Server-side technologies (TCS)

Class 9

Functions

We priorly covered functions in other courses, so we already know they are really helpful and consist into subprograms within the main program performing a single task.

Creating a function

```
function helloWorld() {                    // Creates the function
        echo "Hello world!";
}
helloWorld();                              // Calls the function
                                           // and outputs to screen
```

Note

Functions name are NOT case sensitive.

Function arguments

Information can be passed from arguments to functions. An argument acts just like a variable and are defined within the function's parenthesis. Several arguments can be added and then need to be separated by a comma.

Syntax

```
function myFunction($myArgument) {
        echo "My argument is: $myArgument<br>";
}
myFunction("Hello World");
```

Explanation

The argument ***$myArgument*** has been specified within the function's parenthesis.

The instruction of the function is to output to screen the string *My argument is:* followed by the argument's value.

Above, the value passed to the function is a string (Hello World), but this could have been a variable.

Example:

```
<?php
function carsList($cars) {
        echo "$cars is a nice car<br>";
}
carsList("BMW");
carsList("Ferrari");
carsList("Mercedes");
carsList("Audi");
?>
```

Result:

BMW is a nice car
Ferrari is a nice car
Mercedes is a nice car
Audi is a nice car

Example with multiple arguments:

```
<?php
function carsList($car, $year) {
        echo "My car is a $year $car<br>";
}
carsList("BMW", "1984");
carsList("Ferrari", "1997");
carsList("Mercedes", "2010");
carsList("Lada", "1980");
?>
```

Result:

My car is a BMW 1984
My car is a Ferrari 1997
My car is a Mercedes 2010
My car is a Lada 1980

Using default arguments in functions

In the following example, a default parameter is used ($unknown) to be used if no argument is passed.

Example:

```
<?php
function carsList($unknown="Unspecified car") {
        echo "My car is a: $unknown<br>";
}
carsList("BMW");
carsList("");
carsList("Mercedes");
carsList("Audi");
?>
```

Result:

BMW is a: BMW
Ferrari is a: **Unspecified car**
Mercedes is a: Mercedes
Audi is a: Audi

Returning values in functions

Just like with JavaScript, the statement ***return*** is used to return values.

Example:

```
<?php
function addition($a, $b) {
        $c = $a + $b;
        return $c;
}
echo "1 + 2 = " . addition(1, 2) . "<br>";
echo "2 + 2 = " . addition(2, 2) . "<br>";
echo "3 + 2 = " . addition(3, 2) .  "<br>";
?>
```

Result:

1 + 2 = 3
2 + 2 = 4
3 + 2 = 5

Classes and objects

Classes and functions may seem alike at first glance, although, there are many differences between them. One of the main differences is that a class contains both variables (*called* ***properties***) and functions (*called* ***methods***) that form an entity called ***object***.

A class is a representation of a type of object. You can think of it as a template for the objects that will be created. Several objects (*called* ***instances***) made of real data can be created based on a class.

Creating a class

To create a class, first used the keyword class followed by the class name. You can then add properties (variables) to it.

Syntax:

```
class Person {
        $age;
        $gender;
        $city = "Montreal";        // Declared and assigned default value
}
```

Creating an object

To create a an object which is an instance of a class, the keyword ***new*** must be placed before the name of the object.

Example:

```
$person1 = new Person;
```

Calling an object's method

In order to perform different actions related to object created, a method (function) needs to have been created and then called in relation to a specific object using the operator **->**.

Example:

```
<?php
class Person {                                  // Creates new class
        public $age;                            // Object's properties
        private $_city = "Montreal";
        private $_experience = 0;

        public function speak() {               // Creates function
                echo "I am a person!";
        }
}
$person1 = new Person;                          // Creates new object
$person1->speak();                  // Runs function for the object
```

Properties and methods visibility:

Before every classes' properties, a visibility attribute can be placed. In the above example, you already have noticed ***private*** which means the property can't be modified and is accessible only within the objects created by the class.

Using ***public*** attribute would make it possible to modify the property and to use it anywhere outside of the objects created by the class. This is exactly like the global and local scopes of variables.

In the above example, trying to output the properties *$_city* or *$_experience* would generate a fatal error as the properties can't be used outside of the object. Although, *$_age* could be output without generating an error because it is using the ***public*** attribute.

The visibility is applied the same way to functions.

Note

In respect of *PEAR coding standards*, classes' name starts with a capital letter and private properties with an underscore.

https://pear.php.net/

Accessing objects' properties

The same way it is possible to call a function for an object using the operator **->**, we can use the same operator to access an object's property.

Example:

```
class Person {
        private $_age;
        private $_city = "Montreal";
        private $_experience = 0;
}
$person1 = new Person;
$perso1->_experience = $perso1->_experience + 1;
```

Explanation:

Using the **->** operator, we first access the new object's ***_experience*** property. Using the equal sign (=), we reassign a value by incrementing the default value by 1.

BUT this would generate a fatal error since the property visibility is set to ***private***. But there is a way around...

The pseudo-variable $this

The pseudo variable $this is used to refer to the current object. Although it is impossible to invoke a private property, you can do it by usign ***$this***.

Example:

```
class Person {
        private $_experience = 0;
        public function showExperience() {
                echo $this->_experience;
        }
        public function addExperience() {
                $this->_experience = $this->_experience + 1;
        }
}
$person1 = new Person;
$person1->addExperience();
$person1->showExperience();
}
```