Promises are easy

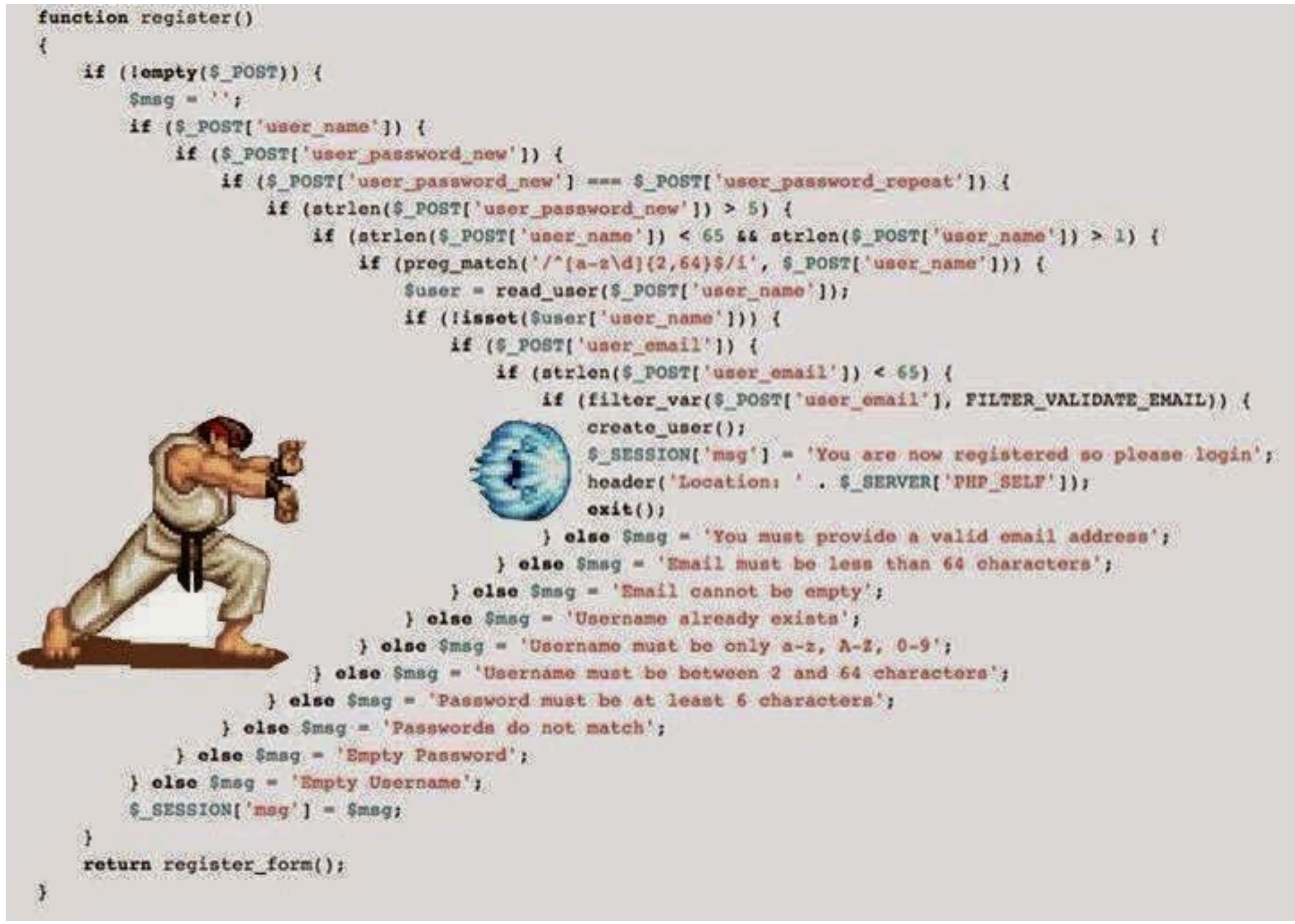
and will make your code better right now

Synchronous Programming

```
var result1 = function1();
var result2 = function2(result1);
print(result2);
```

Asynchronous Programming

```
function1( function(result1) {
    function2(result1, function(result2) {
        print(result2);
    }
});
```



