CTIS 256 Web Technologies II

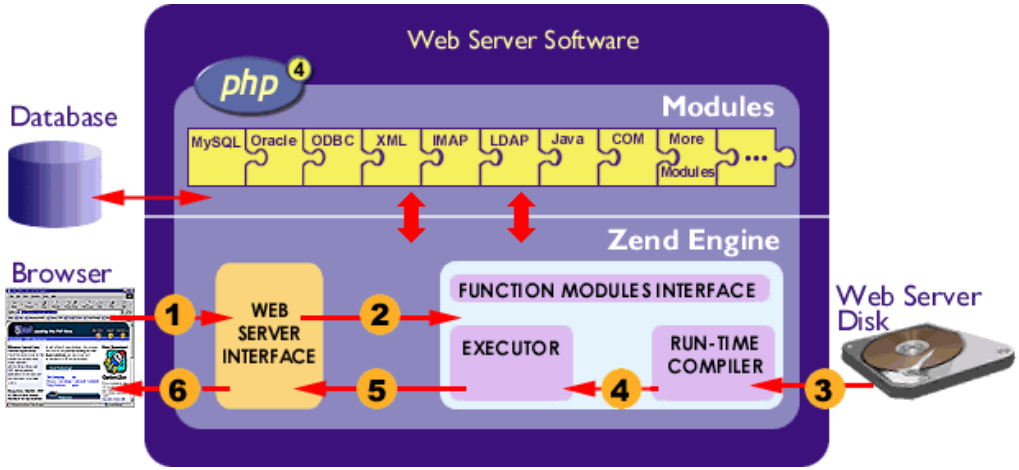
Note # 2

PHP Basics

Serkan GENÇ

PHP

- Php is an object oriented programming language **running on server-side**.
- Php programs produce outputs in **html** , image(jpg, png) , pdf, word, excel, xml, json, in any data file format.
- Its code is transparent (invisible) to the web client (browser).
- It is free and works in all platforms – platform independent.
- A Web server executes php programs using a php engine.
- PHP Engine takes php file and **compiles** into **php bytecodes**, and executes them.
- Using some extensions, PHP Engines can cache precompiled php bytecodes in shared memory to reduce parsing, disk I/O overhead for later requests. Popular cache and optimizers are APC, Xcache, Zend opcache, Zend Platform, WinCache.
- You can use WAMP install package for Apache Web Server, MySQL and PHP. It also makes necessary settings.

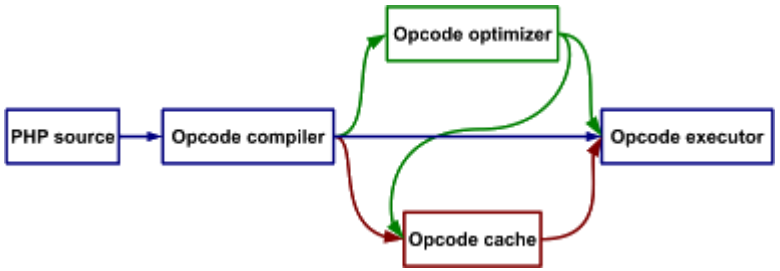


PHP Engine

How PHP Engine works :

1. Read php files from storage (Disk I/O)
2. Parsing of php commands
3. Compile to php's internal instructions (opcode)
4. Execution of opcodes with its virtual machine

