# 9 Introduction to Programming (*in* PHP)

*"The process of preparing programs for a digital computer is especially attractive, not only because it can be economically and scientifically rewarding, but also because it can be an aesthetic experience much like composing poetry or music."*

*– Donald Knuth*

# Content

- Software: Running Program, Data, Document
- Programming Concepts:
  - Representation, Model, Algorithm
- Variables, Operators, Expressions (in PHP)
- HTML Forms

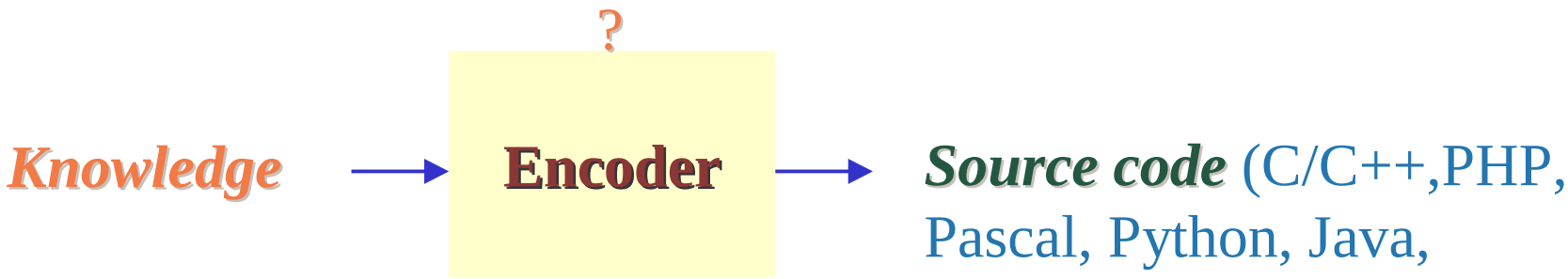
- Note: Some portions of this module is taken from Turgut UYAR's presentations for *"Introduction to Scientific and Engineering Computing"* Course:

  *http://www.cs.itu.edu.tr/~uyar*

# What is Software?

## Software ≡ Knowledge Coding

*"The Business of Software" – Phillip G. Armour*
*Comm. Of the ACM, pp. 13, Vol.44, No.3, March 2001*



*example:*

$$ax^2 + bx + c = 0$$

**Knowledge**

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

→ **Encoder** →

```
cout << endl << “Enter a” ;
cin >> a  ;
cout << endl << “Enter b” ;
cin >> b  ;
cout << endl << “Enter c” ;
cin >> c  ;
D = sqrt( b * b – 4 * a * c ) ;
x1 = ( -b - D ) / (2 * a) ;
x2 = ( -b + D ) / (2 * a) ;
cout << “Roots are ” ;
cout << x1 << “ , ”<<x2 ;
```

**a** →
**b** → PROGRAM → $x_1$
**c** → $x_2$

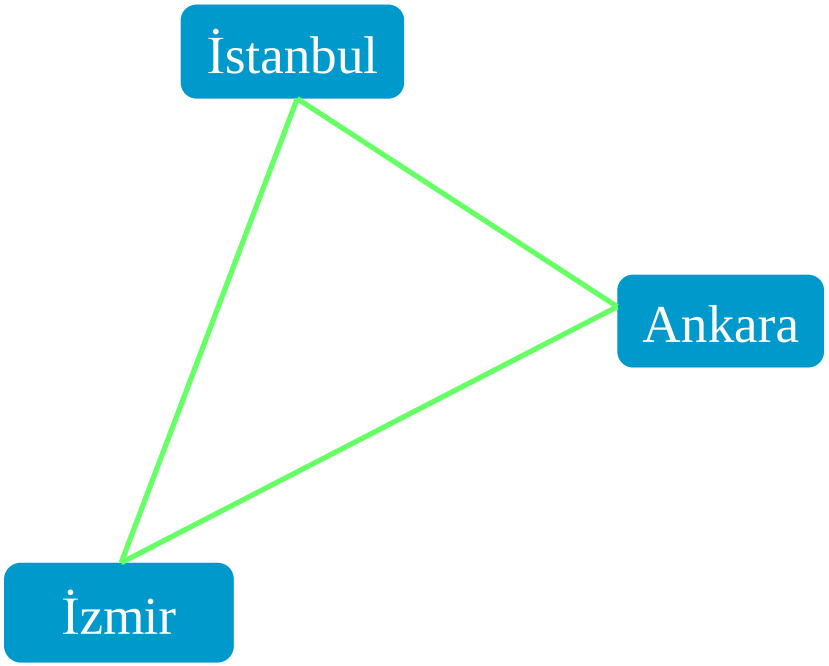
# Computer Programs



- how to represent the problem?
  - computers work on numbers
  - program about the highways in Turkey
  - entities: cities and roads
  - representing a city: name, latitude, longitude
- how to express the solution?

# Representing the Problem

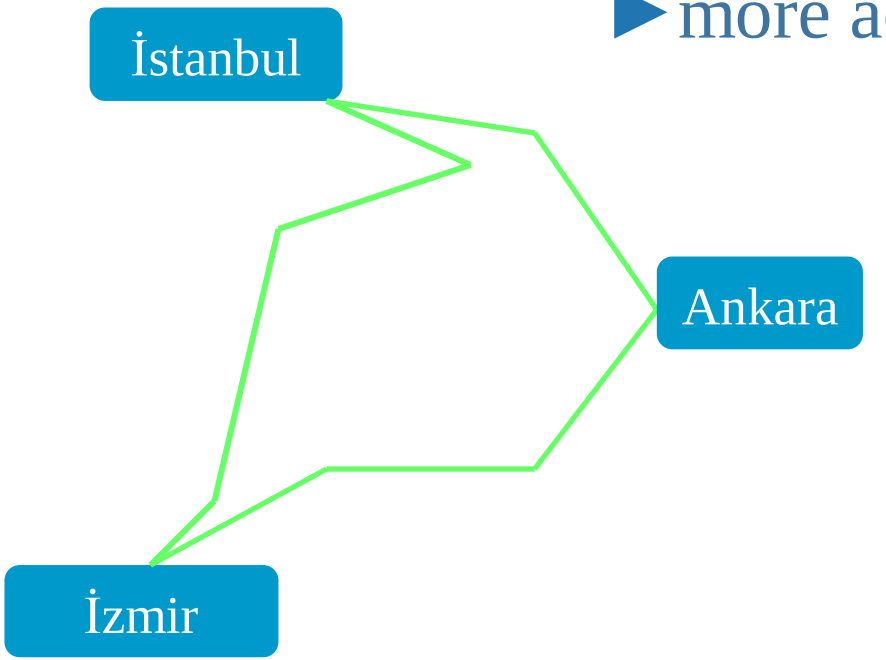





- representing a road: line
  - assume the road is straight
  - start and end cities
- very easy
- very inaccurate

