PHP Notes Chapter 7 Data Formats and Types

XML SOAP REST web services JSON Date and time PHP SPL data structures

XML

XML (eXtensible Markup Language) is a data format for structured document interchange on the Web.
It is a standard defined by The World Wide Web consortium (W3C).

The Basics of XML

Term	Description
SGML	Standardized General Markup Language. XML is a subset of this.
Document Type Declaration	The DTD defines the legal building blocks of an XML document structure with a list of legal elements and attributes.
Entity	An entity can declare names and values that are not permitted in the rest of the XML document. For example, HTML declares < as an entity to represent the less than symbol <. These declarations can also be used as shortcuts and to maintain consistency of spelling and value throughout a document.
Tag	A markup structure that begins with < and ends with >. Start-tag, ex: <book> End-tag, ex: </book> Self-closing tag, ex: <line-break />
Element	Elements are the basic building blocks of an XML document. Begins with a start-tag and ends with an end-tag. Can also consist of a self-closing tag. Elements can be nested and contain elements, or they can contain a value. Elements may have attributes.
Attribute	A markup construct consisting of a name/value pair that exists within a start tag or self-closing tag. <book title="Oliver Twist">
Well-formed	A well-formed document in XML is a document that adheres to the syntax rules specified by the XML 1.0 specification in that it must satisfy both physical and logical structures. 1
Valid	An XML document validated against a DTD is both "Well Formed" and "Valid".

Note: PHP does not require XML documents to be valid but it does require them to be well formed to parse them with standard libraries.

XML Tree Structure

XML documents are formed as element trees.

Example 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

A **prolog** defines the XML version and the character encoding:

```
<?xml version="1.0" encoding="UTF-8"?>
```

The next line is the **root** element of the document:

```
<bookstore>
```

The next line starts a <book> **element**:

```
<book category="cooking">
```

The <book> elements have 4 **child elements**: <title>, <author>, <year>, <price>.

The next line ends the book element :

```
</book>
```

XML Processing Instructions (prolog)

Processing instructions allow documents to contain instructions for applications.
They are enclosed in <? and ?> marks and look like this, for example:

```
<?PITarget PIContent?>
```

One use-case could be to inform an application that an element is to be a particular data type, as in this example:

```
<?var type="string" ?>
```

The most common usage is to include an XSLT or CSS stylesheet, like so:

```
<?xml-stylesheet type="text/xsl" href="style.xsl"?>
```

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

XML Syntax Rules

XML documents must contain one root element that is the parent of all other elements.

The XML Prolog:

- The XML prolog is optional. If it exists, it must come first in the document.
- XML documents can contain international characters, like Norwegian øæå or French êèé.
- To avoid errors, you should specify the encoding used, or save your XML files as UTF-8.
- UTF-8 is the default character encoding for XML documents.

All XML Elements Must Have a Closing Tag

The XML prolog does not have a closing tag! The prolog is not a part of the XML document.

XML Elements Must be Properly Nested:

```
Improper:   <b><i>This text is bold and italic</b></i>

Proper:     <b><i>This text is bold and italic</i></b>
```

XML Attribute Values Must Always be Quoted:

```
<note date="12/11/2007">
  <to>Tove</to>
  <from>Jani</from>
</note>
```

Entity References. Some characters have a special meaning in XML.

If you place a character like "<" inside an XML element, it will generate an error because the parser interprets.

```
<message>salary < 1000</message>
```

To avoid this error, replace the "<" character with an entity reference:

```
<message>salary &lt; 1000</message>
```

There are 5 pre-defined entity references in XML:

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	quotation mark

Comments in XML The syntax for writing comments in XML is similar to that of HTML:

```
<!-- This is a comment -->
```

Two dashes in the middle of a comment are not allowed:

```
<!-- This is an invalid -- comment -->
```

White-space is Preserved in XML

XML does not truncate multiple white-spaces as in HTML.

XML documents that conform to the syntax rules above are said to be "Well Formed" XML documents.

https://www.w3schools.com/xml/xml_syntax.asp

XML Extentions in PHP

extension=dom.so

extension=simplexml.so

extension=xml.so

extension=xmlreader.so

extension=xmliter.so

extension=xsl.so https://www.php.net/manual/en/book.xsl.php

XML Transformations with PHP XSL `extension=xsl.so`

XSL stands for EXtensible Stylesheet Language.

XSL is a language for expressing stylesheets for XML documents. It is like CSS in that it describes how to display an XML document.

XSL consists of four parts:

XSLT	**a language for transforming XML documents**
XPath	**a language for navigating in XML documents**
XSL-FO	a language for formatting XML documents (discontinued in 2013)
XQuery	a language for querying XML documents

What is XSLT?

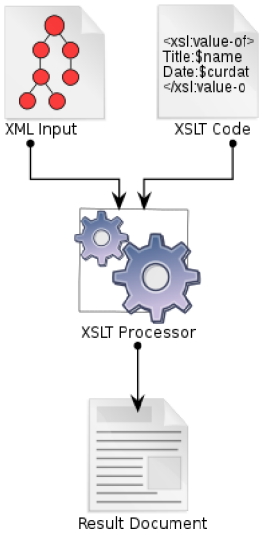
XSLT	**stands for XSL Transformations**
XSLT	**is the most important part of XSL**
XSLT	**transforms an XML document into another XML document**
XSLT	uses XPath to navigate in XML documents
XSLT	is a W3C Recommendation

The PHP XSL extension allows PHP to apply XSLT transformations. `extension=xsl.so`

An XSLT processor takes an input XML file, some XSLT code, and produces a new document

In PHP we can use the XSLTProcessor class.

An XSLTProcessor applies an XSLT stylesheet transformation to an XML document to produce a new XML document as output.




```
class XSLTProcessor {

// two methods were using
...
public importStylesheet(object $stylesheet): bool

...
public transformToXml(object $document): string|false|null
}
```

importStylesheet

This method imports the stylesheet into the XSLTProcessor for transformations.

TransformToXml

This method Transforms the source node (DOMDocument or SimpleXMLElement object) to a string and apply the stylesheet given by the xsltprocessor::importStylesheet() method.

(DOMDocument and SimpleXMLElement to be covered later).

Example 2 (sample example of XSLProcessor):

sample.xml:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="example.xsl"?>
<Tutorial>
   <Title>JavaFX</Title>
   <Authors>
      <Author>Krishna</Author>
      <Author>Rajeev</Author>
   </Authors>
   <Body>Sample text</Body>
</Tutorial>
```

sample.xsl:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:output method="text"/>

   <xsl:template match="/">
      Title - <xsl:value-ofselect="/Tutorial/Title"/>
      Authors: <xsl:apply-templates select="/Tutorial/Authors/Author"/>
   </xsl:template>

   <xsl:template match="Author">
      - <xsl:value-of select="." />
   </xsl:template>
</xsl:stylesheet>
```

sample.php:

```
<?php
   $xsl = new DOMDocument();              ← Loading an XSL document
   $xsl->load("sample.xsl");

   $xml = new DOMDocument();              ← Loading an XML document
   $xml->load("sample.xml");

   $proc = new XSLTProcessor();           ← Creating an XSLTProcessor

   $proc->importStyleSheet($xsl);         ← Importing the XSL document

   $res = $proc->transformToXml($xml);    ← Transforming the style to XML
   print($res);
?>
```

result:

```
    Title - JavaFX
         Authors:
         - Krishna
         - Rajeev
```

https://www.tutorialspoint.com/php/php_function_xsltprocessor_transformtoxml.htm

Parsing XML in PHP

https://www.w3schools.com/php/php_xml_parsers.asp

To read and update, create and manipulate an XML document, you will need an XML parser.

In PHP there are two major types of XML parsers:

- Tree-Based Parsers
- Event-Based Parsers

Tree-Based Parsers

Tree-based parsers holds the entire document in Memory and transforms the XML document into a Tree structure.

It analyzes the whole document, and provides access to the Tree elements (DOM).

This type of parser is a better option for smaller XML documents, but not for large XML document as it causes major performance issues.

Example of tree-based parsers:

- SimpleXML
- DOM

Event-Based Parsers

Event-based parsers do not hold the entire document in Memory, instead, they read in one node at a time and allow you to interact with in real time. Once you move onto the next node, the old one is thrown away.

This type of parser is well suited for large XML documents. It parses faster and consumes less memory.

Example of event-based parsers:

- XMLReader
- XML Expat Parser

The XML Expat parser is a non-validating event based parser that is also built into PHP's core.
It does not require a DTD because it does not validate XML and only requires that XML be well-formed.

XML Parser (XML Expat Parser)

requires extension=libxml.so & expat library (enabled by default)

This toolkit lets you parse, but not validate, XML documents.

It supports three source character encodings also provided by PHP: US-ASCII, UTF-8 and ISO-8859-1 (default). UTF-8 is a multibyte encoding scheme, a single character may be represented by more than one byte.

This extension lets you create XML parsers and then define handlers for different XML events.
Each XML parser also has a few parameters you can adjust.

XML Parser Functions (functions better explaind in Example3, and its step by step explanation)

- **xml_parser_create()** — creates a new XML parser and returns a XMLParser instance to be used by the other XML functions.

```
xml_parser_create(?string $encoding = null): XMLParser
```

- **xml_parser_create_ns()** — Create an XML parser with namespace support

```
xml_parser_create_ns(?string $encoding = null, string $separator = ":"): XMLParser
```

- **xml_set_element_handler()** — Sets the element handler functions for the XML parser.

```
xml_set_element_handler(XMLParser $parser, callable $start_handler, callable $end_handler): bool
```

$start_handler and $end_handler are strings containing the names of functions that must exist when xml_parse() is called for parser.

- **xml_set_character_data_handler()** — Sets the character data handler function for the XML parser parser.

```
xml_set_character_data_handler (XMLParser $parser, callable $handler): bool
```

$handler is a string containing the name of a function that must exist when xml_parse() is called for parser.

Character data handler is called for every piece of a text in the XML document. It can be called multiple times inside each fragment

- **xml_parse()** — Start parsing an XML document.

```
xml_parse(XMLParser $parser, string $data, bool $is_final = false): int
```

The handlers for the configured events are called as many times as necessary.

- **xml_error_string()** — Get XML parser error string.

```
xml_error_string(int $error_code): ?string
```

- **xml_parser_free**($parser) — Free an XML parser.

This function has no effect. Prior to PHP 8.0.0, this function was used to close the resource.

- **xml_set_object**($parser, $object) — Use XML Parser within an object

This function allows to use parser inside object. All callback functions could be set with `xml_set_element_handler()` etc and assumed to be methods of object.

- **parse_into_struct**($parser, $xml, &$valueArr, &$indexArr)

This function parses an XML string into 2 parallel array structures, one (index) containing pointers to the location of the appropriate values in the values array.

Error Codes:

The PHP manual lists several XML error codes. This list is a subset of the 733 error codes of the underlying libxml library.

Prefix Code	Description
XML_ERROR_SYNTAX	The XML is not well-formed.
XML_ERROR_INVALID_TOKEN	You are using an invalid character in XML.
XML_ERROR_UNKNOWN_ENCODING	Your XML could not be parsed because the encoding scheme couldn't be determined.

Option codes:

Prefix Code	Description
XML_OPTION_CASE_FOLDING	Enabled by default and sets element names to uppercase.
XML_OPTION_SKIP_WHITE	Specifies whether to skip values consisting of whitespace characters in the source document.

Example 3 (sample example of XML Parser):

note.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

file.php

```
<?php
// Initialize the XML parser (Step1)
$parser=xml_parser_create();

// Function to use at the start of an element (Step2)
function start($parser,$element_name,$element_attrs) {
  switch($element_name) {
    case "NOTE":          echo "-- Note --<br>";    break;
    case "TO":            echo "To: ";              break;
    case "FROM":          echo "From: ";            break;
    case "HEADING":       echo "Heading: ";         break;
    case "BODY":          echo "Message: ";
  }
}

// Function to use at the end of an element (Step2)
function stop($parser,$element_name) { echo "<br>";}

// Function to use when finding character data (Step2)
function char($parser,$data) {
  echo $data;
}

// Specify element handler (Step3)
xml_set_element_handler($parser,"start","stop");

// Specify data handler (Step4)
xml_set_character_data_handler($parser,"char");

// Open XML file (Step5)
$fp=fopen("note.xml","r");

// Read data (Step6)
while ($data=fread($fp,4096)) {
 xml_parse($parser,$data,feof($fp)) or
 die (sprintf("XML Error: %s at line %d",
 xml_error_string(xml_get_error_code($parser)),
 xml_get_current_line_number($parser)));
}

// Free the XML parser (Step7)
xml_parser_free($parser);
?>
```

result.php

-- Note --
To: Tove
From: Jani
Heading: Reminder
Message: Don't forget me this weekend!

