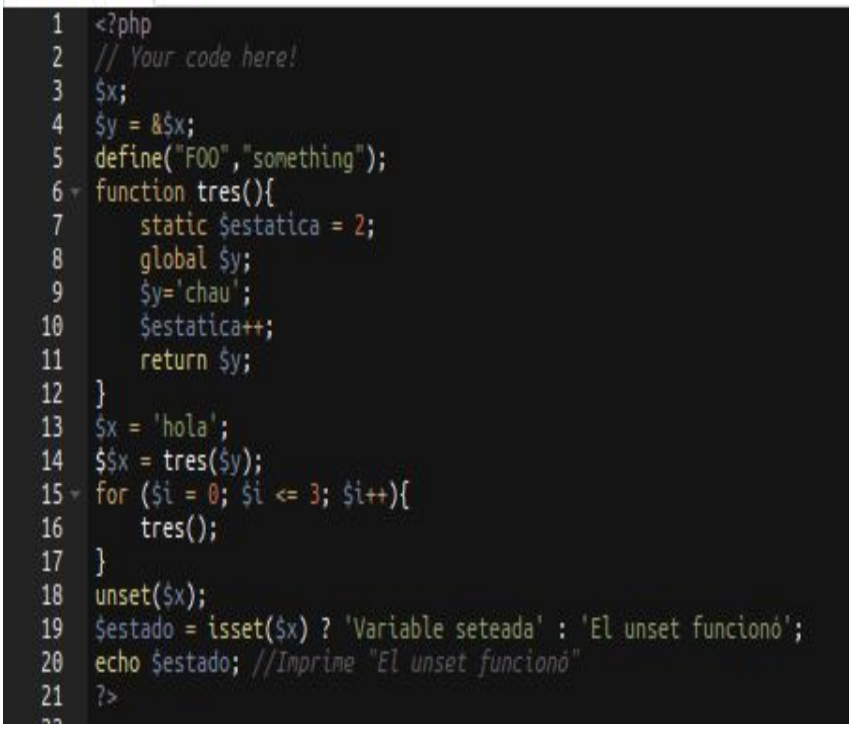
inicializada con 10, por tanto, el resultado en JS es ‘hola’ y PHP, como no se vuelve a generar una nueva referencia, se imprime 10.

Operaciones aritméticas

Ejemplo	JS	PHP
imprimir(‘12’ + 10)	Concatena 12 con 10 y devuelve un string ‘1210’	Realiza operacion aritmetica convirtiendo el ‘12’ de tipo string a 12 integer, por lo que el resultado será el integer 22
a)imprimir(‘12’ - 10) b)imprimir(‘12’ * 10) c)imprimir(‘12’ / 10)	Efectúa la operación aritmética y devuelve un numbers a)2, b)120 y c)1.2	Al igual que en ejemplo anterior se realizará la conversión de tipos, y luego la operación aritmética correspondiente entre tipos enteros dando los resultados a)2, b)120, c) 1.2
arreglo=Array(1,2,3) imprimir(arreglo+’ejemplo3’)	Concatena el arreglo con ‘ejemplo3’ como string ‘123ejemplo3’	Falla ya que no se puede realizar una conversión de tipo automática sobre el arreglo para convertirlo en un operador numérico válido
arreglo=Array(1,2,3) imprimir(arreglo-’ejemplo3’) imprimir(arreglo*’ejemplo3’) imprimir(arreglo/’ejemplo3’)	En este caso, JS devolverá NaN	Falla de igual manera que el ejemplo anterior

---

D)



```php
<?php
// Your code here!
$x;
$y = &$x;
define("FOO","something");
function tres(){
    static $estatica = 2;
    global $y;
    $y='chau';
    $estatica++;
    return $y;
}
$x = 'hola';
$$x = tres($y);
for ($i = 0; $i <= 3; $i++){
    tres();
}
unset($x);
$estado = isset($x) ? 'Variable seteada' : 'El unset funcionó';
echo $estado; //Imprime "El unset funcionó"
?>
```

Identificador	L-valor	R-valor	Alcance	T.V.
$x	dinámica	Basura	3 - 6 / 13 - 21	3 - 18
$y	dinámica	Basura	4 - 5 / 8 - 21	4 - 21
$$x / $$y / $chau	dinámica	‘chau’	14 - 21	14 - 21
$i	dinámica	0	15 - 21	15 - 21
$estatica	estática	2	7 - 12	7 - 21
tres (función)	dinámica	-	6 - 21	6 - 12
FOO	automática	‘something’	5 - 21	5 - 21

En PHP se pueden utilizar las variables variables, es decir que permite utilizar el valor de una variable como el identificador de otra. En este ejemplo se puede ver que lo que devuelve la función dos() se almacena en la variable $$x, en ese momento la variable $x tiene almacenado el String ‘hola’. Esto significa que la variable $$x es la variable $hola, ya que ese es el valor de la variable $x.