# Representing the Problem

- representing a road: consecutive lines
- very hard
- more accurate → more complicated



# Representation: Model

- FIRST STEP: build a correct / accurate (and feasible) model
- what you are solving is the model, not the problem itself
  - incorrect model → incorrect solution
  - inaccurate model → meaningless solution
  - infeasible model → expensive implementation

# Representing the Problem



- representing highways:
  - if you are only interested in total distances, you can use lines
  - if you will talk about “the 274th km of the İstanbul-Ankara highway”, you should use consecutive lines

# Expressing the Solution

- step-by-step guide to the solution: *algorithm*
- recipe for Jamaican rice and peas:
  - put 1 1/2 can of beans in 4-5 cups of water
  - add 1/4 can of coconut milk, 1 sprig of thyme and salt and pepper to taste
  - cook until beans are soft
  - smash the bottom of an escallion and add it to the pot along with 2 cups of rice and 1/4 can of coconut milk and 2 sprigs of thyme
  - remove any access water over 2cm above the rice
  - bring to a boil for 5 min
  - continue to cook covered until rice is tender

# Algorithm

- there must be no room for judgement
  - 4-5 cups? sprig?
  - salt and pepper to taste?
  - beans are soft? rice is tender?
- this cooking recipe is NOT an algorithm
- must be finite
- in a finite number of steps:
  - either find the correct solution
  - or report failure to find a solution
- must not run forever

# Data

- data is represented by *variables*
- symbolic name for the data
- variables take *values*
- city variables: name latitude longitude

  to represent İstanbul:

  name: “İstanbul”

  latitude: 41

  longitude: 29

# Variables

- variables are kept in memory
  - variable is the name of the memory cell
  - value is the content of the memory cell

# Assignment

- block structured programs proceed by assigning values to variables
- notation: latitude ← 41
  - "store the value 41 in the memory cell named latitude"
- left hand side is a variable
- right hand side is an *expression*
  - a computation that yields a value

# Expressions

- can be a single value or variable:
  - 41
  - latitude
- can be combinations of values and variables connected via *operators*:
  - 4 * longitude  multiplication operator
  - latitude + longitude  addition operator

# Assignment

- ASSIGNMENT IS NOT EQUALITY!!!
- 41 ← latitude doesn't make sense
- **i** ← **i** + 1 means: increment the value of **i** by 1
  - if **i** was 5 before the operation, it will become 6 after the operation
- mathematically it would be incorrect:

  **0 = 1**

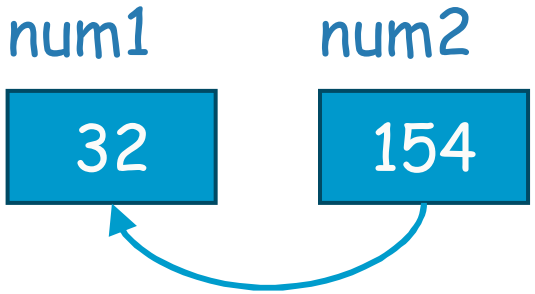
# Swap

- swap the values of two variables:

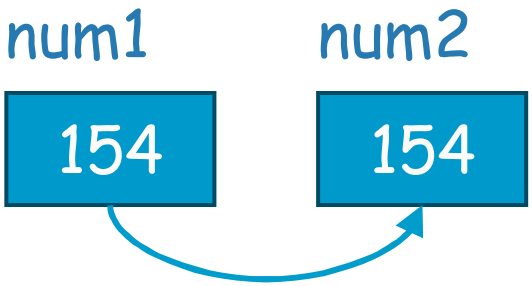
# Swap: Incorrect

– num1 ← num2

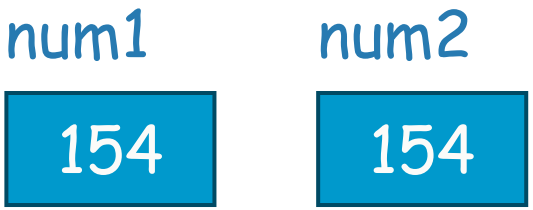
– num2 ← num1




num1 ← num2

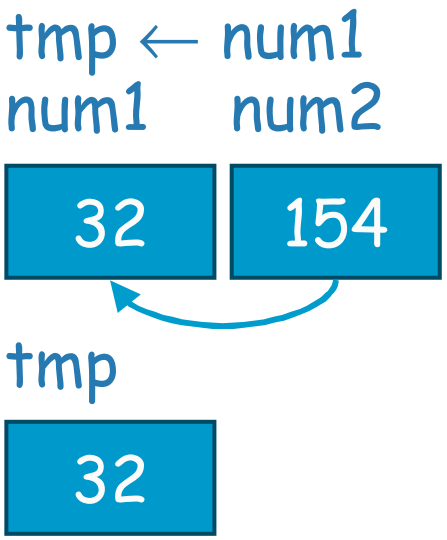


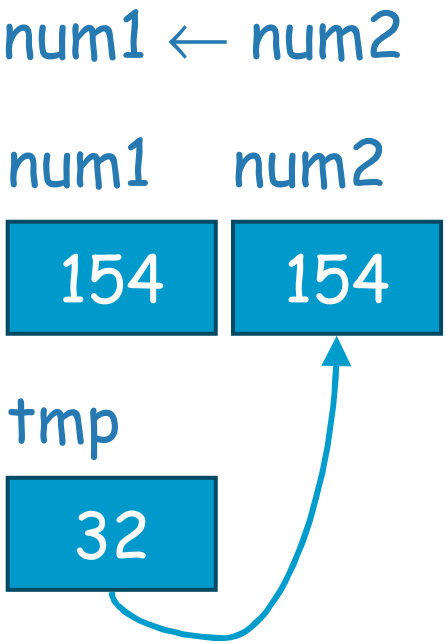

num2 ← num1

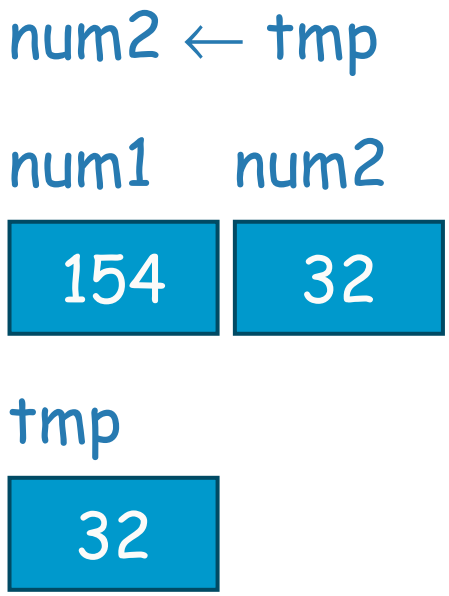


# Swap

- tmp ← num1
- num1 ← num2
- num2 ← tmp

# Data Types

- basic data types:
  - integer
  - real number
  - logical
  - character
  - string
- composite data types: record
- vector data types: array

