# PHP Constants

- Constants are like variables except that once they are defined they cannot be changed or undefined.
- A valid constant name starts with a letter or underscore (no $ sign before the constant name).

⚠️ Unlike variables, constants are automatically global across the entire script.

- To create a constant, use the `define()` function.

```
define(name, value, case-insensitive)
```

- ***name:*** *Specifies the name of the constant*
- ***value:*** *Specifies the value of the constant*
- ***case-insensitive:*** *Specifies whether the constant name should be case-insensitive. Default is false*

## PHP Constants

- The examples below creates constants with a case-sensitive name:

```php
<?php
    define("GREETING", "Welcome to my website!");
    echo GREETING;
?>
```

```php
<?php
    define("GREETING", "Welcome to my website!", true);
    echo GREETING;
?>
```

# PHP Constants

- Constant Arrays
  - *you can create a Array constant using the `define()` function.*
  - *The example below creates an Array constant:*

```php
<?php
    define("cars", [
        "Alfa Romeo",
        "BMW",
        "Toyota"
    ]);
    echo cars[0];
?>
```

## PHP Constants

- Constants are Global
  - *Constants are automatically global and can be used across the entire script.*
  - *The example below uses a constant inside a function, even if it is defined outside the function:*

```
<?php
    define("GREETING", "Welcome to my website!");

    function myTest() {
        echo GREETING;
    }

    myTest();
?>
```

## PHP Operators

- Operators are used to perform operations on variables and values.
- PHP divides the operators in the following groups:
  - *Arithmetic operators*
  - *Assignment operators*
  - *Comparison operators*
  - *Increment/Decrement operators*
  - *Logical operators*
  - *String operators*
  - *Array operators*
  - *Conditional assignment operators*

# PHP Operators

- Arithmetic Operators
  - *The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.*

| *Operator* | ***Name*** | ***Example*** | ***Result*** |
|---|---|---|---|
| + | *Addition* | *$x + $y* | *Sum of $x and $y* |
| - | *Subtraction* | *$x - $y* | *Difference of $x and $y* |
| * | *Multiplication* | *$x * $y* | *Product of $x and $y* |
| / | *Division* | *$x / $y* | *Quotient of $x and $y* |
| % | *Modulus* | *$x % $y* | *Remainder of $x divided by $y* |
| ** | *Exponentiation* | *$x ** $y* | *Result of raising $x to the $y'th power* |

# PHP Operators

- Assignment Operators
  - *The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.*

| *Assignment* | *Same as...* | *Description* |
|---|---|---|
| *x = y* | *x = y* | *The left operand gets set to the value of the expression on the right* |
| *x += y* | *x = x + y* | *Addition* |
| *x -= y* | *x = x - y* | *Subtraction* |
| *x *= y* | *x = x * y* | *Multiplication* |
| *x /= y* | *x = x / y* | *Division* |
| *x %= y* | *x = x % y* | *Modulus* |

# PHP Operators

- Increment / Decrement Operators
  - *The PHP increment operators are used to increment/* decrement *a variable's value.*

| *Operator* | *Name* | *Description* |
|---|---|---|
| ++$x | *Pre-increment* | *Increments $x by one, then returns $x* |
| $x++ | *Post-increment* | *Returns $x, then increments $x by one* |
| --$x | *Pre-decrement* | *Decrements $x by one, then returns $x* |
| $x-- | *Post-decrement* | *Returns $x, then decrements $x by one* |

## PHP Operators

- String Operators
  - *PHP has two operators that are specially designed for strings.*

| *Operator* | *Name* | *Example* | *Result* |
|---|---|---|---|
| . | *Concatenation* | *$txt1 . $txt2* | *Concatenation of $txt1 and $txt2* |
| .= | *Concatenation assignment* | *$txt1 .= $txt2* | *Appends $txt2 to $txt1* |

