

3.6: Data Manipulation

Comparison Operators

PHP supports many of the mathematical comparisons common to programming languages, such as equivalence and relative value comparison. The symbols used however may be different than what you are used to. In the chart below we will look at how to represent each comparison test, and under what condition we can expect the test to come back as true.

Table 3.6.1 Comparison Operators

Example	Name	Result
<code>\$a == \$b</code>	Equal	TRUE if <i>\$a</i> is equal to <i>\$b</i> .
<code>\$a === \$b</code>	Identical	TRUE if <i>\$a</i> is equal to <i>\$b</i> , and they are of the same type. (introduced in PHP 4)
<code>\$a != \$b</code>	Not equal	TRUE if <i>\$a</i> is not equal to <i>\$b</i> .
<code>\$a <> \$b</code>	Not equal	TRUE if <i>\$a</i> is not equal to <i>\$b</i> .
<code>\$a !== \$b</code>	Not identical	TRUE if <i>\$a</i> is not equal to <i>\$b</i> , or they are not of the same type. (introduced in PHP 4)
<code>\$a < \$b</code>	Less than	TRUE if <i>\$a</i> is strictly less than <i>\$b</i> .
<code>\$a > \$b</code>	Greater than	TRUE if <i>\$a</i> is strictly greater than <i>\$b</i> .
<code>\$a <= \$b</code>	Less than or equal to	TRUE if <i>\$a</i> is less than or equal to <i>\$b</i> .
<code>\$a >= \$b</code>	Greater than or equal to	TRUE if <i>\$a</i> is greater than or equal to <i>\$b</i> .

These tests will come in handy as we move into logic structures. The results of these comparisons can help us determine a course of action like what is displayed to the user, how we modify or create data, or respond to user input. Pay close attention to the difference between the equal (==) and identical (===) tests. Equal means the comparison of each side is considered the same. For example, 1 and True are considered equal, because PHP will treat a 1 as both an integer and a binary representation of true. If we want to ensure elements are the same in both value and type, we would use strictly equal. This test returns a false in our example, as 1 and true are not both integers of the value 1. Also, do not forget at least the second =, as just one will be treated as assignment, not a logical test!

Additional notes

Take Note! While using the words “and” and “or” in your logic statements, PHP will not give you an error, as they are in the order of precedence below. Take note that they are below the = sign—this will affect your logic equations. The vast majority of the time you will want to use “&&” and “||”, as they will be evaluated before assignment.

Order of Operations

PHP follows the traditional order of operations used in mathematics, as found below. An associativity of “left” means the parser will read left to right across the equation. “Right” means it will move right to left (i.e.: assign the equation to the element on the left of the = sign). Precedence takes place from the top down, meaning the operators higher in this list will be evaluated before those beneath them. Just as in mathematics, parenthesis will interrupt this list by treating the contents of each set of parenthesis (from the inner most out) as a statement by itself. The portion of the table in yellow highlights the operators most used for development below application level.

Table 3.6.2 Operator Precedence

Associativity	Operators
non-associative	clone new
left	[

non-associative	++ —
right	~—(int) (float) (string) (array) (object) (bool) @
non-associative	instance of
right	!
left	* / %
left	+—.
left	<< >>
non-associative	< <= > >= <>
non-associative	== != === !==
left	&
left	^
left	
left	&&
left	
left	? :
right	= += -= *= /= .= %= &= = ^= <<= >>= ==>
left	and
left	xor
left	or
left	,

Let us look at a few examples to demonstrate precedence in PHP:

1. <code>echo 3 * 4 + 3 + 2;</code>	17 Multiplication takes precedence and all are evaluated left to right
1. <code>echo 3 * (4 + 3 + 2);</code>	27 Parenthesis take precedence so addition is evaluated before multiplication

Given: `$this = true; $that=false`

1. <code>\$result = \$this && \$that</code>	<ul style="list-style-type: none"> • <code>\$result = false</code> true and false is false
1. <code>\$result = \$this and \$that</code>	<ul style="list-style-type: none"> • <code>\$result = true</code> <code>\$this (true)</code> is assigned before <code>\$this</code> and <code>\$that</code> is evaluated

Manipulating Data Streams

Data streams are long strings of characters specially formatted to convey information between systems. They typically focus on the ability to quickly convey all the information in as readable a format as possible, resulting in a compressed syntax to identify the information and its meaning. Two of the most popular methods of streaming data today are JSON and XML.

Data streams do not have to be raw, or complete, records of an entire system. They are frequently used to transmit responses between the back-end system (server or database) and the system that generates content the viewer sees (browser and/or scripting language).

JSON

An acronym for JavaScript Object Notation, JSON delimits objects with nested brackets and quoted values, denoting key and value pairs with colons. This is a very short, concise method of delivering data, but the recipient will need to get the meaning of the information elsewhere like documentation, and the string is not easily human readable. It is useful when the volume of data is high, and speed is important.