# Basic Data Types

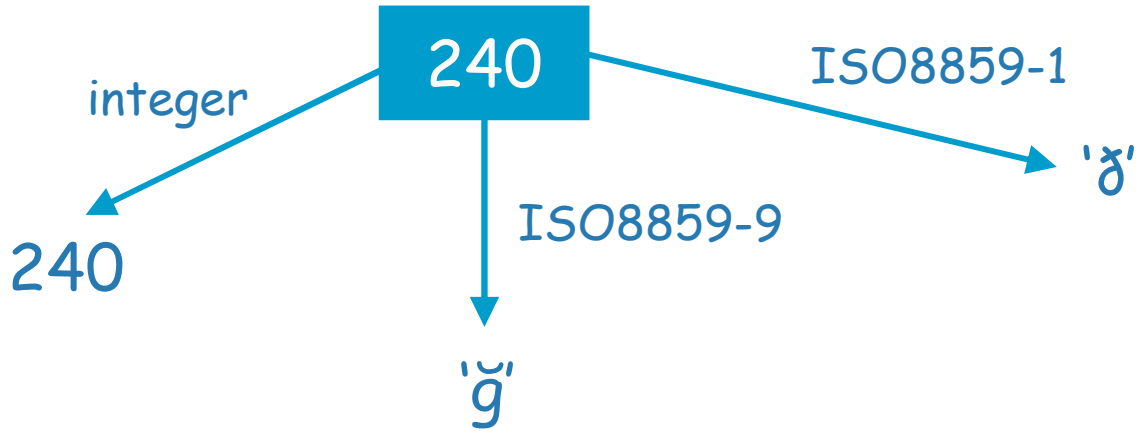
- integer
  - birth year, number of letters in the surname, height in cm
- real numbers
  - height in m, average of several exam scores, square root of a number
- logical: values can be *true* or *false*
  - student successful, older than 18 years

# Character

- any symbol: letter, digit, punctuation mark, ...
  - first letter of surname, the key the user pressed
- mostly written between single quotes:
  - ‘Y’, ‘4’, ‘?’

# Encoding

- numbers correspond to symbols
- ASCII, ISO8859-X, Unicode

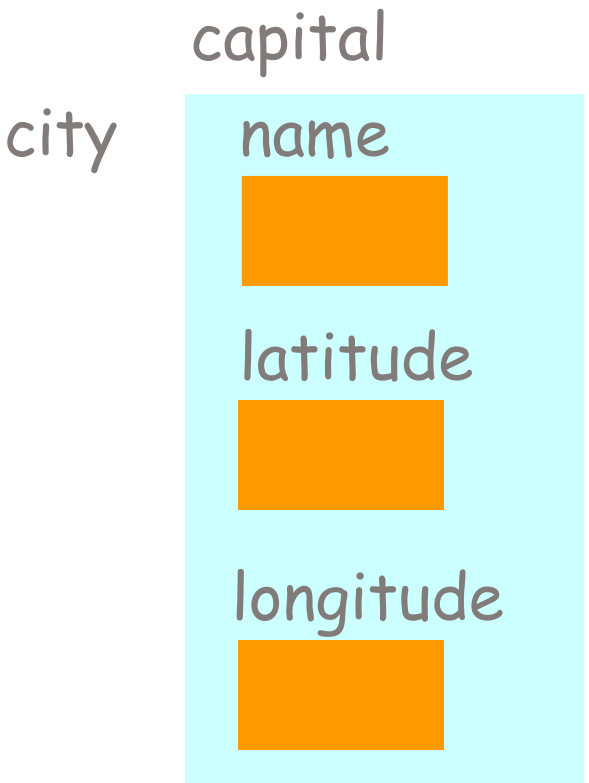


# Strings

- name, word, sentence, ISBN number, ...
- mostly written between double quotes:
  - “Dennis Ritchie”, “ISBN 0-13-110362-8”
- use numbers if you plan to make arithmetic operations on it:
  - student numbers at ITU: 9-digit numbers
  - will you add/multiply/... student numbers?
  - no sense in using integers, use strings

# Composite Data Types

- grouping types to form a new type




- access:
  - capital.name ← “Ankara”
  - NOT city.name ← “Ankara”

