# Тема 1. PHP – определение, сравнение с другите езици за програмиране. Основни елементи на PHP – синтаксис на езика, константи, променливи, изрази, оператори ...

Виолета Божикова

# PHP – определение

- PHP е безплатен, многоплатформен, скриптов, сървърен език за web програмиране, който може да се вгражда в HTML и позволява разработката на динамични Web сайтове.

# Началото на PHP

- Създаден е през 1994г. от датчанина Расмус Лердорф, който го е използвал за да следи кой разглежда личния му сайт. През 1995г. той създава по нова-версия, която вече е достъпна за останалите потребители под името Personal Home Page Tools, от където идва съкращението PHP. Това обаче е много опростена версия на език за програмиране, в която липсва дори оператор за цикъл, и която има много опростен интерпретатор (парсер).
- Едва третата версия (PHP 3 ), създадена през 1998г. от Анди и Зеев е съизмерима с другите езици. Това води до нарастване използването на PHP, броят на домейните с инсталиран PHP 3 достига един милион. Абревиатурата PHP от този момент нататък се дешифрира вече като PHP: Hypertext Preprocessor.

# PHP4

- при PHP4 се въвежда компилиране до междинен код (т.е. до байт код) и последващо изпълнение от виртуална машина, с име Zend Engine (по името на създателите Зеев и Анди).
- от PHP4.1.0 нататък се въвеждат суперглобални променливи като $_GET и $_POST;
- От PHP4.2.0 нататък се забранява конвертирането по подразбиране на входните променливи в глобални променливи (чрез опцията register_globals), което елиминира една от дупките в сигурността на приложенията, написани на PHP

# PHP 5

- нова версия на виртуалната машина и нов обектен модел (съизмерим с този на Java) с манипулатори за достъп – public, private, protected;
- разширена поддръжка на XML чрез интерфейси SAX, DOM, XSLT, включени в библиотека libhml2;
- библиотеката SQLi или MySQL improved, която предоставя процедурен и обектно-ориентиран интерфейс към MySQL4.1 и по нови версии;
- вградена библиотека SQLite за извършване на операции с БД, без да се ползва външен сървър;
- система за прихващане на съобщения, реализирана чрез структура try-catch и др.

# PHP и другите езици за програмиране

- PHP е **сървърен език** за разлика от езиците Java Script, VBScript (клиентски езици) . Думата сървърен означава, че PHP кодът се интерпретира на страната на сървъра (server side), генерира се резултат (HTML страница), който се изпраща на клиента (към клиентския браузър и се визуализира). Java Script и VBScript се интерпретират от браузъра ви (web клиента), тоест са client side езици.
- Сървърът - това е специален компютър (или папка, ако клиентът и сървърът са на един компютър), на който се съхранява php страницата, чийто адрес указвате в адресната лента на своя браузър (например в Internet Explorer).
- Клиентът е всъщност браузерът, с който потребителят разглежда даден Web-сайт.
- Други сървърни езици са ASP (Active Server Pages), JSP (JavaServer Pages) и други.

**Предимства на сървър-базираните страници**.

- Сървърите обикновено са по-бързи и по-мощни от машините, използвани от средния Web клиент. Следователно, сървър-базираните динамични Web страници могат да разполагат с по-големи ресурси и да предоставят по-голямо потенциално съдържание на страниците.
- Обработката от страна на сървъра е единственият начин за съхраняване и извличане на информация във и от база от данни. Данните могат да се организират по-добре и да се съхраняват непрекъснато във файлова система или релационни бази от данни, отколкото това е възможно с използването на клиент-базираните скриптове. Това е особено важно за по-големите сайтове, които имат множество страници и използването на база от данни може да намали значително разходите по поддържане на сайта.
- Обработката от страна на сървъра е независима от браузера. Сървърите изпращат обикновен HTML код и по този начин се избягват проблемите със съвместимостта с различните типове браузери.

- **Трудности при сървър-базираните страници**. Информацията и обработката, която извършва сървърът, са без гражданство и не е известно дали две последователни заявки са от един и същ потребител или от различни. Това изисква всяка заявка да се третира като самостоятелна, а това е свързано с допълнителна обработка.
- Сървър-базираната обработка се случва, само след като е изпратена заявка към сървъра. Това означава, че дори и най-добрите и бързи програмни скриптове могат да действат в малкия интервал от момента, в който потребителят е активирал връзката или е изпратил форма, до момента, когато динамичната страница е върната от сървъра. Тази обработка също така елиминира възможността за всяка промяна на страницата, след като тя е изпратена към браузера.

# PHP и другите езици за програмиране

- PHP е скриптов език, както и Java Script, VBScript . Скриптов език в информационните технологии (Уикипедия) се нарича език за програмиране, при който изходният код на програмите се изпълнява директно при извикване. За да се изпълни директно този изходен код, се използва специална програма, която се нарича интерпретатор. Сорс кодът на програма, която е написана на скриптов език за програмиране се нарича скрипт. Но, докато кодът на Java Script и VBScript се интерпретира от браузъра ви (тоест от web клиента, като за VBScript това е единствено Internet Explorer), тоест това са client side  езици, то PHP кодът се изпълнява на сървъра.
- Определението многоплатформен означава, че PHP скрипт може да се изпълнява под Unix, Windows NT, Macintosh, OS/2 и други сървърни операционни системи. При това вие можете да пренесете своя код на друга платформа почти или въобще без изменения. Силата на PHP е и в съвместимостта му с много типове Бази Данни, браузъри и мрежи с различни протоколи. За сравнение - VBScript работи само в Internet Explorer.

# PHP и другите езици за програмиране

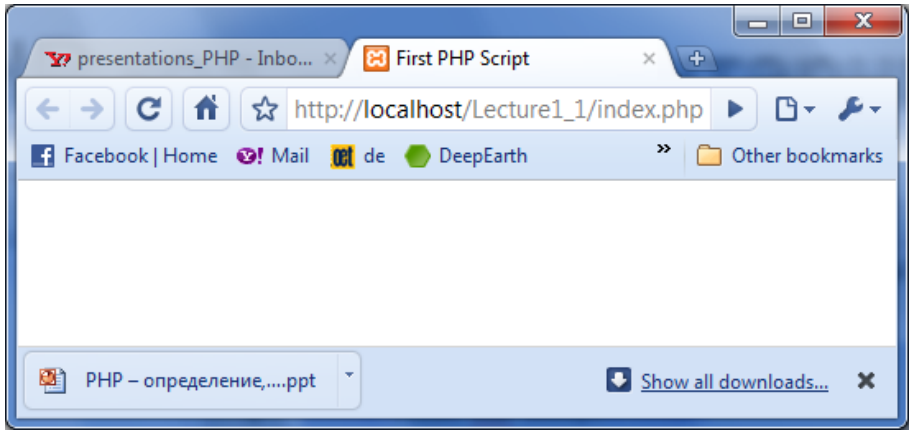
- Словосочетанието вграден в HTML означава, че PHP скрипта се вгражда във файлове, съдържащи HTML код, като се обозначава със специален отварящ и затварящ се таг (като по този начин указвате кода, който трябва да бъде изпратен до сървъра): **<?... Код... ?> или <?php … Код... ?>**
- PHP е безплатен език, създаден единствено да обслужва Web страници, за разлика от други езици като например Perl, който е скриптов език от общ тип.
- PHP позволява разработката на динамични Web сайтове. Известно е, че HTML е подходящ за разработка само на статични Web сайтове. Статичните Web сайтове са изградени само от статични Web страници: всяка Web страница представлява самостоятелен html документ съдържащ всички елементи от страницата: дизайн + съдържание. Когато посетителят на статичен Web сайт кликне върху адреса на някаква страница от сайта, то сървърът директно я изпраща в браузера на посетителя. Всяка HTML страница може само да форматира дадена информация и да я извежда в браузера. Тя не може да изведе динамични данни като час, дата или данни от друг формат (Word, PDF документи или бази данни). Вграждането на PHP скрипт в HTML код ви позволява разработката на динамични Web сайтове: PHP скрипта може да обработва информация от други източници и да я извежда с помощта на HTML, в момента на поискване от потребителя (когато настъпи събитието, например когато потребителя изпрати съобщение или се обърне към даден URL (Uniform Resource Locator - Web-адрес), което прави един Web сайт динамичен (т.е. с променящо се съдържание).