Example explained:

1 step: Initialize the XML parser with the xml_parser_create() function and creating out XMLParser instance.

2 step: Create functions to use with the different event handlers.

3 step: Add the xml_set_element_handler() function to specify which function will be executed when the parser encounters the opening and closing tags.

4 step: Add the xml_set_character_data_handler() function to specify which function will execute when the parser encounters character data.

5 step: Parse the file "note.xml" with the xml_parse() function.

6 step: In case of an error, add xml_error_string() function to convert an XML error to a textual description.

7 step: Call the xml_parser_free() function to release the memory allocated with the xml_parser_create() function.

https://www.php.net/manual/en/intro.xml.php https://www.w3schools.com/php/php_xml_parser_expat.asp
https://www.w3schools.com/php/php_ref_xml.asp

SimpleXML

`requires extension=simplexml.so , supports only version 1.0 of XML specifications`

SimpleXML provides an easy way of getting an element's name, attributes and textual content.

SimpleXML turns an XML document into a data structure you can iterate through like a collection of arrays and objects.

All objects are instances of the `SimpleXMLElement` class.

SimpleXML can be used procedurally (i.e. via functions) or in an object-oriented manner.

Compared to DOM or the Expat parser, it takes a fewer lines of code to read text data from an element but supports only well-formed XML files.

SimpleXML Functions (procedure approach, return a `SimpleXMLElement` inctance)

simplexml_load_file() – Convert the well-formed XML document in the given file to an object.

```
simplexml_load_file( str $filename, ?str $class_name = SimpleXMLElement::class, int $options = 0,
                     str $namespace_or_prefix = "", bool $is_prefix = false ): SimpleXMLElement|false
```

simplexml_load_string() — Takes a well-formed XML string and returns it as an object.

```
simplexml_load_string(str $data, ?str $class_name = SimpleXMLElement::class, int $options = 0,
                  str $namespace_or_prefix = "", bool $is_prefix = false ): SimpleXMLElement|false
```

simplexml_import_dom() — This function takes a node of a DOM document and makes it into a SimpleXML node. This new object can then be used as a native SimpleXML element.

```
simplexml_import_dom(SimpleXMLElement|DOMNode $node,
                        ?string $class_name =  SimpleXMLElement::class): ? SimpleXMLElement
```

Example 4:

```
<?php
//Load an XML file from a string
$xmlstr = file_get_contents('library.xml');
$library = simplexml_load_string($xmlstr);
?>
<?php
//Alternatively, load XML from a file
$library = simplexml_load_file('library.xml');
?>
```

The SimpleXMLElement class (oop approach)

```
class SimpleXMLElement implements Traversable {

      public __construct( $data, $options = 0, $dataIsURL = false,
                                          $namespaceOrPrefix = "", $isPrefix = false)
      ...
}
```

Example 5:

```
<php
//Load from an XML String
$xmlstr = file_get_contents('library.xml');
$library = new SimpleXMLElement($xmlstr);
//Load from an XML file
$library = new SimpleXMLElement('library.xml', null, true);
?>
```

After the `SimpleXMLElement` instance has been created (by oop or procedure), a number of method will be available to used.

Methods	Description
SimpleXMLElement::__construct	Creates a new SimpleXMLElement object
SimpleXMLElement::addAttribute	Adds an attribute to the SimpleXML element
SimpleXMLElement::addChild	Adds a child element to the XML node
SimpleXMLElement::asXML	Return a well-formed XML string based on SimpleXML elem
SimpleXMLElement::saveXML	Alias of SimpleXMLElement::asXML
SimpleXMLElement::attributes	Identifies an element's attributes
SimpleXMLElement::children	Finds children of given node
SimpleXMLElement::count	Counts the children of an element
SimpleXMLElement::getName	Gets the name of the XML element
SimpleXMLElement::getNamespaces	Returns namespaces used in document
SimpleXMLElement::getDocNamespaces	Returns namespaces declared in document
SimpleXMLElement::xpath	Runs XPath query on XML data
SimpleXMLElement::__toString	Returns the string content

Iterate over SimpleXML

Since SimpleXMLElement implements Traversable, loop iteratioans as foreach can be performend on its instances.

`library.xml (file)`

```
<?xml version="1.0"?>
<library>
    <book isbn="0312863551">
        <title>The Moon is a Harsh Mistress</title>
        <author>R.A. Heinlein</author>
        <publisher>Orb</publisher>
    </book>
    <book isbn="0345342968">
        <title>Fahrenheit 451</title>
        <author>R. Bradbury</author>
        <publisher>Del Rey</publisher>
    </book>
    <book isbn="0048231398">
        <title>The Silmarillion</title>
        <author>J.R.R. Tolkien</author>
        <publisher>G. Allen and Unwin</publisher>
    </book>
    <book isbn="0451524934">
        <title>1984</title>
        <author>G. Orwell</author>
        <publisher>Signet</publisher>
    </book>
    <book isbn="031219126X">
        <title>Frankenstein</title>
        <author>M. Shelly</author>
        <publisher>Bedford</publisher>
    </book>
</library>
```

`example5.php (file)`

```
<?php

$libraryObj = new
    SimpleXMLElement('./library.xml', 0, true);

var_export($libraryObj);
?>

SimpleXMLElement::__set_state(array(
   'book' => array (

    0 =>
    SimpleXMLElement::__set_state(array(
       '@attributes' => array ('isbn' =>
                                '0312863551',),
       'title' => 'The Moon is a Harsh Mistress',
       'author' => 'R.A. Heinlein',
       'publisher' => 'Orb',
    )),

    1 =>
    SimpleXMLElement::__set_state(array(
       '@attributes' =>  array ('isbn' =>
        ........
    )),
  ),
))
```

Example5 (foreach loop):

```php
<?php
foreach ($libraryObj as $elem) {
    printf('isbn:  %s <br>', $elem['isbn']);       <- attributes are accessed as array elements
    printf('title: %s <br>', $elem->title);        <- children elem are accessed as object fields
    echo '<br>';
}
?>
```

```
isbn: 0312863551
title: The Moon is a Harsh Mistress

isbn: 0345342968
title: Fahrenheit 451

isbn: 0048231398
title: The Silmarillion

isbn: 0451524934
title: 1984

isbn: 031219126X
title: Frankenstein
```

Example6 (using class methods):

```php
<?php
echo $libraryObj->getName();
echo $libraryObj->count();

foreach ($libraryObj as $elem) {

    print_r($elem->attributes());
    print_r($elem->children());
    echo '<br><br>';
}
?>
```

```
library
5
SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0312863551 ) )
SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0312863551 ) [title] => The Moon is a Harsh Mistress [author] => R.A. Heinlein [publisher] => Orb )

SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0345342968 ) )
SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0345342968 ) [title] => Fahrenheit 451 [author] => R. Bradbury [publisher] => Del Rey )

SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0048231398 ) )
SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0048231398 ) [title] => The Silmarillion [author] => J.R.R. Tolkien [publisher] => G. Allen and Unwin )

SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0451524934 ) )
SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 0451524934 ) [title] => 1984 [author] => G. Orwell [publisher] => Signet )

SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 031219126X ) )
SimpleXMLElement Object ( [@attributes] => Array ( [isbn] => 031219126X ) [title] => Frankenstein [author] => M. Shelly [publisher] => Bedford )
```

XML Namespaces

Namespaces in XML allow for tags to be used in a document from different Document Type Definitions (DTDs).

Example7 :

```
<?xml version="1.0"?>
<library
    xmlns="http://example.org/library"
    xmlns:meta="http://example.org/book-meta"
    xmlns:pub="http://example.org/publisher"
    xmlns:foo="http://example.org/foo">

    <book meta:isbn="0345342968">
        <title>Fahrenheit 451</title>
        <author>Ray Bradbury</author>
        <pub:publisher>Del Rey</pub:publisher>
    </book>
</library>
```

> **Note:** how the isbn and publisher tags come from a defined namespace.
> All other tags will use the default namespace (library).

`SimpleXMLElement::getDocNamespaces()`

An array of namespaces **declared in root**, passing TRUE as the argument will return all namespaces declared also in the children tags

`SimpleXMLElement::getNamespaces()`

An array of namespaces **actually used in root**, passing TRUE as the argument will return all namespaces declared also in the children tags

Example8 :

```
<?php
$libraryObj = new SimpleXMLElement('./library2.xml', 0, true);

$declaredNamespaces = $libraryObj->getDocNamespaces();
var_export($declaredNamespaces);
```

```
array (
  '' => 'http://example.org/library',
  'meta' => 'http://example.org/book-meta',
  'pub' => 'http://example.org/publisher',
  'foo' => 'http://example.org/foo',
)
```

```
$declaredNamespaces = $libraryObj->getDocNamespaces(true);
var_export($declaredNamespaces);
```

```
array (
  '' => 'http://example.org/library',
  'meta' => 'http://example.org/book-meta',
  'pub' => 'http://example.org/publisher',
  'foo' => 'http://example.org/foo',
  'test' => 'http://test.com',
)
```

```
$usedNamespace = $libraryObj->getNamespaces();
var_export($usedNamespace);
```

```
array ('' => 'http://example.org/library',)
```

```
$usedNamespace = $libraryObj->getNamespaces(true);
var_export($usedNamespace);
?>
```

```
array (
  '' => 'http://example.org/library',
  'meta' => 'http://example.org/book-meta',
  'pub' => 'http://example.org/publisher',
)
```

xpath

XPath is a query language used to select nodes within an XML document .

It models an XML document as a series of nodes and uses path expressions for navigating through and selecting nodes from the document.

`SimpleXMLElement::xpath()` runs an XPath query on XML data and returns an array of children that match the path specified.

Unlike PHP structures, XPath results are not zero-based.
The XPath `/college/student[1]/name` will return the first student, not the second.

On the other hand, **PHP arrays containing xpath results are zero-based.**
So, if you store your results in an array variable called `$array` then `$array[0]` will correspond to the `college/student[1]/` name in the previous example.

