# Learning PHP

Data: Working with Text and Numbers

# Data Types

- **String**: a sequence of bytes  (represented by charaters)
- **String can contain**
  - Letters a-z, A-Z
  - Numbers 0-9
  - Punctuation . ; ? ! , ( ) :
  - Spaces
  - Tabs
  - Or any other character
- **Examples**:
  - Gtf46gx!#ddfgfg
  - Life is beautiful

# Defining Strings

- Surround the string with **single-quote** or **double-quote**
  - There are differences between using ‘ and “
- If you want to include a single quote inside a string, put a backslash (\) before it
- Word processors often change straight quotes like ' and " into **curly quotes** like ‘, ’, “, and ”. The PHP engine only understands straight quotes as string delimiters.

```
print 'I would like a bowl of soup.';
print 'chicken';
print '06520';
print '"I am eating dinner," he growled.';
```

```
print 'We\'ll each have a bowl of soup.';
```

# Ecape Sequence

- Backslash (\) is a escape character in PHP (like C,C++,C#,JavaScript, Java)
- inside single-quoted strings, **backslash** and **single quote** are only special characters. Everything else is treated literally.

```
print 'Use a \\ to escape in a string';
```

This prints:

```
Use a \ to escape in a string
```

- Double-Quoted Strings
  - If you use " for strings, there are more special characters
  - Example:
    - Print 'Hi $user'  *this prints Hi $user*
    - Print "Hi $user"  *this prints Hi John if John is the current user*

*Table 2-1. Special characters in double-quoted strings*

| Character | Meaning |
|---|---|
| \n | Newline (ASCII 10) |
| \r | Carriage return (ASCII 13) |
| \t | Tab (ASCII 9) |
| \\ | \ |
| \$ | $ |
| \" | " |
| \0 .. \777 | Octal (base 8) number |
| \x0 .. \xFF | Hexadecimal (base 16) number |

# HERE document

- You can define strings with the *here document* syntax, specialy *HTML code string*
- ***Start** with <<< and a delimiter word, **end** with the same word used at start*
- delimiters can contain
  - **letters**, **numbers**, and the **underscore** character.
  - The **first character** of the delimiter must be a letter or underscore.
  - For **readability**, recommended writing uppercase
- The end delimiter must be **alone on its line**.
- The delimiter can't be **indented** and **no whitespace**, comments, or other characters are allowed after it, only **semi-colon**

*Example 2-1. Here document*

```
<<<HTMLBLOCK
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
<ul>
  <li> Beef Chow-Fun
  <li> Sauteed Pea Shoots
  <li> Soy Sauce Noodles
  </ul>
</body>
</html>
HTMLBLOCK
```

# HERE Document

- **Note**: **print** command must be used,
- but single or double quote not needed

*Example 2-2. Printing a here document*

```
print <<<HTMLBLOCK
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
<ul>
  <li> Beef Chow-Fun
  <li> Sauteed Pea Shoots
  <li> Soy Sauce Noodles
  </ul>
</body>
</html>
HTMLBLOCK;
```

# Combining Strings

- Use a . (period) to combine (concatenate) strings

```
print 'bread' . 'fruit';
print "It's a beautiful day " . 'in the neighborhood.';
print "The price is: " . '$3.95';
print 'Inky' . 'Pinky' . 'Blinky' . 'Clyde';
```

The combined strings print as:

```
breadfruit
It's a beautiful day in the neighborhood.
The price is: $3.95
InkyPinkyBlinkyClyde
```

# PHP String functions

- trim() function removes whitespace from the beginning and end of a string.
- strlen() function tells you the length of a string

*Example 2-3. Checking the length of a trimmed string*

```
// $_POST['zipcode'] holds the value of the submitted form parameter
// "zipcode"
$zipcode = trim($_POST['zipcode']);
// Now $zipcode holds that value, with any leading or trailing spaces
// removed
$zip_length = strlen($zipcode);
// Complain if the zip code is not 5 characters long
if ($zip_length != 5) {
    print "Please enter a zip code that is 5 characters long.";
}
```

# PHP String functions

- Use == to cpmrae two strings

```
if ($_POST['email'] == 'president@whitehouse.gov') {
    print "Welcome, US President.";
}
```

- strcasecmp() function compares two strings without considering their cases (upper or lower case)

```
if (strcasecmp($_POST['email'], 'president@whitehouse.gov') == 0) {
    print "Welcome back, US President.";
}
```

- strtolower() and strtoupper() changes a string to all-lowercase or all-uppercase versions, respectively.