# PHP Operators

- Array Operators
  - *The PHP array operators are used to compare arrays.*

| *Operator* | *Name* | *Example* | *Result* |
|---|---|---|---|
| + | *Union* | *$x + $y* | *Union of $x and $y* |
| == | *Equality* | *$x == $y* | *Returns true if $x and $y have the same key/value pairs* |
| === | *Identity* | *$x === $y* | *Returns true if $x and $y have the same key/value pairs in the same order and of the same types* |
| != | *Inequality* | *$x != $y* | *Returns true if $x is not equal to $y* |
| <> | *Inequality* | *$x <> $y* | *Returns true if $x is not equal to $y* |
| !== | *Non-identity* | *$x !== $y* | *Returns true if $x is not identical to $y* |

# PHP Operators

- Conditional Assignment Operators
  - *The PHP conditional assignment operators are used to set a value depending on conditions:*

| *Operator* | *Name* | *Example* | *Result* |
|---|---|---|---|
| *?:* | *Ternary* | *$x = expr1 ? expr2 : expr3* | *Returns the value of $x.<br>The value of $x is expr2 if expr1 = TRUE.<br>The value of $x is expr3 if expr1 = FALSE* |
| *??* | *Null coalescing* | *$x = expr1 ?? expr2* | *Returns the value of $x.<br>The value of $x is expr1 if expr1 exists, and is not NULL.<br>If expr1 does not exist, or is NULL, the value of $x is expr2.<br>Introduced in PHP 7* |

# PHP Conditional Statements

- Very often when you write code, you want to perform different actions for different conditions. You can use conditional statements in your code to do this.
- In PHP we have the following conditional statements:
  - `if` statement - executes some code if one condition is true
  - `if...else` statement - executes some code if a condition is true and another code if that condition is false
  - `if...elseif...else` statement - executes different codes for more than two conditions
  - `switch` statement - selects one of many blocks of code to be executed

## PHP Conditional Statements

- The if Statement
  - *The* `if` *statement executes some code if one condition is true.*

```php
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
}
?>
```

# PHP Conditional Statements

- The if...else Statement
  - *The `if...else` statement executes some code if a condition is true and another code if that condition is false.*

```php
<?php
$t = date("H");

if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

## PHP Conditional Statements

- The if...elseif...else Statement
  - *The `if...elseif...else` statement executes different codes for more than two conditions.*

```php
<?php
$t = date("H");

if ($t < "10") {
    echo "Have a good morning!";
} elseif ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

# PHP switch Statement

- The `switch` statement is used to perform different actions based on different conditions.
- The `switch` statement to select one of many blocks of code to be executed.

```
switch (n) {
    case label1:
        code to be executed if n=label1;
        break;
    case label2:
        code to be executed if n=label2;
        break;
    case label3:
        code to be executed if n=label3;
        break;
    ...
    default:
        code to be executed if n is different from all labels;
}
```

## PHP switch Statement

```
<?php
$favcolor = "red";

switch ($favcolor) {
    case "red":
        echo "Your favorite color is red!";
        break;
    case "blue":
        echo "Your favorite color is blue!";
        break;
    case "green":
        echo "Your favorite color is green!";
        break;
    default:
        echo "Your favorite color is neither red, 
blue, nor green!";
}
?>
```

## PHP Loops

- Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.
- In PHP, we have the following looping statements:
  - *`while` - loops through a block of code as long as the specified condition is true*
  - *`do...while` - loops through a block of code once, and then repeats the loop as long as the specified condition is true*
  - *`for` - loops through a block of code a specified number of times*
  - *`foreach` - loops through a block of code for each element in an array*

# PHP Loops

- while Loop
  - *The* `while` *loop executes a block of code as long as the specified condition is true.*

```php
<?php
$x = 1;

while($x <= 5) {
    echo "The number is: $x <br>";
    $x++;
}
?>
```

## PHP Loops

- for Loop
  - *The `for` loop is used when you know in advance how many times the script should run.*

```php
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x <br>";
}
?>
```

## PHP Loops

- foreach Loop
  - *The `foreach` loop works only on arrays, and is used to loop through each key/value pair in an array.*

```php
<?php
$colors = array("red", "green", "blue", "yellow");

foreach ($colors as $value) {
    echo "$value <br>";
}
?>
```

