PHP

- Originally: *"Personal Home Page Tools"* (1995).
- Later became: *"PHP: Hypertext Preprocessor"*.
- Programming language for server-side scripting.
- Freely available under an open-source license, widely used.
- Alternative to Microsoft's ASP.NET language
  ↑ which was originally *"Active Server Pages"* (ASP).
- Latest version: PHP 5.6.8 (16 April 2015).

PHP: Strong points

- It’s free.
- It runs on almost every web server.
- PHP scripts, too, are embedded in HTML.
- Its efficiency depends on the web server, not on the client machine.
- Its presence is transparent to the browser / web user.
  ⇒ No browser compatibility issues.
- Running server-side, it can access server-side information, such as company databases.
  ⇒ This is what it is most often used for.

PHP: Weak points

- *Because* it runs server-side, *cannot* interact with events local to the client, such as (user interface) pointer movements, etc.

PHP in practice

- As in JavaScript, script code is embedded in HTML.
- PHP script code is interpreted on the server side.
- PHP syntax is a mix of Java, C and Perl languages.
- Files may have the extension `.php`, so that the web server knows a file to be served should be processed by its PHP interpreter.

PHP: study material

Let's have a look at the required reading.

>

PHP: example

```
<HTML>
  <HEAD> ... </HEAD>
  <BODY>
    <H1>Example</H1>
    <?php echo 'Have a <B>nice</B> day!'; ?>
    (Today is <?php echo date('l, F dS Y.'); ?> .)
  </BODY>
</HTML>
```

- Note that:
  - The script is included *inside the* `<?php ... ... ... ?>` tag.
  - The `echo` statement prints to the HTML document served.
  - HTML tags can be in the output.
  - Every statement must end with the `;` character.

PHP: Comments in the code

- Always clarify your code with human-readable comments!

```
<HTML>
  <HEAD> ... </HEAD>
  <BODY>
    Dutch language lesson 1 <BR>
    <?php
      /*
          Begin with a useful sentence.
      */
      echo ‘<EM>Het is hier gezellig!</EM>’;
      // (This refers positively to the social atmosphere.)
    ?>
  </BODY>
</HTML>
```

- Comments are placed between /* … */ or after // .
- NB: These are not *HTML* comments, but *PHP* comments.

PHP: Variables

```
$dutch_dessert = 'vanillevla';    // string

$x = 28;                          // integer

$pi = 3.1415;                     // double

$valid_input = TRUE;              // boolean
```

- No declaration required: data type is automatically deduced.
- Variable names start with the `$` character.
- Variable names can contain letters, numbers and the underscore (`_`) character – but cannot start with a number.
- Variable names are case-sensitive!

  ⇒ `$Dutch_dessert` is not the same as `$dutch_dessert`.

PHP: String variable evaluation

- There is a difference between using single (‘) and double (“) quotes for strings.
  - With *double quotes*, variable names within the text string are substituted by their value.
  - With *single quotes,* they are **not**.

- Although error-prone, this is useful, an example:

```
$belgian_food = ‘friet’;
$korean_food = ‘kimchi’;

echo “Bart Peeters loves $belgian_food.”;
echo ‘We are using a variable $korean_food.’
```

- Output:

```
Bart Peeters loves friet.
We are using a variable $korean_food.
```

PHP: Strings

- To include nonstandard characters in a text string, you may have to use an escape character.

- The following statements will not work:

```
$wine  = 'Vin de Pays d'Oc';     // problem with quotes
$price = "My car costs $200.";   // problem with $-char
```

- Instead, do:

```
$wine = 'Vin de Pays d\'Oc';     // escape character
$wine = "Vin de Pays d'Oc";      // double quotes
$price = "My car costs \$200."; // escape character
```

PHP: String concatenation

- String concatenation: “pasting strings together”.
- Operator: **.** (dot).