# PHP код – елементарни примери

- **Всеки ред от PHP скрипта задължително трябва да завършва с точка и запетая - (;).**
- **Белите полета (white space) са без значение**
- **Listing 1.1 не извежда нищо.**
- **Документ, съдържащ PHP код трябва да бъде запазен с разширение .php**

Listing 1.1

```
<HTML>
<HEAD>
<TITLE>First PHP Script</TITLE>
</HEAD>
<BODY>
<?PHP
?>
</BODY>
</HTML>
```

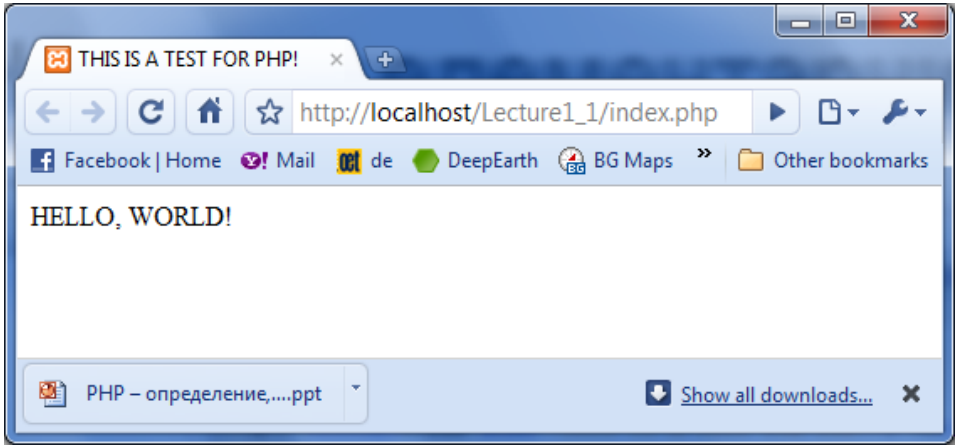


# PHP код – елементарни примери

- **PHP скрипта е вграден в самия HTML документ, при това може да се вгради, където желаете, в един HTML документ.**
- **В примера по-долу се извежда заглавието на страницата чрез PHP код (Листинг 1.2)**

Listing 1.2

```
<HTML>
<HEAD>
<TITLE><?PHP ECHO "THIS IS A TEST FOR PHP!"; ?></TITLE>
</HEAD>
<BODY>
<?PHP
ECHO "HELLO, WORLD!";
?>
</BODY>
</HTML>
```



# echo() vs. print()

Това са езикови конструкции, а не функции въпреки, че print() се държи като функция – връща стойност (винаги 1). Счита се, че echo() е малко по-бърза.

- **void echo** (string $arg1 [, string $... ] ), където args – са параметрите (низове или изрази), които трябва да се изведат

Тъй като **echo()** не е функция, а езикова конструкция, не е задължително използването на кръгли скоби при употребата и, тоест еднакво коректно е:

```
echo ("string1"), (" string2<br>");  //ok   string1 string2
echo "string1", " string2<br>";      //ok   string1 string2
echo (3+4), (3+4), "<br>";  //ok              77
echo 3+4,3+4, "<br>";       //ok              77
```

Нещо повече, ако трябва укажете повече от един параметър в **echo() –** то не трябва да са в кръгли скоби, т.е:

```
echo (3+4,3+4);       //!!! fail
echo ("string1", " string2<br>");       //!!! fail
```

- int **print** (string $arg ) – където arg е извеждания параметър (стринг или израз) – само един.
- също не изисква задължително скоби.

```
print expression;  или print (expression); // ok,
print “string";    или  print ("string“); //Ok
```

**Но:**

```
print (expression), (expression);  или print expression, expression; // fail
print "string", "string";     //Fail
```

# echo() vs. print()

**!!! Забележка: Възможно е конкатениране на изразите или символните низове и с точка, както е възможно и да използваме апострофи, вместо кавички за низовете:**

```
echo "Hello", "World!";     // Ok - в кавички
echo "Hello". "World!";     //Ok - в кавички
echo 'Hello', 'World!';     //в апостроф
echo 'Hello'. 'World!';     //в апостроф
echo (3+4).(5+6); //Ok – !!!задължително отделните изрази да са в скоби
echo 3+4.5+6;     //13.5 – тук точката се интерпретира като десетична точка, а не знак за конкатениране!!!
```

```
print "Hello";     //Ok          но          print "Hello", "World!";     //Fail
                                             print 'Hello', 'World!';     //Fail
print "Hello". "World!";     print 'Hello'. 'World!';     //Ок
print (3+4).(6-7); //Ok     77
```

**ИЗВОД:**

**!!! Както се вижда от примерите, конкатенирането с точка при print предоставя възможност за извеждане на повече от един параметър!!!**

# echo() vs. print()

- Когато стрингът е с кавички, вие можете да включите и безпроблебно променливи в него. Ако използвате апострофи обаче и включите променливи в стринга - резултатът няма да е коректен!!!

...

```
$a=34; $b=56;
echo "<br>variable a=$a, variable b=$b"; //OK
print "<br>variable a=$a, variable b=$b"; //OK
echo '<br>variable a=$a, variable b=$b';  //Not OK
print '<br>variable a=$a, variable b=$b';  //Not OK
```

...

**Резултат:**

variable a=34, variable b=56

variable a=34, variable b=56

variable a=$a, variable b=$b

variable a=$a, variable b=$b

# Елементи на езика

- Литерали (низови константи)
- Идентификатори
- Езикови конструкции (резервирани думи)
- Инструкции (statements)
- Коментари
- Групирани инструкции (code blocks)

# Елементи на езика

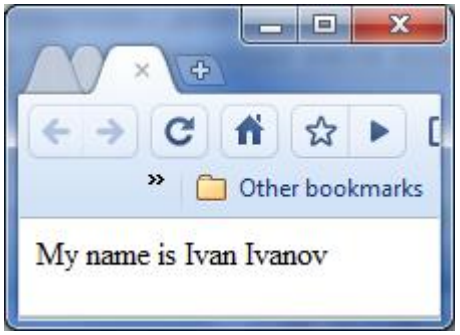
- Литерали (низови константи) – това са стойности, които се появяват директно в кода така както сме ги написали, например “Hello world”.
- В PHP низовите константи могат да се записват в 3 формата:
  - ограничени с двойни кавички
  - ограничени с апострофи
  - Низове в “here-doc” формат

# Низови константи, ограничени с двойни кавички

Този формат предоставя някои възможности, които другите формати не предоставят, а именно:

- включване на управляващи печата символи: символни последователности започващи с \, например \n – преход на нов ред, \r, \t, \\, \", \' и др.(описани са на стр. 33 от учебника ви).
- включване на имена на променливи, които при компилиране се заместват със своята стойност:

```
...
$name="Ivan Ivanov";
echo "My name is $name";
```



# Низови константи, ограничени с апострофи

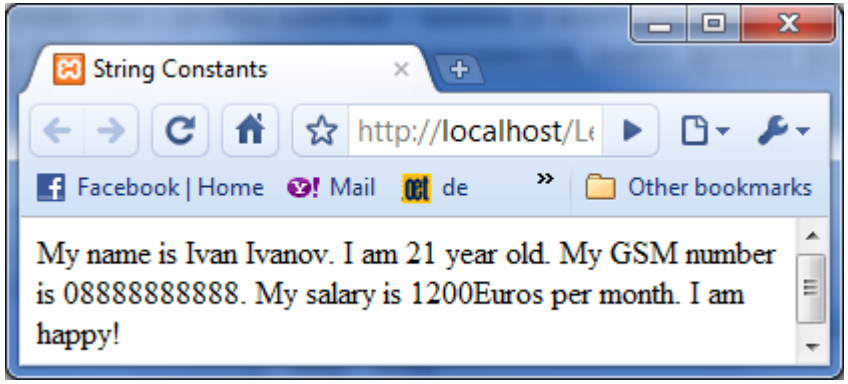
- извежда се каквото е написано;
- Единствено се поддържат 2-та управляващи символа: \\,  \’, тоест обратна наклонена черта и апостроф.