## PHP Functions

- User Defined Functions
  - *Besides the built-in PHP functions, we can create our own functions.*
  - *A function is a block of statements that can be used repeatedly in a program.*
  - *A function will not execute immediately when a page loads.*
  - *A function will be executed by a call to the function.*
- A user-defined function declaration starts with the word *function*:

```
function functionName() {
    code to be executed;
}
```

## PHP Functions

- User Function Arguments
  - Information can be passed to functions through arguments. An argument is just like a variable.
  - Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

# PHP Functions

- User Function Arguments

```
<?php
    function familyName($fname) {
        echo "$fname Refsnes.<br>";
    }

    familyName("Jani");
    familyName("Hege");
    familyName("Stale");
    familyName("Kai Jim");
    familyName("Borge");
?>
```

# PHP Functions

- User Function Arguments

```
<?php
    function familyName($fname, $year) {
        echo "$fname Refsnes. Born in $year <br>";
    }

    familyName("Hege", "1975");
    familyName("Stale", "1978");
    familyName("Kai Jim", "1983");
?>
```

## PHP Functions

- PHP is a Loosely Typed Language
  - In the example above, notice that we did not have to tell PHP which data type the variable is.
  - PHP automatically associates a data type to the variable, depending on its value. Since the data types are not set in a strict sense, you can do things like adding a string to an integer without causing and error.
  - In PHP 7, type declarations were added. This gives us an option to specify the data type expected when declaring a function, and by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.

## PHP Functions

- PHP is a Loosely Typed Language
  - In the following example we try to add a number and a string with without the *strict* requirement:

```php
<?php
function addNumbers(int $a, int $b) {
    return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is NOT enabled "5 days" is changed to
int(5), and it will return 10
?>
```

## PHP Functions

- PHP is a Loosely Typed Language
  - In the following example we try to add a number and a string with with the `strict` requirement:

```php
<?php declare(strict_types=1); // strict requirement

function addNumbers(int $a, int $b) {
    return $a + $b;
}
echo addNumbers(5, "5 days");
// since strict is enabled and "5 days" is not an integer,
an error will be thrown
?>
```

## PHP Functions

- PHP is a Loosely Typed Language
  - To specify `strict` we need to set `declare(strict_types=1);.` This must be the on the very first line of the PHP file. Declaring strict specifies that function calls made in that file must strictly adhere to the specified data types
  - The `strict` declaration can make code easier to read, and it forces things to be used in the intended way.

# PHP Functions

- PHP Default Argument Value
  - he following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

```php
<?php declare(strict_types=1); // strict requirement
function setHeight(int $minheight = 50) {
    echo "The height is : $minheight <br>";
}

setHeight(350);
setHeight(); // will use the default value of 50
setHeight(135);
setHeight(80);
?>
```

## PHP Functions

- PHP Return Type Declarations
  - PHP 7 also supports Type Declarations for the `return` statement. Like with the type declaration for function arguments, by enabling the strict requirement, it will throw a "Fatal Error" on a type mismatch.
  - To declare a type for the function return, add a colon ( `:` ) and the type right before the opening curly ( `{` )bracket when declaring the function.

## PHP Functions

- PHP Return Type Declarations

```php
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : float {
    return $a + $b;
}
echo addNumbers(1.2, 5.2);
?>
```

```php
<?php declare(strict_types=1); // strict requirement
function addNumbers(float $a, float $b) : int {
    return (int)($a + $b);
}
echo addNumbers(1.2, 5.2);
?>
```

## PHP Arrays

- Create an Array in PHP
  - *the `array()` function is used to create an array*
  - *there are three types of arrays:*
    - **Indexed arrays** - Arrays with a numeric index
    - **Associative arrays** - Arrays with named keys
    - **Multidimensional arrays** - Arrays containing one or more arrays

## PHP Arrays

- PHP Indexed Arrays
  - *There are two ways to create indexed arrays:*
    - The index can be assigned automatically (index always starts at 0), like this:

```
$cars = array("Volvo", "BMW", "Toyota");
```

