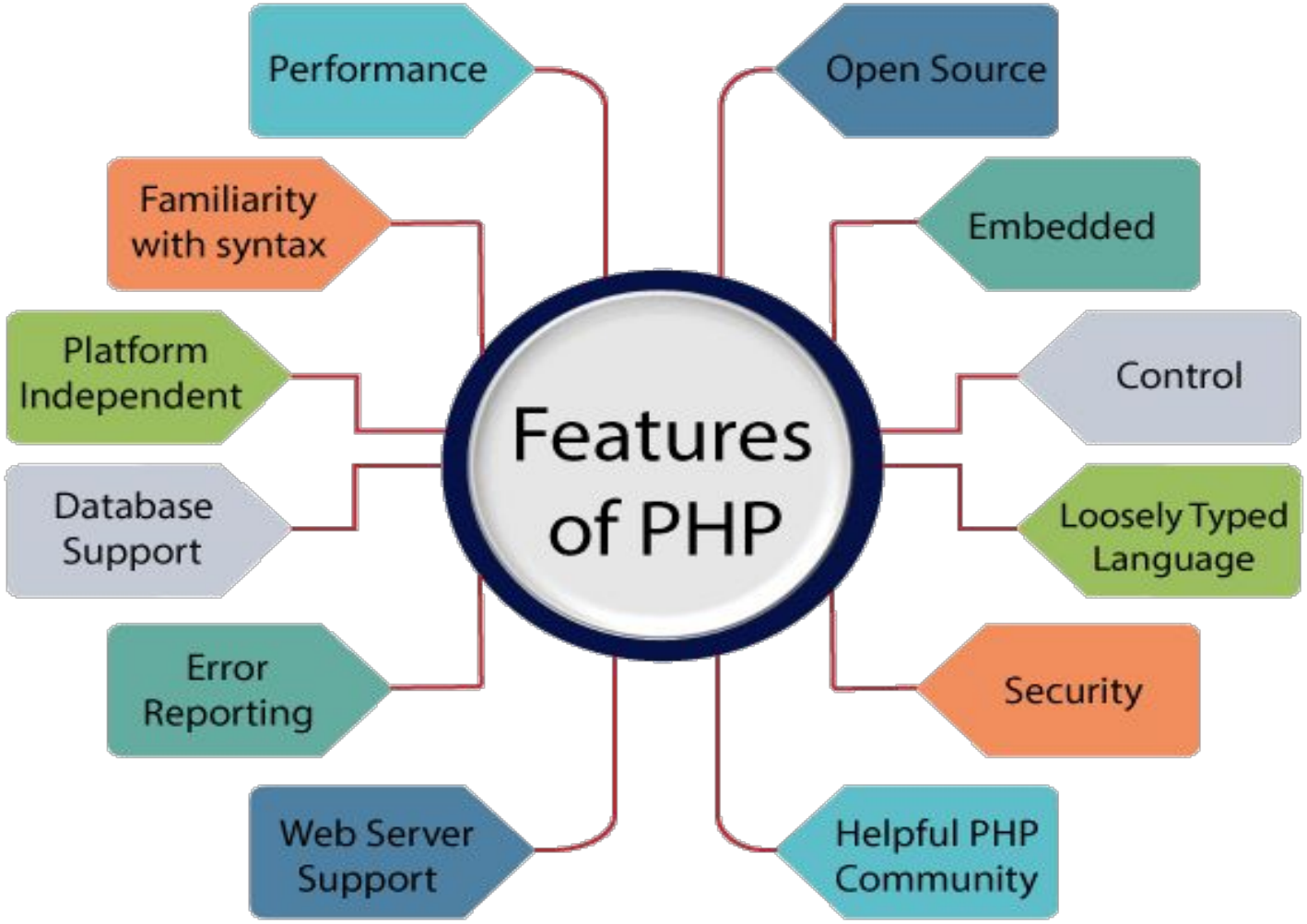
UNIT-3 PHP BASICS

- PHP **Hypertext Pre-processor** is a server scripting language, and a powerful tool for making dynamic and interactive Web pages.
- PHP is an open-source, interpreted, and object-oriented scripting language that can be executed at the server-side. PHP is well suited for web development. Therefore, it is used to develop web applications (an application that executes on the server and generates the dynamic page.).
- PHP was created by **Rasmus Lerdorf in 1994** but appeared in the market in 1995. **PHP 7.4.0** is the latest version of PHP, which was released on **28 November**. Some important points need to be noticed about PHP are as followed:
- PHP is an interpreted language, i.e., there is no need for compilation.
- PHP is faster than other scripting languages, for example, ASP and JSP.
- PHP is a server-side scripting language, which is used to manage the dynamic content of the website.
- PHP can be embedded into HTML.
- PHP is an object-oriented language.
- PHP is an open-source scripting language.
- PHP is simple and easy to learn language.

FEARTURES OF PHP



- **Performance:**

  PHP script is executed much faster than those scripts which are written in other languages such as JSP and ASP. PHP uses its own memory, so the server workload and loading time is automatically reduced, which results in faster processing speed and better performance.