OpCode Cache and Optimizer



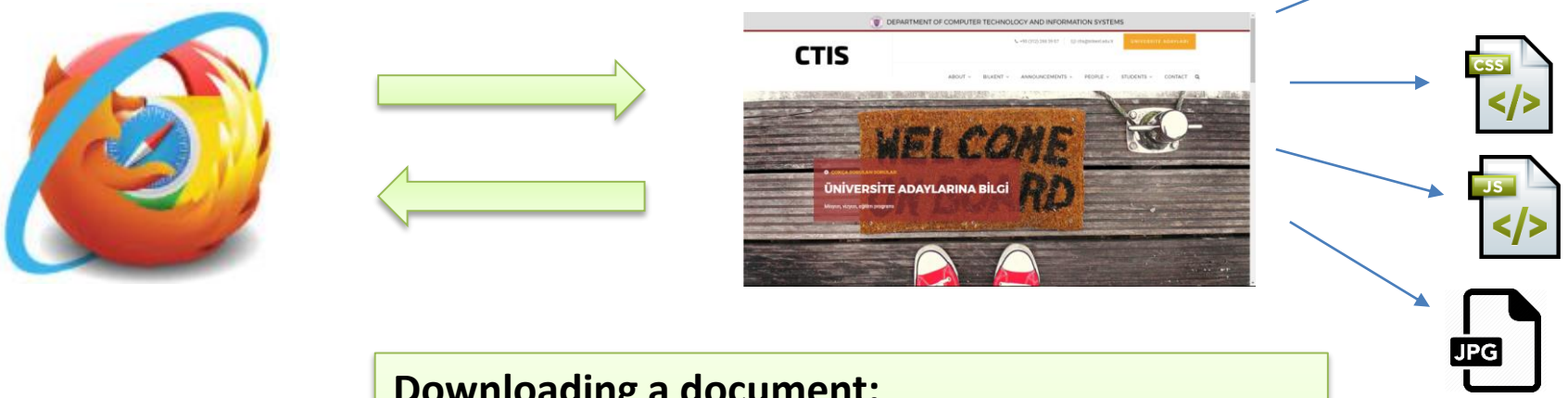
OpCode Cache extensions for Zend PHP Engine:

1. Zend opcache
2. Alternative PHP Cache (APC)
3. Windows PHP Cache

PHP Engine Implementations :

1. Zend PHP : original, complete and most widely used engine powered by Zend Engine, known as PHP. It works as an interpreter, it has a single-request-per-script execution model.
2. HipHop Virtual Machine: (HHVM) developed at Facebook, and available as open source. It converts PHP code into high-level bytecode (opcode) which is then translated into x86-64 machine code dynamically at runtime by Just-in-time (JIT) compiler. 6x performance improvements.

Retrieving Document



Downloading a document:

- A document is composed of many files including HTML, CSS, Javascript, images, font files
- When you request a page, the browser requests all files one by one using HTTP protocol.

PHP Codes

- Php codes are written in **php extension** files; main.php, Personel.class.php
- Codes are placed within PHP code blocks: **<?php ?>**
- A php file can contain pure php codes with a single code block. (**php file**)
- Any number of php code blocks can be embedded into html codes where we generate dynamic content. (**php web page**)
- Within code blocks, C-style single line/multiline comments can be used.

php file

```php
<?php
    // This is called PHP Code Block
    // You are allowed to write PHP codes here.
    // This is single line comment which valid within PHP code block
    /*
     This is multi-line comment
     Similar to C Programming.
     */

?>
```

optional

php web page

```php
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>My First PHP</title>
        <?php
            // Generate content dynamically here.
        ?>
    </head>
    <body>
        <h1>Introduction to PHP</h1>
        <?php
        // put your code here
        ?>
    </body>
</html>
```

Variables

- No explicit types, php decides types internally.
- Variables start with **$** sign.
- C-style variable syntax is used. A variable can not start with a number.
- Variable names are case-sensitive.
- Types: *integer*, *float*, *string*, *boolean*, *array*, *objects* and *null*.
- If a variable is not assigned a value, its type is null.
- const and define() are used to create constants.

```php
<?php

$age = 30 ;        // integer, 30 is an integer literal
$gpa = 3.1415 ;    // float, 3.1415 float literal
$smoking = false; // boolean true/false are boolean literals.

// string can be created two different ways.
$name = "Ali"     ; // using double quotes
$surname = 'Gul' ; // using single quotes

// array is used to store many items under a single name
// accessing an item through indexing.
$colors = array('red', 'green', 'blue') ;

// Objects, in details later.
$person = new Person();

// null is a type to declare the variable logically empty.
// it is available in the memory but
// does not have any meaningful value.
$file = null;

// closing tag is optional for php files.
?>

const PORT = 80 ; // compile-time constant, no $ sign before constant
define("IP", "192.168.1.1") ; // run-time constant

echo IP . ":" . PORT ;
```

