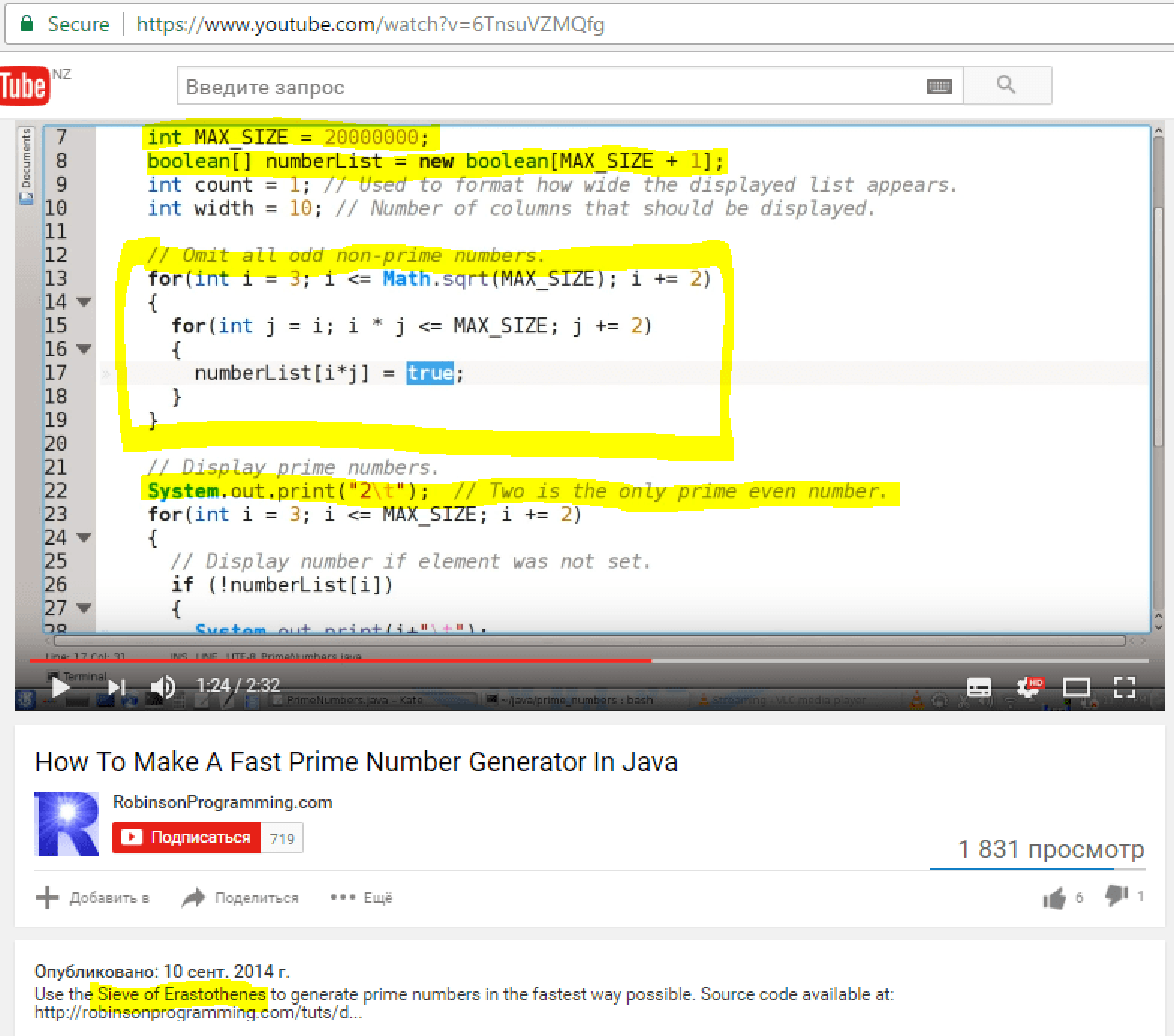
```
$message = 'aaaaaaaaaabbbaxxxxyyyzyx';

function run_length_encode($msg)
{
        $i = $j = 0;
        $prev = '';
        $output = '';

        while ($msg[$i]) {
                if ($msg[$i] != $prev) {

                        if ($i)
                                $output .= $j;

                        $output .= $msg[$i];

                        $prev = $msg[$i];

                        $j = 0;
                }
                $j++;
                $i++;
        }

        $output .= $j;

        return $output;
}

// a10b3a1x4y3z1y1x1
echo run_length_encode($message);
```



