PHP

2017 m Tautvydas

Php basics

https://wwwx.cs.unc.edu/~hays/INLS672/lessons/05php.pdf

PHP introduction

PHP is server side scripting language

- PHP stands for "PHP: Hypertext Preprocessor"
- Very good for creating dynamic content
- PHP = Hypertext preprocessor
- Used for development of dynamical webpages
- Part of typical LAMP combination
- Linux, Apache, MySQL and PHP
- Includes a command line scripting possibility
- Can be used in graphical applications

HOW IT WORKS

PHP code is usually embedded into HTML

Processing the code :

1. The HTML code stands as it is
2. The PHP scripts are executed to create final HTML code
3. Both parts are combined and back
4. Resulting HTML is interpreted by a browser

- Typically file ends in .php
- Separated in files with the tag
- php commands can make up an entire file, or can be contained in html--this is a choice….
- Program lines end in ";" or you get an error
- Server recognizes embedded script and executes

Two Ways

You can embed sections of php inside html (index.html):

```
<body>
    <div>  <?php echo “Hello world!”; ?> </div>
</body>
```

- Or you can call html from php (index.php):

```
<?php echo "<body><div>Hello world!</div></body>” ?>
```

Comments

// komentaras

/* komentaras

komentaras

*/

Writing of the plain text

Echo “text”

print “text”

```
$name = "bil";
echo "Howdy, my name is $name";
echo "What will $name be in this line?";
echo 'What will $name be in this line?';
echo 'What's wrong with this line?';

 if ($name == "bil") {
        // Hey, what's this? echo "got a match!";
}
```

Variables (kintamieji)

Typed by context (but one can force type), so it's loose

- Begin with "$" (unlike javascript!)
- Assigned by value: $foo = "Bob"; $bar = $foo;
- Assigned by reference, this links vars: $bar = &$foo;
- Some are preassigned, server and env vars
  - For example, there are PHP vars, eg.: PHP_SELF, HTTP_GET_VARS

Variables TYPES

PHP has a total of eight data types which we use to construct our variables –

- **Integers** – are whole numbers, without a decimal point, like 4195.
- **Doubles** – are floating-point numbers, like 3.14159 or 49.1.
- **Booleans** – have only two possible values either true or false.
- **NULL** – is a special type that only has one value: NULL.
- **Strings** – are sequences of characters, like 'PHP supports string operations.'
- **Arrays** – are named and indexed collections of other values.
- **Objects** – are instances of programmer-defined classes, which can package up both other kinds of values and functions that are specific to the class.
- **Resources** – are special variables that hold references to resources external to PHP (such as database connections).

VARIABLE TYPES

Numerical

Integer – positive as well as negative, including 0

Float – real numbers, 14 digits accuracy

Logical

Boolean - True x False, not case sensitive

Alphabetical

String – set of characters

```
$true_num = 3 + 0.14159;
$true_str = "Tried and true"
$true_array[49] = "An array element";
$false_array = array();
$false_null = NULL;
$false_num = 999 - 999;
$false_str = "";
```

constant

Constant

```php
<?php
    define("USERNAME", “Tomas”);
    define("MINSIZE", 50);
    define("ONE",      "first thing");
    define("TWO2",     "second thing");

       echo MINSIZE;   // 50
```

WORKING WITH VARIABLES

Settype($var, “integer”)
allows you to set variable according to your wish
Gettype()
write the type of variable

(.)
Connects 2 variables of string type
strlen()
finds the length of a string

Operators

$a and $b	And True if both $a and $b are true.
$a or $b	Or True if either $a or $b is true.
$a xor $b	Xor True if either $a or $b is true, but not both.
! $a	Not True if $a is not true.
$a && $b	And True if both $a and $b are true.
$a || $b	Or True if either $a or $b is true.