*Example 2-10. Changing case*

```
print strtolower('Beef, CHICKEN, Pork, duCK');
print strtoupper('Beef, CHICKEN, Pork, duCK');
```

Example 2-10 prints:

```
beef, chicken, pork, duck
BEEF, CHICKEN, PORK, DUCK
```

# PHP String Functions

- The **ucwords()** function uppercases the first letter of each word in a string.

```
print ucwords(strtolower('JOHN FRANKENHEIMER'));
```

Example 2-11 prints:

```
John Frankenheimer
```

- **substr()** function, you can extract just part of a string.
  - 0 to 30 means first 30 characters
  - If you put negative number, it starts from the end
    - *-4, 4 means last 4 character from the end of the string*

*Example 2-12. Truncating a string with substr()*

```
// Grab the first 30 bytes of $_POST['comments']
print substr($_POST['comments'], 0, 30);
// Add an ellipsis
print '...';
```

If the submitted form parameter comments is:

```
The Fresh Fish with Rice Noodle was delicious, but I didn't like the Beef Tripe.
```

Example 2-12 prints:

```
The Fresh Fish with Rice Noodl...
```

# PHP String Functions

- str_replace() function changes parts of a string. It looks for a substring and replaces the substring with a new string.

*Example 2-14. Using str_replace()*

```
$html = '<span class="{class}">Fried Bean Curd<span>
<span class="{class}">Oil-Soaked Fish</span>';

print str_replace('{class}',$my_class,$html);
```

If `$my_class` has been set to `lunch`, then Example 2-14 prints:

```
<span class="lunch">Fried Bean Curd<span>
<span class="lunch">Oil-Soaked Fish</span>
```

# Numbers

# Numbers

- No special notation needed
- Numbers:
  - **Floating-point**: 34.6, 7.567, 0.765, -8.543
  - **Integers:** 356, 1876, 21, 0, -76

- **NOTE:** Integers are stored **precisely** but floating point numbers may not
- 47 is stored exactly 47 but **46.3 could be stored as 46.299999**

*Example 2-15. Numbers*

```
print 56;
print 56.3;
print 56.30;
print 0.774422;
print 16777.216;
print 0;
print -213;
print 1298317;
print -9912111;
print -12.52222;
print 0.00;
```

# Arithmetic Operators

You can use arithmetic operators ( +, -, *, / ) as you did in primary school

New operators:

- ** for exponentation
- % for modulus division (returing remainder of a devision)

```
print 17 % 3;
```

This prints:

```
2
```

*Example 2-16. Math operations*

```
print 2 + 2;
print 17 - 3.5;
print 10 / 3;
print 6 * 9;
```

The output of Example 2-16 is:

```
4
13.5
3.3333333333333
54
```

# Order of Operators

- Like in Math, operators are executed in order
- Example:
  - 3+4*2 → 11
- Unless, you use brackets (parentheses) to change the order
  - (3+4)*2 → 14
  - 3+(4*2) → 11

- Learn more about **PHP Operator Precedence :** PHP: Operator Precedence - Manual

# Variables

# Variables

- **Variables** hold the data in the memory of computer while your program uses and manipulates it
- In PHP, variables are **denoted by a $** followed by the variable's name.
- **To assign a value to a variable**, use an **equals sign (=).** This is known as the **assignment** operator.

Here are a few examples:

```
$plates = 5;
$dinner = 'Beef Chow-Fun';
$cost_of_dinner = 8.95;
$cost_of_lunch = $cost_of_dinner;
```

Assignment works with here documents as well:

```
$page_header = <<<HTML_HEADER
<html>
<head><title>Menu</title></head>
<body bgcolor="#fffed9">
<h1>Dinner</h1>
HTML_HEADER;

$page_footer = <<<HTML_FOOTER
</body>
</html>
HTML_FOOTER;
```

# Variable Names

- **Variable names may only include:**
  - Uppercase or lowercase Basic Latin letters (A-Z and a-z)
  - Digits (0-9)
  - Underscore (_)
  - Any non-Basic Latin character (such as ç), if you're using a character encoding such as UTF-8 for your program file
  - Additionally, the **first character** of a variable name is not allowed to be a digit.

