Constants, Operators & Control Statements

CONTENTS

Constants

Constants are also variables except for that once they are defined they cannot be changed or undefined. In many other languages, constants are faster than variables and so they are recommended. In PHP, they are small in number and are faster. Advantage of using constants is that they do not have a dollar sign and are visibly different from variables. Unlike variables, constants are automatically global across your entire script.

define() function is used to set constant, it takes two parameters, first argument will be the name of the constant that you want to set, and the second argument will be the value that you want to set.

Example

```
<?php
define("media", “Sakshi”);
print media;
?>
```

Output

```
Sakshi
```

Here we are having an optional parameter for define() function. We have both case-sensitive and case-insensitive constants.

Example for case-sensitive

```
<?php
define("MEDIA", “Sakshi”);
print MEDIA;
?>
```

Output

```
Sakshi
```

Example for case-insensitive

```
<?php
define("MEDIA", “SakshiEducation”,true);
print media;
?>
```

Output

```
Sakshi Education
```

Constant() function

The name itself indicates that, this function will return the value of a constant. This is useful when you want to get the value of a constant, this value is stored in a variable or returned by a function.

Example

```
<?php
define("VALUE", 50);
echo VALUE;
echo constant("VALUE");
?>
```

The two echo statements will give same output.

Output

```
5050
```

Example

```
<?php
define("VALUE", 50);
$my_value =”VALUE”;
echo constant($my_value);
?>
```

Output

```
50
```

Example

```
<?php
define("VALUE", 50);
$my_value =50;
echo constant($my_value);
?>
```

Output

```
Warning: constant(): Couldn't find constant 50
in C:\xampp\htdocs\100bp_profile\index.php on line 4
```

By the above example it is clear that constant() function can carry or show only constant values.

Operators

An operator is a symbol that specifies a particular action in an expression. In PHP, automatic type conversion happens based on the type of operator placed between the two operands.

- Arithmetic Operators
- Assignment Operators
- String Operators
- Increment and Decrement Operators
- Logical Operators
- Equality Operators
- Comparison Operators
- Bitwise Operators

Arithmetic Operators

Arithmetic Operators are commonly used for arithmetic operations like addition, Subtraction, Multiplication, Division etc…

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

Assignment Operators

Assignment Operator is used to assign some value the variable. “=” is the assignment operator in PHP.

Example	Description	Output
$a =15	Assignment	$a equals to 15
$a +=15	Addition and Assignment	$a equals to $a +15
$a *=15	Multiplication and Assignment	$a equals to $a multiplied by 15
$a /=15	Division and Assignment	$a equals to $a divided by 15
$a .=15	Concatenation and Assignment	$a equals to $a concatenated with 15

String Operator

String Operators are used to concatenate strings together, there are two such operators “concatenate operator (.)” and “concatenate assignment operator (.=)”.

Example	Description	Output
$a =”sakshi”.”education”;	Concatenation	$a is assigned to string sakshieducation
$a .=“sakshi”	Concatenation and Assignment	$a equals its current value concatenated with "sakshi"

Increment and Decrement Operators

To increment a value of a variable, we use Increment Operator (++), whereas to decrement the value of a variable we use Decrement Operator (--).

Operator	Example	Operator Name	Output
++	++$a	Pre-increment	Increments $a by one, then returns $a
++	$a ++	Post-increment	Returns $a, then increments $a by one
-	--$a	Pre-decrement	Decrements $a by one, then returns $a
--	$a--	Post-decrement	Returns $a, then decrements $a by one

Logical Operators

Like arithmetic operators, Logical Operators also plays a major role in many of the applications. Logical Operators are most commonly used to check some conditions in applications.

Example	Name	Output
$a &&$b	AND	True if both $a and $b are true
$a AND$b	AND	True if both $a and $b are true
$a || $b	OR	True if either $a or $b is true
$a OR $b	OR	True if either $a or $b is true
!$a	OT	True if $a is not true
NOT $a	NOT	True if $a is not true
$a XOR $b	Exclusive OR	True if only $a or only $b is true

Equality Operator

Equality operators are commonly used to compare two variables or two values.

Example	**Description**	**Output**
$a ==$x	Is equal to	True if $a and $x are equivalent
$a !=$x	Is not equal to	True if $a is not equal to $x
$a === $x	Is identical to	True if $a and $b are equivalent and $a and $x have the same type

The purpose of " = " and " == " is completely different. " = " is used to assign value to variables, whereas " == " will check for equivalent.

Comparison Operators

To compare two values we use Comparison Operators.

Example	**Description**	**Output**
$a <$x	Less than	True if $a is less than $b
$a >$x	Is not equal to	True if $a is greater than $b
$a <= $x	Less than or equal to	True if $a is less than or

		equal to $b
$a >= $x	Greater than or equal to	True if $a is greater than or equal to $b
($a == 10) ? 5 : -1	Ternary	If $a equals 12, return value is 5; otherwise, return value is -1

Bitwise Operator

The bitwise operators have slight variations from some of the logical operators, but results in drastically different outcomes.

Example	**Description**	**Output**
$a &$x	AND	And together each bit contained in $a and $x
$a \|$x	OR	Or together each bit contained in $a and $x
$a ^ $x	XOR	Exclusive—or together each bit contained in $a and $x
~ $x	NOT	Negate each bit in $b
$a << $x	Shift left	$a will receive the value of $x shifted left two bits
$a >> $x	Shift right	$a will receive the value of $x shifted right two bits

Control Structures

Defining execution characteristics such as, whether and how many times a particular code statement will run and as well as when a code block will quit from particular block.