String and Output

- String can be formed by two ways, "**with replacement**" or "**without replacement**".
- String with double quotes replaces the variables with their own values. Escape characters can be used (\n, \t, \r, \\, \", \$, \xHH, H means hex for UTF-8 characters.)
- String with single quotes does not replace the variables, only \' and \\ can be used in the content.
- **print** and **echo** are two main commands to generate output. Basically, they are doing the same thing. **echo** can get more than one parameters separated by comma.
- **var_dump(var)** : to display the variable for debugging purpose.
- **<?= expression ?>** to display the output of the expression in a short hand notation. <?= $lastname ?> to display $lastname variable.

```php
<?php

    $age = 25;
    $name = 'Ali' ;

    // creating string with double quotes.
    $output1 = "$name is $age years old.<br>" ;

    // string with single quotes.
    $output2 = '$name is $age years old.<br>' ;

    // difference between double and single quoted strings.
    print $output1 ;
    echo $output2 ;

    // Several uses of print and echo
    // for constant strings (not involving any variable), use single quote
    print 'Hello World!!<br>' ;
    print "<p>Your age is <span style='color:blue'>$age</span></p>" ;
    echo  '<img src="smiley.png" />' ;
    echo  '<p>Ali\'s car is Opel</p>';
    echo "<p>PSI symbol using UTF-8: \xCE\xA8</p>" ;
    // to debug the value of $output1
    var_dump($output1) ;
```

RESULT

Ali is 25 years old.
$name is $age years old.
Hello World!!

Your age is 25

Ali's car is Opel
PSI symbol using UTF-8: Ψ
string 'Ali is 25 years old.
' (length=24)

String – Heredoc/Nowdoc (optional)

- Heredoc is a block of text with replacement like string with double quotes.
- Nowdoc is a block of text without replacement. In Nowdoc, identifier is surrounded by single quotes. In the example below, notice 'EOT' after <<< .
- In both methods, after EOT and EOT;, there is a new line, there shouldn't be any space, and "EOT;" must be at the beginning of its own line. EOT (End of Text) is not fixed and can be any name.

```php
<?php
$secret = 25 ;
 // heredoc
$out = <<<EOT
     <h1>Unicode character : \xe2\x88\x91</h1>
     <p>The value of secret variable is $secret</p>
     <p>Long text can be written in heredoc area</p>
     <p>After EOT (or any choosen identifier), there should not be
        any space but new line, and "EOT;" must the at the beginning of
        the line and after semicolon, there is no space but a new line.
EOT;
 echo $out ;
```

Unicode character : ∑

The value of secret variable is 25

Long text can be written in heredoc area

After EOT (or any choosen identifier), there should n

```php
$secret = 25 ;
 // nowdoc
$out = <<<'EOT'
     <h1>Unicode character : \xe2\x88\x91</h1>
     <p>The value of secret variable is $secret</p>
     <p>Long text can be written in heredoc area</p>
     <p>After EOT (or any choosen identifier), there should not be
        any space but new line, and "EOT;" must the at the beginning of
        the line and after semicolon, there is no space but a new line.
EOT;
 echo $out ;
```

