

Comparación práctica de los lenguajes PHP y Ruby

Característica	Ruby	PHP
Comentarios	<pre># Una línea estilo shell =begin Esto es un ejemplo de comentarios multilínea =end</pre>	<pre># Una línea estilo shell // Una línea estilo C /* Esto es un ejemplo de comentarios multilínea al estilo C++ */</pre>
Cadenas	<p>Comillas simples: cadenas de texto simples, significan lo que representan.</p> <pre>'Hola Mundo'</pre> <p>Comillas dobles: permiten reemplazar caracteres escapados y variables incluidas.</p> <pre>"Hola Mundo #{var}\n"</pre>	<p>Igual comportamiento.</p> <p>Comillas simples:</p> <pre>'Hola Mundo'</pre> <p>Comillas dobles:</p> <pre>"Hola Mundo {\$var}\n"</pre>
Concatenar y repetir cadenas	<p>Concatenar:</p> <pre>'Hola' + 'Mundo'</pre> <p>Repetir:</p> <pre>'Hola ' * 2</pre>	<p>Concatenar:</p> <pre>'Hola' . 'Mundo'</pre> <p>Repetir:</p> <p>N/A</p>
Extracción de caracteres de una cadena	<pre>cadena[indice] cadena[inicio, longitud] cadena[elemeni ... elemfin]</pre>	<pre>\$cadena[\$indice] substr(\$cadena, \$inicio, \$longitud)</pre>
Arreglos	<pre>arreglo = [1, 2, '3']</pre>	<pre>arreglo = array(1, 2, '3');</pre>
Unir y repetir arreglos	<p>Unir:</p> <pre>a2 = arreglo + ['a', 'b', 'c']</pre> <p>Repetir:</p> <pre>a3 = a2 * 5</pre>	<p>Unir:</p> <pre>\$a2 = array_merge(\$arreglo, array('a', 'b', 'c'));</pre> <p>Repetir:</p> <p>N/A</p>
Extracción de elementos del arreglo	<ul style="list-style-type: none"> • arreglo[indice] • arreglo[inicio, longitud] • arreglo[elemeni ... elemfin] 	<pre>\$arreglo[\$indice] array_slice(\$arreglo, \$inicio, \$longitud);</pre>
Convertir arreglo a cadena	<pre>arreglo.join(separador)</pre>	<pre>implode(\$separador, \$arreglo);</pre>
Convertir cadena a arreglo	<pre>cadena.split(separador)</pre>	<pre>explode(\$separador, \$cadena);</pre>
Conversión de tipos	<pre>cadena = entero.to_s entero = cadena.to_i</pre>	<pre>\$cadena = (string)\$entero; \$entero = (int)\$cadena;</pre>

Comparación práctica de los lenguajes PHP y Ruby

Hashes	<pre>hash = {1 => 2, '2' => '4'}</pre>	<pre>\$hash = array(1 => 2, '2' => '4');</pre>
Acceso, modificación, agregación y eliminación de elementos del hash	<pre>hash[llave] = valor hash[nueva_llave] = valor hash[llave] = nil</pre>	<pre>\$hash[\$llave] = \$valor; \$hash[\$nueva_llave] = \$valor; unset(\$hash[\$llave]);</pre>
Lectura de entrada estándar	<pre>STDIN.gets</pre>	<pre>fgets(STDIN)</pre>
condicionales	<pre>if condición # ... else # ... end unless es un if negado.</pre>	<pre>if (condición) { # ... } else { # ... }</pre>
switch/case	<pre>case variable when opcion1 # ... when opcion2, opcion3 # ... end</pre>	<pre>switch(\$variable) { case opcion1: # ... break; case opcion2, opcion3: # ... break; default: # ... break; }</pre>
while	<pre>while condicion # ... end until es un while negado.</pre>	<pre>while (condicion) { # ... }</pre>
foreach	<pre>for elemento in colección # ... end colección.each{ elemento # ... }</pre>	<pre>foreach (\$coleccion as \$elemento) { # ... }</pre>
for	<pre>for i in (4 .. 6) # ... end</pre>	<pre>for (\$i=4; \$i<=6; \$i++) { # ... }</pre>
Manipulación de las iteraciones	<pre>break: rompe la iteración y termina completamente el ciclo. next: termina la iteración actual y continúa en la siguiente.</pre>	<pre>break: rompe la iteración y termina completamente el ciclo. continue: termina la iteración actual y continúa en la siguiente.</pre>