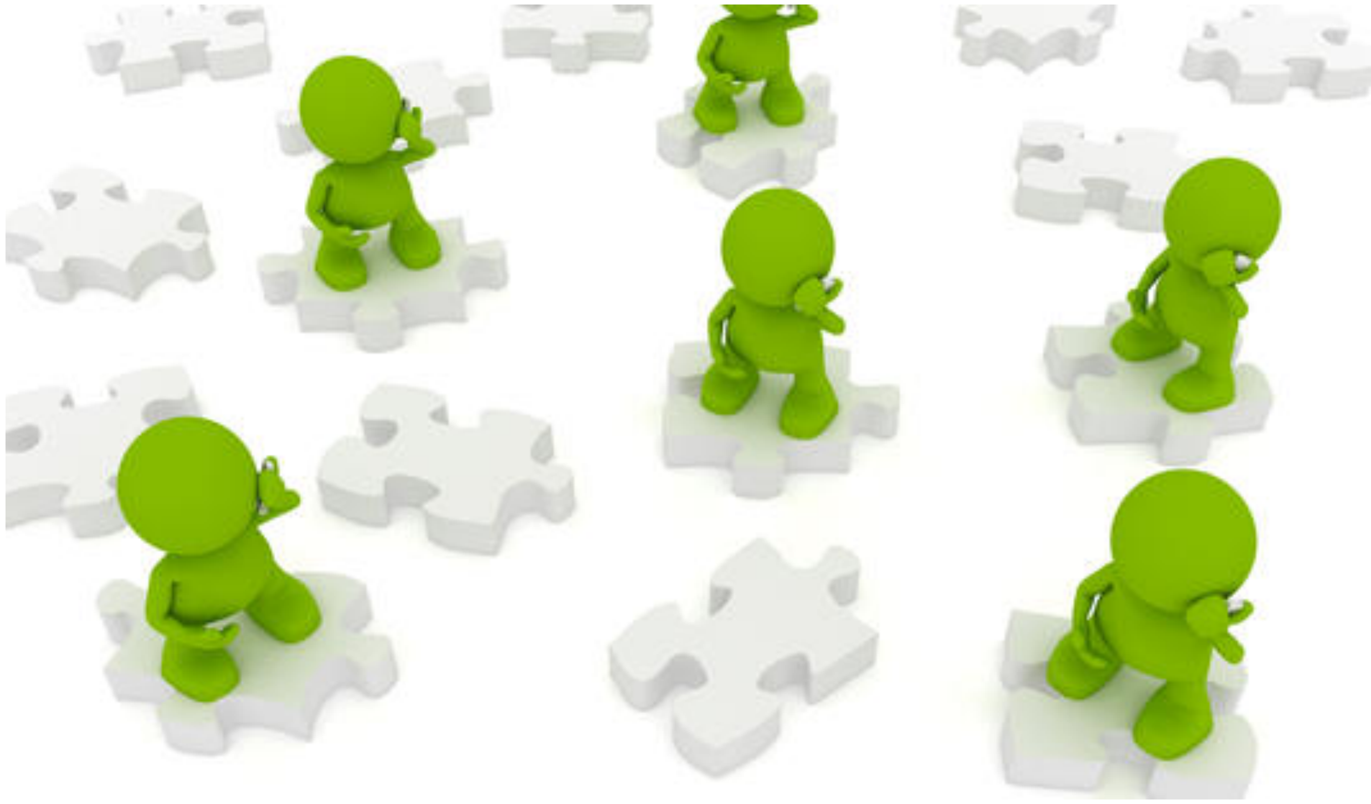
```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```

"Newb, y u do it dis way lol"

```
function printStuff(result) {
    print(result);
}

function continueWithFunction2(result) {
    function2(result, printstuff);
}

function1(continueWithFunction2);
```



Promises

```
function1()
.then(function2)
.then(print)
```

Promises

```
function1()
.then(function2)
.then(print)
```

...which is short for

```
function1()
.then(function(res) {
    return function2(res);
}).then(function(res) {
    print(res)
});
```

Promises

Asynchronous:

```
function1()
.then(function2)
.then(print)
```

Synchronous:

```
var result1 = function1();
var result2 = function2(result1);
print(result2);
```

Promise can be re-used!

```
function setUserImage(user) {
    myUser.image = user.imagePath;
}

function userTest(user) {
    console.log(user);
}

var userPromise = getUser();
userPromise.then(setUserImage)
userPromise.then(userTest);
```

Parallel Execution with Callbacks

```
var results = [];

_.each(asyncOperationsArray, function(asyncOp) {
    var isDoneYet = counterCallback(asyncOperationsArray.length, callWhenDoneFunc);

    asyncOp(function callback() {
        //...
        results.push(asyncOpResult);
        isDoneYet();
    });
}

function counterCallback(count, doneFunction) {
    var counter = count;
    return function() {
        if( --counter  == 0) {
            doneFunction();
        }
    }
}

function callWhenDoneFunc() {
    doSomethingWith(results);
}
```

Parallel Execution with Promises

```
// unknown/irrelevant number of asyncOperations
Q.all([asyncOperationsPromisesArray])
    .then(function(results) {
    // results[0] has the result of the first async operation, etc.
    //...
});

// we know/care about the number and order of asyncOps
Q.all([asyncOperationsArray])
    .spread(function(result1, result2, result 3) {
    //...
});
```

Create a Promise yourself

```
function() {
    var deferred = Q.defer(); //--- 1

    myAsynchronousOperation(function callback() {
        if (everythingWentRight) {
            deferred.resolve(theData); //--- 2
        }
        else { //everything is terrible
            deferred.reject(reasonOrError); //--- 3
        }
    });

    return deferred.promise; //--- 4
}
```

Promises

- Keep your logic in one place instead of spread out over callbacks
- Give you back the control on when and where stuff happens
- Can be reused which decouples unrelated operations
- Are a super-easy drop-in replacement for callbacks

Find User Example

```
git clone https://github.com/mgerlach-klick/promises-lnl.git
```

Kudos 👍*s* Example

Additional resources

Example code with solutions:
https://github.com/mgerlach-klick/promises-lnl.git
(**solutions** branch)

Q API reference:
https://github.com/kriskowal/q/wiki/API-Reference

Promises Anti-patterns:
http://taoofcode.net/promise-anti-patterns/