    - or the index can be assigned manually:

```
$cars[0] = "Volvo";
$cars[1] = "BMW";
$cars[2] = "Toyota";
```

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo "I like " . $cars[0] . ", " . $cars[1] . " and " .
$cars[2] . ".";
?>
```

# PHP Arrays

- The count() Function
  - *The `count()` function is used to return the length (the number of elements) of an array:*

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
echo count($cars);
?>
```

  - *To loop through and print all the values of an indexed array, you could use a `for` loop, like this:*

```
<?php
$cars = array("Volvo", "BMW", "Toyota");
$arrlength = count($cars);

for($x = 0; $x < $arrlength; $x++) {
    echo $cars[$x];
    echo "<br>";
}
?>
```

## PHP Arrays

- PHP Associative Arrays
  - *Associative arrays are arrays that use named keys that you assign to them.There are two ways to create an associative array:*

```
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
```

- or

```
$age['Peter'] = "35";
$age['Ben'] = "37";
$age['Joe'] = "43";
```

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
echo "Peter is " . $age['Peter'] . " years old.";
?>
```

# PHP Arrays

- PHP Associative Arrays
  - *Loop Through an Associative Array*
    - To loop through and print all the values of an associative array, you could use a *foreach* loop, like this:

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");

foreach($age as $x => $x_value) {
    echo "Key=" . $x . ", Value=" . $x_value;
    echo "<br>";
}
?>
```

## PHP Arrays

- PHP Multidimensional Arrays
  - *A multidimensional array is an array containing one or more arrays.*
  - *PHP understands multidimensional arrays that are two, three, four, five, or more levels deep. However, arrays more than three levels deep are hard to manage for most people.*

⚠ The dimension of an array indicates the number of indices you need to select an element.

- For a two-dimensional array you need two indices to select an element
- For a three-dimensional array you need three indices to select an element

# PHP Arrays

- PHP Two-dimensional Arrays
  - *A two-dimensional array is an array of arrays (a three-dimensional array is an array of arrays of arrays).*
  - *First, take a look at the following table:*

| Name | Stock | Sold |
|---|---|---|
| Volvo | 22 | 18 |
| BMW | 15 | 13 |
| Saab | 5 | 2 |
| Land Rover | 17 | 15 |

# PHP Arrays

- PHP Two-dimensional Arrays
  - *We can store the data from the table above in a two-dimensional array, like this:*

```
$cars = array
  (
  array("Volvo",22,18),
  array("BMW",15,13),
  array("Saab",5,2),
  array("Land Rover",17,15)
  );
```

# PHP Arrays

- PHP Two-dimensional Arrays
  - *Now the two-dimensional $cars array contains four arrays, and it has two indices: row and column.*
  - *To get access to the elements of the $cars array we must point to the two indices (row and column):*

```
<?php
echo $cars[0][0].": In stock: ".$cars[0][1].", sold:
".$cars[0][2].".<br>";
echo $cars[1][0].": In stock: ".$cars[1][1].", sold:
".$cars[1][2].".<br>";
echo $cars[2][0].": In stock: ".$cars[2][1].", sold:
".$cars[2][2].".<br>";
echo $cars[3][0].": In stock: ".$cars[3][1].", sold:
".$cars[3][2].".<br>";
?>
```

## PHP Arrays

- PHP Two-dimensional Arrays
  - *We can also put a `for` loop inside another `for` loop to get the elements of the $cars array (we still have to point to the two indices):*

```
<?php
for ($row = 0; $row < 4; $row++) {
  echo "<p><b>Row number $row</b></p>";
  echo "<ul>";
  for ($col = 0; $col < 3; $col++) {
    echo "<li>".$cars[$row][$col]."</li>";
  }
  echo "</ul>";
}
?>
```