E. PARAMETROS

PHP

En PHP los argumentos son una lista de expresiones separadas por comas evaluados de izquierda a derecha.

De forma predeterminada los argumentos se pasan por valor, por esto cuando mandamos una variable como parámetro real, su contenido se copia en el parámetro formal, y cada modificación se hace sobre este, lo que nos garantiza que al retornar de la función la variable pasada como parámetro siga teniendo el mismo valor que antes del llamado.

```
<?php
function funcion1($parametro){
   echo($parametro);        → imprime: 10
   $parametro = 100;
   echo($parametro);        → imprime: 100
}

$x=10;
funcion1($x);
echo($x);                   → imprime: 10
?>
```

En el caso que quisiéramos un pasaje por referencia debemos anteponer el carácter & antes del nombre de la variable en la función.

```
<?php
function funcion1(&$parametro){
   echo($parametro);              → imprime: 10
   $parametro = 100;
   echo($parametro);              → imprime: 100
}

$x=10;
funcion1($x);
echo($x);          → imprime: 100
?>
```

A la vez nos permite iniciar con valores predeterminados los argumentos que recibe una función, en entonces se tomará ese valor predeterminado cuando no se le pase ningún valor en el parámetro real.

```
<?php
function hacer_café($tipo = "capuchino")
{
   return "Hacer una taza de $tipo.\n";
}
echo hacer_café();              → imprime: Hacer una taza de capuchino.
echo hacer_café(null);             → imprime: Hacer una taza de .
echo hacer_café("espresso");                → imprime: Hacer una taza de espresso.
```

```
?>
```

Las listas de argumentos de longitud variable también están soportadas por PHP, las listas de argumentos pueden incluir el token "..." antes del nombre del parámetro para denotar que la función acepta un número variable de argumentos. Los argumentos serán pasados a la variable dada como un array, por ejemplo:

```
<?php
function sum(...$números) {
   $acc = 0;
   foreach ($números as $n) {
      $acc += $n;
   }
   return $acc;
}

echo sum(1, 2, 3, 4);     → imprime: 10
?>
```

En PHP podemos pasar como parámetro real a una función los valores de cualquier tipo soportado por el lenguaje, hasta clases definidas por el usuario. Aunque el lenguaje nos provee un mecanismo para validar si el tipo del argumento pasado a la función es del tipo que es esperado, esto se hace anteponiendo en el parámetro formal el tipo que se espera. En el caso de que el tipo del argumento recibido sea diferente al esperado, PHP intentará hacer una conversión de tipo llevando el argumento pasado al tipo que se espera.
Por ejemplo:

```
<?php
function funcion1(int $parametro){

   echo($parametro);    → imprime: 20
   echo(gettype($parametro));   → imprime: integer
}

funcion1("20");
?>
```

Como podemos ver en el ejemplo, el argumento que recibe la función "funcion1" es de tipo string, y el tipo que se espera recibir es integer, en este caso PHP convierte el string "20", a 20 integer, esto lo pudo hacer ya que el string contiene un número.
Podemos ver en el siguiente ejemplo qué sucede cuando no puede llevarse a cabo ésta conversión.

```
<?php
function funcion1(int $parametro){

   echo($parametro);
   echo(gettype($parametro));
}
```

```
funcion1("aaa");
?>
```

La ejecución de este código resultará en un Fatal Error de PHP, específicamente en un TypeError que corta con la ejecución del programa.

JAVASCRIPT

Los parámetros en la llamada a una función son los argumentos de la función. Los argumentos se pasan a las funciones *por valor*. Si la función cambia el valor de un argumento, este cambio no se refleja globalmente en la llamada de la función. Sin embargo, las referencias a objetos también son valores, pero son especiales, ya que si la función cambia las propiedades del objeto referenciado, ese cambio es visible fuera de la función.