Conditional Statements

Conditional statements will execute based on various input values that we pass to them.

If Statement

If our condition is true then the block of statements inside the If will be executed.

Syntax

```
If(Condition){
     Statement 1;
     Statement 2;
     Statement 3;
          ..
          ..
          ..
          ..
}
```

If you want to print "Sakshi" with some condition then follow the following example.

Example

```
<?php
   $value = "true";
   if ($_POST['name']== $value)
   {
      echo "<p>Sakshi Education</p>";
   }
?>
```

The else Statement

The else statement is an extension to 'If' statement. If the 'If' condition is true then the block of statements within 'if' statement will be executed, if the condition is false then the block of statements in the 'else' condition will be executed.

Syntax

```
If(Condition){
     Statement 1;
     Statement 2;
     Statement 3;
```

```
        ..
        ..
        ..
        ..
} else {
    Statement a;
    Statement b;
    Statement c;
        ..
        ..
        ..
        ..
}
```

Example

```
<?php
  $value = “sakshi”;
  if ($_POST[‘name’]== $value)
  {
     echo "<p>Sakshi Education</p>";
  }else{
echo "<p>Sakshi Post</p>";
         }
?>
```

The if-else-if Statement

The if-else-if combination works nicely in an “either-or-nor” situation—that is, a situation in which only possible outcomes will come.

Syntax

```
If(Condition){
    Statement 1;
    Statement 2;
    Statement 3;
        ..
        ..
```

```
          ..
          ..
} else if(Condition){
      Statement a;
      Statement b;
      Statement c;
          ..
          ..
          ..
          ..
}else {
      Statement a;
      Statement b;
      Statement c;
          ..
          ..
          ..
          ..
}
```

Example

```
<?php
   $value = “sakshi”;
   if ($value==“sakshi”)
   {
      echo "<p>Sakshi Education</p>";
   }else if($value==“sakshi post”)
{
echo "<p>Sakshi Post</p>";
          }
      else if{
echo "<p>Sakshi </p>";
          }
?>
```

The Switch Statement

To compare a large number of variables, we prefer Switch Statement, which are a variant of the if-else combination.

Syntax

```
switch (n) {
   case value1:
      code to be executed if n=value1;
      break;
   case value2:
      code to be executed if n= value 2;
      break;
   case value3:
      code to be executed if n= value 3;
      break;
   ...
   default:
      code to be executed if n is different from all values;
}
```

Example

```
<?php

$category = “education”

switch($category) {

case "news":

echo "<p>Watch news at sakshi post</p>";

break;

case "education":

echo "<p>learn technology at sakshi education</p>";

break;

case "bhavita":

echo "<p>Bhavita to make your life</p>";
break;
default:
echo "<p>Welcome to Sakshi</p>";
}
?>
```

Looping Statements

If we want to execute a particular block again and again then we prefer looping statements. Here we have a set of looping statements in PHP.

- While
- Do-While
- For
- Foreach

While

As long as the specific condition is true the while loop will be executed.

Syntax

```
While(condition){
    statements block;
}
```

Example

```
<?php
  $count = 1;
  while($count<= 10) {
    echo "The count  is: $count<br>";
    $count++;
    }
?>
```

do…while

In do..while, it verifies the condition at the end than at the beginning of the block. So the statements inside the do..while block will execute at least once.

Syntax

```
do {
statements
} while (condition);
```

Example

```
<?php
$count = 5;
     do
        {
           echo " the count is $count <br/>";
          $count--;
       }while($count>3);
?>
```

For Loop

If we know how many times the code to be executed then we prefer or loop.

Syntax

```
For(expression1;expression2;expression3){
      statements block;
}
```

In above syntax

- expression1 will hold the initial value for looping.
- expression2 will deals with condition to execute the loop.
- expression3 will deals the increment or decrement counter of for loop.

The PHP 'foreach' Loop

Foreach works with arrays only, and it deals with keys and value pair for looping.

Syntax

```
For($array as $value){
      statements block;
}
```

Example

```
<?php
$bikes = array("hero", "honda", "Cbz", "yamaha");
foreach ($bike as $value) {
   echo "$value <br>";
}
?>
```

The break and goto Statements

To terminate the execution of a loop then we prefer to use break statement or goto statements, the next statement after the loop will be executed.

Example

```
<?php
$count = 1;
for($i=0;$i<10;$i++){
echo “count is $count”;
if($count == 5){
break;
}
}
?>
```

You can suddenly jump to a specific location outside of a looping or conditional construct

Example

```
<?php
for ($count = 0; $count < 10; $count++)
{
$randomNumber = rand(1,50);
if ($randomNumber < 10)
goto less;
else
echo "Number greater than 10: $randomNumber<br />";
}
less:
```

```
echo "Number less than 10: $randomNumber<br />";
?>
```

The include() Statement

The “include ()” statement will include a file into the location where it was called. Including a file produces the same result as copying the data from the file specified into the location in which the statement appears.

Syntax

```
include(/path/filename)
```

Example

```
<?php

include "/usr/local/lib/php/more/init.inc.php";

?>
```

Example

```
<?php
if (expression)
include ('filename');
else
include ('another_filename');
?>
```