## PHP Sort Functions For Arrays

- In this chapter, we will go through the following PHP array sort functions:
  - `sort()`- *sort arrays in ascending order*
  - `rsort()`- *sort arrays in descending order*
  - `asort()`- *sort associative arrays in ascending order, according to the value*
  - `ksort()`- *sort associative arrays in ascending order, according to the key*
  - `arsort()`- *sort associative arrays in descending order, according to the value*
  - `krsort()`- *sort associative arrays in descending order, according to the key*

## PHP Sort Functions For Arrays

- Sort Array in Ascending Order - sort()

```php
<?php
$cars = array("Volvo", "BMW", "Toyota");
sort($cars);
?>
```

- Result:

```
BMW

Toyota

Volvo
```

## PHP Sort Functions For Arrays

- Sort Array in Ascending Order - sort()

```
<?php
$numbers = array(4, 6, 2, 22, 11);
sort($numbers);
?>
```

- Result:

```
2
4
6
11
22
```

## PHP Sort Functions For Arrays

- Sort Array in Ascending Order - rsort()

```php
<?php
$cars = array("Volvo", "BMW", "Toyota");
rsort($cars);
?>
```

- Result:

```
Volvo
Toyota
BMW
```

## PHP Sort Functions For Arrays

- Sort Array in Ascending Order - rsort()

```php
<?php
$numbers = array(4, 6, 2, 22, 11);
rsort($numbers);
?>
```

- Result:

```
22
11
6
4
2
```

## PHP Sort Functions For Arrays

- Sort Array (Ascending Order), According to Value - asort()

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
asort($age);
?>
```

- Result:

```
Key=Peter, Value=35
Key=Ben, Value=37
Key=Joe, Value=43
```

## PHP Sort Functions For Arrays

- Sort Array (Ascending Order), According to Key - ksort()

```
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
ksort($age);
?>
```

- Result:

```
Key=Ben, Value=37
Key=Joe, Value=43
Key=Peter, Value=35
```

## PHP Sort Functions For Arrays

- Sort Array (Descending Order), According to Value - arsort()

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
arsort($age);
?>
```

- Result:

```
 Key=Joe, Value=43

 Key=Ben, Value=37

Key=Peter, Value=35
```

## PHP Sort Functions For Arrays

- Sort Array (Descending Order), According to Value - krsort()

```php
<?php
$age = array("Peter"=>"35", "Ben"=>"37", "Joe"=>"43");
krsort($age);
?>
```

- Result:

```
Key=Peter, Value=35

Key=Joe, Value=43

Key=Ben, Value=37
```

## PHP Global Variables - Superglobals

- Several predefined variables in PHP are "superglobals", which means that they are always accessible, regardless of scope - and you can access them from any function, class or file without having to do anything special.
- The PHP superglobal variables are:
  - *$GLOBALS*
  - *$_SERVER*
  - *$_REQUEST*
  - *$_POST*
  - *$_GET*
  - *$_FILES*
  - *$_ENV*
  - *$_COOKIE*
  - *$_SESSION*

## PHP Global Variables - Superglobals

- PHP $GLOBALS
  - *$GLOBALS is a PHP super global variable which is used to access global variables from anywhere in the PHP script (also from within functions or methods).*
  - *PHP stores all global variables in an array called $GLOBALS[index]. The index holds the name of the variable.*

## PHP Global Variables - Superglobals

- PHP $GLOBALS
  - *The example below shows how to use the super global variable $GLOBALS:*

```php
<?php
$x = 75;
$y = 25;

function addition() {
    $GLOBALS['z'] = $GLOBALS['x'] + $GLOBALS['y'];
}

addition();
echo $z;
?>
```

# PHP Global Variables - Superglobals

- PHP $_SERVER
  - *$_SERVER is a PHP super global variable which holds information about headers, paths, and script locations.*

```php
<?php
echo $_SERVER['PHP_SELF'];
echo "<br>";
echo $_SERVER['SERVER_NAME'];
echo "<br>";
echo $_SERVER['HTTP_HOST'];
echo "<br>";
echo $_SERVER['HTTP_REFERER'];
echo "<br>";
echo $_SERVER['HTTP_USER_AGENT'];
echo "<br>";
echo $_SERVER['SCRIPT_NAME'];
?>
```