# Низови константи, в „here-doc” формат

- В „here-doc” формат, низът започва с <<<DELIMITER, следва многоредов текст и завършва с DELIMITER, като DELIMITER не трябва да се среща никъде другаде, например:

```
<?PHP
$s=<<<SomeThing
My name is Ivan Ivanov.
I am 21 year old.
My GSM number is 08888888888.
My salary is 1200Euros per month.
I am happy!
SomeThing;
echo $s;
?>
```

Listing 1.4



# Елементи на езика

- Идентификатори

Идентификаторите в PHP са просто имена (могат да започват с долна черта и да съдържат букви (латиница – малки и големи букви), цифри и долна черта.

С тях се именуват:

- Функции: do_something() – обект на следващи лекции
- Класове: class Foo {...} - обект на следващи лекции
- Променливи, пример: $var_name, $_num, $n2.
  - *Имената на променливите започват със знак $,*
  - *Не се декларират явно не се указва типа им;*
  - *Създават се при първото използване и сами си разбират типа, в зависимост от присвоената стойност,* $n2=5;
  - *Чувствителни към регистъра т.е. $help и $HeLp са различни променливи.*

# Променливи - инициализация

Инициализацията на променливите не е задължителна, но се счита за добра практика. Стойността на неинициализираните променливи зависи от контекста в който се използват. Така по подразбиране за променливите от:

- **логически тип - подразбиращата се стойност е FALSE,**
- **за числов тип (integer, float) – 0,**
- **за тип string – празен низ,**
- **за масив (тип array) – празен масив.**

Инициализиране на променливи може да се прави:

- **със стойност ( $y = $x ) ,**
- **с адрес ( $y = &$x )**
- **или индиректно**: съдържанието на променливата се използва като име на променлива:

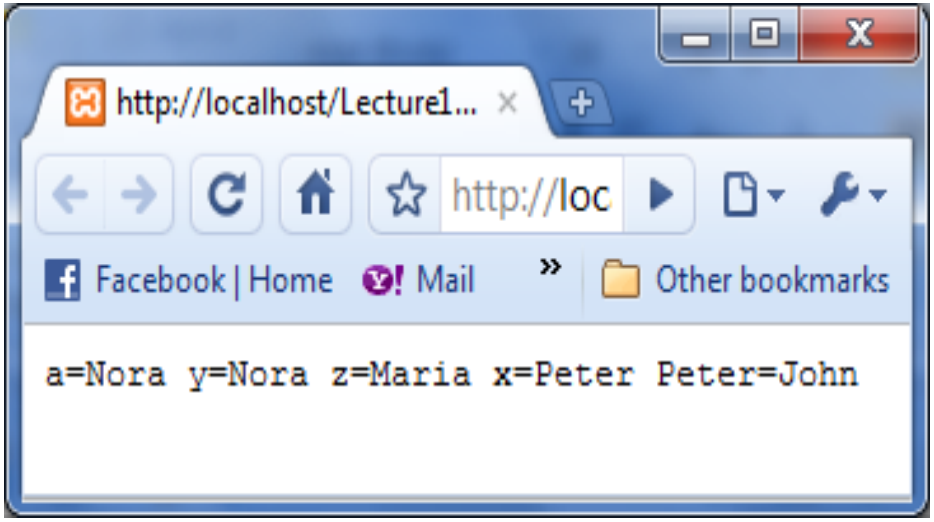
```
$x = 'Peter';
$$x = 'John';
echo $Peter; //ще  изведе John
```

Забележка: съдържанието на променливата **$x** трябва да е символен низ, който да отговаря на изискванията за име на променлива. Последната възможност за инициализация не се среща в други езици и е една от големите предимства на PHP.

# Променливи - инициализация

Пример: Инициализиране на променливи:

```
<?php
$a = 'Nora';
$y=$a;
//инициализация със стойност
$name='Maria';
$z = &$name;
//инициализация с адрес
echo "a=",$a," y=",$y, " z=",$z;
$x = 'Peter';
//индиректна инициализация - съдържанието на променливата $x се използва като име на променлива
$$x = 'John'; //създава се $Peter='John'
echo " x=".$x." Peter=".$Peter; //$Peter ще  изведе John
?>
```




Listing 1.5

# Управление на променливите

- **isset (var1[, var2,….])** - връща TRUE, ако var1 е установена (съществува и е инициализирана) и има стойност различна от NULL.
- **is_null($a)** - връща TRUE ако променливата $a има стойност NULL.
- **empty (var)** – връща TRUE, ако променливата var не съществува или не е установена (тоест има нулева стойност - 0, „0”, ””, FALSE, масив с нулеви стойности ) или има стойност NULL;
- **unset ( var1 [, var2,….])** – унищожаване на стойността на променливите. Поведението на unset() във функция варира в зависимост от това какъв тип променлива искате да унищожите. Ако глобална променлива се унищожава с unset() вътре във функцията, само локалната променлива се разрушава, а променливата във викащата функция не губи стойността си:

Пример:

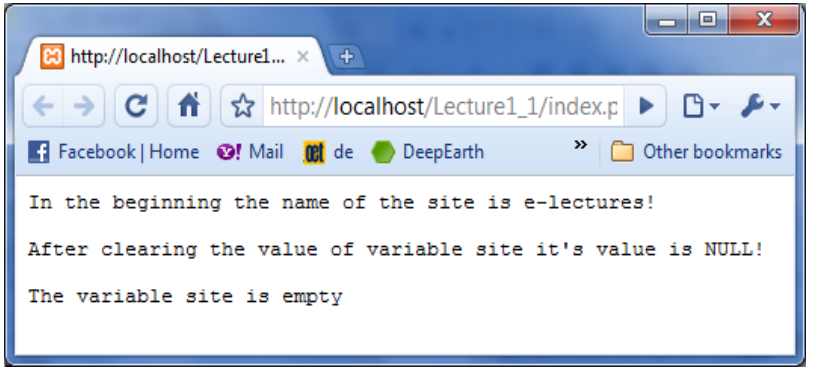
```
<?php
function destroy_foo()
{   global $foo;
    echo $foo;    //bar
    unset($foo);
 /*echo $foo;
...Undefined variable:
foo ...*/
}

$foo = 'bar';
destroy_foo();
echo $foo;
?>//bar
```

# Управление на променливите

Пример: използване на isset ( ), unset()  функции и empty():

```
<?php
$site = 'e-lectures';  //<p>This is some text in a paragraph.</p>
echo "<p>"."In the beginning the name of the site is
    $site!"."</p>";
unset($site);
if (isset($site))
{echo "<p>".$site."</p>";}
else
{echo "<p>"."After clearing the value of variable site
    it's value is NULL! "."</p>";}
if (empty($site))
{echo "<p>The variable site is empty</p>";}
else
{echo "<p>The variable site is Not empty</p>";}
?>
```




Listing 1.6

# Елементи на езика - Идентификатори (продължение)

– Константи (стойността на константата не може да бъде променяна)

- След като бъдат декларирани, константите са достъпни глобално. Не се налага да ги пре-декларирате във всяка функция и PHP файл.
- За деклариране на константа се използва define:

define( “име_на_константа”, стойност[, чувствителност_към_регистъра=false]),

например: **define("NAME", "Ivan");** или

**define('NAME', 'Ivan');** или **define('AGE', 25);**

**define('FAMILY', 'Petrov',true);** //true - non-sensitive

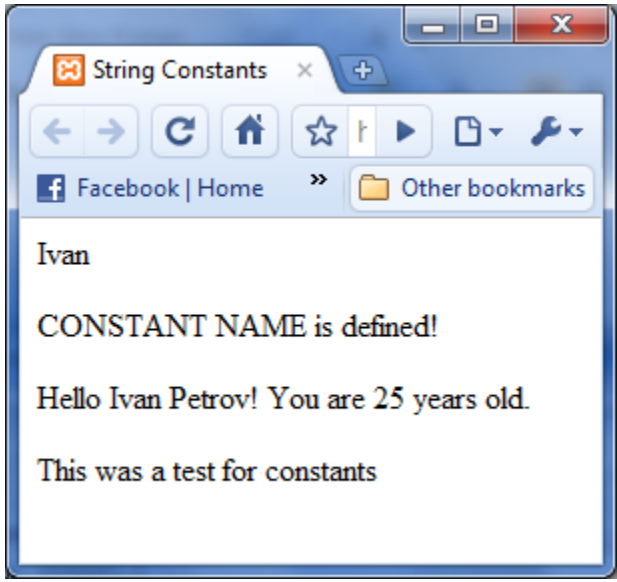
# Идентификатори на константи (продължение)

- Стойността на една константа се извлича чрез името и, но също така и чрез функция constant("име на константа") или constant('име на константа').
- Проверка за дефинирането на константа може да стане чрез функция defined("име на константа") или defined('име на константа'):

- Забележка: Забележете, че константите в показания по-долу пример не са в кавичките, а извън тях.