- Example:

```
echo ‘Chocolade’ . “vla” . ‘<BR>’;

$first = ‘fiets’;
$second = ‘zadel’;
echo ‘<B>’ . $first . $second . ‘</B>’;
```

- Output:

```
Chocoladevla<BR>
<B>fietszadel</B>
```

PHP: Other operators

- For arithmetic, the standard (precedence):

```
$a = 5;
$b = 3;
$c = 6;

echo ($a - $b) * ($c / 2) + 28;
```

PHP: Assignments

- *As elsewhere:* important difference between assignment and comparison operators.

- Assignment operators are used to set (change) the value of a variable:

```
$number = 8;             // set variable $number to value 8
$vla = 'Choco';          // set variable $vla to value 'Choco'

$a += 3;                 // same as: $a = $a + 3;
$a -= $b * 2;            // same as: $a = $a - ($b * 2);
$b .= 'fiets';           // same as: $b = $b . 'fiets';
$c /= 2;                 // same as: $c = $c / 2;
```

PHP: Comparisons

- Comparison operators are used to compare two values.
  ⇒ They return `TRUE` or `FALSE`. *This too should be familiar:*

```
$a = 3;                // assignment
$b = 3;                // assignment
$c = $a + $b - 1;      // assignment ($c = 5)

$a == $b;              // comparison, gives TRUE
$a == $c;              // comparison, gives FALSE
$a != $c;              // comparison (unequal), gives TRUE
$a <= 4;               // comparison, gives TRUE
“fiets” != “Fiets”; // comparison (unequal), gives TRUE

($a < 5) && ($b >= 2);     // && is and, || is or
```

- As always, beware of accidentally using the **=** assignment operator when intending to compare values!

PHP: Arrays

- Array: the familiar data structure containing multiple items.
- *In PHP:* Arrays can contain elements of different data types.

```
$myarray = array('een', 2, 'drie');

echo $myarray[0];         // outputs 'een'
echo $myarray[1];         // outputs '2'

$myarray[1] = 'twee';     // changes element
$myarray[3] = 'vier';     // creates new element

$myarray[] = 'vijf';      // adds new element to array end
echo $myarray[4];         // outputs 'vijf'
```

PHP: Associative arrays

- Indices of arrays can be *strings* too!
- These are called associative arrays.
- *“A bit strange, but quite handy.”*

```
$country[‘friet’] = ‘Belgium’;
$country[‘kimchi’] = ‘Korea’;
$country[‘hummus’] = ‘UNDEFINED - POLITICALLY SENSITIVE’;

$dish = ‘kimchi’;
echo “$dish is from $country[$dish]”;
```

PHP and HTML forms

- *Remember:* data from HTML forms could be sent to the server using either a GET or a POST HTTP request...
- Suppose you make an HTML form with text input fields `firstname` and `lastname`, and use the GET method.
- After clicking the submit button, the requested URL would be something like:

```
http://www.abc.com/welcome.php?firstname=Jan&lastname=Jansen
```

↑ Note that the field names and values are encoded into the URL.

PHP and HTML forms

- A PHP script could handle this data as shown:

```
<?php
  $fname = $_GET[‘firstname’];
  $lname = $_GET[‘lastname’];

  echo “Welcome to this webpage, $fname $lname.”;
?>
```

- PHP automatically creates an *associative array* of all form input fields, called `$_GET`.

- The `$_GET` array contains all name/value pairs of the form fields.

PHP and HTML forms

- If your HTML form uses the POST method, the PHP script would handle it as shown:

```
<?php
  $fname = $_POST['firstname'];
  $lname = $_POST['lastname'];
  echo "Welcome to this webpage, $fname $lname.";
?>
```

- If you do not care whether it is a GET or POST method:

```
<?php
  $fname = $_REQUEST['firstname'];
  $lname = $_REQUEST['lastname'];
  echo "Welcome to this webpage, $fname $lname.";
?>
```

DIY assignment for the next lecture:

(1) Using your favorite plaintext editor and browser, write a simple HTML document from scratch.

(2) Make it contain a form, consisting of a text entry field and a submit button.

(3) Then write JavaScript code which on submit checks whether the user has actually entered any text.

(4) Next lecture, bring your code with you, on your favorite laptop (or other keyboarded device).