DESARROLLO

WEB FULL STACK

-INTERMEDIO

Autor de contenido

Andrés Fernando Pineda Guerra

EMPEZAR

Tabla de Contenido

Presentación

En el curso de desarrollador Full Stack como componente intermedio, podrán adquirir las habilidades y lenguajes necesarios para el desarrollo web, enfocándose en sus grandes pilares, como lo son Front End, Back End, Diseño y modelamiento de aplicaciones y documentación de código.
El curso trata temas emergentes tales como, la seguridad informática, desarrollo de aplicaciones móviles, gestión de base de datos, todo esto basado en la metodología Scrum. De la misma manera, se hace énfasis en el manejo de proyectos tanto en los módulos de desarrollo como los módulos de gestión de proyectos de TI.

Objetivos del curso (competencias)

Objetivo general

Formar a los participantes en el desarrollo web en todo el ciclo de vida del software, en donde adquieran los conocimientos básicos para implementar soluciones web.

Objetivo específico

- Conocer los conceptos y teoría básica del desarrollo web.

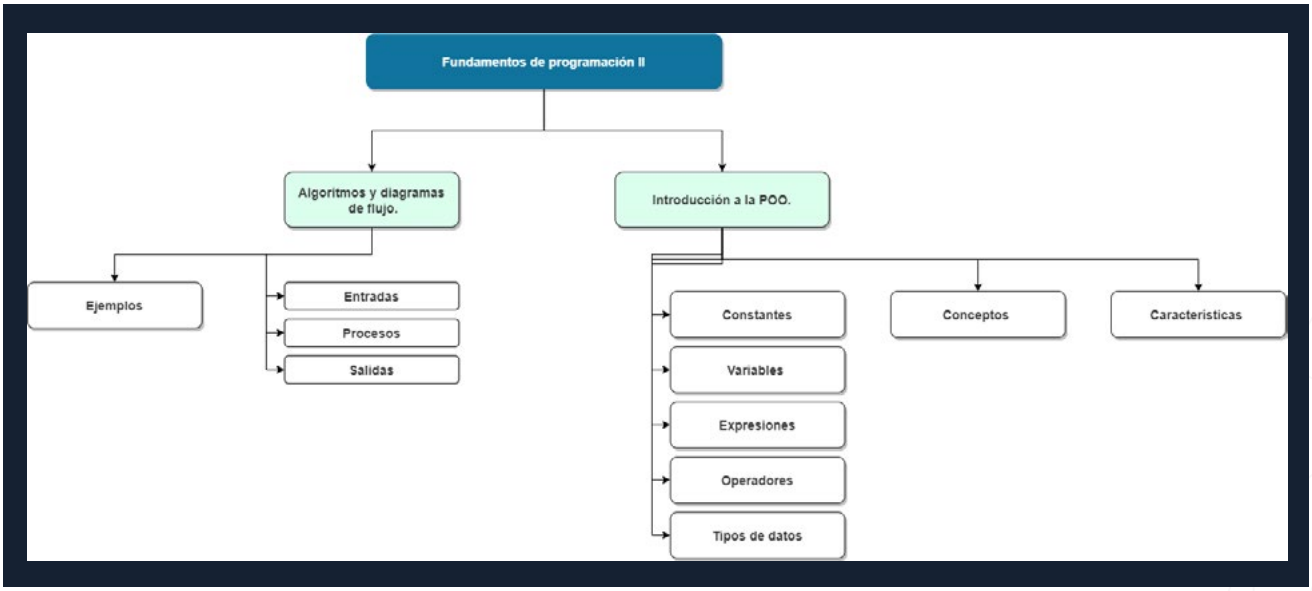
- Identificar y conocer los diferentes lenguajes de programación y herramientas para el desarrollo web.

- Aplicar las diferentes tecnologías web, tendencias y herramientas en el desarrollo de soluciones web enfocadas a proyectos.

- Diseñar, desarrollar e implementar soluciones web básicas en donde se integren los componentes de Front End, Back End, seguridad, redes y buenas prácticas utilizando metodologías ágiles.

- Identificar y conocer los conceptos básicos para el desarrollo móvil, así como aplicar su desarrollo en aplicaciones básicas.

Mapa de contenido de la unidad



Módulo 1

Fundamentos de programación II

Ideas clave

Algoritmos, entradas procesos y salidas.