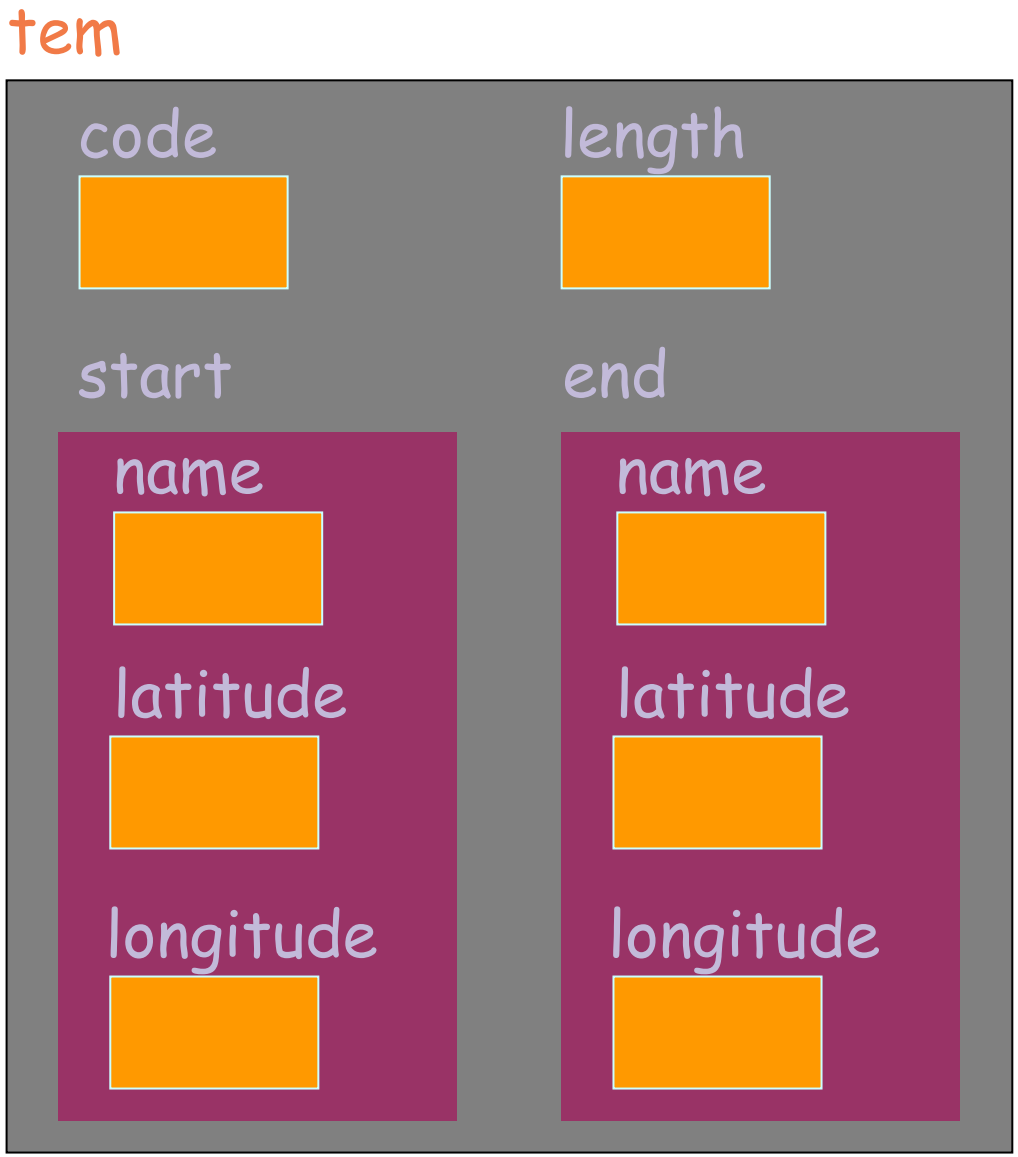
# Composite Data Types



highway

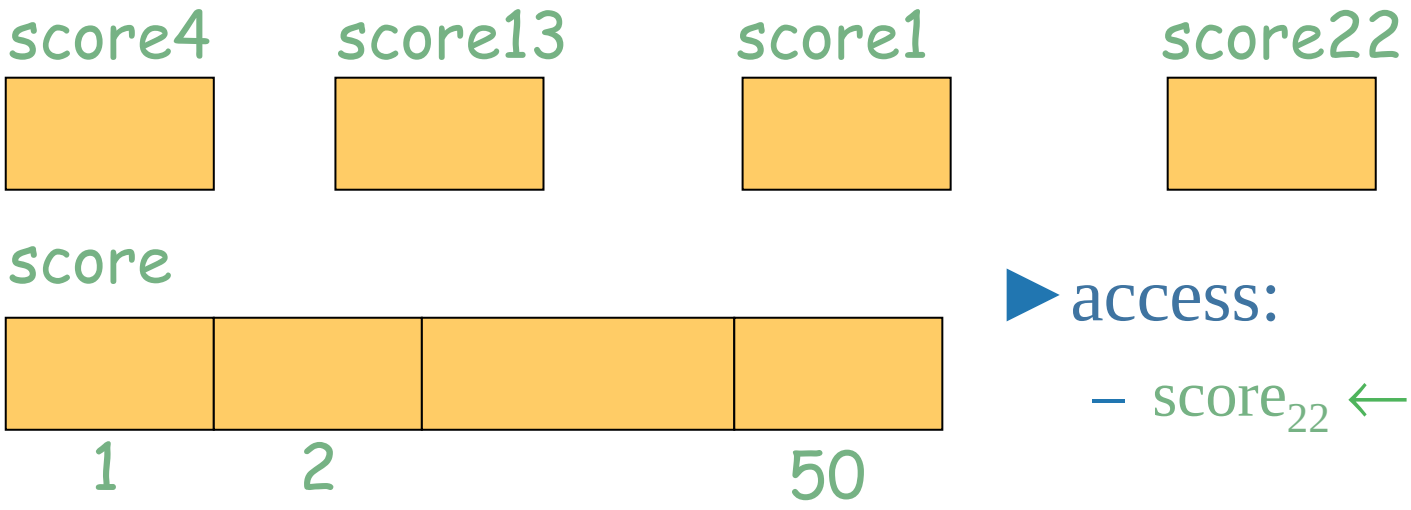
- composite types can be nested

- access:
  - tem.code ← “E-6”
  - tem.end.name ← “Ankara”



# Vector Data Types

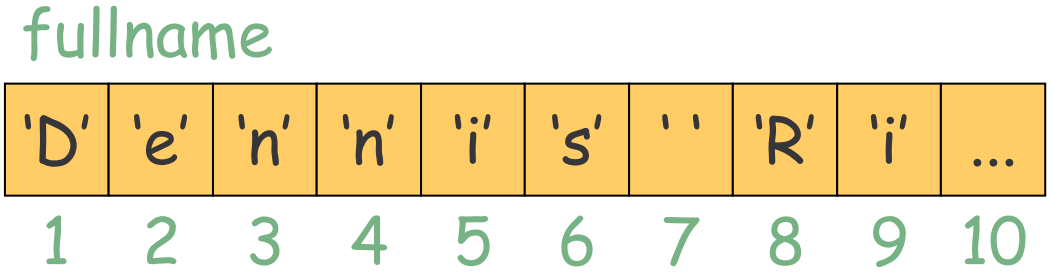
- grouping elements of the same type
  - exam scores for a 50 student class:
    - 50 integer variables: score1, score2, ..., score50
    - an integer array with 50 elements: score




- access:
  - $score_{22} \leftarrow 95$

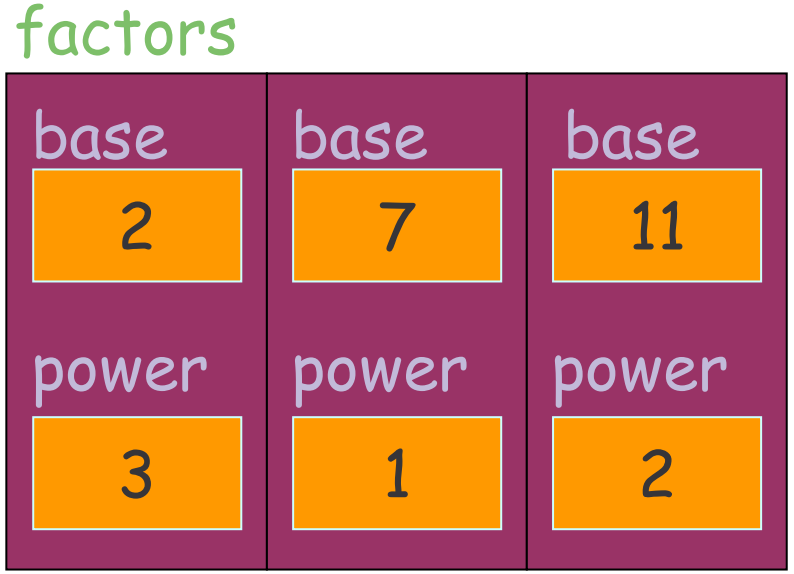
# Strings

- strings are usually arrays of characters



# Arrays of Records

- represent the factors of a number:
  - an array where each member is a factor
  - a factor is a record of a base and a power