Unicode character : \xe2\x88\x91

The value of secret variable is $secret

Long text can be written in heredoc area

After EOT (or any choosen identifier), there should not be any space but n

Arithmetic Operators

- **Binary** operators require two operands while **unary** operators require only one operand.
- Binary operators: **+, - , *, / , %**
- Unary operators: **++, --**
- Arithmetic operators expect two **numeric operands**, if they are not numeric, it **automatically converts** to number.
- A string starting with a numeric character is converted to integer or float.
- If a string cannot be converted to a number, it is converted to **0**.

```php
<?php

  // string "123" -> int 123
  // string "35.0"  -> float 35.0
  // string "5pt"  -> int 5
  $n = ("123" + 12 - "35.0") / "5pt" ;    //  20

  // string starting with non-numeric characters
  // converted to 0 (zero).

  // string "ptz12" -> int 0
  $s = "ptz12" + 5 ;  // 5

  // string "12.5ptz" -> float 12.5
  $s = "12.5ptz" + 5 ;  // 17.5

  // postfix, prefix increment, decrement changes
  // the type of the variable
  $c = "5" ;  // string "5"

  $c++;
  var_dump($c) ; // int 6.
```

Unicode – UTF-8

- Character Set (charset) is a mapping from a number to a symbol such as 65 -> “A”.
- The first charset is ASCII, which represents 256 symbols/chars in 8-bits. (Historically, it supported 128 chars with 7-bits initially, then it was extended to 8-bits or 1 byte)
- ASCII is a charset for English language ONLY. It does not support accent characters from Turkish, German, French and letters from other languages such as Arabic, Hebrew, Chinese, Cyrillic letters, and emoji characters, and many other symbols from math, music, science.
- There are other charsets or code pages for other languages. For example, “latin5”/ “ISO-8859-9” is a charset for Turkish language ONLY. A text prepared in Turkish charset will not be viewed correctly in an ASCII system.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

- Unicode charset solves this problem by mapping **137,439** symbols (up to now). It is a lot bigger than ASCII. It contains all symbols used by human-being. Each year, a Unicode Consortium has been adding new symbols.
- The integer number for a symbol is called Unicode Code Point.

Code Points

Decimal	Hex	Symbol
65	U+0041	A
214	U+00D6	Ö
286	U+011E	Ğ
8721	U+2211	∑
128640	U+1F680	🚀

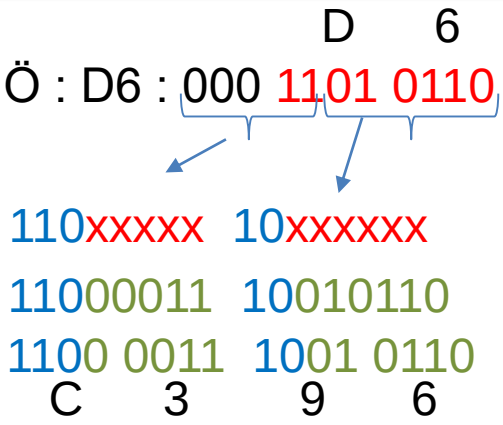
UTF (Unicode Transformation Format)-8

- The question is “How to represent all symbols in Unicode Character Set?”,
- Capacities for bytes: 1 byte : 256, 2 bytes: 65536, 3 bytes: 16,777,216, 4 bytes: 4,294,967,296.
- For all symbols, 2 bytes are not enough (65536 < 137,439). It requires at least 3 bytes. In fact, 3 bytes boundary is not memory architecture friendly, let’s use 4 bytes.
- However, if we use 4 bytes for each character, an English text will take up 4 times as much storage and memory space.
- In this part, Encoding comes into play. How to represent Unicode Code Points wisely? How to represent numbers efficiently? One solution may be to assign fewer bytes for frequently used symbols, and more bytes to rarely used ones. UTF-8 embodies this idea.
- UTF-8 uses 1 byte for the first 127 characters of Unicode Code Points. The first 127 chars of Unicode charset is exactly the same with ASCII because of backward compatibility. However, it can use up to 4 bytes for a symbol.
- If the Unicode code point is between U+0080 (128) and U+07FF (2047), it uses 2 bytes to represent this symbol. For example, Ö : **U+00D6** (214) -> **C3 96** in UTF-8

utf-8 encoding

Number of bytes	Bits for code point	First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
1	7	U+0000	U+007F	0xxxxxxx			
2	11	U+0080	U+07FF	110xxxxx	10xxxxxx		
3	16	U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
4	21	U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Code Point to UTF-8 Conversion