Secure | https://www.youtube.com/watch?v=6TnsuVZMQfg

Введите запрос

```java
    int MAX_SIZE = 20000000;
    boolean[] numberList = new boolean[MAX_SIZE + 1];
    int count = 1; // Used to format how wide the displayed list appears.
    int width = 10; // Number of columns that should be displayed.

    // Omit all odd non-prime numbers.
    for(int i = 3; i <= Math.sqrt(MAX_SIZE); i += 2)
    {
      for(int j = i; i * j <= MAX_SIZE; j += 2)
      {
        numberList[i*j] = true;
      }
    }

    // Display prime numbers.
    System.out.print("2\t");  // Two is the only prime even number.
    for(int i = 3; i <= MAX_SIZE; i += 2)
    {
      // Display number if element was not set.
      if (!numberList[i])
      {
        System.out.print(i+"\t");
```

Line: 17 Col: 31 INS LINE UTF-8 PrimeNumbers.java

1:24 / 2:32

How To Make A Fast Prime Number Generator In Java

RobinsonProgramming.com

Подписаться 719

1 831 просмотр

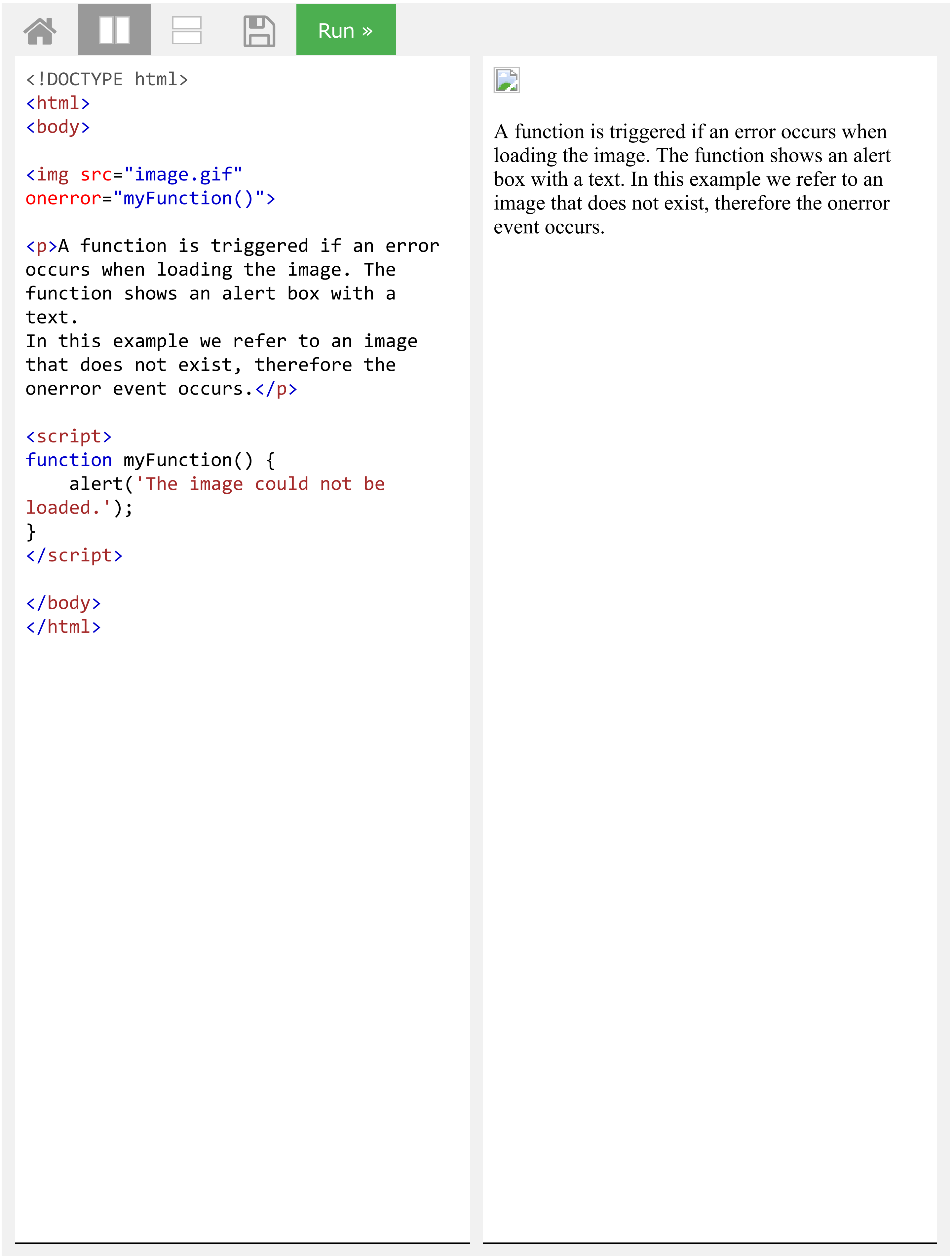
Добавить в   Поделиться   Ещё   6   1

Опубликовано: 10 сент. 2014 г.
Use the Sieve of Erastothenes to generate prime numbers in the fastest way possible. Source code available at: http://robinsonprogramming.com/tuts/d...

```
<?php

function callback($buffer)
{
  // replace all the apples with oranges
  return (str_replace("apples", "oranges", $buffer));
}

ob_start("callback");

?>
<html>
<body>
<p>It's like comparing apples to oranges.</p>
</body>
</html>
<?php

ob_end_flush();

?>
```

The above example will output:

```
<html>
<body>
<p>It's like comparing oranges to oranges.</p>
</body>
</html>
```



```
<!DOCTYPE html>
<html>
<body>

<img src="image.gif"
onerror="myFunction()">

<p>A function is triggered if an error
occurs when loading the image. The
function shows an alert box with a
text.
In this example we refer to an image
that does not exist, therefore the
onerror event occurs.</p>

<script>
function myFunction() {
    alert('The image could not be
loaded.');
}
</script>

</body>
</html>
```

A function is triggered if an error occurs when loading the image. The function shows an alert box with a text. In this example we refer to an image that does not exist, therefore the onerror event occurs.

	MySQL	JS	PHP
True	1	Have 'real' value not 0 non-empty string	
False	0	-0, 0 " "	0, 0.0 " ", "0"
		just 'var x;' ie undefined	
		'Var x=null' ie null	null incl unset var
		10/"A" ie NaN	
			array 0-elem
			Simple XML Obj Created from empty tags

Type	Result
Undefined	`"undefined"`
Null	`"object"` (see below)
Boolean	`"boolean"`
Number	`"number"`
String	`"string"`
Symbol (new in ECMAScript 2015)	`"symbol"`
Host object (provided by the JS environment)	*Implementation-dependent*