En el caso 1, como el valor de la variable “x” no es una referencia a un objeto, el pasaje será por valor, por tanto, cualquier cambio hecho en el bloque de la función no se verá modificado fuera de la misma.

```
x=3
function prueba(x1){ x1 = 2 }
prueba(x)
console.log(x)
```

Los arreglos, diccionarios, WeskSets, etc, son considerados instancias de una clase. Cuando se pasa un arreglo, un diccionario o alguna instancia de clase como parámetros, estos se consideran por referencia. Por ende si su valor es modificado dentro de la función, los cambios se verán reflejados fuera del bloque de la función.

```
console.log(“-----Caso 2-------”)
function prueba2(arreglo1){ arreglo1[1] = 456 }
arreglo=Array(1,2,3)
prueba2(arreglo)
console.log(arreglo)

console.log(“-----Caso 3------”)
function prueba2(diccionario1){ diccionario1[‘marcos’] = ‘pepe’ }
diccionario={‘marcos’:’azcona’}
prueba3(diccionario)
console.log(diccionario)

function cambiarNombrePersona(persona){
        persona.cambiarNombre(‘Franco’)
}
class Persona{
        constructor(){
                this.nombre = ‘Franco’
        }
        cambiarNombre(nuevoNombre){
                this.nombre=nuevoNombre
        }
        getNombre(){
```

```
                return this.nombre
            }
}
persona = new Persona()
console.log("Nombre antes de ser cambiador " + persona.getNombre()) //Franco
cambiarNombrePersona(persona)
console.log("Nombre luego de ser cambiado " + persona.getNombre()) //Franco
```

Existe también los parámetros por defecto, que suelen utilizarse para inicializar los nombres de los parámetros con un valor por defecto en el caso de que no se pase ningún valor o valor indefinido.

```
Function GFG(num1, num2 =2){
        Return num1 * num2
}
console.log(GFG(2))
console.log(GFG(3,3))
```

F. SISTEMA DE TIPOS

Características / Lenguajes	PHP	JavaScript
Fuertemente tipado O Debilmente tipado	**Fuertemente Tipado** ya que provee restricciones sobre cómo llevar a cabo ciertas operaciones entre datos de diferentes tipos.	**Débilmente Tipado** ya que el lenguaje no representa muchas restricciones a los tipos de datos en cuanto a las operaciones que se le pueden realizar.
Reglas de conversión	Posee conversión **Implícita automática**, un ejemplo de esto puede verse cuando utilizamos el operador "+" como fue explicado anteriormente PHP le brinda al programador la oportunidad de realizar conversiones explícitas o "forzar" a una variable para que sea de un determinado tipo donde el nombre del tipo deseado se escribe entre paréntesis antes de la variable que se quiera forzar.	**Implícita automática** (cuando javascript intenta operar con tipos de datos no compatibles, intenta convertirlos al tipo correcto, teniendo así resultados inesperados como por ejemplo ('12' + 3 donde al string 12 lo convierte en tipo number y hace la operación aritmética)) y **Explícita** (por medio de constructores como Number() o String()).

Reglas de inferencia de tipo	PHP posee inferencia de datos, es decir, el tipo de una variable es decidido en tiempo de ejecución por PHP dependiendo del contexto en el que se emplea dicha variable.
	
	$x = 44 //Infiere que es un integer
		Javascript posee una inferencia a nivel de que se deduce el tipo de una variable a partir de las componentes de la misma
		x='Hola' //Infiere que es un string
		y = 22 //Infiere que es un number
		suma = 33 + 55 (como sus componentes son numbers,suma es de tipo numbers)

Tipo de ligadura (estática o dinámica)	Ambos lenguajes presentan tipado dinámico, permitiendo que el tipo de una variable pueda ir cambiando durante la ejecución del programa.	
Nivel de Polimorfismo	PHP soporta herencia simple, lo que permite a una clase tener una sola clase padre. Contando con este mecanismo, varias clases pueden extender a una clase padre donde todas ellas heredan el comportamiento de esta última, pudiendo sobreescribir sus métodos y ante una invocación a estos métodos sobreescritos, comportarse de manera diferente.	Javascript está basado en prototipos, es decir, no diferencia entre clases ni objetos, sino que todos son objetos prototipo ya instanciados, usados como template para obtener las propiedades iniciales de un objeto. Al estar basado en prototipos, la herencia se concibe como la asociación entre un objeto prototipo con una función constructora, donde la herencia de propiedades se hace por medio de una cadena de prototipos. Javascript permite herencia simple (Salvo en ECMAScript6, que por medio de objetos proxy se permite herencia múltiple), donde la subclase prototipo se asocia a la superclase prototipo.
Tipos de datos primitivos	**Tipos Escalares:** boolean integer float (número de punto flotante, también conocido como double) String **Tipos compuestos:** array object callable iterable **Tipos especiales:** resource NULL	**Tipos primitivos**: Symbol Boolean Number String BigInt undefined **Tipos compuestos**: Object Array Map Set WeakSet
Tipos de datos definidos por el Usuario	Clases	Clases/Prototipos Funciones simples