Algoritmos de búsqueda, de ordenamiento, probabilísticos, programación dinámica y voraces.

Diagramas de flujo, sus principales componentes, método de ordenación burbuja.

Programación orientada a objetos y sus principios, encapsulación, abstracción, herencia, polimorfismo.

1.1. Algoritmos y diagramas de flujo complejos.

Algoritmos

Los algoritmos son conjuntos de instrucciones definidas, ordenadas y acotadas, para resolver un problema en específico, realizar algún tipo de cálculo y/o desarrollar alguna actividad o tarea (Jiménez et al., 2015).

Los algoritmos cuentan con tres partes fundamentales:

Input (entrada). Información que damos al algoritmo con el que va a trabajar para ofrecer la solución esperada.

Proceso. Conjunto de pasos para que, a partir de los datos de entrada, llegue a la solución de la situación.

Output (salida). Resultados, a partir de la transformación de los valores de entrada durante el proceso.

Diagramas de flujo

Los diagramas de flujo son indispensables en la fase de análisis y diseño del ciclo de vida del software, ya que estos permiten visualizar procesos representados los pasos, secuencias y decisiones de dichos procesos.

En pocas palabras, es la representación gráfica y secuencial de un flujo o trabajo con tareas y actividades para lograr un objetivo en común (Zanfrillo, s. f.).

Es un esquema para representar gráficamente un algoritmo. Se basan en la utilización de diversos símbolos para representar operaciones específicas, es decir, es la representación gráfica de las distintas operaciones que se tienen que realizar para resolver un problema, con indicación expresa el orden lógico en que deben realizarse.

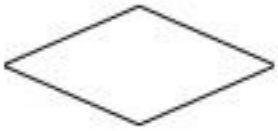
Se les llama diagramas de flujo porque los símbolos utilizados se conectan por medio de flechas para indicar la secuencia de operación. Para hacer comprensibles los diagramas a todas las personas, los símbolos se someten a una normalización; es decir, se hicieron símbolos casi universales, ya que, en un principio cada usuario podría tener sus propios símbolos para representar sus procesos en forma de Diagrama de flujo. Esto trajo como consecuencia que sólo aquel que conocía sus símbolos, los podía interpre-

tar. La simbología utilizada para la elaboración de diagramas de flujo es variable y debe ajustarse a un patrón definido previamente.

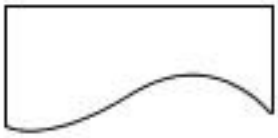
El diagrama de flujo representa la forma más tradicional y duradera para especificar los detalles algorítmicos de un proceso. Se utiliza principalmente en programación, economía y procesos industriales.

Inicio o finalización del diagráma

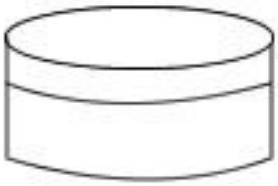
Etapa del proceso o realización de una actividad