Function object (implements [[Call]] in ECMA-262 terms)	`"function"`
Any other object	`"object"`

Examples

```
// Numbers
typeof 37 === 'number';
typeof 3.14 === 'number';
typeof(42) === 'number';
typeof Math.LN2 === 'number';
typeof Infinity === 'number';
typeof NaN === 'number'; // Despite being "Not-A-Number"
typeof Number(1) === 'number'; // but never use this form!


// Strings
typeof '' === 'string';
typeof "bla" === 'string';
typeof (typeof 1) === 'string'; // typeof always returns a string
typeof String('abc') === 'string'; // but never use this form!


// Booleans
typeof true === 'boolean';
typeof false === 'boolean';
typeof Boolean(true) === 'boolean'; // but never use this form!


// Symbols
typeof Symbol() === 'symbol'
typeof Symbol('foo') === 'symbol'
typeof Symbol.iterator === 'symbol'


// Undefined
```

```
typeof undefined === 'undefined';
typeof declaredButUndefinedVariable === 'undefined';
typeof undeclaredVariable === 'undefined';


// Objects
typeof {a: 1} === 'object';

// use Array.isArray or Object.prototype.toString.call
// to differentiate regular objects from arrays
typeof [1, 2, 4] === 'object';

typeof new Date() === 'object';


// The following is confusing. Don't use!
typeof new Boolean(true) === 'object';
typeof new Number(1) === 'object';
typeof new String('abc') === 'object';


// Functions
typeof function() {} === 'function';
typeof class C {} === 'function';
typeof Math.sin === 'function';
```

null

```
// This stands since the beginning of JavaScript
typeof null === 'object';
```

In the first implementation of JavaScript, JavaScript values were represented as a type tag and a value. The type tag for objects was 0. `null` was represented as the NULL pointer (0x00 in most platforms). Consequently, null had 0 as type tag, hence the bogus `typeof` return value. (reference)

A fix was proposed for ECMAScript (via an opt-in), but was rejected. It would have resulted in `typeof null === 'null'`.

Regular expressions

Callable regular expressions were a non-standard addition in some browsers.

```
typeof /s/ === 'function'; // Chrome 1-12 Non-conform to ECMAScript 5.1
typeof /s/ === 'object';   // Firefox 5+  Conform to ECMAScript 5.1
```

Exceptions

All current browsers expose a non-standard host object `document.all` with type Undefined.

```
typeof document.all === 'undefined';
```

Although the specification allows custom type tags for non-standard exotic objects, it requires those type tags to be different from the predefined ones. The case of `document.all` having type tag `'undefined'` must be classified as an exceptional violation of the rules.