# Идентификатори на константи - пример

```
<?php
// define and print constant NAME
define("NAME", "Ivan");
echo constant("NAME");
if(defined("NAME"))
   { echo("<p>CONSTANT NAME is defined!</p>"); }
// define a non-sensitive constant FAMILY
define('FAMILY', 'Petrov',true); //true - non-sensitive
define('AGE', 25);
echo "Hello ". NAME . " " . family. "! You are " . AGE
     . " years old. "; //конкатенация
// echo "Hello ", NAME , " " , family, "! You are " ,
     AGE , " years old. ";       //би могло да е и така
define("CONSTANT2","\x20"); //\x20 - kod for blank
echo "<p>"."This was".CONSTANT2."a test for
     constants"."</p>"; // конкатенация
?>
```



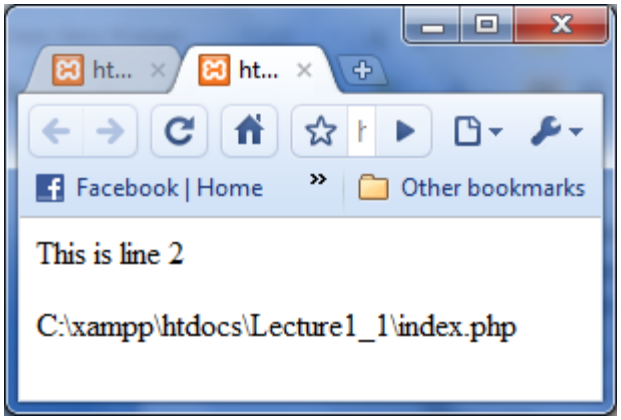

Listing 1.7

# Логическите и и магически константи

- Логическите константи имат стойности TRUE и FALSE.
- Съществуват и магически константи, чиито стойности се променят в зависимост от това къде се изпозват, например:
  - __LINE__ - връща номера на програмния ред, в който се съдържа (echo __LINE__; );

Пример:

```
<?php
echo "<p>"."This is line ". __LINE__."</p>";
echo __FILE__ ;
//echo __FUNCTION__;
// echo __CLASS__;
//echo __METHOD__;
?>
```




Listing 1.8

  - __FILE__ : връща спецификацията на файла на php модула, във вида: C:\...\.php;
  - __FUNCTION__ - връща името на функцията, ако се съдържа във функция;
  - __CLASS__ - връща името на класа, ако се съдържа в клас;
  - __METHOD__- връща името на метода, ако се съдържа в метод;

# Функция get_defined_constants()

- PHP предоставя функцията get_defined_constants(), която връща асоциативен масив, който съдържа двойки:

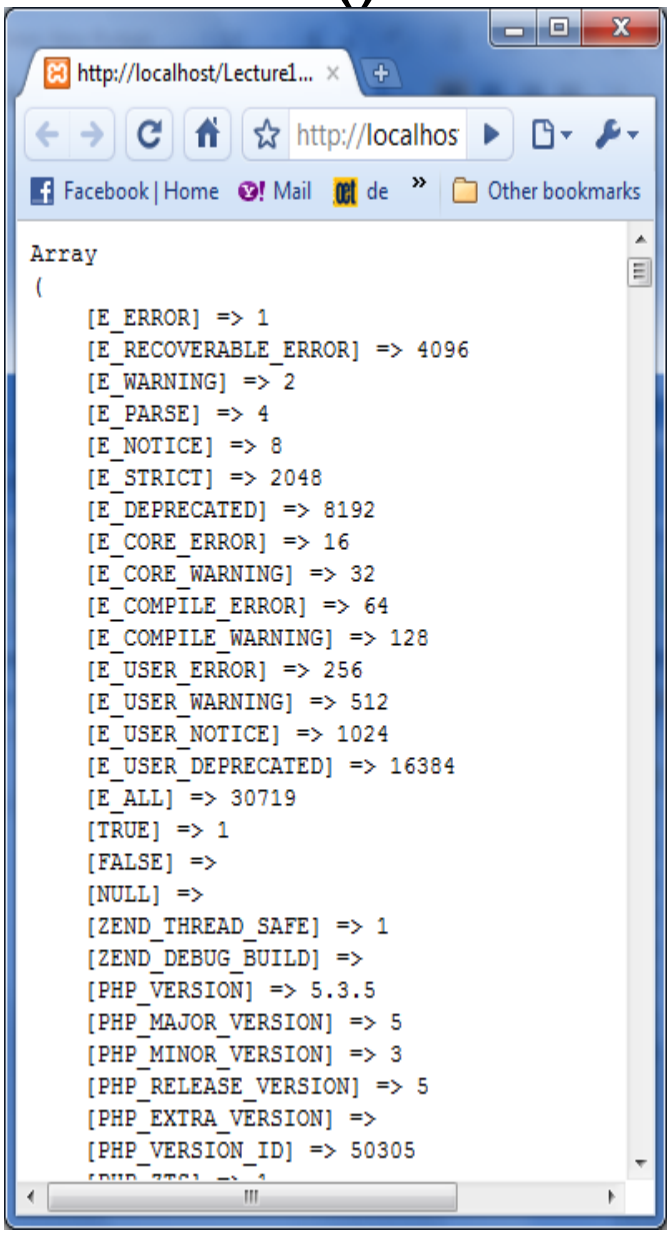
< име на константа, стойност >

за всички декларирани константи.

- За извеждането на масива е добре да използваме Html <pre> таг за преформатиране и функция print_r($array_name), която извежда масив в удобна за потребителя форма: Пример:

Listing 1.9

```
<pre>
<?php
print_r(get_defined_constants());
?>
</pre>
```



# Елементи на езика

- Езикови конструкции (резервирани думи)
  - Резервирани думи от PHP: думи, които не могат да се използват за именуване на променливи, константи, функции и класове:

| | | | | |
|---|---|---|---|---|
| and | or | xor | __FILE__ | exception (PHP 5) |
| __LINE__ | array() | as | break | case |
| class | const | continue | declare | default |
| die() | do | echo() | else | elseif |
| empty() | enddeclare | endfor | endforeach | endif |
| endswitch | endwhile | eval() | exit() | extends |
| for | foreach | function | global | if |
| include() | include_once() | isset() | list() | new |
| print() | require() | require_once() | return() | static |
| switch | unset() | use | var | while |
| __FUNCTION__ | __CLASS__ | __METHOD__ | final (PHP 5) | php_user_filter (PHP 5) |
| interface (PHP 5) | implements (PHP 5) | extends | public (PHP 5) | private (PHP 5) |
| protected (PHP 5) | abstract (PHP 5) | clone (PHP 5) | try (PHP 5) | catch (PHP 5) |
| throw (PHP 5) | cfunction (само PHP 4) | old_function (PHP 4 only) | this (само PHP 5) | |

# Елементи на езика

- Инструкции (statements):
  - Извикване на функции
  - Присвоявания на стойности на променливи
  - Извеждане на данни
  - Изрази (expressions) и др.
- Инструкциите се прекъсват по два начина:
    - точка и запетая - “;”
    - затварящ PHP таг - “?>”
- Последната инструкция преди затварящ PHP таг не изисква “;”

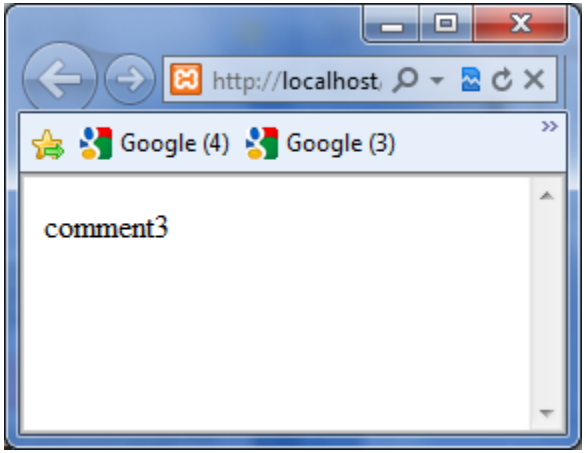
# Елементи на езика

- Коментари
  - Едноредови коментари: след // или #
  - Прекъсват се от \r, \n, \r\n, както и от затварящ PHP таг (?>)

```
<?php
$foo = 1; // Този коментар се прекъсва от края на реда
$bar = 2; # Този коментар се прекъсва от ?> това ще се отпечата
```