- **Open Source:**

  PHP source code and software are freely available on the web. You can develop all the versions of PHP according to your requirement without paying any cost. All its components are free to download and use.

- **Familiarity with syntax:**

  PHP has easily understandable syntax. Programmers are comfortable coding with it.

- **Embedded:**

  PHP code can be easily embedded within HTML tags and script.

- **Platform Independent:**

  PHP is available for WINDOWS, MAC, LINUX & UNIX operating system. A PHP application developed in one OS can be easily executed in other OS also.

- **Database Support:**

  PHP supports all the leading databases such as MySQL, SQLite, ODBC, etc.

- **Error Reporting -**

  PHP has predefined error reporting constants to generate an error notice or warning at runtime. E.g., E_ERROR, E_WARNING, E_STRICT, E_PARSE.

- **Loosely Typed Language:**

  PHP allows us to use a variable without declaring its datatype. It will be taken automatically at the time of execution based on the type of data it contains on its value.

- **Web servers Support:**

  PHP is compatible with almost all local servers used today like Apache, Netscape, Microsoft IIS, etc.

- **Security:**

  PHP is a secure language to develop the website. It consists of multiple layers of security to prevent threads and malicious attacks.

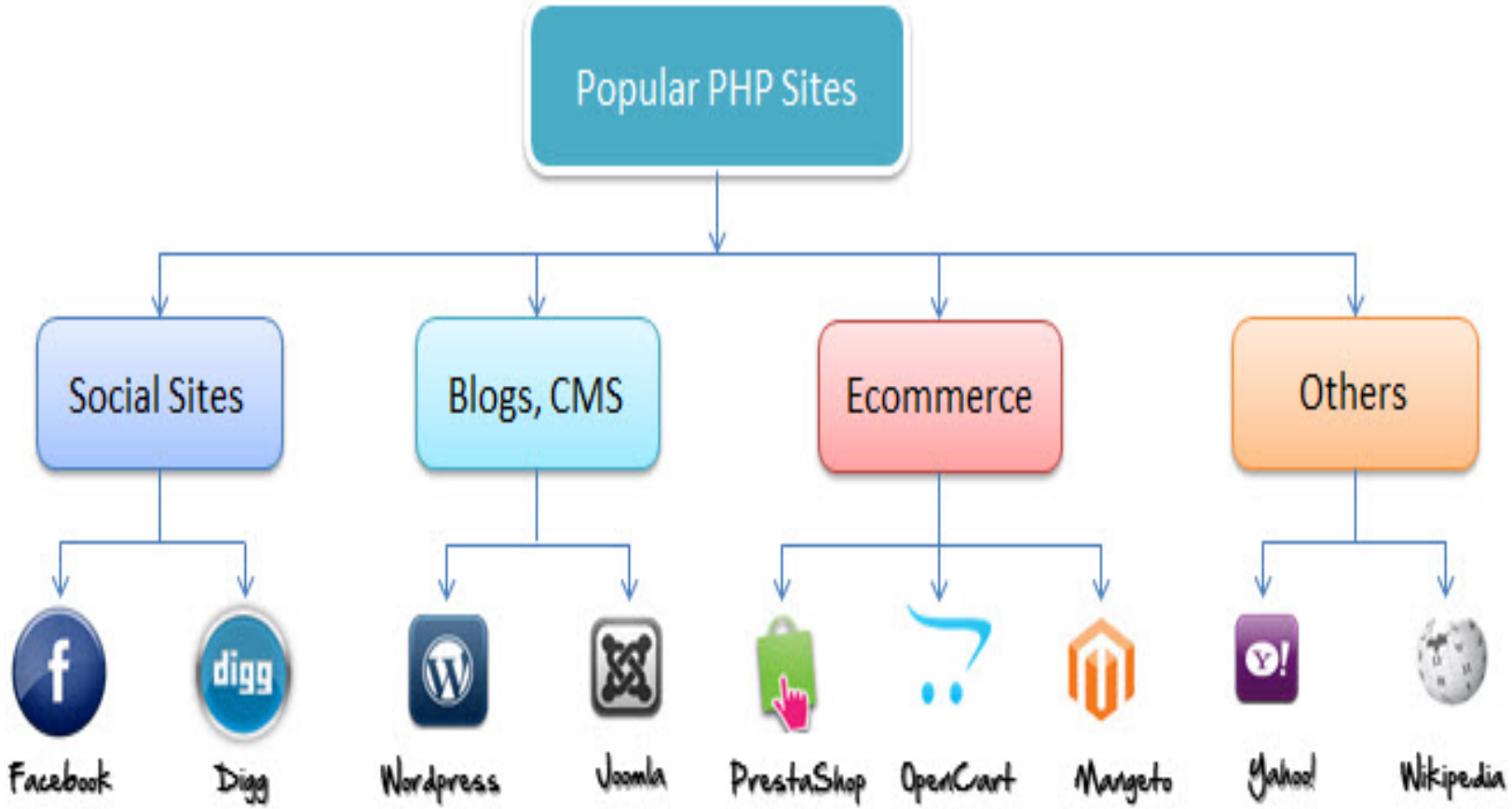
- **Control:**

  Different programming languages require long script or code, whereas PHP can do the same work in a few lines of code. It has maximum control over the websites like you can make changes easily whenever you want.