Control Structures

- if, else, elseif
- while, do-while
- for, foreach
- break, continue, switch
- require, include, require_once, include_once

If , elseif

```
if ($i == 0) {
        echo "i equals 0";
        } elseif ($i == 1) {
                echo "i equals 1";
                } elseif ($i == 2) {
                        echo "i equals 2";
                }
}
```

Switch

```
switch ($i) {
        case 0:   echo "i equals 0";
                  break;
        case 1:   echo "i equals 1";
                  break;
        case 2:   echo "i equals 2";
                  break;
}
```

Nesting Files

require(),

include(),

include_once(),

require_once()

are used to bring in an external file

ARRAYS IN PHP

- Numeric array
  - Each element of array has its ID number (first 0!!)
  - `$names = array("Petr","Joe");`
  - `$names[0] = "Petr";`

- Associative Arrays
  - Each element is assigned its value
  - `$ages = array("Peter"=>32, "Joe"=>34);`
  - `$ages['Peter'] = "32";`

Arrays

$my_array = array(1, 2, 3, 4, 5);

```
$pizza  = "piece1 piece2 piece3 piece4 piece5 piece6";

$pieces = explode(" ", $pizza);

echo $pieces[0]; // piece1

echo $pieces[1]; // piece2
```

Arrays are lists, or lists of lists, or list of lists of lists, you get the idea--Arrays can be multidimensional

• Array elements can be addressed by either by number or by name (strings)

• If you want to see the structure of an array, use the print_r function to recursively print an array inside of pre tags

MULTIDIMENSIONAL ARRAYS

```
$fruits = array (
    "fruits"  => array("a" => "orange", "b" => "banana", "c" => "apple"),
    "numbers" => array(1, 2, 3, 4, 5, 6),
    "holes"   => array("first", 5 => "second", "third")
);
```

Kaip pasiekti “banana”?

```
print_r(array_column($fruits, “b“));
```

Walking Arrays

```
$colors = array('red' , 'blue' , 'green' , 'yellow');

foreach ($colors as $color) {

        echo "Do you like $color?\n";

}
```

You can use print_r()

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
    [3] => 4
)
```

– array_flip() swaps keys for values
– array_count_values() returns an associative array of all values in an array, and their frequency
– array_rand() pulls a random element
– array_unique() removes duppies
– count() returns the number of elements in an array
– array_search() returns the key for the first match in
….

Standard data files

```
<body>
<h1> Failas nuskaitytas:</h1>
<?php
$myfile = fopen("webdictionary.txt", "r") or die("Unable to open file!");
// Output one line until end-of-file
while(!feof($myfile)) {
  echo fgets($myfile) . "<br>";
}
fclose($myfile);
?>

</body>
</html>
```

webdictionary.txt

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

Ruzultatas narsykleje

`Failas nuskaitytas:`

AJAX = Asynchronous JavaScript and XML
CSS = Cascading Style Sheets
HTML = Hyper Text Markup Language
PHP = PHP Hypertext Preprocessor
SQL = Structured Query Language
SVG = Scalable Vector Graphics
XML = EXtensible Markup Language

Useful string functions

str_replace()

- trim(), ltrim(), rtrim()
- implode(), explode()
- addslashes(), stripslashes()
- htmlentities(), html_entity_decode(), htmlspecialchars()
- striptags()

PHP FORMS AND USER INPUT

Used to gain information from users by means of HTML
Information is worked up by PHP

```
<html>
<body>
<form action="welcome.php" method="post">
        Name: <input type="text" name="name" />
        Age: <input type="text" name="age" />
        <input type="submit" />
</form>
</body>
</html>
```

THE $_POST VARIABLE

Used to collect values from a form Information from a form is invisible
No limits on the amount of information to be send

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br />
You are <?php echo $_POST["age"]; ?> years old.
</body>
</html>
```