Example 9 :

```
<?php
$libraryObj = new SimpleXMLElement('./library.xml', 0, true);

$titleElemArr =  $libraryObj->xpath('/library/book/title');
var_export($titleElemArr);
?>
array (
  0 => SimpleXMLElement::__set_state(array( 0 => 'The Moon is a Harsh Mistress',)),
  1 => SimpleXMLElement::__set_state(array( 0 => 'Fahrenheit 451',)),
  2 => SimpleXMLElement::__set_state(array( 0 => 'The Silmarillion',)),
  3 => SimpleXMLElement::__set_state(array( 0 => '1984',)),
  4 => SimpleXMLElement::__set_state(array( 0 => 'Frankenstein',)),
)

<?php
foreach($titleElemArr as $title) echo $title , ', ';
?>
The Moon is a Harsh Mistress, Fahrenheit 451, The Silmarillion, 1984, Frankenstein,

<?php
 $titleElemArr =  $libraryObj->xpath('/library/book[1]/title');
 var_export($titleElemArr);
?>
array ( 0 => SimpleXMLElement::__set_state(array( 0 => 'The Moon is a Harsh Mistress', )), )


<?php
 $titleElemArr =  $libraryObj->book[0]->xpath('title');
 var_export($titleElemArr);
?>
array ( 0 => SimpleXMLElement::__set_state(array( 0 => 'The Moon is a Harsh Mistress', )), )
```

In the table below we have listed some XPath expressions and the result of the expressions (just for ref):

XPath Expression	Result
`/library/book[1]`	Selects the first book element that is the child of the library element
`/library/book[last()]`	Selects the last book element that is the child of the library element
`/library/book[last()-1]`	Selects the 1 before last book element that is the child of the library element
`/library/book[position()<3]`	Selects the first two book elements that are children of the library element
`//title[@lang]`	Selects all the title elements that have an attribute named lang
`//title[@lang='en']`	Selects all the title elements that have a "lang" attribute with a value of "en"
`/library/book[price>35.00]`	Selects all the book elements of the library element that have a price element with a value greater than 35.00
`/library/book[price>35.00]/title`	Selects all the title elements of the book elements of the library element that have a price element with a value greater than 35.00

Example 10 :

```php
<?php
$libraryObj = new SimpleXMLElement('./library.xml', 0, true);

$select = $libraryObj->xpath('/library/book[last()-2]');
var_export($select);

$select = $libraryObj->xpath('/library/book[position()<3]');
var_export($select);
?>
```

```
array (
  0 =>
  SimpleXMLElement::__set_state(array(
     '@attributes' => array ('isbn' => '0048231398',),
     'title' => 'The Silmarillion',
     'author' => 'J.R.R. Tolkien',
     'publisher' => 'G. Allen and Unwin',
  )),
)


array (
  0 =>
  SimpleXMLElement::__set_state(array('@attributes' => array ('isbn' => '0312863551', ),
     'title' => 'The Moon is a Harsh Mistress',
     'author' => 'R.A. Heinlein',
     'publisher' => 'Orb',
  )),
  1 =>
  SimpleXMLElement::__set_state(array('@attributes' => array ('isbn' => '0345342968', ),
     'title' => 'Fahrenheit 451',
     'author' => 'R. Bradbury',
     'publisher' => 'Del Rey',
  )),
)
```

DOM (Document Object Model)

`requires libxml2 extension (Gnome XML Library) and expat library (enabled by default)`

The `DOMDocument` class is useful for working with XML and HTML.

The DOM extension uses UTF-8 encoding.
Use `utf8_encode()` and `utf8_decode()` to work with texts in ISO-8859-1 encoding

`utf8_encode($str): string` — This function converts the $str from the ISO-8859-1 encoding to UTF-8.

`utf8_decode($str): string` — This function converts the $str from the UTF-8 encoding to ISO-8859-1.

Bytes in the string which are not valid UTF-8, and UTF-8 characters which do not exist in ISO-8859-1 are replaced with ' ? '

The `DOMDocument` loads entire document into ram, then creates an internal tree representation.

```
class DOMDocument extentds DOMNode {
      public __construct( string $version = "1.0",  string $encoding = "")
      ...
}
```

After the `DOMDocument` instance has been created, a number of method will be available to used:

Methods	Description
DOMDocument::__construct	Creates a new DOMDocument object

Methods (Load Data to Instance)	Description
DOMDocument::load	Load XML from a file
DOMDocument::loadXML	Load XML from a string
DOMDocument::loadHTMLFile	Load HTML from a file
DOMDocument::loadHTML	Load HTML from a string

Methods (Save Instance)	Description
DOMDocument::save	Dumps the internal XML tree back into a file
DOMDocument::saveXML	Dumps the internal XML tree back into a string
DOMDocument::saveHTMLFile	Dumps the internal document into a file using HTML formatting
DOMDocument::saveHTML	Dumps the internal document into a string using HTML formatting

Methods (Get Elements / Nodes)	Description
DOMDocument::**getElementById**	Searches for an element with a certain id. Return a **DOMElement inctance.**
DOMDocument::**getElementsByTagName**	Searches for all elements with given local tag name. Return a **DOMNodeList inctance.**
DOMDocument::**getElementsByTagNameNS**	Searches for all elements with given tag name in specified namespace. Return a **DOMNodeList inctance.**

Note: Two of these methods return a DOMNodeList object. DOMNodeList object, can be traversed over using foreach().

Methods (Creating Nodes)	Description
DOMDocument::**createTextNode**	Create new text node Return a **DOMText inctance** or false
DOMDocument::**createElement**	Create new element node Return a **DOMElement inctance** or false
DOMDocument::**createElementNS**	Create new element node with an associated namespace. Return a **DOMElement inctanc**e or false

Note: These functions create a new element node or a new text node.
These node will not show up in the document unless it is inserted with `DOMNode::appendChild()`.

DOMDocument class extends DOMNode class and so the appendChild method is inherited.

Example 11 (Basic Usage Example):

```php
<?php
$xml = <<< XML
<?xml version="1.0" encoding="utf-8"?>
<books>
    <book>Patterns of Enterprise Application Architecture</book>
    <book>Design Patterns: Elements of Reusable Software Design</book>
    <book>Clean Code</book>
</books>
XML;

$dom = new DOMDocument;
$dom->loadXML($xml);

$books = $dom->getElementsByTagName('book');

foreach ($books as $book) {
    echo $book->nodeValue, PHP_EOL;
}
?>
```

```
Patterns of Enterprise Application Architecture
Design Patterns: Elements of Reusable Software Design
Clean Code
```

Example 12 (Creating a new element and inserting it as root):

```php
<?php
header('Content-Type: text/xml');

$dom = new DOMDocument('1.0', 'utf-8');

$root = $dom->createElement('root', 'This is the root elemrnt');

$dom->appendChild($root);

$A = $dom->createElement('A', 'This is element A');
$root->appendChild($A);

$B = $dom->createElement('B', 'And is element B');
$root->appendChild($B);

echo $dom->saveXML();
?>
```

```
<root>
	This is the root elemrnt
	<A>This is element A</A>
	<B>And is element B</B>
</root>
```

Note: The following line will produce a Warning (and Error in SimpleXMLElement) :

```php
$B = $dom->createElement('B', '& is element B');
```

& < > ' " has to be replaced , htmlspecialchars('&') can be used.

Methods (inherited from DOMNode)	Description
DOMNode::**appendChild**	Adds new child at the end of the children
DOMNode::**removeChild**	Removes child from list of children
DOMNode::**replaceChild**	Replaces a child
DOMNode::**cloneNode**	Clones a node
DOMNode::**insertBefore**	Adds a new child before a reference node. You need to reference the parent node and also specify the sibling node.
DOMNode::**parentNode**	(Property not method, as on the book) The parent of this node. If there is no such node, this returns null. (ex of other props: firstChild, previousSiblings, childNodes ... etc)

Methods (some Attribute methods)	Description
DOMElement::**setAttribute**	Sets an attribute with name qualifiedName to the given value. If the attribute does not exist, it will be created.
DOMElement::**setIdAttribute**	Declares the attribute specified by name to be of type ID
DOMElement::**setAttributeNS**	Sets an attribute with namespace namespace and name name to the given value. If the attribute does not exist, it will be created.
DOMElement::**getAttributeNS()**	Returns value of attribute
DOMElement::**hasAttributeNS()**	Checks to see if attribute exists
DOMElement::**removeAttributeNS()**	Removes attribute

Using XPath with PHP DOM

The `DOMXPath` class is used to run the XPath query using the appropriately named `query()` method.

The query() method returns an traversable `DOMNodeList` object that contains the list of DOMNode objects. (or false if expression is malformed).

The basic steps in useing DOMXPath Class are:

1. Initializing a **DOMDocument** class instance
2. Initializing a **DOMXPath** object from the DOMDocument object
3. Parsing the **DOMXPath** object

Example 13 :

```php
<?php
header('Content-Type: application/json');

$library = <<<ENDXML
<?xml version="1.0"?>
<library>
    <book isbn="0312863551">
       <title>The Moon is a Harsh..</title>
       <author>R.A. Heinlein</author>
    </book>
    <book isbn="0345342968">
       <title>Fahrenheit 451</title>
    </book>
    <book isbn="0048231398">
       <title>The Silmarillion</title>
       <author>J.R.R. Tolkien</author>
    </book>
</library>
ENDXML;

$domDoc = new DOMDocument();
$domDoc->loadXML($library);

$xpath = new DOMXPath($domDoc);

// quering for any element that have an isbn
   attribut
$elements = $xpath->query("//*[@isbn]");

if (!is_null($elements)) {

    foreach($elements as $e){

        echo $e->nodeName;
        $nodes = $e->childNodes;

        foreach($nodes as $n){
            echo $n->nodeValue;
        }
    }
}
?>
```

```
book    The Moon is a Harsh Mistress    R.A. Heinlein
book    Fahrenheit 451
book    The Silmarillion                J.R.R. Tolkien
```

DOM Functions

dom_import_simplexml() — This function takes the node node of class SimpleXML and makes it into a DOMElement node. This new object can then be used as a native DOMElement node.

```
dom_import_simplexml(object $node): DOMElement
```

Note: This function return an DOMElement Object not a DOMDocument Object

Example 13 (from `SimpleXML` to `DOMDocument`) :

```
<?php
header('Content-Type: text/xml');

$sxml = simplexml_load_string('<books><book><title>blah</title></book></books>');

$domElem = dom_import_simplexml($sxml);


$dom = new DOMDocument('1.0', 'utf-8');

$node = $dom->importNode($domElem, true);
$dom->appendChild($node);

echo $dom->saveXML();

?>
<books>
    <book>
        <title>blah</title>
    </book>
</books>
```

Example 14 (from `DOMDocument to SimpleXML`) :

```
<?php
header('Content-Type: text/xml');
$sml = simplexml_import_dom($dom);
echo $sml->asXML();
?>

<books>
    <book>
        <title>blah</title>
    </book>
</books>
```

Example 15:

```
<?php
$sml = simplexml_import_dom($dom);
echo $sml->book[0]->title;
?>
blah
```

SOAP

SOAP was originally an acronym of Simple Object Access Protocol.

The PHP SOAP extension is used to write SOAP servers and clients. `extension=soap.so`

It requires that libxml is enabled, which is by default.

SOAP cache functions are configured in the php.ini file with the soap.wsdl_cache_* settings.

There are only two SOAP functions:

`is_soap_fault()` returns whether a SOAP call has failed.

`use_soap_error_handler()`
Is used for the SOAP server and sets whether PHP should use the SOAP error handler.
If it is set to false, the PHP error handler is used instead of sending a SOAP error to the client.

The rest of the SOAP functionality is provided in classes:

`SoapClient` class & `SoapServer` class

What SOAP Does:

SOAP allows complex data types to be defined and exchanged and provides a mechanism for various messaging patterns, the most common of which is the Remote Procedure Call (RPC).

SOAP web services are defined by a WSDL (Web Service Description Language). “whizz-dill”.

The WSDL defines the data types using an XML structure. It also describes the methods that may be called remotely, specifying their names, parameters, and return types.

SOAP messages between a server and client are sent in XML structures called SOAP envelopes.

Using a SOAP Service (PHP SOAP Client)

The class SoapClient can take a WSDL file as input, and create an object that mimics the services of the web service (in this example, it has the login method):

Example 15

```php
<?php
$client = new SoapClient("http://example.com/login?wsdl");

$params = array('username'=>'name', 'password'=>'secret');
// call the login method directly
$client->login($params);
// If you want to call __soapCall, you must wrap the arguments in another array as follows:
$client->__soapCall('login', array($params));
?>
```

Some SoapClient Methods	Description
SoapClient::__getFunctions()	Returns list of available SOAP functions (ex from wsdl)
SoapClient::__getTypes()	Returns a list of SOAP types (ex from wsdl)

Debuging Methods	Description
SoapClient::__getLastRequest()	Returns last SOAP request
SoapClient::__getLastRequestHeaders()	Returns the SOAP headers from the last request
SoapClient::__getLastResponse()	Returns last SOAP response
SoapClient::__getLastResponseHeaders()	Returns the SOAP headers from the last response

Note: All Debug methods only works if the SoapClient object was created with the trace option set to true.
Ex: `$client = SoapClient("some.wsdl", array('trace' => 1));`

Offering a SOAP Service (PHP SOAP Server)

The SoapServer class provides a SOAP server.
It can be can be used with or without a WSDL (as in SoapClient class).

Example 16:

```
<?php
$options = ['uri'=>'http://localhost/test'];
$server = new SoapServer(NULL, $options);
$server->setClass('MySoapServer');
$server->handle();
```

In this example:

- First we are create the server with an array of options (we are not supplying a WSDL in the first parameter and so we must supply the URI).
- Once we have an instance of the SoapServer class, we pass in the name of the class that it will use to server requests. The methods in the class will be callable by a SOAP client connecting to the server.
- We can use an object instead of a class by using SoapServer::setObject()
- Finaly, handled a SOAP request.