- **A Helpful PHP Community:**

  It has a large community of developers who regularly updates documentation, tutorials, online help, and FAQs. Learning PHP from the communities is one of the significant benefits.

PHP WEBSITES



PHP SYNTAX

- A PHP script starts with the <?php and ends with the ?> tag.
- The PHP delimiter <?php and ?> in the following example simply tells the PHP engine to treat the enclosed code block as PHP code, rather than simple HTML.
- <?php

```
// Some code to be executed
 echo "Hello, world!";
 ?>
```

- **Embedding PHP within HTML**

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<title>A Simple PHP File</title>
 </head> <body>
<h1><?php echo "Hello, world!"; ?></h1>
</body> </html>
```

- **PHP Comments**
- PHP support single-line as well as multi-line comments.
- To write a single-line comment either start the line with either two slashes (//) or a hash symbol (#).
- For example:

```
<?php
// This is a single line comment
# This is also a single line comment
echo "Hello, world!"; ?>
```

-

```
<?php
/* This is a multiple line comment block that spans across more than one line */
echo "Hello, world!"; ?>
```

PHP VARIABLES

- Variables in a program are used to store some values or data that can be used later in a program. The variables are also like containers that store character values, numeric values, memory addresses, and strings. PHP has its own way of declaring and storing variables. There are few rules, that needs to be followed and facts that need to be kept in mind while dealing with variables in PHP:
- Any variables declared in PHP must begin with a dollar sign ($), followed by the variable name.
- A variable name can only contain alphanumeric characters and underscores (i.e., 'a-z', 'A-Z', '0-9 and '_') in their name. Even it cannot start with a number.
- A constant is used as a variable for a simple value that cannot be changed. It is also case-sensitive.
- Assignment of variables is done with the assignment operator, "equal to (=)". The variable names are on the left of equal and the expression or values are to the right of the assignment operator '='.
- One must keep in mind that variable names in PHP names must start with a letter or underscore and no numbers.
- PHP is a loosely typed language, and we do not require to declare the data types of variables, rather PHP assumes it automatically by analyzing the values. The same happens while conversion. No variables are declared before they are used. It automatically converts types from one type to another whenever required.

```php
<?php
// These are all valid declarations
$val = 5;
$val2 = 2;
$x_Y = "gfg";
$_X = "GeeksforGeeks";

// This is an invalid declaration as it
// begins with a number
$10_ val = 56;
// This is also invalid as it contains
// special character other than _
$f.d = "num";

?>
```

- PHP variables are case-sensitive, i.e., $sum and $SUM are treated differently.

```
<?php
  // Assign value to variable $color = "blue";
  // Try to print variable value
   echo "The color of the sky is " . $color . "<br>";
  echo "The color of the sky is " . $Color . "<br>";
  echo "The color of the sky is " . $COLOR . "<br>"; ?>
```

If you try to run the above example code it will only display the value of the variable $color and produce the **"Undefined variable" warning for the variable $Color and $COLOR.**

- However the some of the keywords, function and classes names are case-insensitive. As a result calling the gettype() or GETTYPE() produce the same result.

```
<?php
   // Assign value to variable $color = "blue";
   // Get the type of a variable
   echo gettype($color) . "<br>";
   echo GETTYPE($color) . "<br>"; ?>
?>
```

PHP VARIABLES SCOPE

- The scope of a variable is defined as its range in the program under which it can be accessed. In other words, "The scope of a variable is the portion of the program within which it is defined and can be accessed."

  PHP has three types of variable scopes:

- Local variable
- Global variable
- Static variable

LOCAL VARIABLES

- The variables declared within a function are called local variables to that function and has its scope only in that particular function. In simple words, it cannot be accessed outside that function.

```
<?php
$num = 60;
function local_var()
{
  // This $num is local to this function
    // the variable $num outside this function
    // is a completely different variable
    $num = 50;
    echo "local num = $num \n";
}

local_var();

// $num outside function local_var() is a
// completely different Variable than that of
// inside local_var()
echo "Variable num outside local_var() is $num \n";
?>
```

GLOBAL VARIABLES

- The variables declared outside a function are called global variables. These variables can be accessed directly outside a function. To get access within a function we need to use the “global” keyword before the variable to refer to the global variable.

```
<?php
$num = 20;
// function to demonstrate use of global variable
function global_var()
{
   // we have to use global keyword before
    // the variable $num to access within
    // the function
    global $num;

    echo "Variable num inside function : $num \n";
}

global_var();

echo "Variable num outside function : $num \n";

?>
```

STATIC VARIABLE

- It is the characteristic of PHP to delete the variable, ones it completes its execution and the memory is freed. But sometimes we need to store the variables even after the completion of function execution. To do this we use static keyword and the variables are then called as static variables. PHP associates a data type depending on the value for the variable.