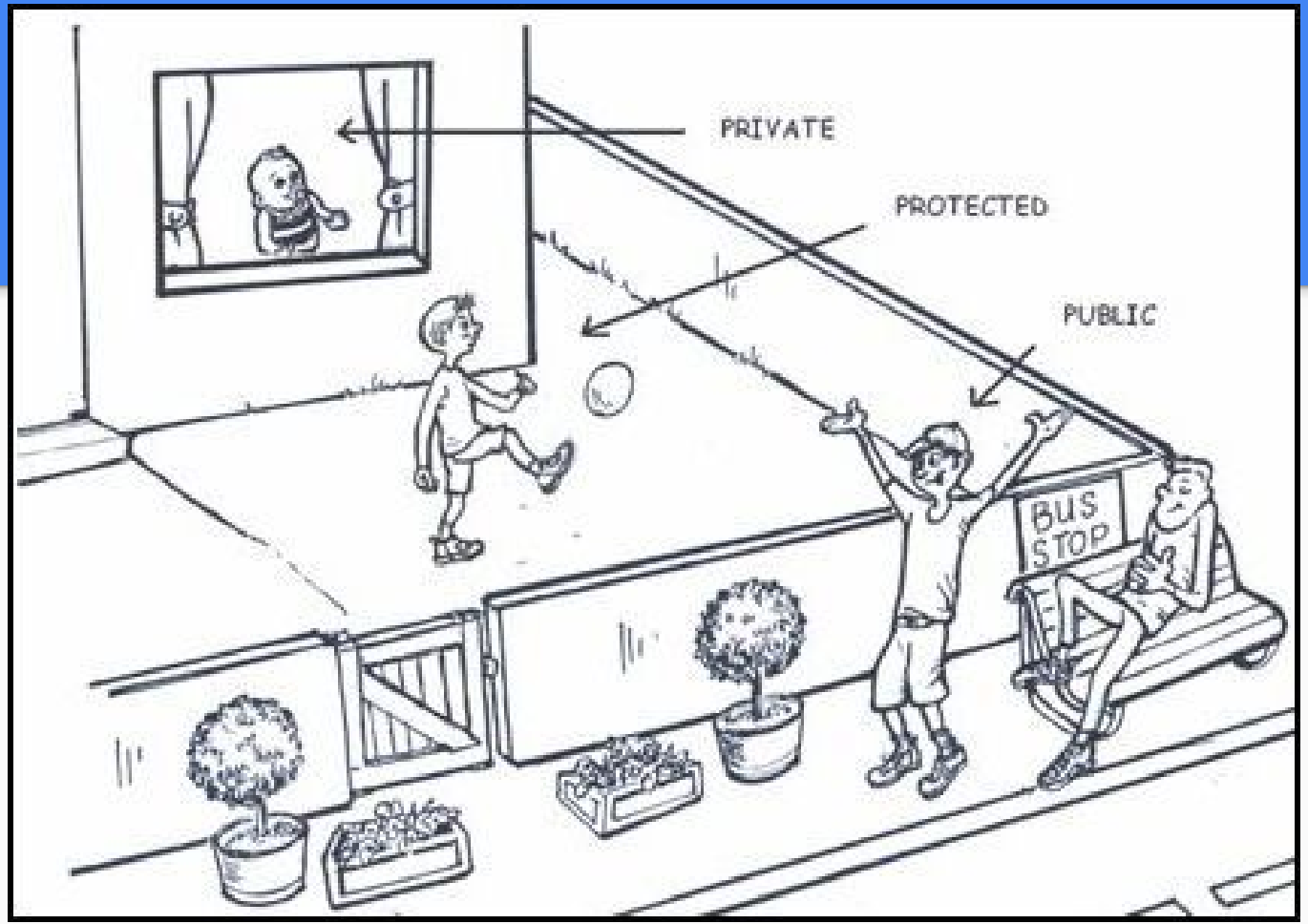
OBJ

http://somabo.de/talks/200606_skien_oop.pdf

Deklaracija

public anyone can access it ;
protected only descendants can access it ;
private only you can access it ;
final no one can re-declare it ;
abstract someone else will implement this

Static - jo reiksme saugoma ir funkcijai pabaigus darba


PRIVATE
PROTECTED
PUBLIC
BUS STOP

```php
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}
```

```php
$obj = new MyClass();
echo $obj->public;          // Public
echo $obj->protected;       // Fatal Error
echo $obj->private;         // Fatal Error
$obj->printHello();
//  Public, Protected and Private
```

Scope

PHP variables can be one of four scope types –

- Local variables
- Function parameters // function ($kintName) { }
- Global variables // global $name; ARBA $name; uz kalses
- Static variables // STATIC $count = 0;

Static - kintamasis nesunaikinamas funkcijai atlikus darba

In contrast to the variables declared as function parameters, which are destroyed on the function's exit,

a static variable will not lose its value when the function exits and will still hold that value should the function be called again.