  - Многоредови Коментари

```
<?php
/* Коментар, който се простира на повече
от един ред */
$foo = 1; //comment 1
$bar = 2; #comment 2 ?> comment3
```



# Елементи на езика

- Групирани инструкции (code blocks)
  - Серия от инструкции, оградени с фигурни скоби

```
<?php
{
foo(); // извикване на функция
$bar = true; // присвояване на стойност
}
?>
```

  - Подходящи за указване на набор от инструкции, които да бъдат изпълнени само при определени условия
  - Могат да бъдат вложени

# Типове данни

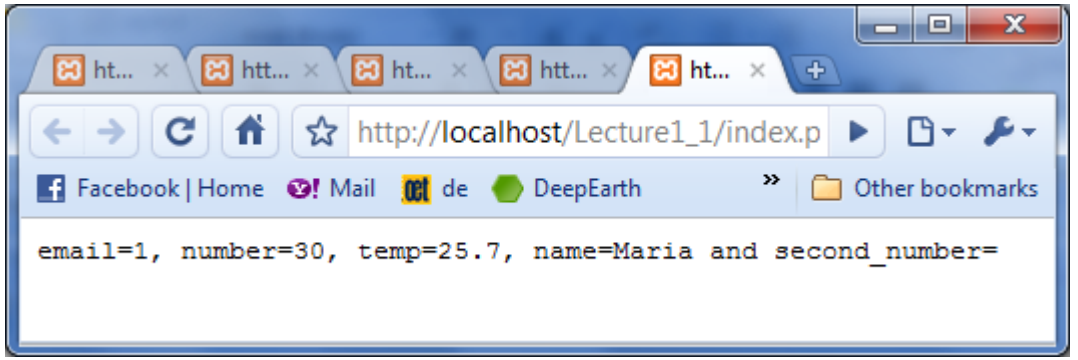
- PHP е слабо-типизиран език
  - Променливите могат да сменят типа на стойностите, които държат за периода на съществуването си
  - Променливите сменят типа на стойностите си в зависимост от контекста, в който се ползват, например:

```
<?php
$foo = 1; echo $foo;          //Извежда 1
$foo = 'Maria'; echo $foo; //Извежда Maria
?>
```

- Скаларни типове: int, float, boolean, string
- Съставни типове: array, обект
- Специални типове: NULL, resource
- Псевдо-типове: mixed, number, callback, void

# Скаларни типове данни - пример

```
<?php
// boolean: true or false)
$validEmail = true;
// integer
$number = 30;
// Floating point number -(float)
$temp = 25.7;
//string
$name = 'Maria';
//Empty type(NULL)
$second_number = NULL;
echo "email=$validEmail, number=$number,
    temp=$temp, name=$name and
    second_number=$second_number";
//email=1, number=30, temp=25.7, name=Maria
    and second_number=
?>
```




Listing 1.10

# Пояснения на примера

- Първата променлива е от булев тип данни и единствените стойности, които може да приеме са true(1) или false(0).
- Следващите две променливи са числени и за тях PHP поддържа два типа данни. Първият е за цели числа без плаваща запетая - 30, докато втория се използва за числа с плаваща запетая като 25.7.
- За нечислените типове PHP предлага низов тип, в който може да се съдържат всички букви, цифри, знаци и специални знаци. Константите както казахме вече задължително трябва да бъде ограждани от апострофи (‘) или (“) кавички.
- Също така имаме $second_number = NULL; Това показва, че променливата $second_number  е от тип **NULL, тоест тя не съдържа никаква информация**. Типът **NULL** е "специален" тип данни в PHP.

# Типове данни

- int (цели числа)
  - Цели числа със знак и без знак
  - Декларират се чрез няколко различни нотации:
    - **Десетична**: 10; -11; 1452
    - **Осмична**: 0666; 0100 (префикс 0)
    - **Шестнадесетична**: 0x123; 0XFF; -0x100 (префиксът 0x не е чувствителен към регистъра - case insensitive)

# Типове данни

- float (числа с плаваща запетая)
  - Още doubles, реални числа
  - Числа, които имат дробна част
  - Също могат да бъдат със знак и без знак
  - Декларират се чрез две различни нотации:
    - **Десетична**: 0.12; 1234.43; -.123
    - **Експоненциална**: 2E7, 1.2e2
      - Числото се умножава по 10 и се повдига на съответната степен.Например 1e2 е равно на 100
      - “E” не е чувствително към регистъра

# Типове данни

- boolean (булеви)
  - Съдържат една от две възможни стойности: TRUE или FALSE
  - Служат като основа на логически операции (предмет на друга лекция)
- string (низ)
  - Подреден списък от знаци
  - PHP има изключително много функции за манипулиране на низове, което е обект на друга лекция

# Типове данни

- array (масив)- Масивите в PHP са асоциативни, те са съвкупност от подредени асоциации (двойки ключ-стойност), за разлика от масивите в другите езици, където са индексирани.
- Стойностите в масивите (числа, boolean, string, object, array) се идентифицират по позиция (0), или именован низов идентификатор
- Създаване на масив - езиковата конструкция array():
  - $my_array = array('foo', 'bar'); тоест: $my_array[0] = 'foo'; $my_array[1] = 'bar';
  - $a=array(3=>"Peter", 'b'=>"John","Horry"); тоест: $a[3] ="Peter”, $a[‘b’] =“John”, $a[4] =“Horry”.
- В PHP има множество функции за работа с масиви, но това е предмет на друга лекция

# Типове данни

- Класове (подробно се разглеждат в друга лекция). Включват:
  - **променливи** (наричани fields в другите езици), но в PHP се наричат атрибути или свойства (properties);
  - **функции**, които описват методите и извършват операции с вътрешните данни на класа;
  - **манипулатори на събития** – специален тип променливи, на които се присвоява адреса на функции, дефинирани извън класа.
- resource (ресурсен идентификатор)
  - Идентифицират външни ресурси, които принципно не са вградени в PHP, но придобиват значение в контекста на някои специални операции, като например работа с файлове, БД и др.
- NULL
  - За първи път тип NULL е включен в PHP4 и се използва за създаване на "празни" променливи в PHP. Променлива от тип NULL не съдържа никаква информация, $a=NULL указва че променливата $a няма стойност.
  - Променлива има стойност NULL, когато й е била присвоена стойността NULL, или когато изобщо не й е била присвоена стойност (is_null()).

**!Важно**: Неправилно е да се мисли, че променлива, чиято стойност е "" е равна на променлива чиято стойност е NULL. Това НЕ е вярно! Ако искате да дефинирате празна променлива използвайте типа данни NULL.

# Други типове данни

- mixed
  - Указва, че параметърът (на функция) може да бъде променлива от различни (не задължително всички) типове
- number
  - Указва, че параметърът може да бъде както int, така и float
- callback
  - Указва, че параметърът е произволна функция, дефинирана от потребителя или метод на обект. В някои случаи се задава като низ, а в други – като масив
- void
  - Като тип за връщане (от функция) показва, че функцията не връща стойност
  - В контекст на параметър показва, че функцията не приема никакви параметри

# Проверка за типа на променлива: bool is_type(var);

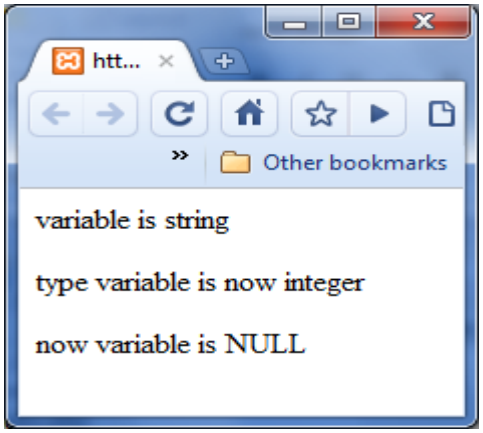
- **is_bool()** - проверява дали дадена променлива е от булев тип. Връща true, ако променливата е от тип bool.