G. EXCEPCIONES

Como ya se dijo en el punto A del informe, ambos lenguajes presentan manejo de excepciones. A continuación entraremos más en detalle respecto a cómo se manejan estos casos especiales en ambos lenguajes.

Tanto PHP como JS presentan la siguiente estructura:

- *Try { Bloque a ejecutar }catch (excepción){ bloque a ejecutar si se levanta excepcion } finally { se ejecuta si hay o no excepción }*

El bloque Try debe contener aquellas instrucciones que puedan llegar a levantar una excepción.

En PHP cada bloque try debe contar con al menos un bloque Catch que contiene el código a ejecutarse de acuerdo al tipo de excepción que maneja.

En el bloque finally se encuentra contenido un bloque de código que se ejecutará siempre, se haya levantado una excepción o no.

En PHP podemos levantar excepciones explícitamente con la palabra clave **throw** donde el objeto lanzado debe ser de la clase **Exception** o subclase de ésta, de otra forma se producirá un Fatal Error de PHP.

Busqueda de manejador en PHP:
En caso de levantarse una excepción en un bloque try que no pueda ser manejada por ningún bloque catch que tenga asociado, si el bloque que levanta la excepción se encuentra contenido dentro de otro bloque try, se comienza a buscar el manejador correspondiente de manera estática, es decir en el bloque try que lo contiene.
Ejemplo:

```
<?php
class MiExcepcion extends Exception { }

try {
   try {
      throw new Exception ;
   } catch (MiExcepción $e) {
      echo("se maneja en el try interno");
   }
} catch (Exception $e) {
   echo ("se maneja en el try externo");
}
?>
```

En este ejemplo se muestra por pantalla “se maneja en el try externo”, esto se debe a que en el try interno, el objeto levantado en el throw es de tipo Exception, pero en el catch asociado solo se tiene definido el manejador para excepciones de tipo MiExcepcion (Excepción definida por el

usuario). Por consiguiente, se busca un manejador correspondiente en el bloque try en el que está contenido, teniendo en cuenta que si este último no tiene el manejador buscado, se producirá un Fatal Error de PHP.

Otro caso que puede presentarse es aquel donde se levanta una excepción dentro de una función y esta no cuenta con un manejador asociado en su bloque estático. Como muestra el siguiente ejemplo.

```
<?php
function funcion1($x) {
   if (!$x) {
      throw new Exception;
   }
   return 1/$x;
 }
try {
      echo funcion1(0) . "\n";
   }
catch (Exception $e) {
   echo ('Excepción capturada en el programa principal');}

?>
```

Puede observarse, que se levanta una excepción dentro de la función “funcion1” y esta no tiene definido un manejador para la misma, lo que hace PHP en estos casos es buscar un manejador correspondiente de manera dinámica, es decir que se va propagando siguiendo la cadena de invocación del proceso. Por esto, ejecutando el ejemplo anterior se tiene como salida “Excepción capturada en el programa principal”.

PHP también nos provee el mecanismo de set_exception_handler(exception_handler), que establece el manejador de excepciones predeterminado si una excepción no es capturada dentro de un bloque try/catch. La ejecución se detendrá después de la llamada a exception_handler.

En **Javascript**, al igual que php, se pueden levantar excepciones por medio de la sentencia *throw excepcion_a_lanzar,* donde en la *excepcion_a_lanzar* se debe especificar el valor a ser levantado. Entre estos valores podemos encontrar los tipos de datos que maneja Javascript ya mencionados, como también estructuras de datos o instancias de la clase Error como la ya mencionada, valueError, SyntaxError, etc.
A diferencia de PHP, aquella excepción que se espera recibir en el bloque (catch (err)), err no especifica el tipo de excepciones que puede manejar el catch sino que es en realidad un *catchID*, un contenedor de la excepción que fue levantada. Por lo tanto, un bloque catch no maneja una excepción específica, sino cualquier excepción que fuese levantada.
Al producirse una excepción, se pueden dar ciertos casos:

Caso 1:

```
try{
        try{
                Throw ValueError
        }catch(ReferenceError){
                console.log("Llegó a esta excepción")
        }
}catch {

}
//Imprime Llegó a esta excepción.
```

Como se explicó al inicio, javascript no intenta matchear el tipo de excepción del manejador con el tipo de excepción levantada, sino que maneja cualquier tipo de excepción.