UTF-8, UCS-2, UTF-16

- **UCS-2** uses 2 bytes for a character, it can represent Unicode code points between U+0000 (0) and U+FFFF (65535). This group of characters (first plane) is called BMP (Basic Multilingual Plane). UCS-2 can represent only BMP. In this way, the length of a string can be calculated easily by dividing the number of bytes by 2. String processing is fast in UCS-2. However, it can not identify planes other than BMP, such as emoji characters. It is deprecated.
- **UTF-16** is successor of UCS-2. It can represent planes other than BMP. It is a variable-length encoding. It represents BMP characters with 2 bytes similar to UCS-2. However, for other planes, it produces two 16-bit code units per code points. For example, Rocket Code Point is U+1F680 which is beyond BMP (U+FFFF), it uses 2 16-bit code units 0xD83D 0xDE80 for the code point U+1F680. These 2 16-bit code units are called surrogate pairs. In *JavaScript Engines, Windows OS, Java* and in many languages and platforms, UTF-16 are used for internal encoding. However, UTF-16 never gained popularity on the web, where **utf-8 is dominant**.

“I am Çağla”

utf-8 text editor →

I		a	m		Ç		a	ğ		l	a
49	20	61	6d	20	c3	87	61	c4	9f	6c	61

how does operating system display the following bytes? (bytes -> symbols)

41	20	f0	9f	9a	80	20	68	65	72	65

UTF-8, UCS-2, UTF-16

- UCS-2 uses 2 bytes for a character, it can represent Unicode code points between U+0000 (0) and U+FFFF (65535). This group of characters (first plane) is called BMP (Basic Multilingual Plane). UCS-2 can represent only BMP. In this way, the length of a string can be calculated easily by dividing the number of bytes by 2. String processing is fast in UCS-2. However, it can not identify planes other than BMP, such as emoji characters. It is deprecated.
- UTF-16 is successor of UCS-2. It can represent planes other than BMP. It is a variable-length encoding. It represents BMP characters with 2 bytes similar to UCS-2. However, for other planes, it produces two 16-bit code units per code points. For example, Rocket Code Point is U+1F680 which is beyond BMP (U+FFFF), it uses 2 16-bit code units 0xD83D 0xDE80 for the code point U+1F680. These 2 16-bit code units are called surrogate pairs. In JavaScript Engines, Windows OS, Java and in many languages and platforms, UTF-16 are used for internal encoding. However, UTF-16 never gained popularity on the web, where utf-8 is dominant.

"I am Çağla"

utf-8 text editor

I		a	m		Ç		a	ğ		l	a
49	20	61	6d	20	c3	87	61	c4	9f	6c	61

You need to know what encoding is used in this string!!

utf-16	禮		🚀		骀		杖		敲		e
utf-8	A		🚀					h	e	r	e
ascii	A		ğ	Ÿ	š	€		h	e	r	e
string	41	20	f0	9f	9a	80	20	68	65	72	65