Some SoapClient Methods	Description
SoapServer::handle()	Handles a SOAP request
SoapServer::setClass()	Sets the class which handles SOAP requests
SoapServer::setObject()	Sets the object which will be used to handle SOAP requests
SoapServer::addFunction()	Adds one or more functions to handle SOAP requests

REST Web Services

REST is an acronym for **Representational State Transfer** and is an architectural style rather than a PHP extension or set of commands.

REST has several verbs that are similar to HTTP request types.

But REST does not have to use HTTP as a transport layer to communicate. HTTP is very convenient for REST because it is stateless and the request types translate well into REST verbs.

REST exposes Uniform Resource Identifiers (URI) that are linked to resources. These links are called REST endpoints.
Depending on the HTTP type used to access them, they will perform an action on the resource (change its state). REST focuses on resources and providing access to those resources.

A resource could be something like a “user”. Much like a database schema represents the user entity, RESTwill represent the user in a JSON or XML structure.

A representation should be readable by both the server and the client. REST can be used to transfer JSON, XML, or both.

In PHP, one of the most common uses for REST APIs is to provide services for an AJAX enabled frontend.

Application and Resource States

A REST server should not remember the state of the application and the client should send all the information necessary for execution.

This means that every request to a server is self-contained.

If a request to a server failed it will not affect the success or failure of other requests. This improves the reliabilityof the application.

The server is not responsible for remembering what state the application is in and relies on the client to send all the information it needs to process the request.

This meansthat the **client stores and maintains the application state** (and not the server).

The resource that REST is providing access to has state that is expected to persist between requests.
Resource state is maintained on the server.

REST verbs

REST verbs specify an action to be performed to alter the state of resource or a collection of resources.

When a request is made by the client, it should send this information in the HTTP request:

- REST verb
- Header information
- Body (optional)

There are quite a few REST verbs available, but six of them are used frequently. They are as follows:

REST Verb	Action	Success	Failure
GET	Fetches a record or set of resources from the server	200	404
OPTIONS	Fetches all available REST operations	200	–
POST	Creates a new set of resources or a resource	201	404, 409
PUT	Updates or replaces the record or create it if it doesn't exist	200, 204	404
PATCH	Modifies the given record without sending the complete representation of it	200, 204	404
DELETE	Deletes the given resource	200	404

HATEOAS

The term HATEOAS stands for the phrase Hypermedia As The Engine Of Application State.

When the browser loads the page, you definitely can see all the content that the page has to offer.

More interestingly, the page also allows you to perform a lot of actions around that data, as clicking on buttons, open new tabs and several more.

But typically, when we perform a REST request, we only get the data and not any actions around it.

With HATEOAS, a request for a REST resource gives both data, and actions related to the data.

```
GET /accounts/12345 HTTP/1.1
Host: bank.example.com

The response is:
HTTP/1.1 200 OK
{
    "account": {
        "account_number": 12345,
        "balance": {
            "currency": "usd",
            "value": 100.00
        },
        "links": {
            "deposits": "/accounts/12345/deposits",
            "withdrawals": "/accounts/12345/withdrawals",
            "transfers": "/accounts/12345/transfers",
            "close-requests": "/accounts/12345/close-requests"
        }
    }
}
```

When you send out this request to retrieve account details, you get both:

- Account number, currency and balance details
- Links that provide actions to do a deposit/withdrawal/transfer/closure

The server is guiding the client through the API by exposing additional URIs that are relevant to the last operation

Request Headers

HTTP allows passing headers in its request.
REST clients will use these to indicate to theserver what they are providing and what they are expecting back.
A REST client should use the accept header to indicate to the server what sort of content (representation) it wants back.
The client will also set a Content-Type header to inform the server of the MIME type of its payload

Example 17 (dispalying JSON):

```php
<?php
header('Content-Type: application/json');

$datadb = ['users' =>
       ['id1' => ['name' => 'Dorothy', 'surname' => 'Cassar', 'age' => 32],
        'id2' => ['name' => 'Martina', 'surname' => 'Taljana', 'age' => 34],]];

$stringdb = json_encode($datadb);
echo $stringdb;
?>
```

```
{
    "users":{
        "id1":{
            "name":"Dorothy",
            "surname":"Cassar",
            "age":32
        },
        "id2":{
            "name":"Martina",
            "surname":"Taljana",
            "age":34
        }
    }
}
```

Example 18 (dispalying XML):

```php
<?php
header('Content-Type: text/xml');

$data = <<< ENDXML
<?xml version="1.0"?>
<users>
    <id1>
        <name>Dorothy</name>
        <surname>Cassar</surname>
        <age>32</age>
    </id1>
    <id2>
        <name>Martina</name>
        <surname>Taljana</surname>
        <age>34</age>
    </id2>
</users>
ENDXML;

echo $data;
?>
```

This XML file does not appear to have any style information associated with it.

```
<users>
<id1>
<name>Dorothy</name>
<surname>Cassar</surname>
<age>32</age>
</id1>
<id2>
<name>Martina</name>
<surname>Taljana</surname>
<age>34</age>
</id2>
</users>
```

Response Headers and Codes

The Content-Type header is sent by the server and defines the MIME type of the body that is being sent. For example, a server may set the content-type to application/json to indicate that the body of the response contains JSON formatted text.

2xx family (successful)	Indicate that an operation was successful.
200 (Successful Operation)	Is the most common type of response status code in REST
201 (Successfully Created)	Returned when a POST operation successfully creates a resource on the server
202 (Accepted)	The resource was accepted for processing, but has not yet been processed
204 (No content)	Is issued when a client needs a status but not any data back
3xx family (redirection)	These status codes are used to convey redirection messages.
301	Issued when a resource is moved permanently to a new URL endpoint. It is essential when an old API is deprecated. It returns the new endpoint in the response with the 301 status.
4xx family (client error)	These are the standard error status codes which the client needs to interpret and handle further actions. These have nothing to do with the server.
400 (Bad Request)	Returned when the server cannot understand the client request.
401 (Unauthorized)	Returned when the client is not sending the authorization info in the header.
403 (Forbidden)	Returned when the client has no access to a certain type of resources.
404 (Not Found)	Returned when the client request is on a resource that is nonexisting.
405 (Method Not Allowed)	Returned if the server bans a few methods on resources.
5xx family (server error)	These are the errors from the server. The client request may be perfect, but due to a bug in the server code, these errors can arise.
500 (Internal Server Error)	This status code gives the development error which is caused by some buggy code or some unexpected condition
501 (Not Implemented)	Returned when the server is no longer supporting the method on a resource
502 (Bad Gateway)	Returned when the server itself got an error response from another service vendor
503 (Service Unavailable)	Returned when the server is down due to multiple reasons, like a heavy load or for maintenance
504 (Gateway Timeout)	Returned when the server is waiting a long time for a response from another vendor and is taking too much time to serve the client

Sending Requests

The curl extension is a common way to send REST requests in PHP. Curl lets you specify headers and request types.

JSON

JSON is an acronym of JavaScript Object Notation. In PHP, it is used a lot with Ajax, which is an acronym for Asynchronous JavaScript and XML.

JSON lets you serialize an object as a string so that it can be transported between services.

Ajax is a means to transport the string.

Together these technologies allow you to communicate between JavaScript applications in the browser and PHP applications on the server.

The JSON extension is loaded in PHP by default

JSON Functions

json_decode(), json_encode(), json_last_error(), json_last_error_message()

json_decode() — Takes a JSON encoded string and converts it into a PHP variable

```
json_decode(str $json, ?bool $associative = null, int $depth = 512, int $flags = 0 ):  mixed
```

`$associative`

When true, JSON objects will be returned as associative arrays;

When false, JSON objects will be returned as objects.

When null, JSON objects will be returned as associative arrays or objects depending on whether JSON_OBJECT_AS_ARRAY is set in the flags.

`$depth` Maximum nesting depth of the structure being decoded.

Example 19:

```
<?php
$json = '{"a":1,"b":2,"c":3}';

var_dump(json_decode($json));
var_dump(json_decode($json, true));
?>
```

```
object(stdClass)#1 (5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)
}

array(5) {
    ["a"] => int(1)
    ["b"] => int(2)
    ["c"] => int(3)

}
```

Example 20:

```
Accessing elements within an object that
contain characters

<?php

$json = '{"foo-bar": 12345}';

$obj = json_decode($json);

print $obj->{'foo-bar'};

?>
```

```
12345
```

Example 21:

```php
<?php
// Encode some data with a depth  of 5 (array -> array -> array -> array -> int)

$arr = ['language' => [
    'english' => ['numbers' => [1, 2, 3]],
    'german' =>  ['numbers' => [1, 2, 3]],
]];

$json = json_encode($arr);
$arr2 = json_decode($json, True, 4);

var_export($arr2);
echo 'Last error: ', json_last_error_msg();
?>
```

```
NULL
Last error: Maximum stack depth exceeded
```

```php
// Updating maximun depth to 5
<?php
$arr3 = json_decode($json, True, 5);
var_export($arr3);
echo 'Last error: ', json_last_error_msg();
?>
```

```
array ( 'language' => array (
      'english' => array ( 'numbers' => array ( 0 => 1, 1 => 2, 2 => 3, ), ),
      'german'  => array ( 'numbers' => array ( 0 => 1, 1 => 2, 2 => 3, ), ),
      ), )

Last error: No error
```

Example 22 (json_decode() of large integers):

```php
<?php
$json = '{"number": 12345678901234567890}';

var_dump(json_decode($json));
var_dump(json_decode($json, false, 512, JSON_BIGINT_AS_STRING));
?>
```

```
object(stdClass)#1 (1) {["number"] => float(1.2345678901235E+19)}
object(stdClass)#1 (1) {["number"]=> string(20) "12345678901234567890"}
```

json_encode() — Returns a string containing the JSON representation of the supplied value.

```
json_encode(mixed $value, int $flags = 0, int $depth = 512): string|false
```

Example 23:

```
<?php
$arr = array('a' => 1, 'b' => 2, 'c' => 3, 'd' => 4, 'e' => 5);

echo json_encode($arr);
?>
```

```
{"a":1,"b":2,"c":3,"d":4,"e":5}
```

Example 24 (A json_encode() example showing some flags in use):

```
<?php
$a = array('<foo>',"'bar'",'"baz"','&blong&', "\xc3\xa9");

echo "Normal"     json_encode($a);

echo "Tags: ",    json_encode($a, JSON_HEX_TAG);

echo "All: ",     json_encode(
      $a, JSON_HEX_TAG | JSON_HEX_APOS | JSON_HEX_QUOT | JSON_HEX_AMP | JSON_UNESCAPED_UNICODE
)
?>
```

```
Normal: ["<foo>","'bar'","\"baz\"","&blong&","\u00e9"]

Tags:   ["\u003Cfoo\u003E","'bar'","\"baz\"","&blong&","\u00e9"]

All:    ["\u003Cfoo\u003E","\u0027bar\u0027","\u0022baz\u0022","\u0026blong\u0026","é"]
```

Example 25 (JSON_FORCE_OBJECT):

```
<?php

$b = array();

echo "Empty array output as array: ", json_encode($b);
echo "Empty array output as object: ", json_encode($b, JSON_FORCE_OBJECT);

$c = array(array(1,2,3));

echo "Non-associative array output as array: ", json_encode($c);
echo "Non-associative array output as object: ", json_encode($c, JSON_FORCE_OBJECT);

$d = array('foo' => 'bar', 'baz' => 'long');

echo "Associative array always output as object: ", json_encode($d);
echo "Associative array always output as object: ", json_encode($d, JSON_FORCE_OBJECT);
```

```
Empty array output as array: []
Empty array output as object: {}

Non-associative array output as array: [[1,2,3]]
Non-associative array output as object: {"0":{"0":1,"1":2,"2":3}}

Associative array always output as object: {"foo":"bar","baz":"long"}
Associative array always output as object: {"foo":"bar","baz":"long"}
```

json_last_error_msg () — Returns the error strinof the last `json_encode()` or `json_decode()` call. (which did not specify JSON_THROW_ON_ERROR).

```
json_last_error_msg(): string
```

json_last_error() — Returns the last error (if any) occurred during the last JSON encoding/decoding. (which did not specify JSON_THROW_ON_ERROR).

```
json_last_error(): int
```

Note: If JSON_THROW_ON_ERROR is specifed in `json_encode()` or `json_decode()` as a flag, an `Exception` which will be an instance of `JsonException class` will be thrown.