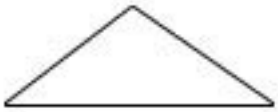
Etapa de análisis o toma de decisión



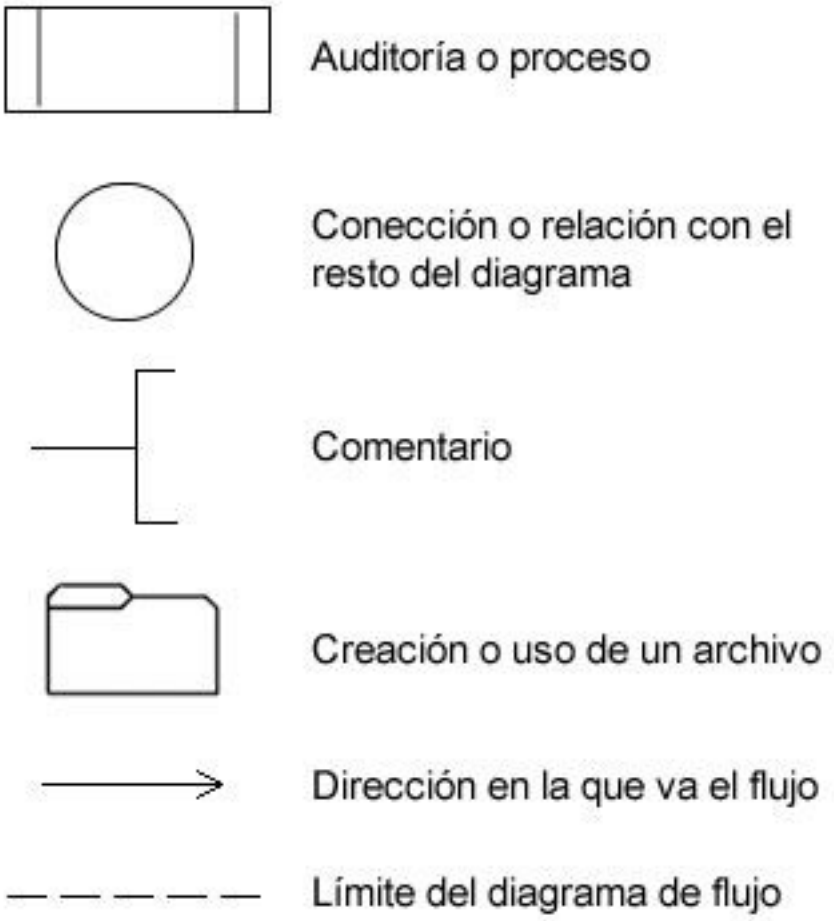
Realización de un documento



Creación de base de datos