```
<?php
// function to demonstrate static variables
function static_var()
{
   // static variable
    static $num = 5;
    $sum = 2;
    $sum++;
    $num++;
    echo $num, "\n";
    echo $sum, "\n";
}
// first function call
static_var();
// second function call
static_var();

?>
```

DECLARATION OF VARIABLE

- The $ operator in PHP is used to declare a variable. In PHP, a variable starts with the $ sign followed by the name of the variable. For example, below is a string variable:

- $var_name = "Hello World!";

  The *$var_name* is a normal variable used to store a value. It can store any value like integer, float, char, string etc.

  On the other hand, the *$$var_name* is known as reference variable where $var_name is a normal variable.

  The $$var_name used to refer to the variable with the name as value of the variable $var_name.

- $Hello = "Geeks for Geeks"

  $var = "Hello"

  echo $var

  echo $$var

  Output : Hello Geeks for Geeks

  Explanation: In the above example, *$var* stores the value “Hello”, so $$var will refer to the variable with name Hello i.e., $Hello.

DATA TYPES USED BY PHP

- Integers
- Doubles
- NULL
- Strings
- Booleans
- Arrays
- Objects
- Resources

- PHP data types are used to hold different types of data or values. PHP supports 8 primitive data types that can be categorized further in 3 types:
- Scalar Types (predefined)
- Compound Types (user-defined)
- Special Types

Scalar Types

- It holds only single value. There are 4 scalar data types in PHP.

boolean

integer

float

string

Compound Types

- It can hold multiple values. There are 2 compound data types in PHP.

array

object

Special Types

resource

NULL

- Boolean

  Booleans are the simplest data type works like switch. It holds only two values: **TRUE (1)** or **FALSE (0)**.

```
<?php
  if (TRUE)
       echo "This condition is TRUE.";
  if (FALSE)
       echo "This condition is FALSE.";
?>
```

- **Output:**

This condition is TRUE.

- **Integer**
- Integer means numeric data with a negative or positive sign. It holds only whole numbers, i.e., numbers without fractional part or decimal points.
- **Rules for integer:**
- An integer can be either positive or negative.
- An integer must not contain decimal point.
- Integer can be decimal (base 10), octal (base 8), or hexadecimal (base 16).
- The range of an integer must be lie between 2,147,483,648 and 2,147,483,647 i.e., -2^31 to 2^31.
- **Example:**

```
<?php
    $dec1 = 34;
    $oct1 = 0243;
    $hexa1 = 0x45;
    echo "Decimal number: " .$dec1. "</br>";
    echo "Octal number: " .$oct1. "</br>";
    echo "HexaDecimal number: " .$hexa1. "</br>";
?>
```

Output:

Decimal number: 34 Octal number: 163 HexaDecimal number: 69

- **Float:** A floating-point number is a number with a decimal point. Unlike integer, it can hold numbers with a fractional or decimal point, including a negative or positive sign.
- **Example:**

```
<?php
  $n1 = 19.34;
  $n2 = 54.472;
  $sum = $n1 + $n2;
  echo "Addition of floating numbers: " .$sum;
?>
```

Output: Addition of floating numbers: 73.812

- **String** is a non-numeric data type. It holds letters or any alphabets, numbers, and even special characters.
- String values must be enclosed either within **single quotes** or in **double quotes**. But both are treated differently.
- **Example:**

```
<?php
    $company = "JavaBase";
   //both single and double quote statements will treat different
    echo "Hello $company";
    echo "</br>";
    echo 'Hello $company';
?>
```

Output: Hello JavaBase
Hello $company

- **Array** is a compound data type. It can store multiple values of same data type in a single variable.
- **Example:**

```
<?php
 $bikes = array ("Royal Enfield", "Yamaha", "KTM");
 var_dump($bikes);    //the var_dump() function returns the datatype and values
    echo "</br>";
    echo "Array Element1: $bikes[0] </br>";
    echo "Array Element2: $bikes[1] </br>";
    echo "Array Element3: $bikes[2] </br>";
?>
```

- **Output:**

```
array(3) { [0]=> string(13) "Royal Enfield" [1]=> string(6) "Yamaha" [2]=> string(3) "KTM" }
Array Element1: Royal Enfield
Array Element2: Yamaha
Array Element3: KTM
```

- **Objects** are the instances of user-defined classes that can store both values and functions. They must be explicitly declared.
- **Example:**

```
<?php
class bike {
      function model() {
         $model_name = "Royal Enfield";
            echo "Bike Model: " .$model_name;
      }
   }
 $obj = new bike();
 $obj -> model();
?>
```

- **Output:**

Bike Model: Royal Enfield

- **Null** is a special data type that has only one value: **NULL**. There is a convention of writing it in capital letters as it is case sensitive.
- The special type of data type NULL defined a variable with no value. The special NULL value is used to represent empty variables in PHP. A variable of type NULL is a variable without any data. NULL is the only possible value of type null.
- **Example:**

```
<?php
$a = NULL;
var_dump($a);
echo "<br>";
$b = "Hello World!";
$b = NULL;
var_dump($b);
?>
```

When a variable is created without a value in PHP like $var; it is automatically assigned a value of null. Many novice PHP developers mistakenly considered both $var1 = NULL; and $var2 = ""; are same, but this is not true. Both variables are different

- A resource is a special variable, holding a reference to an external resource.
- Resource variables typically hold special handlers to opened files and database connections.