Constant	Meaning
JSON_ERROR_NONE	Confirms whether a JSON error occurred or not.
JSON_FORCE_OBJECT	Outputs an object rather than an array, even if array is empty.
JSON_BIGINT_AS_STRING	Decodes large integers as their original string value.
JSON_ERROR_SYNTAX	Confirms if there was a syntax error parsing JSON and helps detect encoding errors.
JSON_ERROR_UTF8	Malformed UTF-8 characters, possibly incorrectly encoded.
JSON_ERROR_DEPTH	The maximum stack depth has been exceeded.
JSON_OBJECT_AS_ARRAY	Decodes JSON objects as PHP array. This option can be added automatically by calling json_decode() with the second parameter equal to true.
JSON_HEX_TAG	All < and > are converted to \u003C and \u003E.
JSON_HEX_AMP	All & are converted to \u0026.
JSON_HEX_APOS	All ' are converted to \u0027.
JSON_HEX_QUOT	All " are converted to \u0022.
JSON_PRESERVE_ZERO_FRACTION	Ensures that float values are always encoded as a float value.

The JsonSerializable interface

Objects implementing JsonSerializable can customize their JSON representation when encoded with `json_encode()`.

```
interface JsonSerializable {
      // Methods
      public jsonSerialize(): mixed
}
```

Date and Time

PHP supplies several functions that retrieve the date and time from the server.
In PHP 5.2 the DateTime class was introduced , which deals with a wide range of date and time calculations.

time() — Returns the current time measured in the number of seconds since the Unix timestamp (January 1 1970 00:00:00 GMT).

```
time(): int
```

Example 26:

```
<?php
$now = time();

$nextWeek = $now + 7 * 24 * 60 * 60;

echo $now , '<br>', $nextWeek;
?>
```

```
1635575979
1636180779
```

date() — Format a local time/date

```
date(string $format, ?int $timestamp = null): string
```

Returns a string formatted according to the given format string using the given integer timestamp or the current time if no timestamp is given.

Example 27:

```
<?php
echo date('l jS \of F Y h:i:s A');
?>
```

```
Saturday 30th of October 2021 09:27:14 AM
```

```
<?php
$in3days = time() + 3 * 24 * 60 * 60;
echo date('l jS \o\f F Y h:i:s A', $in3days);
?>
```

```
Tuesday 2nd of November 2021 08:31:22 AM
```

Note:
The 'o' in 'of' needed to be excaped. In the second example I have excaped both letters just to be sure.

Code	Description	Example(s)
D	A textual representation of a day (three letters)	Mon - Sun
M	A short textual representation of a month (three letters)	Jan - Dec
Y	A four digit representation of a year	1970
d	The day of the month	01 to 31
m	A numeric representation of a month	01 to 12
y	A two digit representation of a year	70
l	A full textual representation of the day of the week	Sunday - Saturday
F	A full textual representation of a month	January - December
h	12-hour format of an hour	01 to 12
H	24-hour format of an hour	00 to 23
i	Minutes with leading zeros	00 to 59
s	Seconds, with leading zeros	00 to 59
u	Microseconds (added in PHP 5.2.2)	654321
a	Lowercase am or pm	am
A	Uppercase AM or PM	AM
T	Timezone abbreviations (Examples: EST, MDT)	EST, MDT, CEST
e	The timezone identifier	Europe/Berlin
O	Difference to Greenwich time (GMT) in hours	+0200
P	Difference to Greenwich time (GMT) in hours:minutes	+02:00

date_default_timezone_set() — Sets the default timezone used by all date/time functions in a script

```
date_default_timezone_set(string $timezoneId): bool
```

date_default_timezone_get() — date_default_timezone_get — Gets the default timezone used by all date/time functions in a script

```
date_default_timezone_get(): string
```

Returns the default timezone by:

1. Reading the timezone set using the `date_default_timezone_set()` function (if any)
2. Reading the value of the `date.timezone` in `php.ini` option (if set)
3. Return the system timezone defined in `/etc/localtime`. A warning may shown when this stage is reached.

Note: Instead of using `date_default_timezone_set()` to set the default timezone in your script, you can also use the INI setting date.timezone to set the default timezone.

Example: `ini_set('date.timezone','UTC');` and to get echo `ini_get('date.timezone');`

Example 28:

```
<?php
echo date_default_timezone_get(),;              // Europe/Berlin

echo date('h:i:s a');                           // 10:45:23 am

date_default_timezone_set('UTC');

echo date_default_timezone_get();               // UTC

echo date('h:i:s a');                           // 08:45:23 am
?>
```

Example 29:

```
<?php
var_dump(ini_get('date.timezone'));             // string(0) ""

ini_set('date.timezone','America/New_York');

echo ini_get('date.timezone');                  // America/New_York

echo date('h:i:s a');                           // 04:47:52 am
?>
```

strftime() — Format a local time/date according to locale settings.

```
strftime(string $format, ?int $timestamp = null): string|false
```

Example 30:

```
<?php
if (setlocale(LC_ALL, 'de_DE.UTF-8')){

    date_default_timezone_set('America/New_York');

    echo strftime('%A %d-%m-%Y %H:%M:%S %Z', time());

    echo date('l d-m-Y H:i:s T', time());

}
?>
Samstag  30-10-2021 07:26:39 EDT
Saturday 30-10-2021 07:26:39 EDT
```

Note: `date()` is only able to return month/day names in English. Also **date() and strftime() use different Formats.**

mktime() — Returns the Unix timestamp corresponding to the arguments given.

```
mktime(
    int $hour,
    ?int $minute = null,
    ?int $second = null,
    ?int $month = null,
    ?int $day = null,
    ?int $year = null
): int|false
```

Example 31:

```
<?php
echo  mktime(1, 2, 3, 4, 5, 2006);                          // 1144191723
echo date('d-m-Y H:i:s T', mktime(1, 2, 3, 4, 5, 2006));    // 05-04-2006 01:02:03 CEST

date_default_timezone_set('UTC');
echo date('d-m-Y H:i:s T', mktime(1, 2, 3, 4, 5, 2006));    // 05-04-2006 01:02:03 UTC
?>
```

Note: As shown in this example. `mktime()` will returns the Unix timestamp of the arguments given.

However if your PHP is set in a diferent timezone and you use the returned timestamp, PHP won't know that it is in UTC. You need to set the `date.timezone` to UTC to work correctly.

strtotime() — Parse about any English textual datetime description into a Unix timestamp

```
strtotime(string $datetime, ?int $baseTimestamp = null): int|false
```

Example 32:

```
<?php
echo strtotime("now");                                      // 1635587447
echo strtotime("10 September 2000");                        // 968536800
echo strtotime("+1 day");                                   // 1635677447
echo strtotime("+1 week");                                  // 1636195847
echo strtotime("+1 week 2 days 4 hours 2 seconds");         // 1636383049
echo strtotime("next Thursday");                            // 1635980400
?>
```

Example 33:

```php
<?php
date_default_timezone_set('UTC');
echo date('d-m-Y', strtotime("last Monday"));
echo date('l', strtotime("first day of December"));
?>
```

```
25-10-2021
Wednesday
```

```php
<?php
echo date('d-m-Y', strtotime("Christmas 2021"));
var_dump( strtotime("Christmas 2021"));
?>
```

```
01-01-1970
bool(false)
```

Note: `strtotime` will return false when if it is unsuccessful. This is equivalent to `int 0`.
You should always check for failer when using strtotime:

```php
if (strtotime('string') === false) {...}
```

date_parse() — Returns associative array with detailed info about given date/time

```php
date_parse(string $datetime): array
```

Example 34:

```php
<?php
print_r(date_parse("2006-12-12 10:00:00.5"));
?>
```

```
Array
(
    [year] => 2006
    [month] => 12
    [day] => 12
    [hour] => 10
    [minute] => 0
    [second] => 0
    [fraction] => 0.5
    [warning_count] => 0
    [warnings] => Array()
    [error_count] => 0
    [errors] => Array()
    [is_localtime] =>
)
```

PHP DateTime classes

PHP provides a set of date and time classes that allow you to work with the date and time in an object-oriented way.

`DateTime`, `DateTimeImmutable`, `DateTimeZone`, `DateInterval`, `DatePeriod` & `DateTimeInterface`

DateTime Class

```
class DateTime implements DateTimeInterface {

      public __construct(string $datetime = "now", ?DateTimeZone $timezone = null)
      ...
      ...
}
```

The valid formate for the `$datetime sting` are the same as the `strtotime()` function. And for the timezone is a `DateTimeZone object`.

Example 35:

```
<?php

$datetime = new DateTime();

var_export($datetime);

?>
```

```
DateTime::__set_state(array(
   'date' => '2021-11-01 08:42:20.431908',
   'timezone_type' => 3,
   'timezone' => 'Europe/Berlin',
))
```

Example 36:

```
<?php
$datetime = new DateTime('tomorrow 7:15pm', new DateTimeZone('Europe/London'));
var_export($datetime);
?>
```

```
DateTime::__set_state(array(
   'date' => '2021-11-02 19:15:00.000000',
   'timezone_type' => 3,
   'timezone' => 'Europe/London',
))
```

Note: Using '/' as date delimiter will be interpreted as USA date format that is mm/dd/yy example:

`new DateTime('12/08/21')` result: `..,'date' => '2021-12-08 00:00:00.000000', ..`

While using the '-' will be interpreted as EU date format

`new DateTime('12-08-21')` result: `..,'date' => '2021-08-12 00:00:00.000000', ..`

(this can be updated using the static `DateTime::createFromFormat(), will be covered later`)

Methods	Description
DateTime::setTimezone()	Sets the time zone for the DateTime object.
DateTime::setTime()	Sets the time.
DateTime::setDate()	Sets the date.
DateTime::setTimestamp()	Sets the date and time based on an Unix timestamp
DateTime::add()	Adds an amount of days, months, years, hours, minutes and seconds to a DateTime object.
DateTime::sub()	Subtracts an amount of days, months, years, hours, minutes and seconds from a DateTime object.

static DateTime::createFromFormat() -- date_create_from_format()
Parses a time string according to a specified format

DateTime::format() -- DateTimeImmutable::format() -- DateTimeInterface::format() -- date_format()
Returns date formatted according to given format.

Example 37:

```
<?php

$datetime = new DateTime();
echo $datetime->format('m/d/Y g:i A T');

$datetime->setTimezone(new DateTimeZone('America/New_York'));
echo $datetime->format('m/d/Y g:i A T');

?>
11/01/2021 9:38 AM CET
11/01/2021 4:38 AM EDT
```

The DateTimeInterface interface

DateTimeInterface was created so that parameter, return, or property type declarations may accept either DateTime or DateTimeImmutable as a value.

DateTime and DateTimeImmutable classes are the only classes that can implement

Methods	Description
DateTime::diff()	Returns the difference between two DateTime objects
DateTime::format()	Returns date formatted according to given format
DateTime::getOffset()	Returns the timezone offset from GMT / UTC
DateTime::getTimestamp()	Gets the Unix timestamp
DateTime::getTimezone()	Return DateTimeZone Object relative to given DateTime

DateTime::__wakeup()	The __wakeup handler

The DateTimeZone class

Representation of time zone.