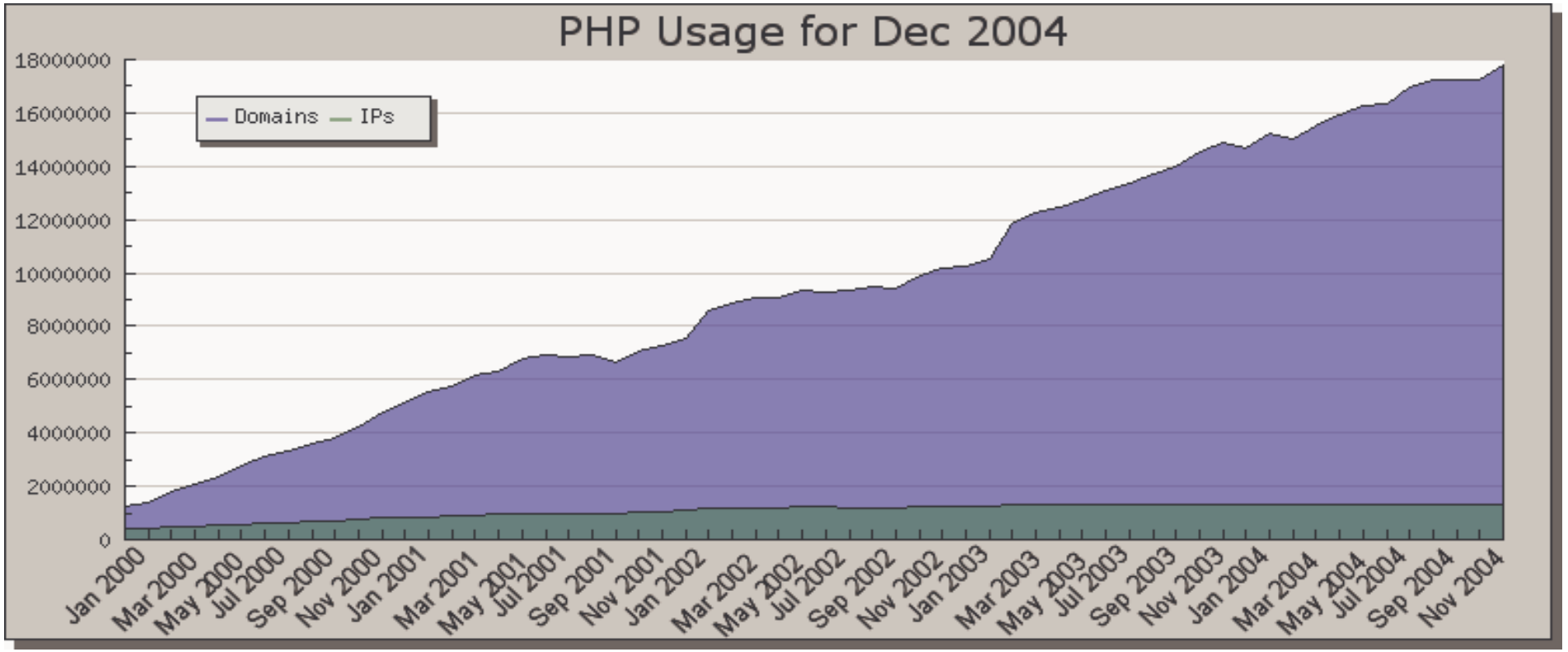

- access:
  - $factors_2.base \leftarrow 7$

# What is PHP

- PHP nowadays stands for "PHP: Hypertext Preprocessor", but originally it was called "Personal Home Page Tools". PHP is created by Rasmus Lerdorf.
- PHP is attached to HTML such that you can create dynamic websites, that is, the content is created at the moment you retrieve that webpage

# What is PHP

- There are currently about 18 million web servers running/ using PHP



# A Simple example



- HTML (Hypertext Markup Language) uses tags to define the structure of a page
  - all pages are already defined on the server
  - the user can only see these pages, follow hyperlinks etc.
- In pure HTML, there is no way to change the content of pages on the fly.

# Pure html

```
<!--first simple example in pure-->
<HTML>
<HEAD>
<TITLE>Simple example</TITLE>
</HEAD>
<BODY>
This is my first html example
</BODY>
</HTML>
```

Example in a browser

# PHP-HTML



```
<!--first simple example in PHP-HTML-->
<HTML>
<HEAD>
<TITLE>Simple example</TITLE>
</HEAD>
<BODY>
<? Print "This is my first php-html
example" ?>
</BODY>
</HTML>
```

# PHP-HTML



```
<!--second simple example in PHP-HTML-->
<HTML>
<HEAD>
<TITLE>Second Simple example</TITLE>
</HEAD>
<BODY>
Today, it is
<? Print Date("l F d, Y"); ?>
</BODY>
</HTML>
```