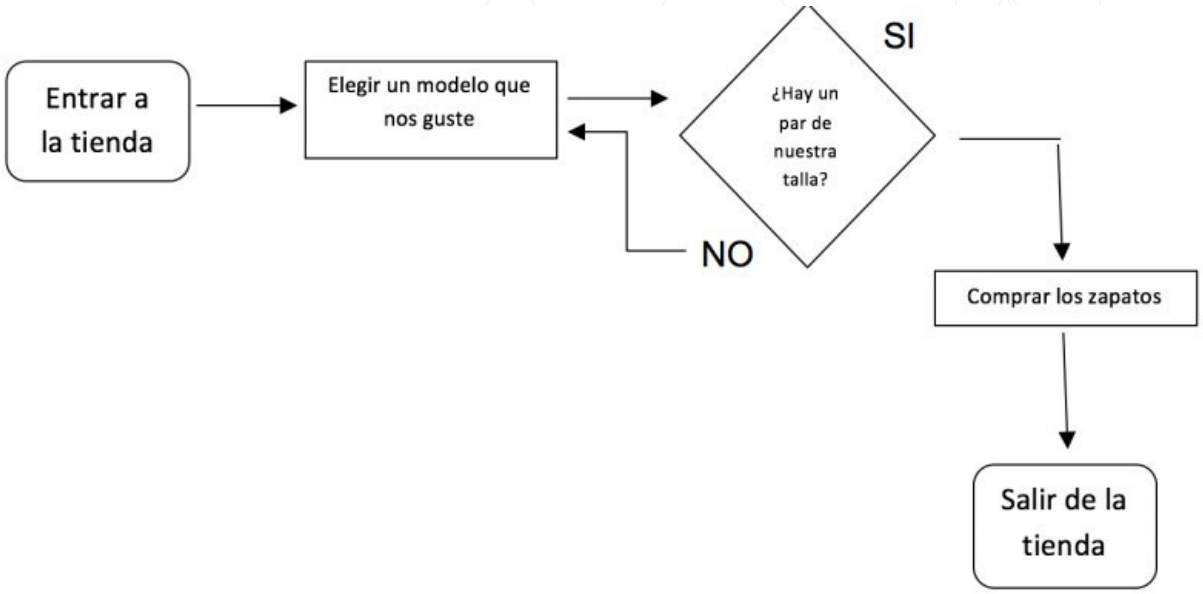


Actividad de control

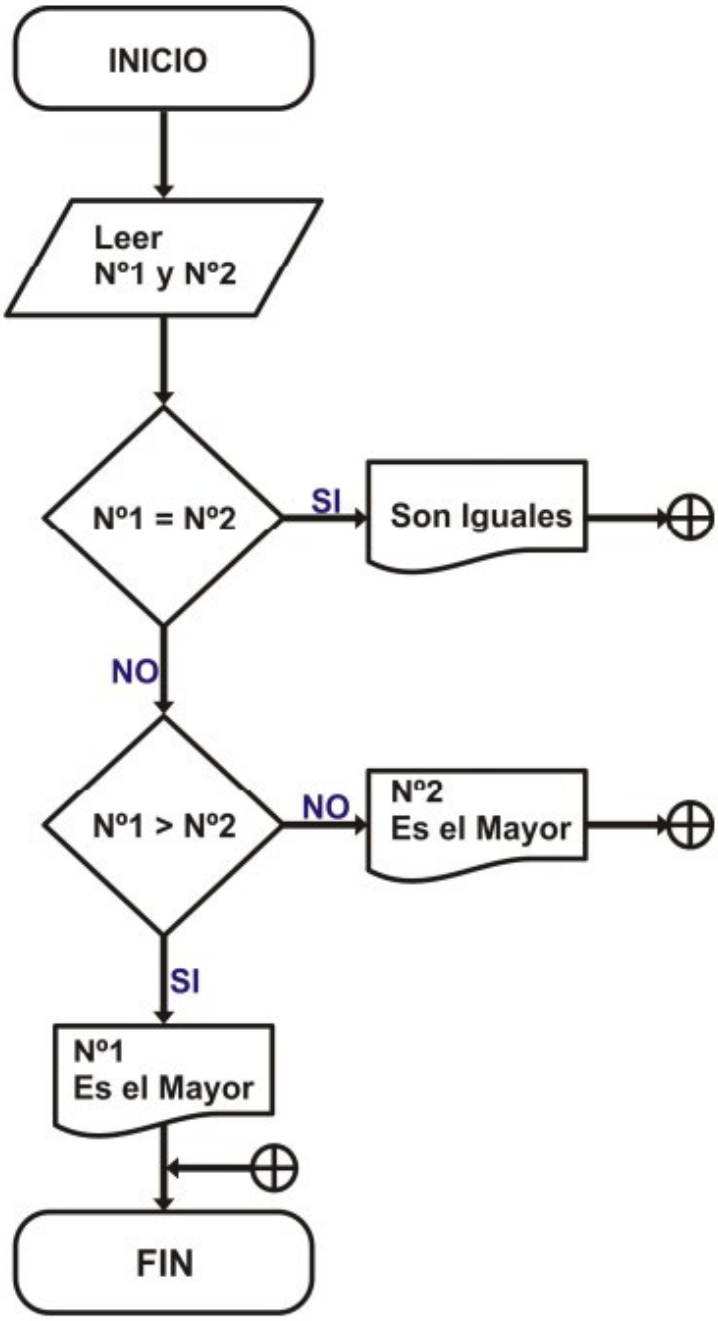


Ejemplo de diagramas de flujo.

Diagrama de flujo básico para la compra de unos zapatos:



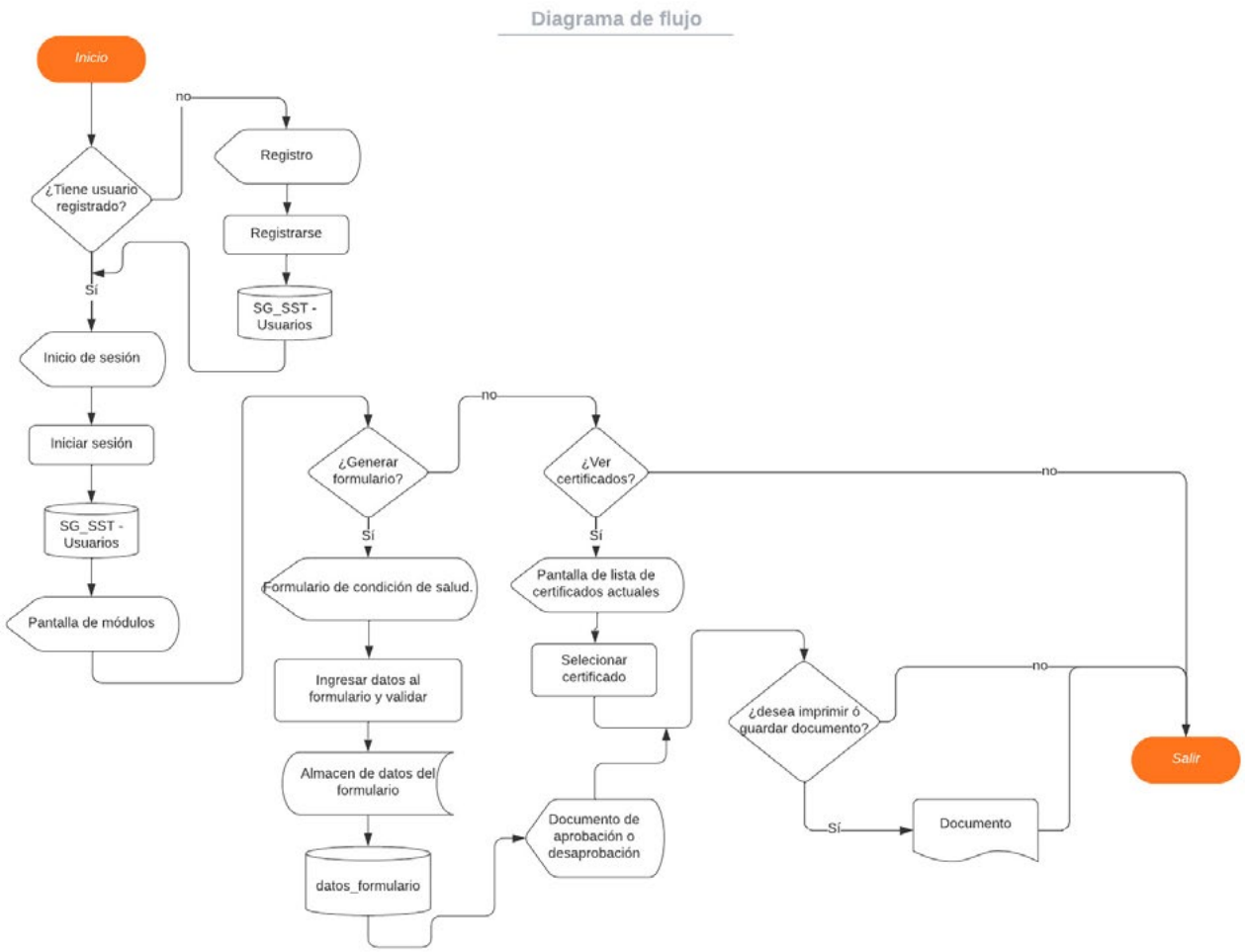
Ejemplo enfocado a los algoritmos de programación



MÉTODO DE ORDENACIÓN POR BURBUJA

El método de ordenación por burbuja es un algoritmo de ordenamiento, que funciona revisando cada elemento de la lista que va a ser ordenada con el siguiente, e intercambiándolos de posición si esta está en el orden erróneo. Es estrictamente necesario realizar este proceso varias veces de modo que no necesiten más intercambios, hasta que esté ordenada. En este método de ordenación se realiza el intercambio entre los más grandes y los más pequeños (los más grandes quedan arriba y los pequeños abajo). Estos intercambios suceden en dos ciclos un ciclo que es interno que realiza las comparaciones y se asegura de que en la primera revisión que los elementos más grandes suban a la posición más alta, y el otro ciclo que es un poco más externo se encarga de realizar las revisiones.

Análisis de diagramas de flujo complejos.



11.2. Énfasis a la POO.

La programación orientada a objetos - POO, es un modelo de lenguaje de programación el cual se organiza por objetos y se construye por medio de datos y funciones, mediante estos se pueden generar relaciones como herencia, cohesión, abstracción, encapsulamiento y polimorfismo. De esta manera se permite la flexibilidad y creación de objetos heredados y con la capacidad de ser transmitidos sin necesidad de ser modificados continuamente (MUÑOZ et al., 2007).

Conceptos.

Encapsulación

La encapsulación contiene toda la información importante de un objeto dentro del mismo y solo expone la información seleccionada al mundo exterior.

Esta propiedad permite asegurar que la información de un objeto esté oculta para el mundo exterior, agrupando en una Clase las características o atributos que cuentan con un acceso privado, y los comportamientos o métodos que presentan un acceso público.

La encapsulación de cada objeto es responsable de su propia información y de su propio estado. La única forma en la que este se puede modificar es mediante los propios métodos del objeto. Por lo tanto, los atributos internos de un objeto deberían ser inaccesibles desde fuera, pudiéndose modificar sólo llamando a las funciones correspondientes. Con esto conseguimos mantener el estado a salvo de usos indebidos o que puedan resultar inesperados.