Methods	Description
DateTimeZone::getLocation()	Returns location information for a timezone
DateTimeZone::getName()	Returns the name of the timezone
DateTimeZone::getOffset()	Returns the timezone offset from GMT / UTC
DateTimeZone::getTransitions()	Returns all transitions for the timezone
DateTimeZone::listAbbreviations()	Returns associative array containing dst, offset and the timezone name
DateTimeZone::listIdentifiers()	Returns a numerically indexed array containing all defined timezone identifiers

Example 38:

```
<?php

$dateTimeNY = new DateTime('now', new DateTimeZone('America/New_York'));

var_export($dateTimeNY->getTimezone());

var_export($dateTimeNY->getTimezone()->getLocation());

echo $dateTimeNY->getTimezone()->getName();

?>
```

```
DateTimeZone::__set_state(array(

   'timezone_type' => 3,
   'timezone' => 'America/New_York',
))

array (
  'country_code' => 'US',
  'latitude' => 40.71416,
  'longitude' => -74.00638,
  'comments' => 'Eastern (most areas)',
)

America/New_York
```

Example 39 (Returns the timezone offset from GMT in seconds):

```
<?php
$romeTimeZone = new DateTimeZone('Europe/Rome');

echo $romeTimeZone->getOffset(new DateTime('12-1-2021'));
echo $romeTimeZone->getOffset(new DateTime('12-8-2021'));
?>
```

```
3600
```

```
7200
```

Note: the new DayTime Object passed as argument is only to check if it is winter or summer daylight saving time. **Not the timezone**.

Example 40 (using **`getOffset()`** on a DateTime object):

```php
<?php
$winter = new DateTime('2010-12-21', new DateTimeZone('America/New_York'));
$summer = new DateTime('2008-06-21', new DateTimeZone('America/New_York'));

echo $winter->getOffset();
echo $summer->getOffset();
?>
```

```
-18000
-14400
```

Example 41 (`setTime(), setDate, and setDateTimeZone`):

```php
<?php
$dateTime = new DateTime( 'today 6:00am', new DateTimeZone('Europe/London'));


$dateTime->setTime(12, 30, 30)->setDate(1990, 01, 12);
$dateTime->setTimezone( new DateTimeZone('Asia/Taipei'));

echo $dateTime->format('l  dS F Y -- g:iA T'), PHP_EOL;

$dateTime->setTimezone(new DateTimeZone('Asia/Taipei'))->setTime(12, 30, 30)->setDate(1990, 01, 12);

echo $dateTime->format('l  dS F Y -- g:iA T'), PHP_EOL;
?>
```

```
Friday  12th January 1990 -- 8:30PM CST
Friday  12th January 1990 -- 12:30PM CST
```

Note: Take note of the order in which the methods has passed.
When changing the timezone the current time will alse be update relative to the difference in timezones.

DateTime::createFromFormat()

createFromFormat() is a static method on the DateTime class which returns a DateTime Object parsed from a given format. (used insted new DateTime).

```php
DateTime::createFromFormat( str $format, str $datetime,
                                        ?DateTimeZone $timezone = null ) DateTime|false
```

or you can use a Procedural function:

```
date_create_from_format (str $format, str $datetime,
                                        ?DateTimeZone $timezone = null): DateTime|false
```

Example 42:

```
<?php
$format = 'Y.m.d.H.i.s.u';
$date = '2000.01.30.08.45.30.999999';

$dateTime = DateTime::createFromFormat($format, $date);

var_export($dateTime);

echo $dateTime->format(DateTime::COOKIE);
?>
```

```
DateTime::__set_state(array(
   'date' => '2000-01-30 08:45:30.999999',
   'timezone_type' => 3,
   'timezone' => 'Europe/Berlin',
))

Sunday, 30-Jan-2000 08:45:30 CET
```

Note: When using `DateTime::createFromFormat()` if the date is supplied but time is not spacifed, the current time will be set. Example: `DateTime::createFromFormat('Y.m.d', '1990.1.12')`

However in `new DateTime` if the date is supplied but time is not spacifed it will be set to midnight.
Exemple: `new DateTime('12-1-1990');`

The **DateTime::COOKIE** is one of the constants inherited from the `DateTimeInterface interface`.

```
interface DateTimeInterface {

      // Constants
      const string ATOM = "Y-m-d\TH:i:sP";
      const string COOKIE = "l, d-M-Y H:i:s T";
      ...
      const string W3C = "Y-m-d\TH:i:sP";
```

Comparing DateTime

Example 43:

```
<?php
$dateTime1 = new DateTime('05/25/2021 9:15 AM');
$dateTime2 = new DateTime('05/25/2021 9:14 AM');

var_dump($dateTime1  <  $dateTime2);     bool(false)
var_dump($dateTime1  >  $dateTime2);     bool(true)
var_dump($dateTime1 ==  $dateTime2);     bool(false)
```

```
var_dump($dateTime1 <=> $dateTime2);      int(1)
?>
```

Example 44 (comparing TimeStamps):

```
<?php
echo $t1 = $dateTime1->getTimestamp();          1621926900
echo $t2 = $dateTime2->getTimestamp()           1621926840

echo $t1 - $t2 . 'seconds';                     60seconds
?>
```

Using the diff() method

The DateTimeInterface::diff() method allows you to compare the difference between two DateTime objects or DatetimeImmurable objects.
It returns a DateInterval object that contains the period of time between the two dates being represented.

Remember to take in account the time zone and daylight savings time conversions.

```
DateTimeInterface::diff(DateTimeInterface $targetObject, bool $absolute = false): DateInterval
```

Example 45:

```
<?php
$dateTime1 = new DateTime('12-8-1984 4:50 AM');
$dateTime2 = new DateTime('12-1-1990 9:14 AM');
?>
```

```
<?php
var_export($dateTime1->diff($dateTime2));
?>

DateInterval::__set_state(array(
   'y' => 5,
   'm' => 5,
   'd' => 0,
   'h' => 4,
   'i' => 24,
   's' => 0,
   'f' => 0.0,
   'weekday' => 0,
   'weekday_behavior' => 0,
   'first_last_day_of' => 0,
   'invert' => 0,
   'days' => 1979,
   'special_type' => 0,
   'special_amount' => 0,
   'have_weekday_relative' => 0,
   'have_special_relative' => 0,
))
```

```
<?php
var_export($dateTime2->diff($dateTime1));
?>

DateInterval::__set_state(array(
   'y' => 5,
   'm' => 5,
   'd' => 0,
   'h' => 4,
   'i' => 24,
   's' => 0,
   'f' => 0.0,
   'weekday' => 0,
   'weekday_behavior' => 0,
   'first_last_day_of' => 0,
   'invert' => 1,
   'days' => 1979,
   'special_type' => 0,
   'special_amount' => 0,
   'have_weekday_relative' => 0,
   'have_special_relative' => 0,
))
```

Note: In this example, the different method has subtract the `$datetime1` from the `$datetime2`.

If the differens is positive Example: `$today->diff($tomorrow)`, the invert = 0.
If the differens is negative Example: `$today->diff($testerday)`, the invert = 1.

The DateInterval class

A date interval stores either a fixed amount of time (in years, months, days, hours etc) or a relative time string in the format that `DateTime`'s constructor supports.

More specifically, the information in an object of the `DateInterval class` is an instruction to get from one date/time to another date/time. This process is not always reversible.

A common way to create a DateInterval object is by calculating the difference between two date/time objects through `DateTimeInterface::diff()`.

```
class DateInterval {

public int $y;    public int $m;    public int $d;    public int $h;
public int $i;    public int $s;    public float $f;  public int $invert;
public mixed $days;

public __construct(string $duration)
public static createFromDateString(string $datetime): DateInterval|false
public format(string $format): string
}
```

DateInterval::__construct() — Creates a new DateInterval object

The format of the `string $duration` starts with the letter P, for period. Each duration period is represented by an integer value followed by a period designator.

If the duration contains time elements, that portion of the specification is preceded by the letter T.

Period	Description
Y	years
M	months
D	days
W	weeks. These get converted into days, so can not be combined with D.
H	hours
M	minutes
S	seconds
P14D	14 days
P2W	Two weeks
P2W5D	This is invalid; you may not specify weeks and days together in one string; the weeks will be ignored
P2WT5H	Two weeks and five hours

DateInterval::format()

Formats the interval, similat to `DateTime::format()` however the format parameter string is different. Each format character must be prefixed by a percent sign (%), more similar to `strftime()`.

Example 46:

```
<?php
$interval = new DateInterval('P2Y4DT6H8M');
var_export($interval);

echo $interval->format('%y years, %d day, %h hours and %i minute');
?>
```

```
DateInterval::__set_state(array(
   'y' => 2,
   'm' => 0,
   'd' => 4,
   'h' => 6,
   'i' => 8,
   's' => 0,
   'f' => 0.0,
   'weekday' => 0,
   'weekday_behavior' => 0,
   'first_last_day_of' => 0,
   'invert' => 0,
   'days' => false,
   'special_type' => 0,
   'special_amount' => 0,
   'have_weekday_relative' => 0,
   'have_special_relative' => 0,
))
```

```
2 years,  4 day, 6 hours and 8 minute
```

Note: The days is set to `false`. This is the total number of days as a result of a `DateTime::diff()`. Compare this result to that of the previous example.

static DateInterval::createFromDateString

A static method that create a `DateInterval object` from a string, that is supported by the parser used for `strtotime()`.

Example 47 (Each set of intervals is equal):

```
$i = new DateInterval('P1Y1D');
$i = DateInterval::createFromDateString('1 year + 1 day');

$i = new DateInterval('P1DT12H');
$i = DateInterval::createFromDateString('1 day + 12 hours');

$i = new DateInterval('PT3600S');
$i = DateInterval::createFromDateString('3600 seconds');
```

Date Calculations

The DateTime class allows you to `add()` or `sub()` a `DateInterval object` from a `DateTime object`.

Example 48:

```
// This is just to find the day differace between Leah bday and Charlotte bday
<?php
$leah = new DateTime('04-04-2014');
$charlotte = new DateTime('29-11-2015');

echo $leah->diff($charlotte)->days;
?>
```

```
604
```

```
// add() method
<?php
echo $leah->add(new DateInterval('P604D'))->format(DateTime::COOKIE);
?>
```

```
Sunday, 29-Nov-2015 00:00:00 CET
```

```
// sub() method
<?php
echo $charlotte->sub(
                    DateInterval::createFromDateString('604 days')
      )->format(DateTime::COOKIE);

?>
```

```
Friday, 04-Apr-2014 00:00:00 CEST
```

Note: If the interval is set to 1. Example in:

```
$dateInterval = new DateInterval('P604D');     $dateInterval->interval = 1;
```

The `add()` and `sub()` would be inverted, `add()` would be subtracting and VS.

Also simple calculations can be performed using the `DateTime` class method `modify()`.

Example 49:

```
<?php
$date = new DateTime('2006-12-12');
$date->modify('+1 day');
echo $date->format('Y-m-d');
?>
```

```
2006-12-13
```

Warning: Beware when using add() sub() and modify() you are changing the original object.

Example50 (check final the value of example 48):

```php
<?php
var_export($leah);
?>
```

```
DateTime::__set_state(array(
   'date' => '2015-11-29 00:00:00.000000',
   'timezone_type' => 3,
   'timezone' => 'Europe/Berlin',
))
```

```php
<?php
var_export($charlotte);
?>
```

```
DateTime::__set_state(array(
   'date' => '2014-04-04 00:00:00.000000',
   'timezone_type' => 3,
   'timezone' => 'Europe/Berlin',
))
```

Example51 (also beware when adding or subtracting months):

```php
<?php
$date = new DateTime('2000-12-31');

$date->modify('+1 month');
echo $date->format('Y-m-d');

$date->modify('+1 month');
echo $date->format('Y-m-d');
?>
```

```
2001-01-31
2001-03-03
```

The DateTimeImmutable class

As we saw in the previous example, when adding subtracting or modify and object, it will affect the original object, even when assigning.

Example52 :

```php
<?php
$from = new DateTime();
$to   = $from->add(new DateInterval('P1D'));

echo 'FROM ' . $from->format('d-m-Y') . ' TO: ' . $to->format('d-m-Y');
?>
```

```
FROM 03-11-2021 TO: 03-11-2021
```

Example53 :

```php
<?php
$A = new DateTime();
$B = $A;

$B->add(new DateInterval('P1Y2M3D'));

echo $A->format('d-m-Y');          04-01-2023
echo $B->format('d-m-Y');          04-01-2023
?>
```

One way to deal with this is to use the clone keyword.

Example54 :

```php
<?php
$A = new DateTime();
$B = clone $A;

$B->add(new DateInterval('P1Y2M3D'));

echo $A->format('d-m-Y');          01-11-2021
echo $B->format('d-m-Y');          04-01-2023
?>
```

However PHP provid us with the **DateTimeImmutable** class:

```php
class DateTimeImmutable implements DateTimeInterface {

      public __construct(string $datetime = "now", ?DateTimeZone $timezone = null)
      ...
      ...
}
```

The `DateTimeImmutable object` is almost the same as `DateTime object` except when using any of its modification methods, they will return a new `DateTimeImmutable instance` (that need to be assigned).