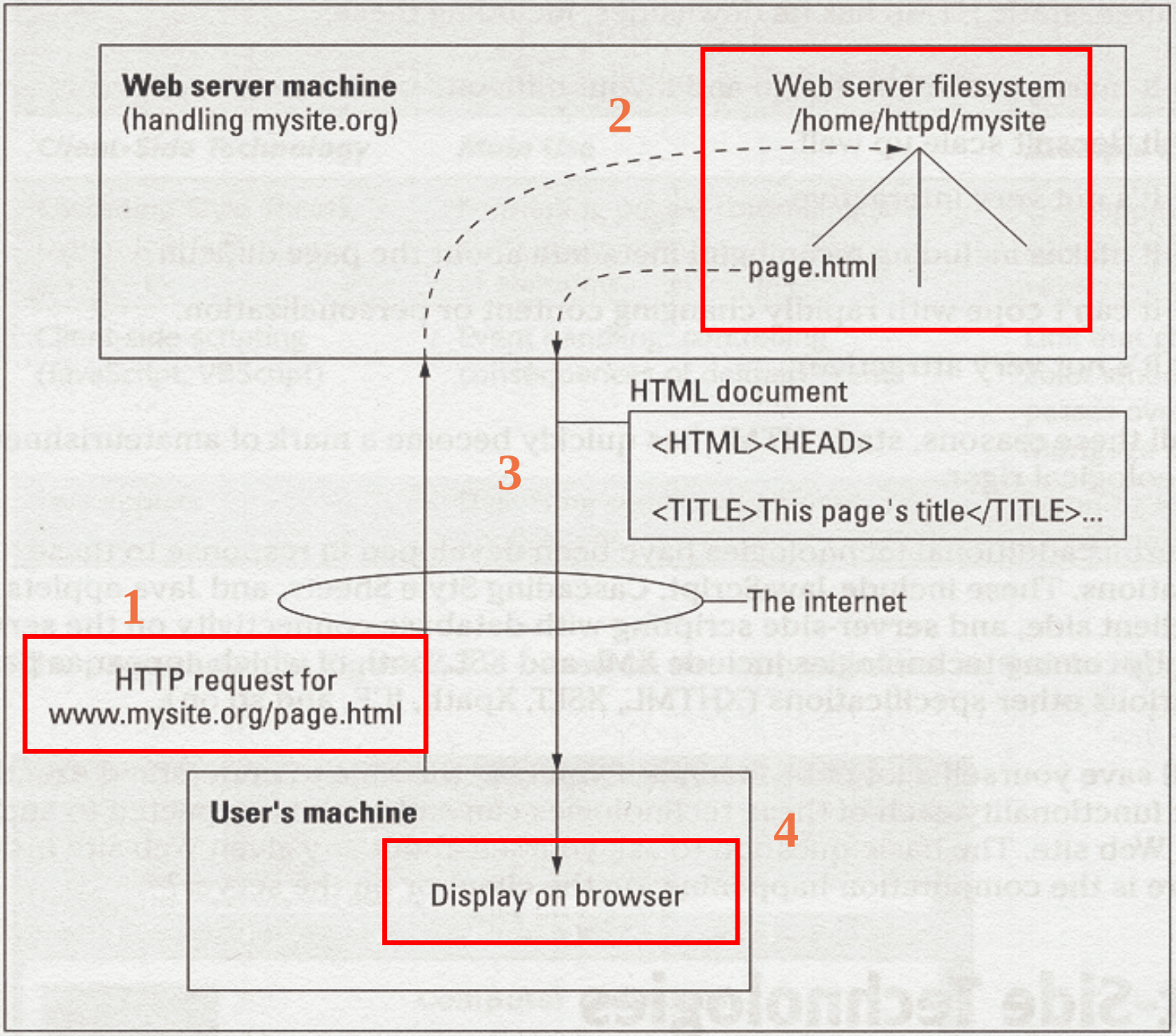



**Figure 2-2:** A simple HTTP request

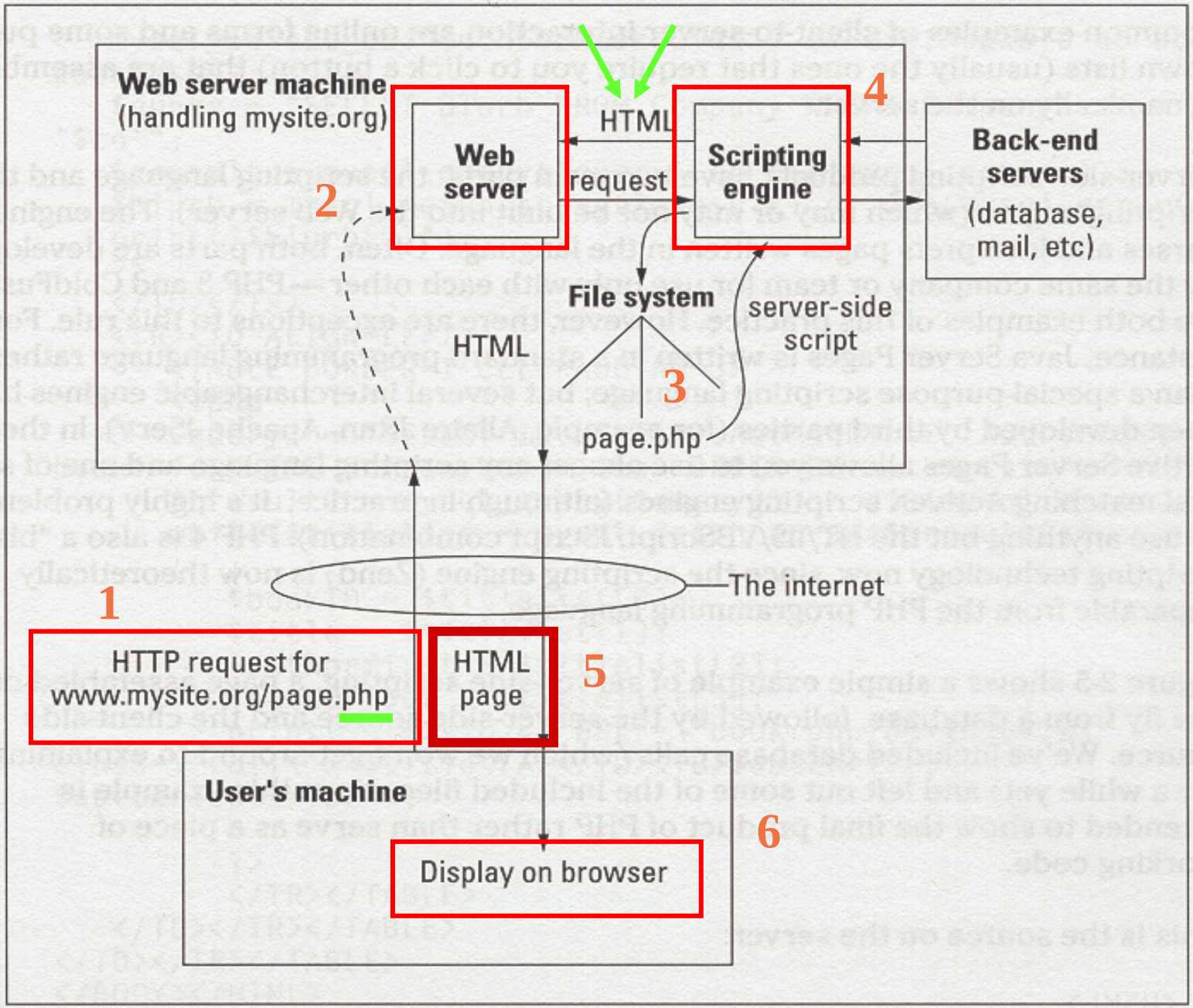



**Figure 2-4:** Server-side tasks

# Adding PHP to HTML



- Escaping from HTML into PHP can be done by including your code in the tag:

`<?PHP ?>` or just `<? ?>`

- Everything in the tag is treated as PHP, at least if the file extension is .php
- If you use frontpage, use:

```
<SCRIPT LANGUAGE="PHP"> </SCRIPT>
```

- You can also switch between php and pure html in a single page by using more php-tags.

```
<HTML>
<HEAD>
<TITLE>Third Simple example</TITLE>
</HEAD>
<BODY>
<? $name="Rasmus"; ?>
Today, it is <? Print Date("l F d, Y"); ?>
<BR>
How are you <? Print $name; ?>?
</BODY>
</HTML>
```

# Output



- By using echo or print
- The output goes to the users' browser (through an HTML file)
- Echo can have multiple arguments:

```
<?
Print ("Print this and that and so <BR>");
Echo "print this", " and that", " and so";
//is also a valid statement
?>
```

# Output

Note that in the previous example, the code is written on one line if you look at the source (By using <view> and <source> from the menu bar)

You can create separate lines in the source by using the "\n" character:

- This makes the source code more readable
- This doesn’t change the layout in the browser