Abstracción

La abstracción es cuando el usuario interactúa solo con los atributos y métodos seleccionados de un objeto, utilizando herramientas simplificadas de alto nivel para acceder a un objeto complejo.

En la programación orientada a objetos, los programas suelen ser muy grandes y los objetos se comunican mucho entre sí. El concepto de abstracción facilita el mantenimiento de un código de gran tamaño, donde a lo largo del tiempo pueden surgir diferentes cambios.

Así, la abstracción se basa en usar cosas simples para representar la complejidad. Los objetos y las clases representan código subyacente, ocultando los detalles complejos al usuario. Por consiguiente, supone una extensión de la encapsulación

Herencia

La herencia define relaciones jerárquicas entre clases, de forma que atributos y métodos comunes puedan ser reutilizados. Las clases principales extienden atributos y comportamientos a las clases secundarias. A través de la definición en una clase de los atributos y comportamientos básicos, se pueden crear clases secundarias, ampliando así la funcionalidad de la clase principal y agregando atributos y comportamientos adicionales.

Polimorfismo

El polimorfismo consiste en diseñar objetos para compartir comportamientos, lo que nos permite procesar objetos de diferentes maneras. Es la capacidad de presentar la misma interfaz para diferentes formas subyacentes o tipos de datos. Al utilizar la herencia, los objetos pueden anular los comportamientos principales compartidos, con comportamientos secundarios específicos. El polimorfismo permite que el mismo método ejecute diferentes comportamientos de dos formas: anulación de método y sobrecarga de método.

Constantes

Una constante es un identificador (nombre) para un valor simple. Como el nombre sugiere, este valor no puede variar durante la ejecución del script (a excepción de las constantes mágicas, que en realidad no son constantes). Por defecto, una constante distingue mayúsculas y minúsculas. Por convención, los identificadores de constantes siempre se declaran en mayúsculas.

El nombre de una constante sigue las mismas reglas que cualquier otra etiqueta de PHP. Un nombre de constante válido empieza por una letra o guión bajo, seguido por cualquier número de letras, números o guiones bajos. Usando una expresión regular, se representaría de la siguiente manera: [a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*

Ejemplos:

```
/*Declaración de constantes*/

const y = 89;
```

```
const x = 1.5;
const z = 56;
const i = 9;
const CONSTANTE = 'Hola Mundo';
```

Variables

En PHP las variables se representan con un signo de dólar seguido por el nombre de la variable. El nombre de la variable es sensible a minúsculas y mayúsculas. Los nombres de variables siguen las mismas reglas que otras etiquetas en PHP. Un nombre de variable válido tiene que empezar con una letra o un carácter de subrayado (underscore), seguido de cualquier número de letras, números y caracteres de subrayado. Como expresión regular se podría expresar como: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

Ejemplos:

```php
/*Declaración de variables*/

$y = 89;
$x = 1.5;
$z = 56;
$i = 9;
$CONSTANTE = 'Hola Mundo';
```

Expresiones

Las expresiones son los bloques de construcción más importantes de PHP. En PHP casi todo lo que se escribe es una expresión. La manera más simple y acertada de definir lo que es una expresión es «cualquier cosa que tiene un valor».

Las formas más básicas de expresiones son las constantes y las variables. Cuando se escribe "$a = 5", se está asignando '5' a $a. '5', obviamente, tiene el valor 5, o en otras palabras, '5' es una expresión con el valor de 5 (en este caso, '5' es una constante entera).

Después de esta asignación, se espera que el valor de $a sea 5 también, por lo que si se escribe $b = $a, se espera que esto se comporte tal como si se escribiera $b = 5. En otras palabras, $a es también una expresión con el valor 5. Si todo funciona bien, esto es exactamente lo que sucederá.
Un ejemplo de expresiones algo más complejo son las funciones. Por ejemplo, considere la siguiente función:

```php
/*EXTRESIONES*/

function foo ()
{
    return 5;
}
```

Asumiendo que está familiarizado con el concepto de función (si no lo está, échele una ojeada al capítulo sobre funciones), asumirá que escribir $c = foo() es esencialmente igual que escribir $c = 5. Y está en lo cierto. Las funciones son expresiones con el valor de sus valores devueltos. Ya que foo() devuelve 5, el valor de la expresión 'foo()' es 5. Normalmente las funciones no sólo devuelven un valor estático, sino que computan algo.