Static

```php
<?php
function test()
{
    static $a = 0;
    echo $a;
    $a++;
}
test();   // $a == 0
test();   // $a == 1
test();   // $a == 2
?>
```

`$a` is initialized only in first call of function and every time the *test()* function is called it will print the value of `$a` and increment it.

```php
<?php
function test2()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        test2();
    }
}
?>
// $count == 10
```

OBJ eg.

Codeacademy PHP obj:

https://www.codecademy.com/courses/web-beginner-en-bH5s3/0/2

What are the features of OOP?

Encapsulation (Jusu kintamuju ir funkciju matomumas: private, pubilc,...)

Inheritance (Paveldimumas)

Polymorphism (Taip paciai besivadinanti funkcija atlieka skirtingus veiksmus)

Encapsulation

Jusu kintamuju ir funkciju matomumas: private, pubilc,...

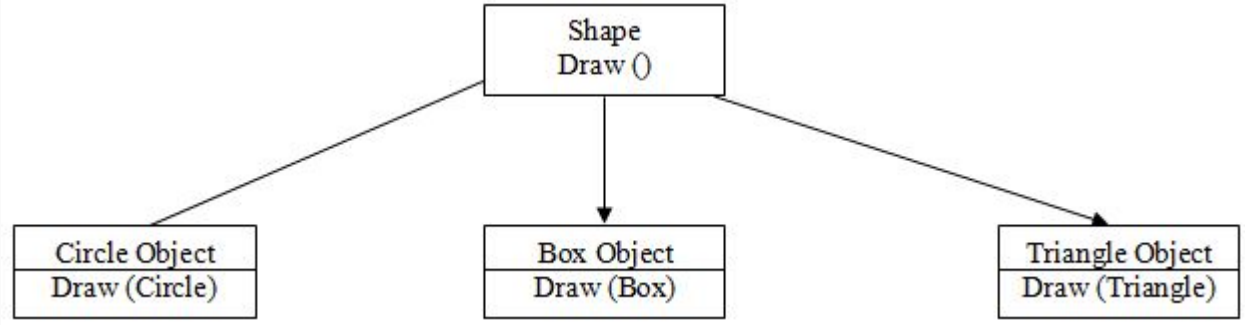
In basic terms, it's the way we define the visibility of our properties and methods.

```
class CanadianPet extends Pet
{
   public function setPetName( $newName )
   {
      $this->petName = $this->checkPetName( $newName );
   }
   private function checkPetName( $petNameToCheck )
   {
      return str_replace( 'a', '', $petNameToCheck );
   }
}
```