```
<?php
// Open a file for reading
  $handle = fopen("note.txt", "r"); var_dump($handle);
   echo "<br>";
   // Connect to MySQL database server with default setting
   $link = mysqli_connect("localhost", "root", "");
   var_dump($link);
?>
```

PHP OPERATORS

- PHP Operator is a symbol i.e used to perform operations on operands. In simple words, operators are used to perform operations on variables or values.

For example:

$num=10+20;

//+ is the operator and 10,20 are operands

- Arithmetic Operators
- Assignment Operators
- Bitwise Operators
- Comparison Operators
- Incrementing/Decrementing Operators
- Logical Operators
- String Operators
- Array Operators
- Type Operators

ARITHMETIC OPERATORS

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

Operator	Name	Example	Explanation
+	Addition	$a + $b	Sum of operands
-	Subtraction	$a - $b	Difference of operands
*	Multiplication	$a * $b	Product of operands
/	Division	$a / $b	Quotient of operands
%	Modulus	$a % $b	Remainder of operands
**	Exponentiation	$a ** $b	$a raised to the power $b

Assignment Operators

Operator	Name	Example	Explanation
=	Assign	$a = $b	The value of right operand is assigned to the left operand.
+=	Add then Assign	$a += $b	Addition same as $a = $a + $b
-=	Subtract then Assign	$a -= $b	Subtraction same as $a = $a - $b
*=	Multiply then Assign	$a *= $b	Multiplication same as $a = $a * $b
/=	Divide then Assign (quotient)	$a /= $b	Find quotient same as $a = $a / $b
%=	Divide then Assign (remainder)	$a %= $b	Find remainder same as $a = $a % $b

Bitwise Operators

Operator	Name	Example	Explanation
&	And	$a & $b	Bits that are 1 in both $a and $b are set to 1, otherwise 0.
|	Or (Inclusive or)	$a | $b	Bits that are 1 in either $a or $b are set to 1
^	Xor (Exclusive or)	$a ^ $b	Bits that are 1 in either $a or $b are set to 0.
~	Not	~$a	Bits that are 1 set to 0 and bits that are 0 are set to 1
<<	Shift left	$a << $b	Left shift the bits of operand $a $b steps
>>	Shift right	$a >> $b	Right shift the bits of $a operand by $b number of places

Comparison Operators

Operator	Name	Example	Explanation
==	Equal	$a == $b	Return TRUE if $a is equal to $b
===	Identical	$a === $b	Return TRUE if $a is equal to $b, and they are of same data type
!==	Not identical	$a !== $b	Return TRUE if $a is not equal to $b, and they are not of same data type
!=	Not equal	$a != $b	Return TRUE if $a is not equal to $b
<>	Not equal	$a <> $b	Return TRUE if $a is not equal to $b
<	Less than	$a < $b	Return TRUE if $a is less than $b

>	Greater than	$a > $b	Return TRUE if $a is greater than $b
<=	Less than or equal to	$a <= $b	Return TRUE if $a is less than or equal $b
>=	Greater than or equal to	$a >= $b	Return TRUE if $a is greater than or equal $b
<=>	Spaceship	$a <=>$b	Return -1 if $a is less than $b Return 0 if $a is equal $b Return 1 if $a is greater than $b

- Incrementing/Decrementing Operators

Operator	Name	Example	Explanation
++	Increment	++$a	Increment the value of $a by one, then return $a
		$a++	Return $a, then increment the value of $a by one
--	decrement	--$a	Decrement the value of $a by one, then return $a
		$a--	Return $a, then decrement the value of $a by one

- Logical Operators

Operator	Name	Example	Explanation
and	And	$a and $b	Return TRUE if both $a and $b are true
Or	Or	$a or $b	Return TRUE if either $a or $b is true
xor	Xor	$a xor $b	Return TRUE if either $ or $b is true but not both
!	Not	! $a	Return TRUE if $a is not true
&&	And	$a && $b	Return TRUE if either $a and $b are true
||	Or	$a || $b	Return TRUE if either $a or $b is true

- String Operators

Operator	Name	Example	Explanation
.	Concatenation	$a . $b	Concatenate both $a and $b
.=	Concatenation and Assignment	$a .= $b	First concatenate $a and $b, then assign the concatenated string to $a, e.g. $a = $a . $b

- Array Operators

Operator	Name	Example	Explanation
+	Union	$a + $y	Union of $a and $b
==	Equality	$a == $b	Return TRUE if $a and $b have same key/value pair
!=	Inequality	$a != $b	Return TRUE if $a is not equal to $b
===	Identity	$a === $b	Return TRUE if $a and $b have same key/value pair of same type in same order
!==	Non-Identity	$a !== $b	Return TRUE if $a is not identical to $b
<>	Inequality	$a <> $b	Return TRUE if $a is not equal to $b

- **Type Operators**
- The type operator **instanceof** is used to determine whether an object, its parent and its derived class are the same type or not. Basically, this operator determines which certain class the object belongs to. It is used in object-oriented programming.