String Operations

- **Dot operator (.)** is used to concatenate two strings.
- **strlen(), mb_strlen()** : return the number of bytes in the string.
- Whitespace(**ws**) is a term for " ", "\t", and "\r", "\n".
- **trim()** : removes ws from the start and end of the string, while **ltrim()** and **rtrim()** removes start or end of the string respectively .
- **explode()** : breaks the string into an array. It makes parsing easier.
- **join()/implode()**: joins the array elements to produce a string.
- **substr(), mb_substr()**: copy a part of the string.
- **strpos(), mb_strpos()**: is used to search a substring in the source string.
- **strtoupper()**, **mb_strtoupper()**, **strtolower()**, **mb_strtolower()**, **ucfirst()**
- **strcmp():** to compare two strings.
- **mb_convert_encoding():** converts from one encoding to another.

```php
<?php

  $name = 'Özge' ;
  $surname = 'Yarman' ;
  $fullname = $name . ' ' . $surname ;

  // multi line string with dot operator
  $text = '<p style=\'color:red\'>' .
          "Your name is $fullname" .
          '</p>' ;

  // length of a string
  $len = strlen( 'Çağıl') ; // 8
  $lenMB = mb_strlen( 'Çağıl' , 'utf8') ; // 5

  $uname = '    Ali   ' ;
  $uname = trim($uname) ;  // 'Ali'

  $pos = mb_stripos('Acaba Çağıl burada mı', 'ÇAĞIL', 0, 'utf8') ;
```

Relational and Logical Operators

- Relational ops: **==, != , ===, !==, >, >=, <, <=**
- Logical ops: **&&, ||, !**
- Relational expression compares two operands and returns a boolean value (true, false) using relational operators.
- Logical expression applies a logical operator for two boolean values, and returns a boolean value.
- "Entirely number" is a string that contains a number(integer or float).
- Relational operators convert "entirely number" string to number.
- **Equality** ops(==, !=) implicitly convert its operands, but **Identity** ops (===, !==) do not make a type conversion.
- **Boolean False**: false, empty string '', '0', 0, 0.0, empty array [], null/NULL are considered false values.

```php
<?php

// Equality operator implicitly conver types.
// '3' entirely number string is converted number.
3 == '3' ;  // '3' -> 3 , true

// Types and values of operands should be the same
3 === '3' ;  // int and string types, false

// Relation operators convert
// "entirely number string" to number,
// then it compares.  "012" -> 12 -> 12.0
// "12.0" -> 12.0
// This is not string comparison.
'012' == '12.0' ;  // true


// if at least one of the operands is a number
// or "entirely number string"
// then the operator converts both operands to number.
'012' > '0012' ;  // false "012" -> 12, "0012"->12
  '0' == '0000' ; // true

// if both operands are string and not "entirely number",
// string comparison is done, lexicographic comparison
"cherry" > "banana" ; // true
```

Assignment Operators

- Assignment operator (=) **copies** values from right hand side (rhs) to left hand side (lhs).
- Other assignment operators: +=, -=, *=, %=, /= for arithmetic expressions, and .= for string appending.
- A reference operator (=&) is used to create an alias to a memory area. It shows the same memory area with the rhs.

```php
<?php

// copies value 5 into $a
$a = 5 ;
$a += 10 ;
$a *= 2 ;

// string operator, dot(.) operator for concatenation.
$name = 'ali' ;
$surname = 'gul' ;
$fullname = $name . ' ' . $surname . '<br>' ;

// Appending to the string.
$text = '<p style="color:blue">';
$text .= "My name is $fullname" ;
$text .= '</p>' ;

// Reference does not copy the value of rhs.
// It creates an alias to the same memory area.
$b = 10 ;
$c =& $b ;  // $c is a reference/alias to $b
$b++;
var_dump($c) ; // c = 11, b = 11
```