Caso 2:

```
 try{
        try{
                Throw ValueError
        }catch(ReferenceError){
                console.log("Llegó a esta excepción")
                Throw ('Excepción del bloque más interno')
        }
}catch (err){
        console.log(err)
}
```

En este caso se imprime 'Excepción del bloque más interno', ya que en el bloque más interno se maneja una excepción levantando otra excepción, para lo que el manejador de la misma se comienza a buscar de forma estática. Como en su registro estático se encuentra el manejador, se procede a capturar e imprimir por consola.

Caso 3:

```
function p1(){
        try{
                throw 'Error en p1'
        }finally {
                console.log("Fin de bloque p1")
        }
}
function p2(){
        try {
                p1()
        }catch (err) {
                console.log(err)
        }
}
```

Se imprime "Fin de bloque p1 Error en p1". Como en p1 se levanta una excepción y no se encuentra el manejador, busca en su bloque de forma estática. Al no encontrarlo, procede al finally, imprime por consola y luego se propaga de forma dinámica a p2(). Como en p2() se encuentra dentro un bloque try{}catch{} se captura y maneja imprimiendo en consola.

H. ENTREVISTA a Sergio B. Saraví

1. **¿Por qué elegiste trabajar con estos lenguajes? ¿Y ahora?**

“Fue una decisión que se tomó en el centro de cómputo en su momento, cuando tenía que elegir un lenguaje open source, cómo no había forma de investigar todas las opciones del mercado, se optó por la que teníamos más documentación y por la que nos parecía más apta para el trabajo.
Cuando arrancamos a trabajar en PHP 3 comenzamos con la programación estructurada y usando la base de datos Access. Al tiempo implementamos JavaScript para el front end.
Hoy en día implementamos la programación orientada a objetos, donde ya tenemos unas plantillas armadas para los distintos estereotipos de páginas webs.”

2. **Que tipo de aplicaciones hacen con estas herramientas**

“Donde más implementamos estos lenguajes son en aplicaciones administrativas con bases de datos (MySQL), apps de seguimiento y páginas web para distintos automotores.”

3. **¿Qué ventajas te trajo trabajar con estos lenguajes?**

“Tanto en su momento como hoy en día, las ventaja que nos trae es que al momento de hacer cambios o actualizar algo es mandarlo a desarrollo y listo.”

4. **¿Cuál fue la complejidad de aprender estos lenguajes?**

“En su momento no fue tan fácil por la poca documentación con la que contábamos, pero hoy en día pienso que se aprende más rápido ya que pueden encontrarse distintas metodologías de aprendizaje.”

5. **¿Qué te parece la sintaxis en relación a los otros lenguajes? ¿Y la semántica?**

“Es lo mismo que el resto, ni más facil, ni más dificil”

6. **¿Encontras alguna facilidad o conveniencia al utilizar los parámetros?**

“No, es igual a todos los lenguajes”.

7. **¿Soles utilizar excepciones en alguno de estos lenguajes? ¿En qué casos las utilizan?**

“Intentamos evitarlas constantemente, solo las ponemos si no tenemos tiempo para corregir esos errores, aunque preferimos hardcodearlo, pero a la larga preferimos arreglar el código antes de usar las excepciones”

8. **¿Recomendás estos lenguajes hoy en día?**

“Sí, me parecen dos lenguajes muy útiles hoy en día, y que se puede aprender mucho de ellos.”

I. CONCLUSIONES SOBRE EL TRABAJO

Este trabajo nos ayudó a reforzar los conocimientos teóricos-prácticos que vimos a lo largo de la cursada de la materia. A medida que íbamos avanzando en las prácticas, fuimos investigando y reforzando los conceptos asociandolos con los lenguajes asignados.

Nos gustó mucho el proceso de aprender de cero estos dos lenguajes, nos ayudó a ver lo mucho que avanzamos a lo largo de la carrera con respecto a lo que son los lenguajes de programación, como funcionan y porque funcionan de esa manera. Y al ir agarrando cada vez más conceptos teóricos se nos hacía más fácil el entender el porqué las cosas eran de esa forma.