```
<?php
 //class declaration
   class Developer
   {}
   class Programmer
   {}
   //creating an object of type Developer
   $charu = new Developer();
  //testing the type of object
   if( $charu instanceof Developer)
   {
     echo "Charu is a developer.";
   }
   else
   {
       echo "Charu is a programmer.";
   }
   echo "</br>";
  var_dump($charu instanceof Developer);          //It will return true.
  var_dump($charu instanceof Programmer);       //It will return false.
?>
```

CONDITIONAL STRUCTURE

- A control/conditional structure is a block of code that decides the execution path of a program depending on the value of the set condition.
- IF
- IF Else
- Nested IF
- While
- Do while
- For loop
- For each
- switch

IF Else - simplest control structure. It evaluates the conditions using Boolean logic
When to use if… then… else
You have a block of code that should be executed only if a certain condition is true

- You have two options, and you have to select one.
- **Syntax** :<?php

```
 if (condition is true)
   {
       block one else block two
    }
?>
```

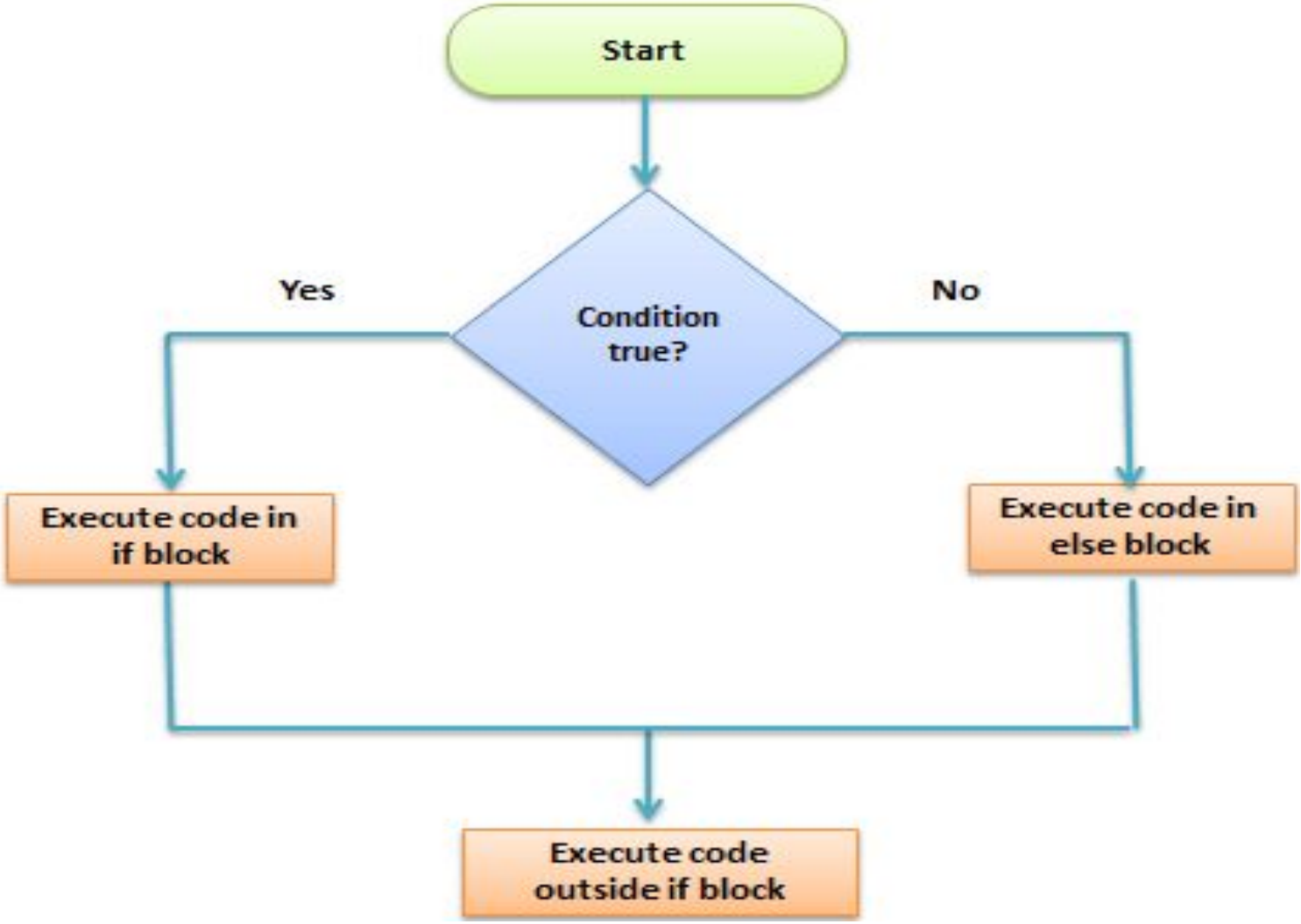
- **HERE,**