Por supuesto, los valores en PHP no tienen que ser enteros, y con frecuencia no lo son. PHP soporta cuatro tipos de valores escalares: valores enteros (integer), valores de coma (punto) flotante (float), valores de cadena (string) y valores booleanos (boolean) - (valores escalares son aquellos que no se pueden descomponer en piezas más pequeñas, a diferencia de las matrices, por ejemplo). PHP también soporta dos tipos compuestos (no escalares): matrices (arrays) y objetos. Cada uno de estos tipos de valores pueden ser asignados a variables o devueltos desde funciones.

PHP lleva las expresiones mucho más allá, de la misma manera que lo hacen otros lenguajes. PHP es un lenguaje orientado a expresiones, en el sentido de que casi todo es una expresión. Considere el ejemplo que ya hemos tratado, '$a = 5'. Es fácil de ver que aquí hay dos valores involucrados, el valor de la constante entera '5', y el valor de $a que ha sido actualizado a 5 también. Aunque la verdad es que existe aquí un valor adicional involucrado, que es el valor de la asignación misma. La asignación evalúa al valor asignado, en este caso 5. En la práctica, esto significa que '$a = 5', sin importar lo que haga, es una expresión con el valor 5. De este modo, escribir algo como '$b = ($a = 5)' es igual que escribir '$a = 5; $b = 5;' (el punto y coma marca el final de una sentencia). Ya que las asignaciones se analizan de derecha a izquierda, también se puede escribir '$b = $a =5'.

Otro buen ejemplo de orientación a expresiones es el pre- y post-incremento y decremento. Los usuarios de PHP y de otros muchos lenguajes pueden estar familiarizados con la notación variable++ y variable--. Éstos son los operadores de incremento y decremento. En PHP, al igual que en C, hay dos tipos de incrementos - pre-incremento y post-incremento. Ambos esencialmente incrementan la variable, y el efecto sobre la variable es idéntico. La diferencia está con el valor de la expresión de incremento. Pre-incremento, que se escribe '++$variable', evalúa al valor incrementado (PHP incrementa la variable antes de leer su valor, de ahí el nombre de 'pre-incremento'). Post-incremento, que se escribe '$variable++' evalúa el valor original de $variable, antes de que sea incrementado (PHP incrementa la variable después de leer su valor, de ahí el nombre de 'post-incremento') .

Un tipo de expresiones muy comunes son las expresiones de comparación. Estas expresiones evalúan si algo es false (falso) o true (verdadero). PHP soporta > (mayor que), >= (mayor o igual que), == (igual), != (distinto), < (menor que) y <= (menor o igual que). El lenguaje también soporta un conjunto de operadores de equivalencia estricta: === (igual y del mismo tipo) y !== (diferente o de distinto tipo). Estas expresiones se usan mayormente dentro de ejecuciones condicionales, tales como la sentencia if.

El último ejemplo de expresiones que trataremos aquí es una combinación de expresiones operador-asignación. Ya sabe que si quiere incrementar $a en 1, puede simplemente escribir '$a++' o '++$a'. Pero si lo que quiere es añadir más de uno, por ejemplo 3, podría escribir '$a++' varias veces, pero esto, obviamente, no es una manera muy eficiente o cómoda. Una práctica mucho más común es escribir '$a = $a + 3'. '$a + 3' evalúa al valor de $a más 3, y se vuelve a asignar a $a, lo que resulta en incrementar $a en 3. En PHP, como en otros lenguajes como C, se puede escribir esto de una manera más abreviada, lo que con el tiempo se podría convertir en una forma más clara y rápida de entenderlo. Añadir 3 al valor actual de $a puede ser escrito '$a += 3'. Esto significa exactamente "toma el valor de $a, añádele 3 y asígnale de nuevo a $a". Además de ser más corto y claro, también resulta en una ejecución más rápida. El valor de '$a += 3', al igual que el valor de una asignación normal, es el valor asignado. Observe que NO es 3, sino el valor combinado de $a más 3 (éste es el valor que es asignado a $a). Se puede usar cualquier operador compuesto de dos partes en este modo de operador-asignación, por ejemplo '$a -= 5' (restar 5 del valor de $a), '$b *= 7' (multiplicar el valor de $b por 7), etc.