Logical Statements

- To change the flow of the program testing some states using logical expressions.
- **if** statement and **switch** statement are used to make decisions.
- Their syntax are exactly the same with C Programming language.
- In switch statement, test variable can be a string, which is different than C.
- **Ternary logical operator** (**?:**) is a shortcut for an if statement. A variable can be assigned to a value based on a condition.

```php
<?php

// Basic if statement syntax.
$bgColor = '#DDD' ;
// if $i is even, background color will be #AAA
if (  ( $i % 2 ) == 0 ) {
    $bgColor = '#AAA' ;
}

$paragraph = "<p style='background: $bgColor'>Content is here</p>" ;

// Complete if statement syntax.
$a = 5 ;
if ( $a < 5) {
    print 'Number is less than 5' ;
} else if ( $a > 5 ) {
    print 'Number is greater than 5' ;
} else {
    print 'Number is equal to 5' ;
}

// Switch to select from multiple choice.
switch( $gameState ) {
    case 'running' : print 'Game is in running state...' ; break ;
    case 'gameover' : print 'Display Gameover text' ; break ;
    default : print 'In other states..' ;
}

// Ternary operator, shorcut for if statement.
$bgColor = ( $i % 2) == 0 ? '#AAA' : '#DDD' ;
```

condition

true

false

Loop Statements

- To repeat some codes desired number of times or based on a certain condition.
- **for**, **do..while** and **while** statements are exactly the same with C programming language.
- **break** is used to get out of the loop.
- **continue** is used to start the next iteration.
- **for** and **foreach** loops are usually used to traverse the array items.
- **Array** is a variable type to store many values under a single name. Using index, any item can be accessed.
- Usual mistake is to forget **$** in front of index variable in ***for*** loops.

```php
<?php

// Basic for loop to display the squares of 3 numbers.
for ( $i = 0; $i < 3; $i++) {
    print "<p>Square of $i is " . ($i * $i) . "</p>" ;
}

// Basic do..while syntax.
$i = 0 ;
do {
    $i++;
    print "$i" ;
} while( $i < 5) ;

// Basic while syntax
$j = 0 ;
while ( $j < 10 ) {
    if ( $j % 2 == 0 ) continue;  // skip even numbers.
    print "$j<br>" ;
    $j++;
}

// Simple sequential Array
$color = array( 'red', 'green', 'blue') ;
// to add a new item.
$color[] = 'black' ;
// Traversing array items
for ( $i=0; $i < count($color); $i++) {
    print $color[$i] . '<br>' ;
}
// or
foreach( $color as $item) {
    print "$item<br>";
}
```

Alternative Syntax

- PHP generates html outputs
- To make it easier, there is a new syntax for "if, for, foreach, while and switch" statements.
- "**{**" is changed as "**:**", and "}" is replaced by "endif, endfor, endforeach, endwhile and endswitch"

```
<?php if ( $age < 12) : ?>
  <p>You cannot attend to this event.</p>
<?php else :  ?>
  <p>Prepare your equipments</p>
  <p><a href="rules.html">Read the rules here</a></p>
<?php endif; ?>

<table>
<?php foreach( $students as $student) : ?>
    <tr><td><?= $student ?></td>
<?php endforeach; ?>
</table>

<?php while($i < 10) : ?>
  echo $i , "<br>" ;
<?php endwhile; ?>
```

Arrays - Sequential

- Array types are sequential(index), associative and multidimensional.
- Sequential or index array is the array similar to C arrays. Using an **integer index**, an item can be accessed.
- *for* loop can be used for sequential array to traverse the items.
- Associative array is called "dictionary" or "hashmap"
- Each item is indexed by a **string index**. It maps a key to a value. It is called key-value pair.
- *for* loop does **not** work for associative arrays. "*foreach*" loop should be used.
- In Multidimensional (Mixed) array, an item can be a sequential or associative array.

```php
<?php
// 1. Sequential arrays
// a. how to define an array
$names = [] ; // empty array.
$colors = ["white" , "green" , "blue" ] ; // array with 3 items.
// b. get an item from the array. Use index number, starts from 0.
$bgColor = $colors[1] ; // green
// c. modify an item in the array.
$colors[2] = "black" ;
// d. add a new item to the array. (index part is empty )
$colors[] = "blue" ;
$colors[] = "yellow" ;
array_push($colors, "purple") ; // another way of adding item.
// e. creates and initialize an array.
$surname[] = "kumbaracı" ;
// f. get the size of the array.
$size = count( $colors ) ;
// g. display/travers all items in the array.
// 1st way:
print "<ul>" ;
for ( $i=0 ; $i< count($colors); $i++) {
  print "<li>" . $colors[$i] . "</li>" ;
}
print "</ul>" ;
// 2nd way : using foreach statement ( no need $i, count(), $i++
print "<ul>" ;
foreach ( $colors as $item) {
  print "<li>$item</li>" ;
}
print "</ul>" ;
// h. Embedding an array item in a string.
print "<p>The first color is {$colors[0]}</p>" ;
```