While the original object is immutable, (retain its values).

Example55 :

```php
<?php
$A = new DateTimeImmutable('noon', new DateTimeZone('Europe/London'));

$B = $A->setDate(1984, 8, 12);
$C = $A->modify('-10 years');
$D = $A->setTimezone(new DateTimeZone('America/New_York'));

echo $A->format(DateTime::COOKIE);       Tuesday,   02-Nov-2021 12:00:00 GMT
echo $B->format(DateTime::COOKIE);       Sunday,    12-Aug-1984 12:00:00 BST
echo $C->format(DateTime::COOKIE);       Wednesday, 02-Nov-2011 12:00:00 GMT
echo $D->format(DateTime::COOKIE);       Tuesday,   02-Nov-2021 08:00:00 EDT
?>

<?php
$A->add(new DateInterval('PT30M'));
$E = $A->add(new DateInterval('PT30M'));

echo $A->format(DateTime::COOKIE);       Tuesday,   02-Nov-2021 12:00:00 GMT
echo $E->format(DateTime::COOKIE);       Tuesday,   02-Nov-2021 12:30:00 GMT
?>
```

The DatePeriod class

A date period allows iteration over a set of dates and times, recurring at regular intervals, over a given period.

```
class DatePeriod implements Traversable {

  // Constants
  const int EXCLUDING_START_DATE = 1;

  // Properties
  public int $recurrences;
  public bool $unclude_start_date;
  public DateTimeInterface $start;
  public DateTimeInterface $current;
  public DateTimeInterface $end;
  public DateInterval $interval;


  // Methods
  public __construct(
    DateTimeInterface $start, DateInterval $interval, int $recurrences, int $options = 0 )

  public __construct(

    DateTimeInterface $start, DateInterval $interval,DateTimeInterface $end, int $options = 0 )

  ...
   }
```

Example55 (method 1 - using recurrences) :

```
<?php
$startDate = new DateTime('12th January 1900', new DateTimeZone('Europe/Rome'));

$interval  = new DateInterval('P1M');

$period = new DatePeriod($startDate, $interval, 3);

foreach($period as $datetime) {
    echo $datetime->format(DateTime::COOKIE), PHP_EOL;
}
?>
```

```
Friday,   12-Jan-1900 00:00:00 CET
Monday,   12-Feb-1900 00:00:00 CET
Monday,   12-Mar-1900 00:00:00 CET
Thursday, 12-Apr-1900 00:00:00 CET
```

Example56 (method 1 - using recurrences and excluding start date) :

```php
<?php
$startDate = new DateTime('12th January 1900', new DateTimeZone('Europe/Rome'));

$interval  = new DateInterval('P1M');

$period = new DatePeriod($startDate, $interval, 3, DatePeriod::EXCLUDE_START_DATE );

foreach($period as $datetime) {
    echo $datetime->format(DateTime::COOKIE), PHP_EOL;
}
?>
```

```
Monday,   12-Feb-1900 00:00:00 CET
Monday,   12-Mar-1900 00:00:00 CET
Thursday, 12-Apr-1900 00:00:00 CET
```

Example57 (method 2 – using enddate) :

```php
<?php
$startDate    = new DateTime('12th August  1984');
$endDate      = new DateTime('12th January 1990');
$timeInterval = new DateInterval('P1Y');

$period = new DatePeriod($startDate, $timeInterval, $endDate);

foreach($period as $datetime) {
    echo $datetime->format(DateTime::COOKIE), PHP_EOL;
}
?>
```

```
Sunday, 12-Aug-1984 00:00:00 CEST
Monday, 12-Aug-1985 00:00:00 CEST
Tuesday, 12-Aug-1986 00:00:00 CEST
Wednesday, 12-Aug-1987 00:00:00 CEST
Friday, 12-Aug-1988 00:00:00 CEST
Saturday, 12-Aug-1989 00:00:00 CEST
```

Corresponding procedural functions

Most of the methods we cover has a procedural Alias function. Below is a short list of some of them:

Function	Alias of
date_create()	DateTime::__construct()
date_create_format()	DateTime::createFromFormat()
date_timezone_set()	DateTime::setTimezone()
date_timezone_get()	DateTime::getTimezone()
date_time_set()	DateTime::setTime()
date_date_set()	DateTime::setDate()
date_modify()	DateTime::modify()
date_offset_get()	DateTime::getOffset()

PHP SPL Data Structures

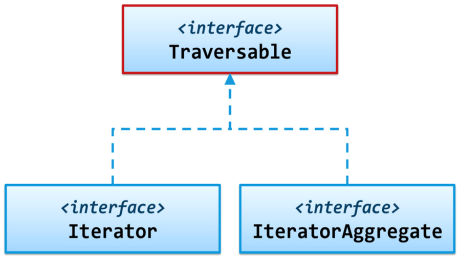
The standard PHP library (SPL) is a collection of interfaces and classes that are meant to solve common problems. It includes several classes that help you work with standard data structures.

(In Chapter 5 we have cover most of these in greater detaile)

Traversable interface

The Traversable interface used to detect if a class is traversable, can be looped over using foreach().

It can not be used alone. Instead it must be implemented by either IteratorAggregate or Iterator



Iterator interface

This interface extends from the Traversable interface and contain the following 5 methods that need to be implemented:

Method	Purpose
current()	Returns the current element
key()	Returns the key of the current element
next()	Moves forward to next element
rewind()	Rewinds the iterator to the first element
valid()	Checks if current position is valid

```php
<?php
class hillstations implements Iterator {
    private $places = [];
    private $count = 0;
    private $index = 0;

   public function current(){
      echo "Current called---";
      return $this->places[$this->index];
   }
   public function next() {
      echo "Next Called---";
      $this->index++;
   }
   ...
   ...
```

https://webmobtuts.com/backend-development/php-iterators-part-1-traversable-iterable-iterator-and-iterator-aggregate/

IteratorAggregate interface

the IteratorAggregate interface like the iterator interface, extends from traversable but unlike the Iterator interface it has only one method getIterator():

Method	Purpose
getIterator()	Retrieve an external iterator

```php
<?php

class myData implements IteratorAggregate {
      public $places = ["USA", "London", "Berlin", "Paris"];

      public function getIterator() {
            return new ArrayIterator($this->places);
      }
}
?>
```

Interface that Extends Iterator Interface	
OuterIterator	`getInnerIterator()` which is like: `IteratorAggregate::getIterator()`. (So this interface is like a combo of both interfaces). This itrator is implemented by the SPL class `IteratorIterator`
RecursiveIterator	Classes implementing RecursiveIterator can be used to iterate over iterators recursively. `getChildren()`— Returns an iterator for the current entry `hasChildren()` — Returns if an iterator can be created for the current entry
SeekableIterator	`seek()` — Seeks to a given position in the iterator.

ArrayAccess interface

This interface provides the ability to access objects as arrays. To do so, you need to implement four methods:

Method	Purpose
offsetExists	Whether an offset exists
offsetGet	Offset to retrieve
offsetSet	Assign a value to the specified offset
offsetUnset	Unset an offset

```
class Foo implements ArrayAccess {
    protected $_data=array();

    public function offsetExists ($offset){
        return array_key_exists($offset, $this->_data);
    }

    public function offsetGet ($offset) {
        return $this->_data[$offset];
    }

    public function offsetSet ($offset, $value) {
        $this->_data[$offset] = $value;
    }

    public function offsetUnset ($offset) {
        unset($this->_data[$offset]);
    }

}
```

Countable interface

If your class implements the Countable interface, you will be able to use the count() function to find how many elements it has.

The Countable interface has an abstract method called count.

This method will becalled when you call the PHP function count() on an object instantiated from a class thatimplements the interface.

```
<?php
class myCounter implements Countable {
    private $count = 0;

    public function count() {
        return ++$this->count;
    }
}
```

Lists data structures

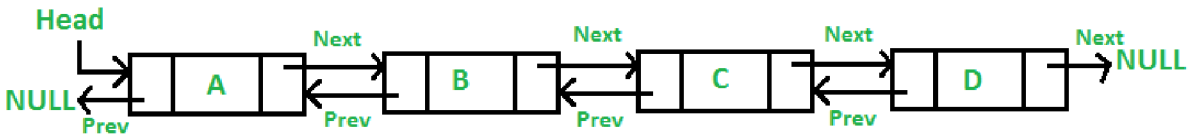
A list is an ordered collection of elements. The same value may appear more than once in a list.

Doubly Linked List

A Doubly Linked List is a list that contains links to both next and previous nodes.

A Doubly Linked List (DLL) contains an extra pointer, typically called previous pointer, together with next pointer and data which are there in singly linked list.

Unlike singly linked lists where traversal is only one way, doubly linked lists allow traversals in both ways.



SplDoublyLinkedList Class

The `SplDoublyLinkedList` class is a PHP Library class which provides the main functionalities of a doubly linked list in PHP.

SplDoublyLinkedList class implements the Iterator, ArrayAccess, Countable and Serializable interfaces.

In addition, it implements methods that let you change the iterator behavior as well as add or remove items to the front or back of the list:

```
$list_name->push(value);
$list_name->unshift(value);
```

```
$list_name->pop();
$list_name->shift();
```

`$list_name->top();`	This will return the last node from the list list_name.
`$list_name->bottom();`	This will return the node present at the beginning of the list list_name.
`$list_name->add(index, value);`	This will add the value at position index in the list list_name.
`$list_name->isEmpty();`	This function returns True if doubly linked list is empty.

Constants and Modes

`SplDoublyLinkedList::setIteratorMode` — Sets the mode of iteration

There are two orthogonal sets of modes that can be set:

The direction of the iteration (either one or the other):

`SplDoublyLinkedList::IT_MODE_LIFO`	(Stack style) – last in first out
`SplDoublyLinkedList::IT_MODE_FIFO`	(Queue style) – first in first out

The behavior of the iterator (either one or the other):

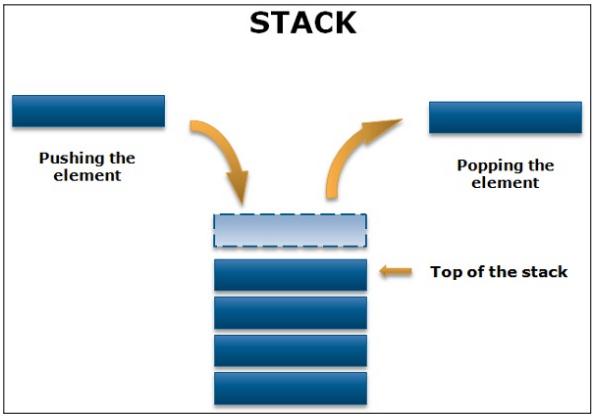
`SplDoublyLinkedList::IT_MODE_DELETE`	(Elements are deleted by the iterator)
`SplDoublyLinkedList::IT_MODE_KEEP`	(Elements are traversed by the iterator)

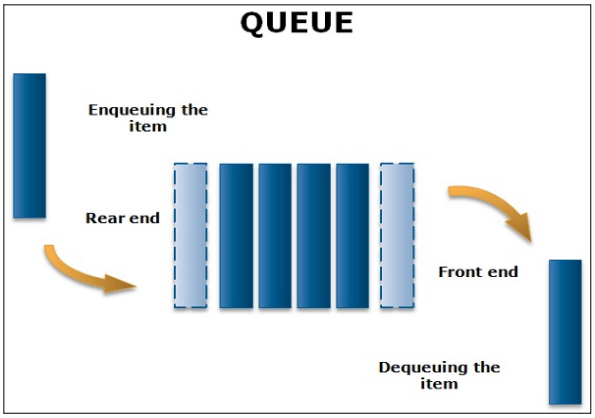
The default mode is: `SplDoublyLinkedList::IT_MODE_FIFO` | `SplDoublyLinkedList::IT_MODE_KEEP`

```
const int IT_MODE_LIFO = 2;
const int IT_MODE_FIFO = 0;
const int IT_MODE_DELETE = 1;
const int IT_MODE_KEEP = 0;
```

Stack and Queur

Stack and Queur are linear data structure that follows the specific order to perform insertion and deletion operations.