Аналогично:

- **is_numeric()** - проверява дали дадена променлива е от числов тип
- **is_int()** - проверява дали дадена променлива е от целочислен тип
- **is_float()** - проверява дали дадена променлива е от тип float
- **is_string()** - проверява дали дадена променлива е от тип string
- **is_null()** - проверява дали дадена променлива е "празна”
- **is_array()** - проверява дали дадена променлива е масив
- **is_object()** - проверява дали дадена променлива е обект

# Проверка за типа на променлива: функции var_dump() и gettype()

- gettype() - връща 'boolean', 'integer', 'double', 'string', 'array', 'object', 'resource', 'NULL', 'unknown type‘




Listing 1.11

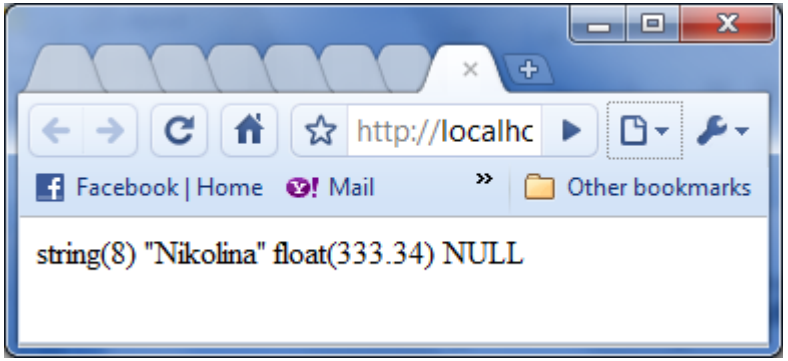
```
<?php
$variable = 'Type';
echo "<p>variable is ".gettype($variable)."</p>";
$variable = 333;
echo "<p>type variable is now ".gettype($variable)."</p>";
$variable=Null; // или NULL
if (is_null($variable))
{echo "<p>now variable is ".gettype($variable)."</p>";} //NULL
?>
```

# Проверка за типа на променлива: функции var_dump() и gettype()

- var_dump() - за проверка на типа и стойността на даден израз

Listing 1.12

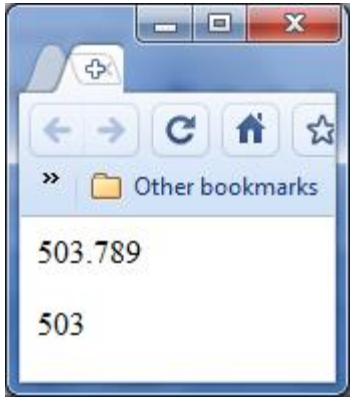
```
<?php
$variable = 'Nikolina';
echo var_dump($variable);
$variable = 333.34;
echo var_dump($variable);
$variable=Null; // или NULL
echo var_dump($variable);
//string(8) "Nikolina" float(333.34) NULL
?>
```



# Преобразуване от един тип данни в друг

- В **PHP** можете да извършвате преобразувания между типовете данни, като например число с плаваща запетая да превърнете в цяло число.
- Вижте примера за да разберете как става това:

```
<?php
$speed = 503.789;
echo "<p>$speed</p>";
$intSpeed = (integer)$speed;
// или $intSpeed = (int)$speed;
echo $intSpeed;
?>
```



# Преобразуване от един тип данни в друг - продължение

- Преобразуване на типа на променлива (type casting)

$foo = 10;

$bar = (boolean) $foo // $bar holds TRUE

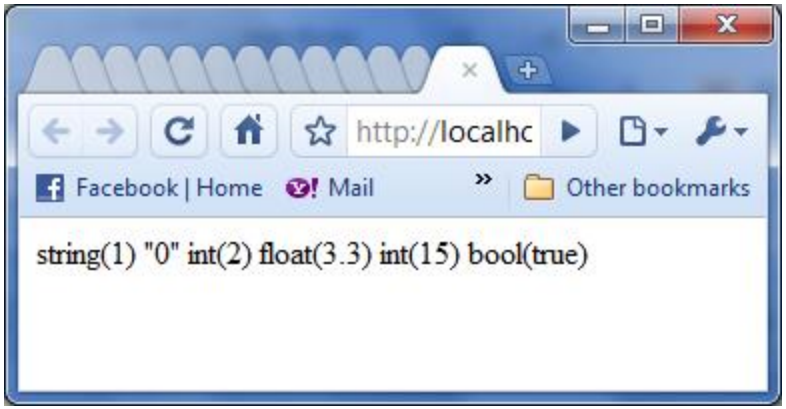
  - (int), (integer), (bool), (boolean)
  - (float), (double), (real)
  - (string), (binary) // PHP 6+
  - (array), (object)

- Промяна на типа на променлива
  - $bar = 2;
  - settype($bar,'string');

# Преобразуване от един тип данни в друг

- Автоматично преобразуване на типовете

```
<?php
$foo = "0";  // низ - "0"
echo var_dump($foo);
$foo += 2;  // цяло число (стойност=2)
echo var_dump($foo);
$foo = $foo + 1.3;  // float(стойност=3.3)
echo var_dump($foo);
$foo = 5 + "10 apples"; // цяло число (стойност=15)
echo var_dump($foo);
$bar = (boolean) $foo; // bool - стойност=TRUE
echo var_dump($bar);
//string(1) "0" int(2) float(3.3) int(15) bool(true)
?>
```




Listing 1.13

# Оператори

- Могат да се разграничат следните видове
  - Оператори за присвояване
  - Аритметични оператори
  - Оператори за низове
  - Оператори за сравнение
  - Логически оператори
  - Побитови оператори
  - Оператори за контрол на грешките
  - Оператори за изпълнение
  - Инкрементиращи / декрементиращи оператори
  - Оператори за типове
  - Оператори за масиви

# Някои оператори в PHP стр.35-37

- Аритметични: - +, - , *, /, % (остатък от деление);
- Логически: &&,||, ! – и, или, не.
- Оператор за съединяване на низове (.)

```
$string = 'Ana' . ' Ivanova'; // 'Ana Ivanova'
$string2 = ' Dimova';
$string .= $string2;              // 'Ana Ivanova Dimova‘
```

- Условен оператор:

*условие ? израз1 : израз2;*

Пример:
```
<?php $a=5;
        $b=($a) ? (1) : (2);        echo $b;
        ?>
```
*Резултат: 1*

Ако стойността на $a е TRUE (а тук е така), целия израз се изчислява като 1, в противен случай - 2.

# Оператори за сравнение

| Пример | Наименование | Резултат |
|---|---|---|
| $a == $b | Равно | TRUE ако $a е равна на $b. |
| $a === $b | Идентично | TRUE ако $a е равна на $b и ако са от един и също тип. (въведено в PHP 4) |
| $a != $b | Не-равно | TRUE ако $a не е равна на $b. |
| $a <> $b | Не-равно | TRUE ако $a не е равна на $b. |
| $a !== $b | Не-идентично | TRUE ако $a не е равна на $b или ако не са от един и същи тип. (въведено в PHP 4) |
| $a < $b | По-малко | TRUE ако $a е строго по-малка от $b. |
| $a > $b | По-голямо | TRUE ако $a е строго по-голяма от $b. |
| $a <= $b | По-малко или равно | TRUE ако $a е по-малка или равна на $b. |
| $a >= $b | По-голямо или равно | TRUE ако $a е по-голяма или равна на $b. |

```
<?php
$a = 5; $b='5';$c=5;
if ($a == $b) echo "a==b!!!  ";
if ($a !== $b) echo " a!==b!!!  ";
if ($a === $c) echo " a===c!!!  ";
//a==b!!! a!==b!!! a===c!!!
?>
```

# Логически Оператори

| Пример | Наименование | Резултат |
|---|---|---|
| $a and $b | И | TRUE ако и $a и $b са TRUE. |
| $a or $b | Или | TRUE ако $a или $b е TRUE. |
| $a xor $b | Изключващо или | TRUE ако или $a или $b е TRUE, но не и двете |
| ! $a | Не | TRUE ако $a не е TRUE. |
| $a && $b | И | TRUE ако и $a и $b са TRUE. |
| $a \|\| $b | Или | TRUE ако $a или $b е TRUE. |

- Комбинират 1 или 2 булеви стойности и като резултат от израза се получава 3-та, отново булева стойност
- Ако левият операнд на 'AND' се оцени като FALSE, интерпретаторът изобщо няма да оценява десния
- Причината за съществуването, както на 'and' и 'or', така и '&&' и '||' се крие в различния им приоритет. Следващите примери са еквивалентни:

```
$x and $y || $z
$x && ($y || $z)
$x and ($y or $z)
```

# Оператори

- Приоритет на операторите
  - Редът, в който операторите в един израз се оценяват се определя от техния относителен приоритет

3 + 2 * 5; // операторът за умножение има по-висок приоритет, резултат 13

От предния пример стана ясно, че имаме &&, || са с по-висок приоритет от and, or.