"if (condition is true)" is the control structure

"block one" is the code to be executed if the condition is true

{…else…} is the fallback if the condition is false

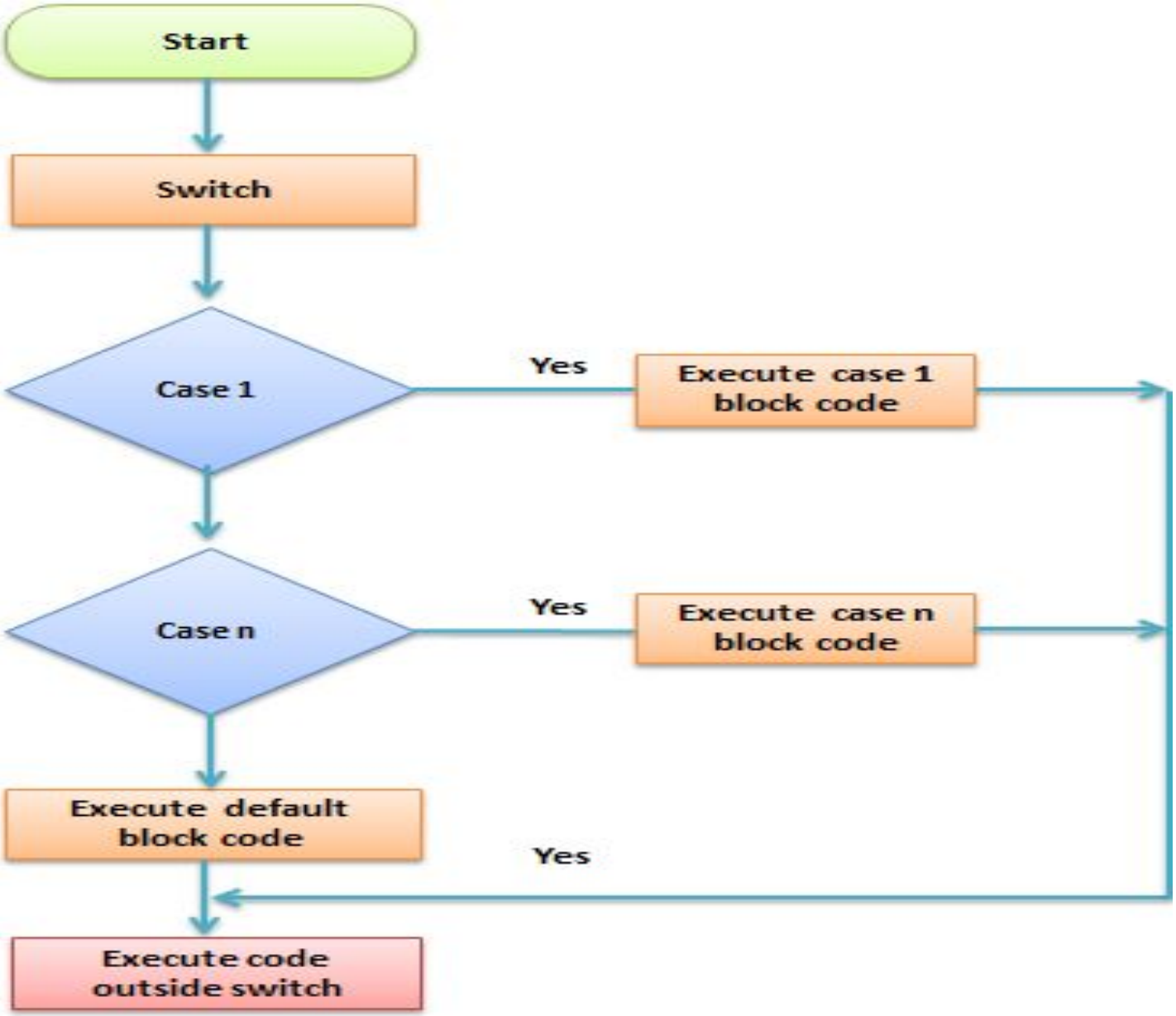
FLOW CHART OF IF ELSE



- Example

```
<?php
  $first_number = 7;
  $second_number = 21;
  if ($first_number > $second_number)
    {
   echo "$first_number is greater than
   $second_number";
    }
   else {
      echo "$second_number is greater than
   $first_number";
    }
?>
```

- **Switch Case**
- **Switch… case** is similar to the **if then… else** control structure. It only **executes** a single block of code depending on the **value** of the condition.
- If no condition has been met then the default block of code is executed.
- **HERE,**
- **“switch(…){…}”** is the control structure block code
- **“case value: case…”** are the blocks of code to be executed depending on the value of the condition
- **“default:”** is the block of code to be executed when no value matches with the condition


Start
Switch
Case 1
Yes
Execute case 1 block code
Case n
Yes
Execute case n block code
Execute default block code
Yes
Execute code outside switch