If we asked our system to give us the family members from Family Guy, we might get the following:

```
1. {"Griffins":{"Peter":"Father", "Lois":"Mother", "Stewie":"Son", "Chris":"Son", "Meg":"Daughter", "Brian":"Dog"}} }
```

If we asked for the Griffins *and* Quagmire, we might get:

```
1. {"Griffins":  
2. {"Peter":"Father", "Lois":"Mother", "Stewie":"Son", "Chris":"Son", "Meg":"Daughter", "Brian":"Dog"},  
3. {"Quagmire":"Neighbor"}  
4. }
```

XML

An abbreviation of eXtensible Markup Language, XML wraps pieces of information in tags, similar to HTML, but the names of the tags are user-defined. Grouping of information is done by nesting the tags within each other. Using our Family Guy example above, our XML response would be as follows:

```
1. <Response>  
2. <Griffin>  
3. <Peter>father</Peter>  
4. <Lois>mother</Lois>  
5. <Stewie>son</Stewie>  
6. <Chris>son</Chris>  
7. <Meg>daughter</Meg>  
8. <Brian>dog</Brian>  
9. </Griffin>  
10. <Quagmire>  
11. <Glen>neighbor</Glen>  
12. </Quagmire>  
13. </Response>
```

Useful Feature

You can test validate JSON and XML strings that you create or receive by copying and pasting them into validation sites like jsonlint.com and xmlvalidation.com. They can help you identify problem areas in your strings to make sure your data is stored correctly.

Take note that I specify that this is how your code *might* look in these examples. The actual output's format would vary based on how the developers decides to create the response string, and also based on any options available to the user as to how they want the information organized. For example, we might want character ages instead of relationships like father or daughter, or the developer might group the results by gender and put first and last names as the value pairs.

It is important to note that when you actually interact with data streams they will not look as they do above, but will be long strings without the spacing and line breaks, as this reduces the size of the string. The formatting you see above is often referred to as the "pretty print" format, which adds extra spacing and formatting to make it more human readable.

We can create both XML and JSON in PHP. You can do this by creating the exact string necessary to format it, or we can use functions in PHP to help us along. The SimpleXML package allows us to create, navigate, and edit XML content, while the `json_encode` and `json_decode` functions allow us an easy means to convert JSON to and from arrays.

For brevity, we will consider examples of receiving data in these two formats. While converting JSON into, an out of, arrays is easily done with `json_encode()` and `json_decode()`, creating data by hand in these formats would necessitate a much deeper look at both XML and JSON. Your journey there can begin with the Learn More section. I would recommend you explore at least one

format in depth, as you will come into contact with these formats when you interact with APIs. Current trending has JSON getting more attention in new development, but there are plenty of already built XML systems in place, and plenty more that offer both.

An easy way to interact with XML or JSON data in PHP is to convert it into arrays that we can more easily traverse. When we are working with XML we can use the SimpleXML package integrated in PHP to read our string or file using `$data = simplexml_load_string($ourXML);` or `$data = simplexml_load_file("ourXmlFile.xml");`. We can open JSON files to string or receive them from another source, and decode them using `$data = json_decode($ourJson)`. Just like we did with arrays we created earlier, we can see our data by using `print_r($data);`.

```
• $ourJson = '{"Griffins":{"Peter":"Father", "Lois":"Mother", "Stewie":"Son", "Chris":"Son", "Meg":"Daughter", "Brian":"Dog"}, }';
• $familyGuy = json_decode($ourJson,1);
• print_r($familyGuy);

Array ( [Griffins] => Array ( [Peter] => Father [Lois] => Mother [Stewie] => Son [Chris] => Son [Meg] => Daughter [Brian] => Dog )
)
```

Be sure to place the 1 as our second option in our `json_decode()` call, as it instructs the function to return the data as an array instead of objects. The same transfer to array for XML becomes a little more complicated, as PHP does not natively support this type of conversion, so we need to do more to get our full list displayed as arrays:

```
1. $ourXML= '<Response>
2. <Griffin>
3. <Peter >Father</Peter>
4. <Lois>Mother</Lois>
5. <Stewie>Son</Stewie>
6. <Chris>Son</Chris>
7. <Meg>Daughter</Meg>
8. <Brian>Dog</Brian>
9. </Griffin>
10. <Quagmire>
11. <Glen>Neighbor</Glen>
12. </Quagmire>
13. </Response>';
14. $familyGuy = simplexml_load_string($ourXML);
15. $familyGuy = (array) $familyGuy;
16. foreach ($familyGuy as &$group){ $group = (array) $group;}
17. print_r($familyGuy);

Array ( [Griffin] => Array ( [Peter] => Father [Lois] => Mother [Stewie] => Son [Chris] => Son [Meg] => Daughter [Brian] => Dog )
[Quagmire] => Array ( [Glen] => Neighbor ) )
```

While we were able to make the outermost layer of the data an array just by re-declaring its type, the type casting conversion in PHP is not recursive. However, `simplexml_load_string` turns our XML into objects not arrays, so by looping through our array again and recasting each element to an array, we can correct the data in the second layer. This process would need to be repeated for each nested layer of data.

Learn more

Keywords, search terms: json, xml, data formatting, data structures

Essential XML Quick Reference: <http://bookos.org/book/491155/a86a21>

Json.org: <http://www.json.org/>

Data Structures Succinctly (Pt 1): <http://www.syncfusion.com/resources/techportal/ebooks/datastructurespart1>

3.6: Data Manipulation is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.

- [3.6: Data Manipulation](#) by [Michael Mendez](#) has no license indicated.