- Асоциативност на операторите
  - Редът, в който операторите с един и същи приоритет се оценяват се определя от тяхната асоциативност

2 / 2 * 2; // резултатът е 2 понеже '*' и '/' са с лява асоциативност

$a = 1; $b = 2;

$a += $b += 3; // дясна асоциативност: $b += 3; $a += $b;

Резултат: $a = 6; $b = 5

# Оператори за инкрементиране и декрементиране

| Пример | Наименование | Резултат |
|---|---|---|
| ++$a | Предварително инкрементиране | Увеличава $a с едно, след което връща $a. |
| $a++ | Последващо инкрементиране | Връща $a, след което увеличава $a с едно. |
| --$a | Предварително декрементиране | Намалява $a с едно, след което връща $a. |
| $a-- | Последващо декрементиране | Връща $a, след което намалява $a с едно. |

- Приложими са само върху променливи
- Инкрементиране на нулеви (NULL) стойности връща 1, декрементирането им не е възможно
- Низови стойности (само от ASCII диапазона) се инкрементират, но не се декрементират

```
$str = 'Y';
echo ++$str; // 'Z'
echo ++$str; // 'AA'
echo ++$str; // 'AB'
```

```
<?php
$a=5;
echo --$a; //4
echo $a;   //4
?>
```

```
<?php
$a=5;
echo $a--; //5
echo $a;   //4
?>
```

- Оператор за потискане на извеждане на грешка - @;

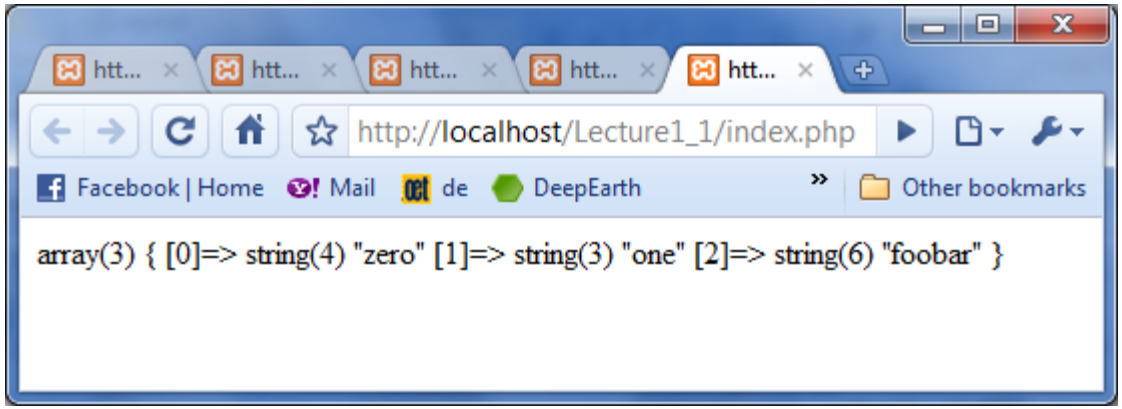
# Оператори

| Асоциативност | Оператори | Информация |
|---|---|---|
| без асоциативност | new | new |
| лява | [ | array() |
| без асоциативност | ++ -- | инкрементиране/декрементиране |
| без асоциативност | ! ~ - (int) (float) (string) (array) (object) @ | типове |
| без асоциативност | instanceof | типове |
| дясна | ! | логически |
| лява | * / % | аритметични |
| лява | + - . | аритметични и низови |
| лява | << >> | побитови |
| без асоциативност | < <= > >= | сравнителни |
| без асоциативност | == != === !== | сравнителни |
| лява | & | побитови и референции |
| лява | ^ | побитови |
| лява | \| | побитови |
| лява | && | логически |
| лява | \|\| | логически |
| лява | ? : | третични |
| дясна | = += -= *= /= .= %= &= \|= ^= <<= >>= | присвоителни |
| лява | and | логически |
| лява | xor | логически |
| лява | or | логически |
| лява | , | множество употреби |

# Оператори за масиви

| Пример | Наименование | Резултат |
|---|---|---|
| $a + $b | Обединение | Обединението на $a и $b. |
| $a == $b | Равенство | TRUE ако $a и $b имат едни и същи двойки ключ/стойност. |
| $a === $b | Идентичност | TRUE ако $a и $b имат едни и същи двойки ключ/стойност в един и същи ред и са от един и същи тип. |
| $a != $b | Неравенство | TRUE ако $a не е равно на $b. |
| $a <> $b | Неравенство | TRUE ако $a не е равно на $b. |
| $a !== $b | Неидентичност | TRUE ако $a не е идентично на $b. |

```php
<?php
$a = array('0' => 'zero', '1' => 'one');
$b = array('0' => 'foo', '1' => 'bar', '2' => 'foobar');
var_dump($a + $b);
//array(3) { [0]=> string(4) "zero" [1]=> string(3) "one"
//[2]=> string(6) "foobar" }
?>
```



# Оператори за масиви

| Пример | Наименование | Резултат |
|---|---|---|
| $a + $b | Обединение | Обединението на $a и $b. |
| $a == $b | Равенство | TRUE ако $a и $b имат едни и същи двойки ключ/стойност. |
| $a === $b | Идентичност | TRUE ако $a и $b имат едни и същи двойки ключ/стойност в един и същи ред и са от един и същи тип. |
| $a != $b | Неравенство | TRUE ако $a не е равно на $b. |
| $a <> $b | Неравенство | TRUE ако $a не е равно на $b. |
| $a !== $b | Неидентичност | TRUE ако $a не е идентично на $b. |

```
$a=array(1,2,3);
$b=array('1',2,3);
    if ($a==$b) echo "a==b";
    if ($a!==$b) echo "  a!==b";
//a==b, a!==b
```

# Променливи - обхват на действие

- Локален vs. глобален обхват.
- Променливите в PHP имат строго определен обсег на действие. Когато една променлива бъде декларирана във функция, тя може да бъде използвана само от тази функция и се нарича локална променлива.
- Друг вид променливи са т.нар. глобални променливи, които могат да се ползват от всички функции в скрипта. За да се възприеме една променлива като глобална е нужно вътре във функцията изрично да се декларира, че тя е такава. Това се прави като пред името на променливата, употребена в дадена функция, се пише ключовата дума global или с масива $GLOBALS . Ако това не бъде направено PHP ще възприеме променливата като локална по подразбиране. В следващия пример променливата $a е декларирана като глобална във функцията **my_finc()**:

# Променливи - обхват на действие

- Локален vs. глобален обхват - пример

```
<?php
function my_finc()
{   global $a;
    echo $a;    //1
    echo $b;   //локален обхват, $b = null
                //Notice: Undefined variable...
}
$a = 1; $b=2;
echo $a; //1
echo $b; //2
my_finc();
?>
```

Достъпът до външна променлива може да стане и с масива $GLOBALS:

```
<?php
$a = 1;
$b = 2;
function Sum()
{    $GLOBALS['b'] += $GLOBALS['a'];}
Sum();
echo $b; //3
?>
```

## Масив $GLOBALS: Съдържа референция към всички променливи, налични в глобалния обхват

```
<?php
function test()
{   $foo = "локална променлива";
    echo "<br>".'$foo в глобален обхват: ' . $GLOBALS["foo"] .
    "\n";
    echo "<br>".'$foo в текущия обхват: ' . $foo . "\n";
}
$foo = "Примерно съдържание";
test();
//$foo в глобален обхват: Примерно съдържание
//$foo в текущия обхват: локална променлива
?>
```