Comparación práctica de los lenguajes PHP y Ruby

	redo: termina la iteración actual y la reinicia nuevamente.	
Declaración de una clase	<pre>class Nombre # ... end</pre>	<pre>class Nombre { # ... }</pre>
Declaración de un método	<pre>class Nombre def miMetodo(parametros) # ... end end</pre>	<pre>class Nombre { function miMetodo(\$parametros) { # ... } }</pre>
Declaración de un constructor	<pre>class Nombre def initialize(parametros) # ... end end</pre>	<pre>class Nombre { function __construct (\$parametros) { # ... } }</pre>
Referencia a la misma clase	self	\$this
Creación de una nueva instancia	instancia = Clase.new (parametros)	\$instancia = new Clase (\$parametros);
Llamado de métodos	<pre>instancia.metodo(parametros)</pre> <p>Los paréntesis son opcionales.</p>	\$instancia -> metodo(parametros);
Herencia	<pre>class Hijo < Padre # ... end</pre>	<pre>class Hijo extends Padre { # ... }</pre>
Referencia a la clase padre	super	parent
Encapsulamiento	<pre>public protected private</pre>	<pre>public protected private</pre>
Módulos/interfaces	<pre>Module miModulo # ... end</pre> <p>Acceso a los recursos del módulo:</p> <pre>miModulo.metodo(parametros) miModulo::miConstante</pre> <p>Incluir el módulo en el ambiente actual:</p> <pre>include miModulo</pre>	<pre>interface miInterfaz { # ... } class miClase implements miInterfaz { # ... }</pre>
Constantes	Nombre = valor	define('Nombre', 'valor');

Comparación práctica de los lenguajes PHP y Ruby

	El nombre de las constantes empieza por una letra mayúscula.	
Variables globales	<code>\$nombre = valor</code>	<code>\$GLOBALS['nombre'] = valor;</code>
Atributos de instancia	<pre>class miClase def sumar(valor) @resultado = resultado + valor end end</pre> <p>Inicialmente tiene el valor de <code>nil</code>.</p>	<pre>class miClase { var \$resultado; function sumar(\$valor) { \$this -> resultado = \$this -> resultado + \$valor; } }</pre> <p>Inicialmente tiene el valor de <code>null</code>.</p>
Variables locales	<pre>nombre = valor</pre> <p>El nombre de la variable inicia por minúsculas o <i>underscore</i> ('_').</p> <p>La primera asignación actúa como declaración.</p>	<pre>\$nombre = \$valor;</pre> <p>Depende del ámbito donde se utilice.</p>
Verificar la existencia de una variable	<pre>defined? variable</pre> <p>Retorna una descripción de la variable si esta existe, <code>nil</code> de lo contrario.</p>	<pre>isset(\$variable)</pre> <p>Retorna <code>true</code> si existe la variable, <code>false</code> de lo contrario.</p>
Manejo de excepciones	<pre>begin # Código que puede lanzar # excepciones rescue # Código de manejo solo en # caso de excepciones ensure # Código que se ejecuta # hayan sucedido excepciones # o no end</pre> <p><code>retry</code> intenta ejecutar el bloque <code>begin</code> nuevamente.</p>	<pre>try { # Código que puede lanzar # excepciones } catch (ClaseExcepcion \$e) { # Código de manejo solo en # caso de excepciones }</pre>
Lanzar excepciones	<pre>raise "Mensaje de la excepción"</pre> <p>El mensaje de la excepción se puede obtener posteriormente con la variable <code>\$!</code>.</p>	<pre>throw new Exception('Mensaje de la excepción');</pre> <p>El mensaje de la excepción se puede obtener posteriormente a través del método <code>getMessage()</code> del objeto <code>Exception</code>.</p>
Representación de cadena de un objeto	<pre>class miClase def inspect return "esta es la representación" end end</pre>	<pre>class miClase { function __toString() { return "esta es la representación"; } }</pre>

Comparación práctica de los lenguajes PHP y Ruby

```
obj = miClase.new
print obj
```

```
}
}
$obj = new miClase();
print ($obj);
```

Mas recursos:

1. Guía del usuario de Ruby. <http://es.tldp.org/Manuales-LuCAS/doc-guia-usuario-ruby/>
2. Ruby quick reference: <http://www.digilife.be/quickreferences/QRC/Ruby%20Language%20QuickRef.pdf>
3. Ruby desde otros lenguajes: <http://www.ruby-lang.org/es/documentation/ruby-from-other-languages/>