Functions

Intro PHP

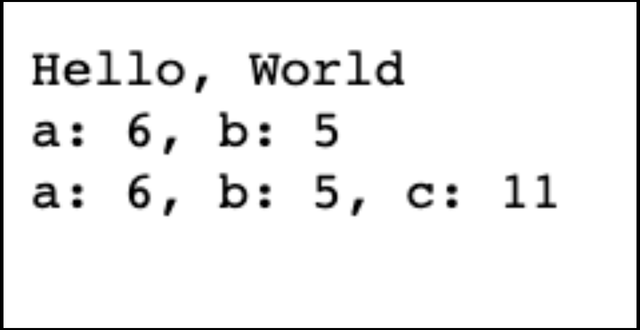
Download sample code from Nexus

Functions in PHP

PHP functions work in the same way as functions in other languages. They are the main way in which we can modularize our code.

```
function add($num1, $num2)
{
  $a = $num1 + $num2;
  return $a;
}

$a = 6;
$b = 5;
echo "a: $a, b: $b <br />";
$c = add($a, $b);
echo "a: $a, b: $b, c: $c <br />";
```



```
Hello, World
a: 6, b: 5
a: 6, b: 5, c: 11
```

In PHP, function scope is isolated from global scope.

In Javascript, the add function as written here, would have overwritten the value of $a in the global scope

01_scope.php

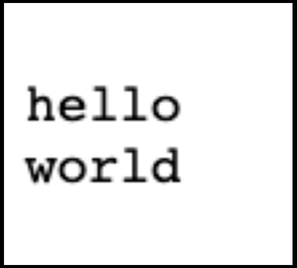
Naming functions

While variable names in PHP are case sensitive, function names are not case sensitive.

```php
function DOSOMETHING($str) {
   return STRTOLOWER($str);
}

$a = DOSOMETHING('HELLO');
$b = dosomething('WORLD');

echo $a;
echo $b;
```



```
hello
world
```

The names of PHP functions, as well as our own function names, are case insensitive.

DOSOMETHING === dosomething

STRTOLOWER === strtolower

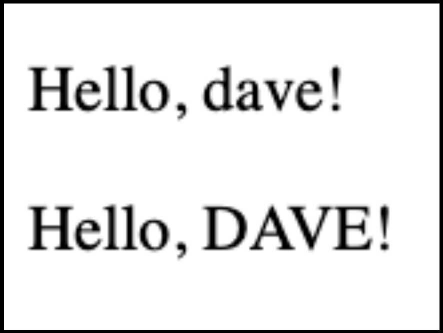
Please do not take advantage of this “feature”

02_naming.php

Optional parameters

By giving a parameter a default value in the function signature, the parameter becomes optional. If not passed in, it will have the default value.

```
function sayHello($name, $upper = false) {
     if($upper) {
          $name = strtoupper($name);
     }
     return 'Hello, ' . $name . '!';
}
```



```
Hello, dave!

Hello, DAVE!
```

Parameter values must be passed in to the function in the same order in which they appear in the function signature.

03_default_values.php

Avoiding naming collisions

Naming a function the same as a previously declared function will cause a fatal error.

```
if(!function_exists('doSomething')) {

   function doSomething()
   {
      return 'No other doSomethings around here!’;
   }

}

echo doSomething();
```

If you’re not sure, check... using function_exists(). This is not really necessary unless you are importing code from a number of sources, in which case collisions may be more likely.

04_collisions.php

Passing values by reference

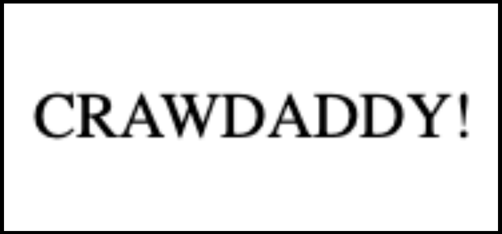
By default, variables passed into functions are copied. The function receives a new copy of the variable value, and any changes made to it will not affect the original.

```
$name = 'Crawdaddy!';

function changeName(&$a)
{
  $a = strtoupper($a);
}

changeName($name);
```

Functions can be written, however, that receive parameters by reference, not by copy. If the parameter is modified inside the function, it is the original variable that is being modified.


CRAWDADDY!

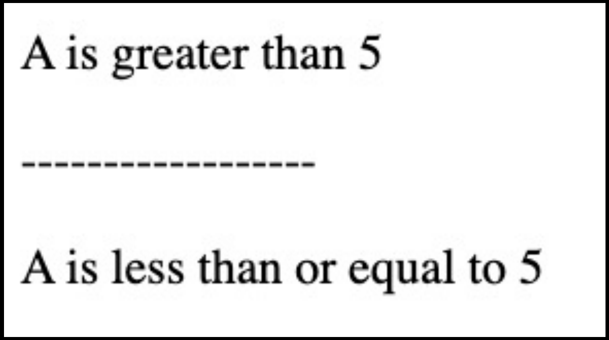
Note: objects are always passed by reference.

