User-defined functions & date + time handling

This week:

- User-defined functions
- Handling dates and time in PHP

PHP user-defined functions

PHP Functions: User-defined functions

In PHP, defining your own functions is done through the use of the function keyword.

This keyword should be immediately followed by the function's signature, itself immediately followed by a curly brace bounded block of code that contains the function's statements.

The function signature is written in the following format:

<identifier>(<optionally typed, comma-separated, list of parameters>) : <return type>

Example:

```
function my_function(int $int_param, string $string_param, mixed $optional_param = null) : bool {
    // Insert here a series of statements that ultimately return a bool
}
```

PHP Functions: User-defined functions (continued)

PHP functions have many possibilities and functionalities that go beyond basic simple functions; particularly with respect to the parameters. Here's a few examples.

Optional parameters with default values:

PHP supports defining parameters that are optional, which means that when called, it is not required to pass an argument for these parameters. These are defined by assigning a default value to them in the function signature. Optional parameters must always be declared after all the required ones in the function signature.

```
function my_function(mixed $required_param, mixed $optional_param = null) : void {
    // statements
}
```

Calling my_function($value) works just fine. $required_param will take the value of $value and $optional_param will take the value null during that function call.

PHP Functions: User-defined functions (continued)

Passed-by-reference parameters:

By default PHP passes arguments to parameters by value (while still true, object arguments behave like passing-by reference because the values of objects are references). This means that a copy of the value will be created and assigned to the parameters during the function call. If you want a function that actually modifies the original passed variable, you need to use the reference operator (‘&’) before the parameter name.

```
function my_function(array $array_ref) : void {
        // statements. The passed array argument will be copied into $array_ref. Any changes made to $array_ref
        // will not apply outside of the function block.
}
function my_function_by_ref(array &$array_ref) : void {
        // statements. Because of the use of the ‘&’, The reference of the array argument will be copied into $array_ref. This
        // makes the $array_ref an alias for the original argument and any changes made to it will apply outside of the
        // function block.
}
```

PHP Functions: User-defined functions (continued)

Variadic functions:

Variadic functions are functions that can accept an unspecified number of arguments (any number of them). This is done by adding a special parameter that uses the splat (or “pack”) operator (‘...’) at the end of the parameter list. The function will “pack” arguments passed in that parameter spot and any subsequent undefined ones in an array that will be assigned to the parameter. The splat operator can also be used to “unpack” an array into values in a function call.

```
function my_variadic_function(int ...$integers) : void {
        foreach($integers as $an_int) { echo $an_int; }
}
```

Calling my_variadic_function(1, 2, 3, 4) works fine and will output each integer sequentially.

```
$int_array = [10, 9, 8, 7, 6, 5];
```

Calling my_variadic_function(...$int_array) works fine even if the type of the parameter is int, because we unpack the array argument in the function call; this makes the call equivalent to calling my_variadic_function(10, 9, 8, 7, 6, 5).

PHP Functions: User-defined functions (continued)

Union-type parameters and return type:

PHP supports union types as parameter types and return value types. In essence, union types allows you to define multiple acceptable types for a parameter/return value more precisely than by using the mixed type that accepts anything. This is done by defining the type as a listing the acceptable types each separated by the bitwise OR operator (‘|’). the function will then accept values of either of the listed types.

```
function union_type_function(int|array $integer_or_array) : string|bool {
        // can receive either an array or an int as the argument for $integer_or_array and must return either a
        // string or a bool
}
```

PHP Functions: User-defined functions (continued)

Recursive functions:

Recursive functions are functions that call themselves inside their block of instructions. As such, unless limited in some way, they can form infinite loops, and when properly limited, can behave like other loop-like structures. However, since recursive functions use nested function calls, they consume much more memory than other loop-like structures, since none of the calls will terminate until the whole loop completes.

Example:

```
function recursive() : void {
        if (<someCondition>) {
                recursive();
        }
}
```

PHP Functions: User-defined functions (continued)

Anonymous functions:

If functions can be defined with a name (identifier), they can also be used without, as *anonymous functions*. These can be used as callable arguments for other functions that require some, or can be stored in variables for later use.

Anonymous functions are also called ***inline functions***, ***lambda expressions*** or ***closures***.

Example:

```
$function = function() : void {
        // function body
}
```

PHP Date and Time

PHP Date & Time – Timestamps & date functions

Because PHP is not fundamentally an object oriented language, its original way to handle dates and times is through specifically formatted date strings, UNIX ***timestamps*** (integers representing the number of seconds elapsed since January 1st 1970, 00:00:00 GMT) and functions to create and manipulate those.

The most basic date and time function in PHP is time() : int ; it returns an int representing the current timestamp of the server. Note that timestamps do not contain any timezone information; this is something that we need to take into account when dealing with worldwide-reaching applications.

The date(string $format, ?int $timestamp = null) : string function allows us to convert timestamps into date strings of practically any format; the $format string parameter accepts a string composed of specific markers that will define the format of the output date string. You can find the details and list of markers and their effects on the date() manual page and the DateTimeInterface::format() manual page. The $timestamp parameter is optional, it accepts timestamp integer arguments for the date and time value to format; if no value or null is given, then the date() function will use the current date and time of the server.

PHP Date & Time – Timestamps & date functions (continued)

There are predefined format string available (as PHP constants) that can simplify formatting dates according to international standards:

- DATE_ATOM: Atomic clock style (2013-04-12T15:52:01+00:00)
- DATE_ISO8601: ISO 8601 standard format (2013-04-12T15:52:01+0000)
- DATE_RFC3339_EXTENDED: RFC 3339 Extended format (PHP>7.0) (2019-01-18T16:34:01.000+00:00)
- DATE_W3C: World Wide Web Consortium (2019-01-18T14:13:03+00:00)

It is also possible to create a timestamp from ***some*** date strings by using the strtotime(string $datetime, ?int $baseTimestamp = null) : int|false function. It is able to parse some date strings (mostly from English-language usual formats) into timestamps.

PHP Date & Time – DateTime objects

See the DateTime class for details

Because handling date and time through strings and timestamps can be quite complicated, PHP includes, since version 5.2.0, the DateTime class and other affiliated classes.

The DateTime class allows much easier creation and manipulation of dates and times; its instances represent a certain point in time (date and time) and support specifying time zones as well as modifying the specified point in time. You can create DateTime objects through the class constructor or through certain static methods of the class:

- new DateTime(string $datetime = "now", ?DateTimeZone $timezone = null)
- DateTime::createFromFormat(string $format, string $datetime, ?DateTimeZone $timezone = null)
- DateTime::createFromImmutable(DateTimeImmutable $object)
- DateTime::createFromInterface(DateTimeInterface $object)

PHP Date & Time – DateTime objects (continued)

DateTime objects have the added ability, compared to strings and timestamp, to be manipulated and modified, notably by adding or subtracting time periods to them in the form of DateInterval objects.

The object methods add(DateInterval $interval) : DateTime and sub(DateInterval $interval) : DateTime allow us to respectively add or subtract a time interval from a DateTime object, returning the modified object.

Similarly, the methods getTimezone() : DateTimeZone|false and setTimezone(DateTimeZone $timezone) : DateTime allow us to handle changing the time zone (in the form of a DateTimeZone object) of the associated DateTime object.

Finally, just as timestamps can be formatted into strings, DateTime objects can be exported as strings also, by the use of the format(string $format) : string method. The string $format argument works the same way as the date() function’s.

Next week

- PHP and the web: creating web applications