*Table 2-2. Allowable variable names*

| |
|---|
| `$size` |
| `$drinkSize` |
| `$SUPER_BIG_DRINK` |
| `$_d_r_i_n_k_y` |
| `$drink4you2` |
| `$напиток` |
| `$သောက်စရာ` |
| `$🄳🅁🄸🄽🄺` |
| `$😀` |

- **NOTE**:Even though you can use special charcaters, but **NOT RECOMMENDED**

- **Variable names are case-sensitive.**
  - $dinner, $Dinner, and $DINNER are separate and distinct
  - **Avoid using variables names with same letters but differ in cases**

*Table 2-3. Disallowed variable names*

| Variable name | Flaw |
|---|---|
| `$2hot4u` | Begins with a number |
| `$drink-size` | Unacceptable character: - |
| `$drinkmaster@example.com` | Unacceptable characters: @ and . |
| `$drink!lots` | Unacceptable character: ! |
| `$drink+dinner` | Unacceptable character: + |

# Arithmetic Operation on Variables

- Arithmetic and string operators work on variables containing numbers or strings just like they do on literal numbers or strings.

*Example 2-17. Operating on variables*

```
$price = 3.95;
$tax_rate = 0.08;
$tax_amount = $price * $tax_rate;
$total_cost = $price + $tax_amount;

$username = 'james';
$domain = '@example.com';
$email_address = $username . $domain;

print 'The tax is ' . $tax_amount;
print "\n"; // this prints a line break
print 'The total cost is ' .$total_cost;
print "\n"; // this prints a line break
print $email_address;
```

Example 2-17 prints:

```
The tax is 0.316
The total cost is 4.266
james@example.com
```

# Putting Variables Inside Strings

- Frequently, you print the values of variables combined with other text
  - **Example:** display an HTML table with calculated values in the cells
- Use double-quoted strings and variable inside them

*Example 2-22. Interpolating in a here document*

```
$page_title = 'Menu';
$meat = 'pork';
$vegetable = 'bean sprout';
print <<<MENU
<html>
<head><title>$page_title</title></head>
<body>
<ul>
<li> Barbecued $meat
<li> Sliced $meat
<li> Braised $meat with $vegetable
</ul>
</body>
</html>
MENU;
```

Example 2-22 prints:

```
<html>
<head><title>Menu</title></head>
<body>
<ul>
<li> Barbecued pork
<li> Sliced pork
<li> Braised pork with bean sprout
</ul>
</body>
</html>
```

```
$email = 'jacob@example.com';
print "Send replies to: $email";
```

Example 2-21 prints:

```
Send replies to: jacob@example.com
```

# Putting Variables Inside Strings using {}

- You could surround the variable with curly braces to remove the confusion

*Example 2-23. Interpolating with curly braces*

```
$preparation = 'Braise';
$meat = 'Beef';
print "{$preparation}d $meat with Vegetables";
```

Example 2-23 prints:

```
Braised Beef with Vegetables
```

- If you could put {} around $preparation, it becomes $preparationd which is not the variable

# Exercises

# Exercises

Find the errors in this PHP program:

```
<? php
print 'How are you?';
print 'I'm fine.';
??>
```

Write a PHP program that computes the total cost of this restaurant meal: two hamburgers at $4.95 each, one chocolate milkshake at $1.95, and one cola at 85 cents. The sales tax rate is 7.5%, and you left a pre-tax tip of 16%.

Write a PHP program that sets the variable `$first_name` to your first name and `$last_name` to your last name. Print out a string containing your first and last name separated by a space. Also print out the length of that string.

Write a PHP program that uses the increment operator (++) and the combined multiplication operator (*=) to print out the numbers from 1 to 5 and powers of 2 from 2 ($2^1$) to 32 ($2^5$).