05_reference.php

Returning values from functions

Functions can perform work, and then do one of two things: perform an action that affects the state of the program (echo output, for example), or they can return a value.

```
function saySomething($a)
{
  if($a > 5) {
    return 'A is greater than 5’;
  }
    return 'A is less than or equal to 5';
}
```



```
A is greater than 5

-------------------

A is less than or equal to 5
```

A function can only return once. If $a is greater than five, the program flow will never pass the first return statement

06_return.php

Recursive functions

Recursion occurs when a function invokes itself. This can be useful at times, but it can also be risky if not thought out clearly. The equivalent of an infinite loop might occur.

```
function upperArray($array)
{
  foreach($array as $key => $value) {
    if(is_array($value)) {
      $array[$key] = upperArray($value);
    } else {
      $array[$key] = strtoupper($value);
    }
  }
  return $array;
}
```

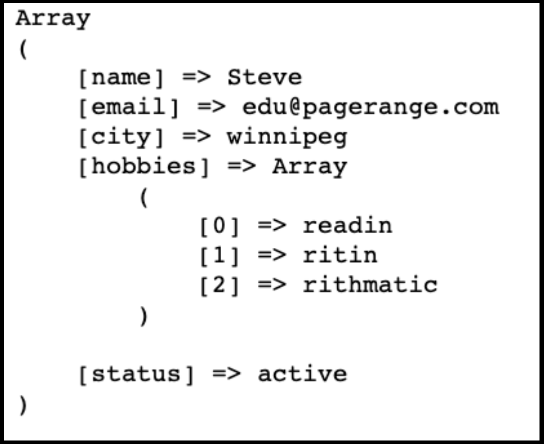
This function loops through an array and converts every value to upper case. If the value is an array, however... the function invokes itself, and passes the array in as a value.

07_recursion.php

Recursive functions

When passed the following array, we get this result

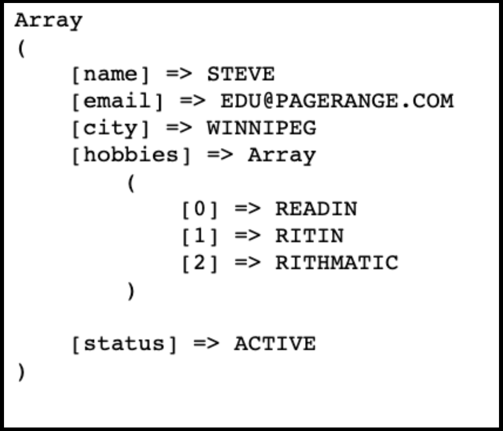
```
$user = [
  'name' => 'Steve’,
  'email' => 'edu@pagerange.com’,
  'city' => 'winnipeg',
‘  hobbies' => ['readin', 'ritin', 'rithmatic’],
  'status' => 'active'
];
```



```
Array
(
    [name] => Steve
    [email] => edu@pagerange.com
    [city] => winnipeg
    [hobbies] => Array
        (
            [0] => readin
            [1] => ritin
            [2] => rithmatic
        )

    [status] => active
)
```

original



```
Array
(
    [name] => STEVE
    [email] => EDU@PAGERANGE.COM
    [city] => WINNIPEG
    [hobbies] => Array
        (
            [0] => READIN
            [1] => RITIN
            [2] => RITHMATIC
        )

    [status] => ACTIVE
)
```

result

07_recursion.php

Type hinting for functions

In PHP we can enforce type restrictions on functions by providing a hint for the function parameters, as well as the function return value.

```php
function happyBirthday(int $age, string $name, string $gender):string
{
   return '<p>Happy Birthday, ' . $name .
          '! A big ' . $gender . ' who is ' . $age . '!</p>';
}

echo happyBirthday('twelve', 'Dave', 'boy');
```

```
Fatal error: Uncaught TypeError: happyBirthday(): Argument #1 ($age) must be of type int, string given
```

08_type_hinting.php

Closures

In PHP, like Javascript, functions are first class: they can be assigned to variables, or used as function parameters. They can also capture variables from the current global scope at a moment in time.

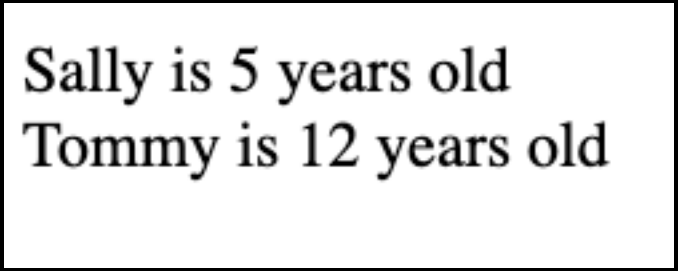
```
$age = 12;
$name = 'Tommy';

$hello = function () use($age, $name) {
   return "$name is $age years old";
};

$age = 5;
$name = 'Sally';

echo "$name is $age years old";
echo '<br />';
echo $hello();
```

09_closure.php



```
Sally is 5 years old
Tommy is 12 years old
```

Although we change the value of $name to Sally, right after we assign the anonymous function to $hello, $hello returns the original value of $name when it is called... the value $name had at the time the function was assigned.

Positional Parameters

Parameters are positional. We cannot skip any parameters when invoking a function, even if we only want to pass in the last one.

```
function do_something(string $param1 = null, int $param2 = null,
                      array $param3 = null, string $param4 = null)
{
    // do something in here
}
```

If we only wanted to pass $param4 into this function, we would have to provide null values for the other params.

```
do_something(null, null, null, 'Hello, World');
```

10_named_parameters.php

Named Parameters

As of PHP 8, parameters can be named when they are passed into a function when invoked.

```
function do_something(string $param1 = null, int $param2 = null,
                      array $param3 = null, string $param4 = null)
{
    // do something in here
}
```

Since PHP 8, we can now name the parameters we want to pass in, ignoring the other positional parameters entirely. If we name all the parameters, we can pass them in any order we like

```
do_something(param4: 'Hello, World');
```

10_named_parameters.php

Next:

Including Files