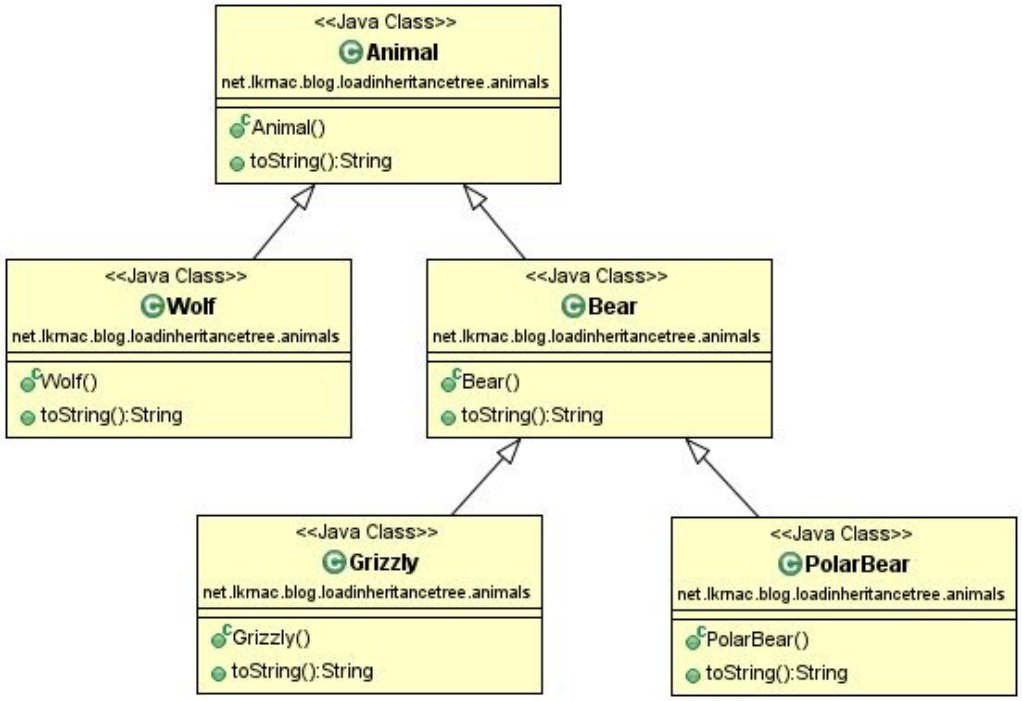
Polymorphism

Taip paciai besivadinanti funkcija atlieka skirtingus veiksmus

Polymorphism describes a pattern in object oriented programming in which classes have different functionality while sharing a common interface.



Inheritance (Paveldimumas)



```php
class Foo
{
    public function printItem()
    {
        echo 'Foo: ';
    }
}

class Bar extends Foo
{

}
```

```php
$foo = new Foo();
$bar = new Bar();
$foo->printItem(); // Output: 'Foo: '
$bar->printItem(); // Output: 'Foo: '
```

The Problem of Code Duplication

Code duplication contradicts maintainability. You often end up with code that looks like this:

```
function foo_to_xml($foo) {
// generic stuff ….
// foo-specific stuff ...
}
function bar_to_xml($bar) {
// generic stuff   ...
// bar specific stuff ...}
```

Solution - Inheritance

```
class Base {
    public function toXML() { /*...*/ }
 }
Class Foo extends Base {
    // foo-specific stuff ...
}
Class Bar extends Base {
    // bar specific stuff …
}
```

Inheritance eg.

```
class Humans {
        public function __construct($name) { /*...*/ }
        public function eat() { /*...*/ }
        public function sleep() { /*...*/ }
        public function wakeup() { /*...*/ }
}
class Women extends Humans {
        public function giveBirth() { /*...*/ }
}
class Men extends Humans {
         public function snore() { /*...*/ }
        }
```

Interface

- An interface is similar to a class except that it cannot contain code.
- An interface can define method names and arguments, but not the contents of the methods.
- Any classes implementing an interface **must** implement all methods defined by the interface.
- A class can implement multiple interfaces.

Interface

```
interface MyInterface {
    public function doThis();
    public function setName($name);
}

// VALID
class MyClass implements MyInterface {
    protected $name;
    public function doThis() {
        // code that does this
    }
    public function doThat() {
        // code that does that
    }
    public function setName($name) {
        $this->name = $name;
    }
}
```

INVALID Interface

```
interface MyInterface {
    public function doThis();
    public function setName($name);
}

// VALID
class MyClass implements MyInterface {
    protected $name;
    public function doThat() {
        // code that does that
    }
    public function setName($name) {
        $this->name = $name;
    }
}
```

Abstract Class

- An abstract class is a mix between an interface and a class.
- It can define functionality as well as interface (in the form of abstract methods).
- Classes extending an abstract class **must** implement all of the abstract methods defined in the abstract class.

Regular methods can be defined in an abstract class just like in a regular class, as well as any abstract methods (using the '`abstract`' keyword).

Abstract methods behave just like methods defined in an interface, and must be implemented exactly as defined by extending classes.

```
abstract class MyAbstract {
    public $name;
    public function doThis() {
        // do this  ...
    }
    abstract public function doThat();          // tures but aprasytas paveldejimo metu
    abstract public function setName($name); // tures but aprasytas paveldejimo
metu
}
```

abstraction + Inheritance

```
abstract class Humans {
        abstract public function gender();
        public function eat() {          /*...*/ }
        public function sleep() {       /*...*/ }
}
class Women extends Humans {
         public function gender() {        return 'female'; }
         public function giveBirth() {      /*...*/ }
}
 class Men extends Humans {
        public function gender() {        return 'male'; }
         public function snore() {         /*...*/ }
}
```

Overloading or Polymorphism the other way round

Unlike other languages PHP does not and will not offer overloading polymorphism for method calling. Thus the following will never work in PHP

```php
<?php
class Test {
    function toXML(Personal $obj) //...
    function toXML(Professional $obj) //...
}
?>
```

function toXML (String $s) { }
function toXML (Integer $i) { }