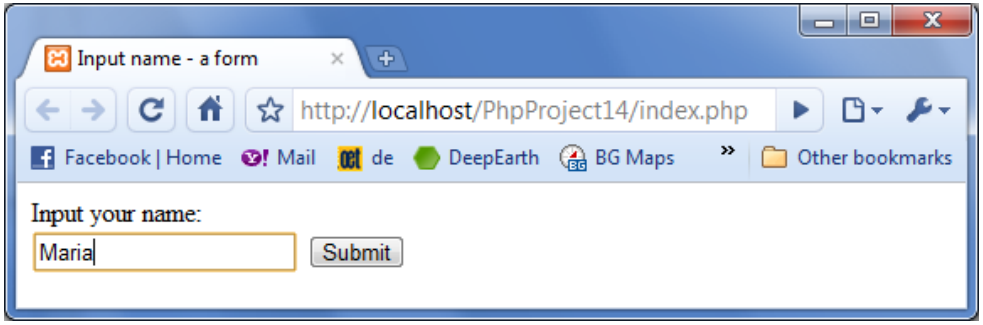
# Супер-глобални променливи (Superglobals)

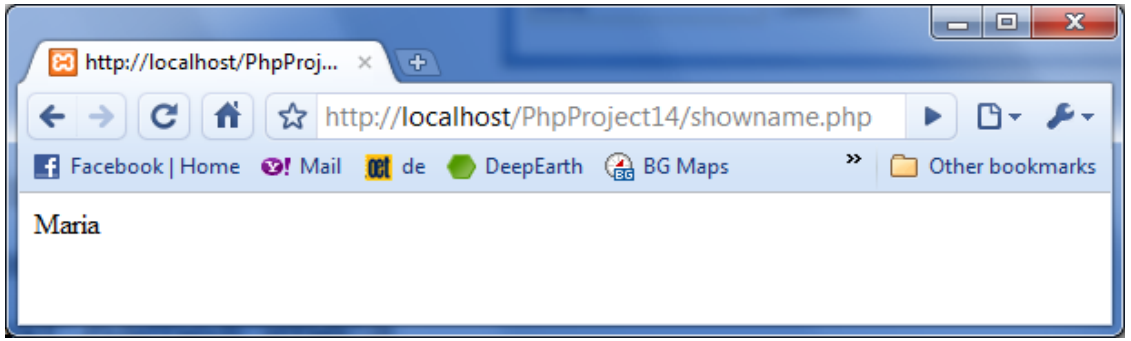
- Съществуват и още един вид променливи - така наречените **суперглобални** променливи, които са предварително дефинирани в езика и могат да бъдат ползвани в произволен брой скриптове. $_POST и $_GET са именно такива променливи. Те са от типа променливи array (масив), затова освен суперглобални променливи биват наричани още **суперглобални масиви** (superglobal arrays). PHP не разполага с механизъм чрез който да създавате собствени суперглобални променливи, т.е. могат да се ползват единствено тези, които са предварително вградени в езика.
- Чрез **суперглобални масиви** $_POST и $_GET може да бъде достъпвана информацията, която потребителят въвежда в HTML формуляр. Когато даден потребител попълни съответния HTML формуляр и натисне бутона за извършване на действието (Submit), въведените във формуляра данни биват изпратени за обработка от PHP скрипта, който е указан чрез атрибут action във HTML формуляра. Чрез масивите $_POST и $_GET, PHP (или $_REQUEST, който може да бъде използван вместо тях), PHP скриптът достъпва попълнената във формуляра информация.

# Супер-глобални променливи (Superglobals) - пример

**form.php**

```
<html>
<head>
<title>Input name - a form
    </title>
</head>
<body>
<form action="showname.php"
    method="POST">
Input your name:<br />
<input type="text"
    name="ime">
<input type="submit"
    value="Submit">
</form>
</body>
</html>
```




**showname.php**

```
<?php
echo $_POST["ime"];
?>
```

**Имаме формуляр form.php (или form.html) и обработващ скрипт showname.php**

# Супер-глобални променливи (Superglobals) - пример

**Имаме формуляр form.php (или form.html) и обработващ скрипт showname.php.** При написване на нещо в полето на формуляра и натискане на бутона “Submit" обработващият скрипт **showname.php** ще покаже въведената от потребителя информация. Как става това?

Чрез масива $_POST в **showname.php** вземаме от формуляра стойността на полето с название ime (имаме $_POST["ime"]).

Названието на полето, в случая ime, представлява "ключ", чрез който може да се обърнем и да достъпваме стойността, към която "ключа" сочи. Тази стойност ще бъде името, което потребителят въвежда във формуляра. Съответно echo $_POST["ime"] **извлича** стойността, сочена от ключа ["ime"] и я извежда на екрана.

Повече информация за ключовете и стойностите на масивите ще получите в една от следващите лекции.

# Супер-глобални променливи (Superglobals)

Освен $_GET, $_POST и $_REQUEST, други предварително дефинирани променливи, винаги достъпни - с глобална видимост са:

- $_COOKIE
- $_FILES
- $_SERVER
- $_ENV
- $_SESSION

# Предварително дефинирани (вградени в езика) променливи

- Пълен списък с променливите на средата може да видите като използвате вградената функция phpinfo(). Направете си php страница в която сложете следния код

```
<?php
phpinfo();
?>
```

- В изведената информация ще видите списък на променливите на средата със съответните им стойности, показването на характеристиките на сървъра и PHP софтуера, с който работи вашия компютър.

# Информация за обкръжението - PHP $_SERVER[]

- $_SERVER е PHP супер глобален  масив, съдържащ информация за средата (или обкръжението), като заглавия (headers), пътища и местоположения на скриптове.
- Елементите на този масив се създават от уеб сървъра. Няма гаранция, че всеки уеб сървър ще предостави който и да е от тези елементи; сървърите може да пропуснат някои, или да предоставят други, които не са описани тук.

- **Чрез елементите на $_SERVER[] може да се извлече различна информация за посетителя на една страница - какъв е IP адреса и URL му (REMOTE_ADDR, HTTP_REFERER) , каква операционна система и браузър ползва (HTTP_USER_AGENT ) и т.н:**
- **$_SERVER['HTTP_USER_AGENT'] - дава информация за браузъра и ОС на посетителя (напр. Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1;...)**
- **$_SERVER['REMOTE_ADDR'] - дава информация за IP адреса на посетителя, например: 127.0.0.1**
- **$_SERVER['SERVER_SOFTWARE'] - дава информация за използвания web сървър (напр. Apache/2.2.17 (Win32) )**
- **$_SERVER['PHP_SELF'] - връща името и пътя на текущо изпълнявания скрипт (спрямо root folder-a, който при xampp инсталация е htdocs). Например, /PhpProject15/index.php), относително спрямо коренната директория за документи (document root) - htdocs.**

- **$_SERVER['SERVER_NAME' ]** - дава информация за името на сървърния хост (например, **localhost**), под който се изпълнява текущият скрипт. Ако скриптът работи на виртуален хост, това ще бъде стойността, дефинирана за този виртуален хост.
- **$_SERVER['HTTP_REFERER' ]** - дава информация за адреса (**URL-а**) на страницата (ако има такава), която е стартирала текущата страница (т.е дава информация за адреса на посетителя). Този елемент се попълва от браузъра.
- И др.

**Примери на:**

**http://php.net/manual/bg/reserved.variables.server.php**
**http://www.w3schools.com/php/php_superglobals.asp**
**http://www.w3resource.com/php/super-variables/$_SERVER.php**

# Информация за средата (обкръжението) чрез PHP $_SERVER[] - пример

```
<?php
echo "SERVER_NAME=".$_SERVER['SERVER_NAME'];
echo "<br>";
echo "PHP_SELF=".$_SERVER['PHP_SELF'];
echo "<br>";
echo "SERVER_ADDR=".$_SERVER['SERVER_ADDR'];
echo "<br>";
echo "REMOTE_ADDR=".$_SERVER['REMOTE_ADDR'];
echo "<br>";
echo "Вашия web сървър е
      ".$_SERVER['SERVER_SOFTWARE'];
echo "<br>";
echo "Вашите браузър и операционната система са
      ".$_SERVER['HTTP_USER_AGENT'];
echo "<br>";
if(isset($_SERVER['HTTP_REFERER'])) {
          echo "идвате от ";
 echo $_SERVER['HTTP_REFERER'];}
?>
```

SERVER_NAME=localhost
PHP_SELF=/PhpProject15/index.php
SERVER_ADDR=127.0.0.1
REMOTE_ADDR=127.0.0.1

Вашия web сървър е Apache/2.2.17 (Win32) mod_ssl/2.2.17 OpenSSL/0.9.8o PHP/5.3.4 mod_perl/2.0.4 Perl/v5.10.1

Вашите браузър и операционната система са Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; Trident/5.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET CLR 3.0.30618; InfoPath.2; Media Center PC 5.0; SLCC1)