A **Stack** follows the principle LIFO (**Last In First Out)** in which the insertion and deletion take place from one side known as a top.

The two operations are performed in LIFO are, push & pop (add to end, take from end).

A **Queue** is an ordered list that follows the principle FIFO (**First In -First Out)**. In the case of Queue, insertion is performed from one end, and that end is known as a rear end.

The two operations are performed in LFIFO are, push & shift (add to end, take from beginning).

Note: SPLDoublyLinkedList combines the functionality of both the queue and the stack.

SplStack Class & SplQueue Class

In PHP we have `SplStack` and `SplQueue` classes. Both of which inherit from the `SplDoublyLinkedList`.

In `SplDoublyLinkedList` Class we saw that we have two orthogonal sets of modes that can be set by:

```
public SplDoublyLinkedList::setIteratorMode(int $mode): int
```

The two modes are:

- the direction of the iteration
- and the behavior of the iterator

Now in both the `SplStack` and `SplQueue` classes, there is only **one mode we can change, the behavior**.

We can either set it to : `IT_MODE_DELETE (Elements are deleted by the iterator)`

(default for all classes) or to: `IT_MODE_KEEP (Elements are traversed by the iterator)`

The direction of iteration can not be changed for `SplStack` and `SplQueues` classes:

In `SplStack` it is set to `SplDoublyLinkedList::IT_MODE_LIFO`

and in `SplQueue` it is set to `SplDoublyLinkedList::IT_MODE_FIFO`

Note: SplQueue class has two additional methoded which are alias of other methods:

`SplQueue::enqueue()` alias of `SplDoublyLinkedList::push()`

`SplQueue::dequeue()` alias of `SplDoublyLinkedList::shift()`

Example60 (SplDoublyLinkedList):

```php
<?php
// Created my class from SplDoublyLinkedList & added a method to diplay the list
class MySplDoublyLinkedList extends SplDoublyLinkedList {

    public function displayData () {
        foreach ($this as $k => $v) {echo "$k => $v \n";}
        echo "\n"; }
}
$linkedList = new MySplDoublyLinkedList;

var_export($linkedList->isEmpty());      // true
?>

<?php
$linkedList->push('A');
$linkedList->push('B');
$linkedList->push('C');

$linkedList->displayData();
?>
0 => A
1 => B
2 => C

<?php
$linkedList->unshift('Z');
$linkedList->add(2, 2);

$linkedList->displayData();
?>
0 => Z
1 => A
2 => 2
3 => B
4 => C

<?php
$linkedList->shift();
$linkedList->pop();

$linkedList->displayData();
?>
0 => A
1 => 2
2 => B

<?php
echo $linkedList->top();                 // B
echo $linkedList->bottom();              // A

var_export($linkedList->isEmpty());      // false
?>
```

Example61 (SplStack):

```php
<?php
// Extending from SplStack and added a diplay method
class MySplStack extends SplStack{

    public function displayData () {
        foreach ($this as $k => $v) {echo "$k => $v \n";}
        echo "\n"; }
}

$stackList = new MySplStack;

$stackList->push('A');
$stackList->push('B');
$stackList->push('C');
$stackList->push('D');
$stackList->push('E');

$stackList->displayData();
?>
```

```
4 => E
3 => D
2 => C
1 => B
0 => A
```

```php
<?php
$stackList->pop();
$stackList->pop();
$stackList->displayData();
?>
```

```
2 => C                      <- Note Last In First Out
1 => B
0 => A
```

Note: Since **SplStack** extends from SqlDoublyLinkedList it has also inherted some method like add unshift & shift. However since it's a Stack DataStructure our add & subtract method should be limeted to **push** & **pop**.

```php
// Do not do:
<?php
$stackList->add(2, 2);
$stackList->add(2, 2);
$stackList->displayData();
?>
```

```
4 => C
3 => B
2 => A
1 => 2
0 => 2
```

```php
<?php
$stackList->unshift('3');
$stackList->unshift('3');
$stackList->shift();
$stackList->displayData();
?>
```

```
5 => C
4 => B
3 => A
2 => 2
1 => 2
0 => 3
```

Example62 (SplQueue)

```php
<?php
// As previous examples we are extending from SplQueue and added a diplay method
class MySplQueue extends SplQueue {

    public function displayData () {
        foreach ($this as $k => $v) {echo "$k => $v \n";}
        echo "\n"; }
}

$QueueList = new MySplQueue;
?>

<?php
echo PHP_EOL, PHP_EOL;

$QueueList->push('A');
$QueueList->push('B');
$QueueList->enqueue('C');
$QueueList->enqueue('D');
$QueueList->enqueue('E');

$QueueList->displayData();
?>
```

```
0 => A
1 => B
2 => C
3 => D
4 => E
```

```php
<?php
$QueueList->shift();
$QueueList->dequeue();
$QueueList->dequeue();
$QueueList->displayData();
?>
```

```
0 => D                  <- Note First In First Out
1 => E
```

Note: A explained in the previous example.

Since we are creating a **Queue DataStructure** object our add & subtract method should be limeted to push & shift or there alias **enqueue** & **dequeue**.

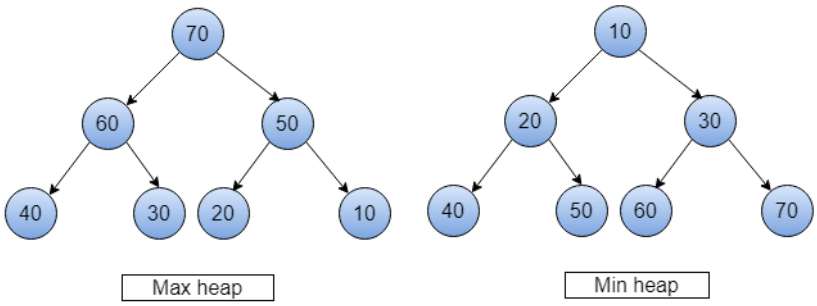
Heaps data structures

Heaps are specialized tree-like data structures which satisfy the heap property – the node value (key) of any parent is always ordered with respect to its child node values across the entire tree.

There are several heap variants. For instance:

If parent keys are ordered such that they are of equal or greater value than their children, and the highest key is at the root, the heap is said to be a **maxheap**.

If parent keys are ordered such that they are of equal or lower value than their children, with the lowest key at the root, the heap is called a **minheap.**



Hearps in PHP

PHP provides with four SPL Classes:

an abstract class `SplHeap`

a `SplMaxHeap` class and a `SplMinHeap` class (which extends `SplHeap`)

and a specialized class `SplPriorityQueue`

The abstract class SplHeap

The SplHeap class provides the main functionalities of a Heap. It implements the Iterator, Countable interfaces.

```
abstract class SplHeap implements Iterator, Countable {
      public __construct()
      abstract protected compare(mixed $value1, mixed $value2): int
      public top(): mixed
      public extract(): mixed
      public insert(mixed $value): bool
      public isCorrupted(): bool
      public isEmpty(): bool
      public recoverFromCorruption(): bool
      public count(): int
      public rewind(): void
      public current(): mixed
      public key(): int
      public next(): void
      public valid(): bool
}
```

Note: The `next()` method will also delete the top node of the heap.

Example58 :

If you dump the heap structure you'll notice that the items aren't in any particular order. However, if you loop through the heap, all nodes will be ordered:

```php
<?php
class MyMaxHeap extends SplHeap {
    protected function compare($value1, $value2) {
        return $value1 <=> $value2;
    }
}
$list = [10, 2, 12, 42, 5, 100, 99, 101, 17, -1, 7];
$max = new MyMaxHeap;

foreach ($list as $item) {
    $max->insert($item);
}
print_r($max);
?>
```

```
MyMaxHeap Object(
    [flags:SplHeap:private] => 0
    [isCorrupted:SplHeap:private] =>
    [heap:SplHeap:private] => Array (
            [0] => 101
            [1] => 100
            [2] => 99
            [3] => 17
            [4] => 7
            [5] => 10
            [6] => 42
            [7] => 2
            [8] => 12
            [9] => -1
            [10] => 5
        )
)
```

```php
<?php
while($max->current()) {
    echo $max->current()
    $max->next();
}
var_dump($max->isEmpty());
?>
```

```
100
99
42
17
12
10
7
5
2
-1
bool(true)
```

The compare method can perform any arbitrary comparison, as long as it returns:

In the case of a **MaxHeap**:

A positive integer if `$value1` is greater than `$value2`, 0 if they are equal, or a negative integer otherwise.

If extending **MinHeap**:

It should instead return a positive integer if `$value1` is less than `$value2`.

It's not recommended to have multiple elements with the same value in a heap as they may end up in an arbitrary relative position

Example59 (alternative looping) :

```php
<?php
while (!$harp->isEmpty()) {
    echo $harp->extract();
}
?>
```

```php
<?php
for ($harp->top(); $max->valid(); $harp->next()) {
    echo $harp->current();
}
?>
```

Note: `SplMinHeap` and `SplMaxHeap` are just classes that extend `SplHeap` and implement the `compare()` to provide directional sorting. So if extended or use directly the `compare()` is already defined.

SplPriorityQueue

The SplPriorityQueue class provides the main functionalities of a prioritized queue, implemented using a max heap.

In short, when using the insert method, a second argument is required, proprity. Data in the SplPriorityQueue is ordered as according the value of priority using the structure of a Max Heap.

```php
public SplPriorityQueue::insert(mixed $value, mixed $priority): bool
```

Example60:

```php
<?php
$objPQ = new SplPriorityQueue;
$objPQ->insert('A', 1);
$objPQ->insert('B', 4);
$objPQ->insert('C', 2);
$objPQ->insert('D', 3);
$objPQ->insert('E', 0);
$objPQ->insert('F', 0);
$objPQ->insert('G', 0);
$objPQ->insert('X', 0);

echo $objPQ->count();
echo $objPQ->top();
?>
```

8
B

```php
<?php
//mode of extraction (to display array containing both data and proprity)
$objPQ->setExtractFlags(SplPriorityQueue::EXTR_BOTH);

while($objPQ->valid()){
    var_export($objPQ->current());
    echo '<br>';
    $objPQ->next();
}
?>
```

```
array ( 'data' => 'B', 'priority' => 4, )
array ( 'data' => 'D', 'priority' => 3, )
array ( 'data' => 'C', 'priority' => 2, )
array ( 'data' => 'A', 'priority' => 1, )
array ( 'data' => 'E', 'priority' => 0, )
array ( 'data' => 'X', 'priority' => 0, )
array ( 'data' => 'G', 'priority' => 0, )
array ( 'data' => 'F', 'priority' => 0, )
```

Arrays Data Stractures (already covered with example in Chapter 5)

The SplFixedArray class provides the main functionalities of array.

The main differences between a SplFixedArray and a normal PHP array is that the SplFixedArray is of fixed length and allows only integers within the range as indexes.

The advantage is that it uses less memory than a standard array and is faster than ordinary array.

Maps Data Stractures (already covered with example in Chapter 5)

The SplObjectStorage class is a data structure object container in the SPL standard library, used to store a set of objects like in an array.

The objects it stored are unique and can be map to data (info).

A unique identifier called hash is assigned to each contained object.

The PHP SPL SplObjectStorage class implements four interfaces: Countable, Iterator, Serializable and ArrayAccess. It can realize statistics, iteration, serialization, array access and other functions.

Summary of SPL Data Structures Classes	
SplHeap	A heap is a tree collection where the children of a parent must always have a value lower than their parent. There are different types of heap.
SplMaxHeap	This is a type of heap where the maximum is kept at the top of the heap.
SplMinHeap	In this type of heap, the minimum is kept at the top.
SplPriorityQueue	This is a queue where each element also has a "priority" associated with it. An example of a use-case is bandwidth management wherein traffic of a certain type has a higher precedence over other traffic.
SplFixedArray	This is a faster implementation of an array, but it limits you to using an array of fixed l ength that only contains integers.
SplObjectStorage	This class provides a convenient way to map objects and their data.

set_time_limit – Limits the maximum execution time()