Operadores

Un operador es algo que toma uno más valores (o expresiones, en la jerga de programación) y produce otro valor (de modo que la construcción en sí misma se convierte en una expresión).

Los operadores se pueden agrupar de acuerdo con el número de valores que toman. Los operadores unarios toman sólo un valor, por ejemplo ! (el operador lógico de negación) o ++ (el operador de incremento). Los operadores binarios toman dos valores, como los familiares operadores aritméticos + (suma) y - (resta), y la mayoría de los operadores de PHP entran en esta categoría. Finalmente, hay sólo un operador ternario, ? :, el cual toma tres valores; usualmente a este se le refiere simplemente como "el operador ternario" (aunque podría tal vez llamarse más correctamente como el operador condicional).

Una lista completa de operadores de PHP sigue en la sección Precedencia de Operadores. La sección también explica la precedencia y asociatividad de los operadores, las cuales gobiernan exactamente cómo son evaluadas expresiones que contienen varios diferentes operadores (MUÑOZ et al., 2007).

El operador se utiliza para realizar la operación.

Los operadores se dividen principalmente en tres grupos.

- Operadores Unarios que toman un valor
- Operadores binarios que toman dos valores
- operadores ternarios que toman tres valores

Los operadores se dividen principalmente en tres grupos que son en total diecisiete tipos.

a. Operador aritmético

Operador	Símbolo
Suma	+
Resta	-
Multiplicación	*
División	/
Módulo	%
Exponenciación	**

b. Operador de asignación

Operador	Símbolo
Igual a	=

c. Operador de matriz

Operador	Símbolo
Unión	+
Igualdad	==
Identidad	===
Desigualdad	!=
Desigualdad	<>
Sin identidad	!==

d. Operador bit a bit

Operador	Símbolo
y	&
xor	^
no	|
desplazar a la izquierda	<<
desplazar a la derecha	>>

e. Operador de comparación

Operador	Símbolo
igual	==
idéntico	===
no igual	!=
no idéntico	!==
no igual	<>
menor que	<
menor que o igual	<=
mayor que	>
mayor o igual	>=

Activar Windows
Ve a Configuración para

f. Operador de incremento/decremento

Operador	Símbolo
PreIncremento	++$a
PostIncremento	$a++
PreDecremento	--$a
Posdecremento	$a--

g. Operador lógico

Operador	Símbolo
Y	&&
O	||
No	!
y	y
xor	xor
o	o

h. Operador de cadena

Operador	Símbolo
Operador de concatenación	.
Operador de asignación de concatenación	.=

i. Tipos de datos

Los valores posibles para la cadena devuelta son:

Dato	Descripción
Boolean	Un valor que puede ser verdadero (true) o falso (false)
int	Un valor numérico entero con signo
Float	Un valor numérico de punto flotante con signo
String	Cadena de caracteres - texto
null	Representa la ausencia de valor para una variable.
resource	Recursos no representados de una forma nativa por PHP, por ejemplo las conexiones de una base de datos y manejadores de archivos.

Otros materiales para profundizar

Recursos de video

CompilaTec (Director). (2016, octubre 2). Algoritmo y Diagrama de Flujo. https://www.youtube.com/watch?v=SZTXmCbfjP0

DiscoDurodeRoer (Director). (2020, marzo 30). Ejercicios PHP - POO #1—Personas. https://www.youtube.com/watch?v=vzna0A3W5b4

Referencias bibliográficas de la unidad

Arias, Á. (2016). Fundamentos de Programación y Bases de Datos: 2a Edición. IT Campus Academy.

Insuasti, J. (2016). Problemas de enseñanza y aprendizaje de los fundamentos de programación. Revista Educación y Desarrollo Social, 10(2), Art. 2. https://doi.org/10.18359/reds.1966


ALCALDÍA MAYOR
DE BOGOTÁ D.C.
SECRETARÍA DE EDUCACIÓN


ATENEA
AGENCIA DISTRITAL PARA LA EDUCACIÓN
SUPERIOR LA CIENCIA Y LA TECNOLOGÍA


BOGOTÁ


UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS
Acreditación Institucional de Alta Calidad