```php
<?php
	$favourite_site = 'Code';
switch ($favourite_site) {
  case 'Business':
    echo "My favourite site is business.tutsplus.com!";
    break;
  case 'Code':
    echo "My favourite site is code.tutsplus.com!";
    break;
  case 'Web Design':
    echo "My favourite site is webdesign.tutsplus.com!";
    break;
  case 'Music':
    echo "My favourite site is music.tutsplus.com!";
    break;
  case 'Photography':
    echo "My favourite site is photography.tutsplus.com!";
    break;
  default:
    echo "I like everything at tutsplus.com!";
}
?>
```

- **While Loop**
- The while loop is used when you want to execute a piece of code repeatedly until the while condition evaluates to false.
- You can define it as shown in the following pseudo-code.

```
while (expression)
{
  // code to execute as long as expression evaluates to TRUE
```

```php
<?php
$max = 0;
echo $i = 0;
echo ",";
echo $j = 1;
echo ",";
$result=0;

while ($max < 10 )
{
    $result = $i + $j;

    $i = $j;
    $j = $result;

    $max = $max + 1;
    echo $result;
    echo ",";
}
?>
```

- **Do-While Loop in PHP**
- The do-while loop is very similar to the while loop, with the only difference being that the while condition is checked at the end of the first iteration. Thus, we can guarantee that the loop code is executed at least once, irrespective of the result of the while expression.

```
do
{
 // code to execute
} while (expression);
```

```php
<?php
$handle = fopen("file.txt", "r");
if ($handle)
{
    do
    {
        $line = fgets($handle);

        // process the line content

   } while($line !== false);
}
fclose($handle);
?>
```

For Loop

- Generally, the for loop is used to execute a piece of code a specific number of times. In other words, if you already know the number of times you want to execute a block of code, it's the for loop which is the best choice.

```
for (expr1; expr2; expr3)
{
  // code to execute
}
```

The expr1 expression is used to initialize variables, and it's always executed. The expr2 expression is also executed at the beginning of a loop, and if it evaluates to true, the loop code is executed. After execution of the loop code, the expr3 is executed. Generally, the expr3 is used to alter the value of a variable which is used in the expr2 expression.

```
<?php
for ($i=1; $i<=10; ++$i)
{
  echo sprintf("The square of %d is %d.</br>", $i, $i*$i);
}
?>
```

For Each

- The for each loop is used to iterate over array variables. If you have an array variable, and you want to go through each element of that array, the foreach loop is the best choice.

```
<?php
$fruits = array('apple', 'banana', 'orange', 'grapes');
foreach ($fruits as $fruit)
{
  echo $fruit;
  echo "<br/>";
}
```

```
    $employee = array('name' => 'John Smith', 'age' => 30, 'profession' =>
    'Software Engineer');
foreach ($employee as $key => $value)
{
  echo sprintf("%s: %s</br>", $key, $value);
  echo "<br/>";
}
?>
```

If you want to access array values, you can use the first version of the foreach loop, as shown in the above example. On the other hand, if you want to access both a key and a value, you can do it as shown in the $employee example above.

Breaking Out of the Loop

- There are times when you might want to break out of a loop before it runs its course. This can be achieved easily using the break keyword. It will get you out of the current for, foreach, while, do-while, or switch structure.
- You can also use break to get out of multiple nested loops by supplying a numeric argument. For example, using break 3 will break you out of 3 nested loops. However, you cannot pass a variable as the numeric argument if you are using a PHP version greater than or equal to 5.4.

```
echo 'Simple Break';
for($i = 1; $i <= 2; $i++) {
    echo "\n".'$i = '.$i.' ';
    for($j = 1; $j <= 5; $j++) {
        if($j == 2) {
            break;
        }
        echo '$j = '.$j.' ';
    }
}
```

```
echo 'Multi-level Break';
for($i = 1; $i <= 2; $i++) {
    echo "\n".'$i = '.$i.' ';
    for($j = 1; $j <= 5; $j++) {
        if($j == 2) {
            break 2;
        }
        echo '$j = '.$j.' ';
    }
}
```

- Another keyword that can interrupt loops in PHP is continue. However, this only skips the rest of the current loop iteration instead of breaking out of the loop altogether. Just like break, you can also use a numerical value with continue to specify how many nested loops it should skip for the current iteration.

```
<?php
 echo 'Simple Continue';
for($i = 1; $i <= 2; $i++) {
    echo "\n".'$i = '.$i.' ';
    for($j = 1; $j <= 5; $j++) {
        if($j == 2) {
            continue;
        }
  echo '$j = '.$j.' ';
    }
}
```

```
echo 'Multi-level Continue';
for($i = 1; $i <= 2; $i++) {
    echo "\n".'$i = '.$i.' ';
    for($j = 1; $j <= 5; $j++) {
        if($j == 2) {
          continue 2;
        }
  echo '$j = '.$j.' ';
    }
}
?>
```

THANK YOU!!!!!!!