# Adding comments

- It is very important that you add comments to your source code. First of all we require that you write your name on the first line.
- But if scripts become more complicated, you need to add comments to explain what you do at that stage. This in order to make it easier to change the code afterwards.

```
Comments in html:
  <!-- here is a comment -->
In php:
  // single line comment
  /* multiple line
   comment */
```

# Variables



- A variable is a container that holds data
- Variables always starts with a $-sign, followed by the name itself.
- The name must always start with a character (A..Z, a..z)
- After the first character, you may use numbers and the underscore (e.g. $name1, $first_name etc.)
- Variable names are case sensitive so $FirstName is different from $firstname
- Always use logical names
- Never use the same name with different type casting

# Variables: Examples



```
$name = "Rasmus";
/* assigns the data (in this case a string)  Rasmus to the variable called $name */
$amount = 2.2;
/* assign the value 2.2 to the variable called $amount */
```

# Variables: Summary

- All variables start with a $-sign
- The value is based on the most recent assignment
- Assignment through the "=" operator
- Each statement ends with a ;

```
$my_num = "this is a string";
$my_num = 5; //this is an integer
Print "$my_num";  //this prints a 5
```

# Comments

- It is very important to document your programming code
- Multiline comments:

```
/* this is a multiline
Comment */
```

- Single line comments

```
# this is a single line comment
//this is also a single line comment
```

# Comments

- In this skills training, each html and php file should include your name and id number in the first line.
- Note: in most cases you will not start with php code in the first line but with html code. In that case, a comment starts with <!-- and ends with -->

# Types in PHP

- Integer (whole numbers)
- Double (floating point number)
- String (sequence of characters)
- Boolean (have value TRUE or FALSE)
- Array (collection of other types)
- Objects (not treated in the course)

```
$a = 2;  //an integer
$a = 2.2; //a double
$a = "two"; //a string
$a = FALSE; //a Boolean
```

- Can be used after each other, of course the last assignment will be the one that is stored in the variable $a
- Each of the above lines is called a statement and a statement is closed with a semicolon ;

# Common mistake

- 90% of all mistakes are caused by just one character, the:

;

Each statement ends with a ;

# Strings

- Strings: a collection of characters enclosed with single or double quotes.
- Note the difference between single and double quoted strings (single almost no interpretation)
- PHP has many string functions to manipulate strings

# Single and Double quoted strings

- PHP preprocesses double quoted strings
- Escape sequences are possible in double quoted strings
- Single quoted strings are not preprocessed
- Only escape sequences for ' and \ in single quoted strings

Example:

```
/*assign chicken to the variable animal*/
$animal = "chicken";
/*assign this to saved_string, use single quotes */
$saved_string = 'The animal is $animal';
/*assign to saved_string2, now use double quotes */
$saved_string2 = "The animal is $animal";
/*assign a new string to the variable animal*/
$animal = "dog";
print ("The animal is $animal <BR>\n");
print ("the first saved string is : $saved_string <BR>\n");
print ("the second saved string is : $saved_string2
  <BR>\n");
```

Example in browser

# Concatenation



- Adding two strings together is simple done by a dot-operator

```
$firstname=“Jack";
$lastname =“Bauer";
$name=$firstname . " " . $lastname;
Print("$name");
```

# Many useful string functions

- Strpos(string, stringtofind) gives the position of the first instance of stringtofind in string, starting at 0

```
strpos("Jack","c") returns a 2
```

- Strlen gives the length of the string
- Substr(string,pos) returns the substring of string, starting at position pos (actually pos+1)

```
substr("example",5) returns "le"
```

# Addslashes and Stripslashes

- Two functions are very useful if we want to store data in a database. Adding a single quote in a text to a database is problematic, for this reason we use these two functions.
- Addslashes adds slashes to all kinds of special characters and stripslashes removes them

# Example

- Suppose you want to store the name: Jeanne d'Arc in a database or the text: he said: "PHP is very powerful".
- In both cases PHP has some troubles, for instance if you want to store the second example in a string:
- $text = "He said: "PHP is very powerful. "";
- This is impossible and the solution is to add slashes:
- $text = "He said: \"PHP is very powerful.\""; But if the data come from the input of a user, e.g. by using a form, we don't know in advance where to put the slashes. The function addslashes does this for us.

# PHP Operators – Arithmetic Operators

| Operator | Description | Example | Result |
|---|---|---|---|
| + | Addition | x=2<br>x+2 | **4** |
| - | Subtraction | x=2<br>5-x | **3** |
| * | Multiplication | x=4<br>x*5 | **20** |
| / | Division | 15/5<br>5/2 | **3**<br>**2.5** |
| % | Modulus (division remainder) | 5%2<br>10%2 | **1**<br>**0** |
| ++ | Increment | x=5<br>x++ | **x=6** |
| -- | Decrement | x=5<br>x-- | **x=4** |

# PHP Operators – Assignment Operators

| Operator | Example | Is The Same As |
|---|---|---|
| = | x=y | x=y |
| += | x+=y | x=x+y |
| -= | x-=y | x=x-y |
| *= | x*=y | x=x*y |
| /= | x/=y | x=x/y |
| %= | x%=y | x=x%y |

# PHP Operators – Comparison Operators

| Operator | Description | Example |
|---|---|---|
| == | is equal to | 5==8 returns false |
| != | is not equal | 5!=8 returns true |
| > | is greater than | 5>8 returns false |
| < | is less than | 5<8 returns true |
| >= | is greater than or equal to | 5>=8 returns false |
| <= | is less than or equal to | 5<=8 returns true |

# PHP Operators – Logical Operators

| Operator | Description | Example |
|---|---|---|
| && | and | x=6<br>y=3<br>(x < 10 && y > 1) returns true |
| \|\| | or | x=6<br>y=3<br>(x==5 \|\| y==5) returns false |
| ! | not | x=6<br>y=3<br>!(x==y) returns true |

# PHP Operators

- Similar to those in C++ / Java / Perl
- Be careful with a few operators
  - / in PHP is always floating point division
    - To get integer division, we must cast to int

```
$x = 15;
$y = 6;
echo ($x/$y), (int) ($x/$y), "<BR />";
```

      - Output is `2.5 2`
  - Inequality operators do not compare strings
    - Will cast strings into numbers before comparing

# PHP Operators

- To compare strings, use the C-like string comparison function, strcmp()