Arrays - Associative

- Array types are sequential(index), associative and multidimensional.
- Sequential or index array is the array similar to C arrays. Using an **integer index**, an item can be accessed.
- *for* loop can be used for sequential array to traverse the items.
- Associative array is called "dictionary" or "hashmap"
- Each item is indexed by a **string index**. It maps a key to a value. It is called key-value pair.
- *for* loop does **not** work for associative arrays. "*foreach*" loop should be used.
- In Multidimensional (Mixed) array, an item can be a sequential or associative array.

```php
// 2. Assosicative Arrays/Hash Map/Dictionary
//a. create an associative array. (indices are string not a number )
$person = ["name"=>"ahmet", "surname" => "Kama" , "id"=> 232454];
// b. get an item, store the value "ahmet" of the key/index "name"
$pName = $person["name"] ;
// c. modify an item.
$person["id"] = 4343232 ;
// d. add a new item/property.
$person["cgpa"] = 3.45 ;
// e. new array with init.
$car["brand"] = 'Volvo' ;  // creates a new array "car"

// f. traverse the associative array.
print "<h4>The properties/items in the associative array</h4>" ;
print "<ul>" ;
foreach ( $person as $key=>$value) {
    print "<li>$key : $value</li>" ;
}
print "</ul>" ;

// g. get keys of the array. ( array_keys() )
print "<p>Keys : ";
foreach( array_keys($person) as $key) {
    print "$key " ;
}
print "</p>";

 // g. get values of the array. ( array_values() )
print "<p>Values : ";
foreach( array_values($person) as $val) {
    print "$val " ;
}
print "</p>";
```

Arrays - Mixed

- Array types are sequential(index), associative and multidimensional.
- Sequential or index array is the array similar to C arrays. Using an **integer index**, an item can be accessed.
- *for* loop can be used for sequential array to traverse the items.
- Associative array is called "dictionary" or "hashmap"
- Each item is indexed by a **string index**. It maps a key to a value. It is called key-value pair.
- *for* loop does **not** work for associative arrays. "*foreach*" loop should be used.
- In Multidimensional (Mixed) array, an item can be a sequential or associative array.

```
// 3. Mixed arrays ( sequential and associative array )
// students is a sequential array with 4 items
// each of which is an associative array containing id and cgpa.
$students = [
    [ "id" => 12345, "cgpa" => 2.34] ,
    [ "id" => 45344, "cgpa" => 3.34] ,
    [ "id" => 63453, "cgpa" => 2.44] ,
    [ "id" => 45753, "cgpa" => 1.44] ,
    [ "id" => 63453, "cgpa" => 2.44] ,
    [ "id" => 53453, "cgpa" => 3.12]
] ;

// how to process/traverse items in the array.
print "<table border=1>" ;
print "<tr><td>ID</td><td>CGPA</td></tr>" ;
foreach ( $students as $std ) {
    print "<tr>" ;
    print "<td>{$std["id"]}</td>" ;
    // find the color based on cgpa
    // red for U, gray for S, green for H and HH.
    if ( $std['cgpa'] < 2.0) {
        $c = "red" ;
    } else if ( $std['cgpa'] < 3.0) {
        $c = "gray" ;
    } else {
        $c = "green";
    }

    print "<td style='color:$c'>{$std['cgpa']}</td>" ;
    print "</tr>" ;
}
print "</table>" ;
```