In JavaScript, `undefined` means a variable has been declared but has not yet been assigned a value, such as:

```
var TestVar;
alert(TestVar); //shows undefined
alert(typeof TestVar); //shows undefined
```

`null` is an assignment value. It can be assigned to a variable as a representation of no value:

```
var TestVar = null;
alert(TestVar); //shows null
alert(typeof TestVar); //shows object
```

From the preceding examples, it is clear that `undefined` and `null` are two distinct types: `undefined` is a type itself (undefined) while `null` is an object.

```
null === undefined // false
null == undefined // true
null === null // true
```

and

```
null = 'value' // ReferenceError
undefined = 'value' // 'value'
```

> ### native object
>
> object in an ECMAScript implementation whose semantics are fully defined by this specification rather than by the host environment.
>
> NOTE Standard native objects are defined in this specification. Some native objects are built-in; others may be constructed during the course of execution of an ECMAScript program.

Source: http://es5.github.com/#x4.3.6

> ### host object
>
> object supplied by the host environment to complete the execution environment of ECMAScript.
>
> NOTE Any object that is not native is a host object.

Source: http://es5.github.com/#x4.3.8

A few examples:

Native objects: `Object` (constructor), `Date` , `Math` , `parseInt` , `eval` , string methods like `indexOf` and `replace` , array methods, ...

Host objects (assuming browser environment): `window` , `document` , `location` , `history` , `XMLHttpRequest` , `setTimeout` , `getElementsByTagName` , `querySelectorAll` , ...

JS	PHP
undef 'var x;' only declared, no value assigned	
null ~~object~~ 'var x=null'	Null isset() [empty() check false] assign const [illegible] NULL " nothing yet unset ()
boolean	boolean
number	float, integer
string	string
symbol	
function object	
object	object
	array
	resource
Host Object	

There is no "undefined" data type in PHP. You can check for a variable being set with `isset`, but this cannot distinguish between a variable not being set at all and it having a `null` value:

```
var_dump(isset($noSuchVariable)); // false

$nullVariable = null;
var_dump(isset($nullVariable)); // also false
```

However, there is a trick you can use with `compact` that allows you to determine if a variable has been defined, even if its value is `null`:

```
var_dump(!!compact('noSuchVariable')); // false
var_dump(!!compact('nullVariable')); // true
```

Live example.

Both `isset` and the `compact` trick also work for multiple variables at once (use a comma-separated list).

You can easily distinguish between a `null` value and total absence when dealing with array keys:

```
$array = array('nullKey' => null);

var_dump(isset($array['nullKey'])); // false
var_dump(array_key_exists($array, 'nullKey')); // true
```

Live example.

When dealing with object properties there is also `property_exists`, which is the equivalent of `array_key_exists` for objects.

The special **NULL** value represents a variable with no value. **NULL**is the only possible value of type null.

A variable is considered to be null if:

- it has been assigned the constant **NULL**.
- it has not been set to any value yet.
- it has been unset().

Syntax

There is only one value of type null, and that is the case-insensitive constant **NULL**.

```
<?php
$var = NULL;
?>
```

See also the functions is_null() and unset().

Casting to NULL

Casting a variable to null using *(unset) $var* will *not* remove the variable or unset its value. It will only return a **NULL** value.

User Contributed Notes 8 notes

add a note

42 quickpick 5 years ago

```
Note: empty array is converted to null by non-strict equal '==' comparison. Use
is_null() or '===' if there is possible of getting empty array.

$a = array();

$a == null  <== return true
$a === null < == return false
is_null($a) <== return false
```

Objects are passed (and assigned) by reference. No need to use address of operator.

Granted what I typed is an oversimplification but will suit your purposes. The documentation states:

> One of the key-points of PHP5 OOP that is often mentioned is that "objects are passed by references by default". This is not completely true. This section rectifies that general thought using some examples.
>
> A PHP reference is an alias, which allows two different variables to write to the same value. As of PHP5, an object variable doesn't contain the object itself as value anymore. It only contains an object identifier which allows object accessors to find the actual object. When an object is sent by argument, returned or assigned to another variable, the different variables are not aliases: they hold a copy of the identifier, which points to the same object.

For a more detailed explanation (explains the oversimplification as well as identifiers) check out this answer.