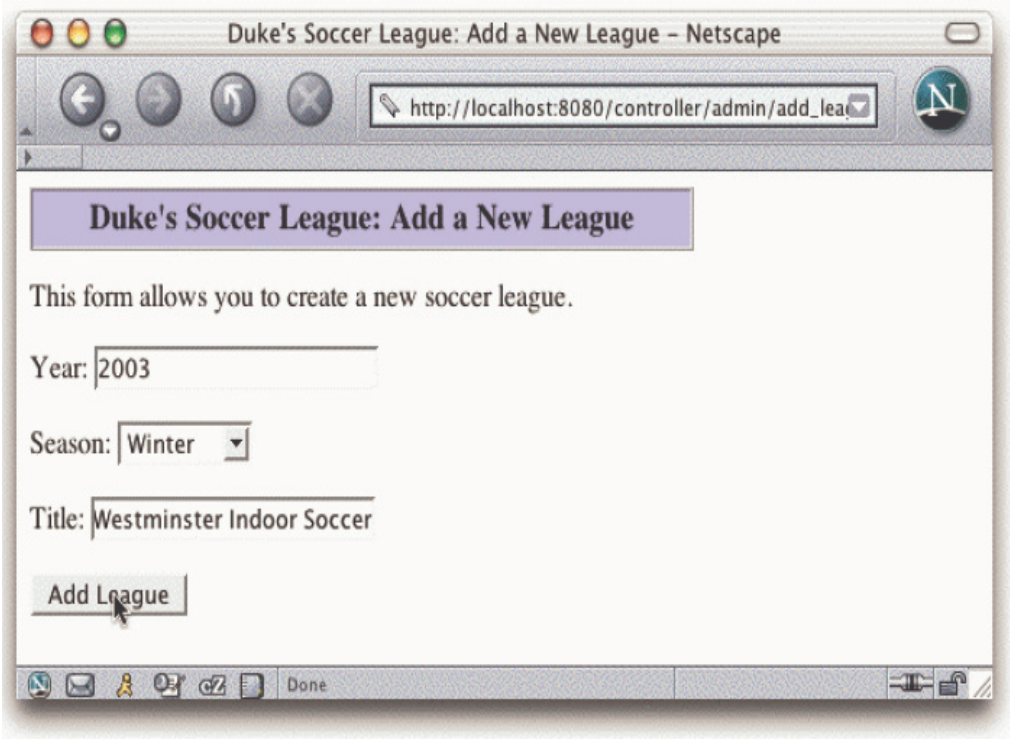
- Even the == operator has odd behavior in PHP when operands are mixed
  - Ex: Consider comparing a string and an int
    - Any non-numeric PHP string value will “equal” 0
    - Any numeric PHP string will equal the number it represents
  - Example: Consider comparing a string and a boolean
    - Regular PHP string value will “equal” true
    - “0” string will equal false
  - This behavior is consistent but confusing to the programmer and is probably best avoided

# PHP Operators

- An additional equality operator and inequality operator are defined
  - === returns true only if the variables have the same value and are of the same type
    - If casting occurred to compare, the result is false
  - !== returns true if the operands differ in value or in type

- Precedence and associativity are similar to C++/Java

# Creating an HTML Form



# The FORM Tag



- The following is a partial structure of an HTML form:

<form action='URL TO CONTROLLER' method='GET or POST'>
<!-- PUT FORM COMPONENT TAGS HERE -->
</form>

- For example:

<form action='add_league.php' method='POST'>
Year: [textfield tag]
Season: [drop-down list tag]
Title: [textfield tag]
[submit button tag]
</form>

- A single web page can contain many forms.

# Textfield Component

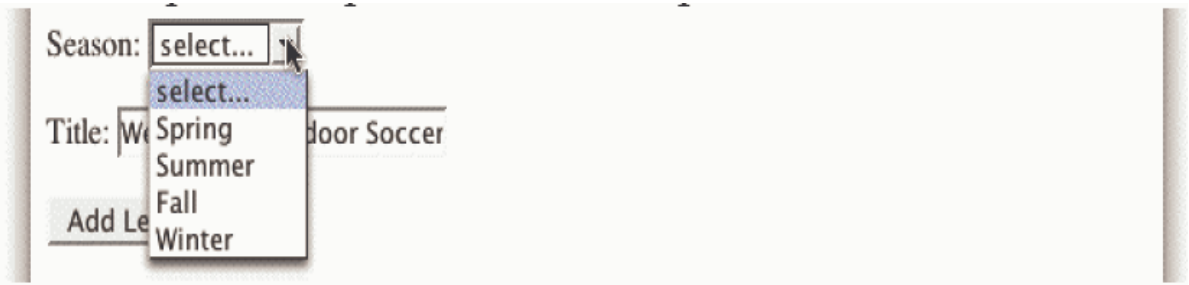
- In Netscape, a textfield component looks like this:


- The HTML content for this component is:

```
16 <p>
17 This form allows you to create a new soccer league.
18 </p>
19
20 <form action='add_league.do' method='POST'>
21 Year: <input type='text' name='year' /> <br/><br/>
```

# Drop-Down List Component

- In Netscape, a drop-down list component looks like this:



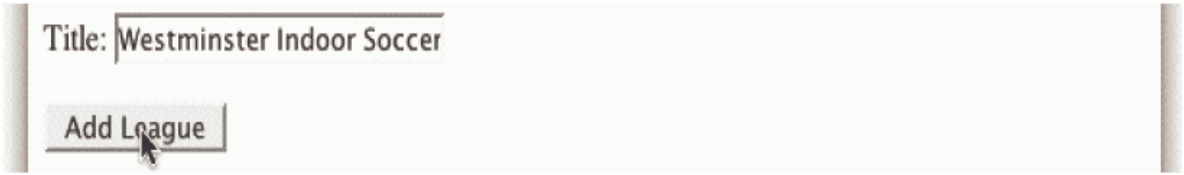

- The HTML content for this component is:

```
22 Season: <select name='season'>
23 <option value='UNKNOWN'>select...</option>
24 <option value='Spring'>Spring</option>
25 <option value='Summer'>Summer</option>
26 <option value='Fall'>Fall</option>
27 <option value='Winter'>Winter</option>
28 </select> <br/><br/>
```

# Submit Button



- In Netscape, a submit button component might look like this:




- The HTML content for this component is:

```
29 Title: <input type='text' name='title' /> <br/><br/>
30 <input type='submit' value='Add League' />
31 </form>
```

# PHP: Form Handling

- **$_POST** retrieves our form data and output's it directly to our browser.
- The best way to do this, is to make variables for each element in our form, so we can output this data at will, using our own variable names.

# PHP: Form Handling



```
<?php
$Fname = $_POST["Fname"];
$Lname = $_POST["Lname"];
$gender = $_POST["gender"];
$food = $HTTP_POST_VARS["food"];
$quote = $_POST["quote"];
$education = $_POST["education"];
$TofD = $_POST["TofD"];
?>
```

# PHP: Form Handling



```
<form method="post" action="<?php echo
    $PHP_SELF;?>">
First Name:<input type="text" size="12" maxlength="12"
    name="Fname"><br />
Last Name:<input type="text" size="12" maxlength="36"
    name="Lname"><br />
Gender:<br />
Male:<input type="radio" value="Male"
    name="gender"><br />
Female:<input type="radio" value="Female"
    name="gender"><br />
```