X. I. E.
Mahim, Mumbai

EXPERIMENT - I

Page No. :

Date :

1 WHAT IS PHP?

— PHP [HYPERTEXT PROCESSOR] IS A SERVER SIDE SCRIPTING LANGUAGE USED FOR WEB DEVELOP. IT IS USED FOR GENERATING DYNAMIC CONTENT ON A WEB PAGE. IT IS AN OPEN SOURCE AND HAS A LARGE COMMUNITY

FEATURES OF PHP

1. OPEN SOURCE
2. CROSS PLATFORM
3. SUPPORT FOR DATABASES
4. EXTENSIONS

ADVANTAGES OF PHP

1. RELATIVELY EASY TO LEARN
2. COMPATIBLE WITH MULTIPLE OPERATIG SYSTEM
3. COST EFFECTIVE FOR WEB PROJECTS
4. VAST COMMUNITY SUPPORT.
5. ALLOWS DEVELOPMENT OF DYNAMIC WEB APPS

WHAT IS MYSQL

— MYSQL IS A RELATIONAL DATABASE MANAGMENT SYSTEM OFTEN USED WITH (PHP) TO CREATE DYNAMIC DATA DRIVEN WEB APPS.

ADVANTAGES OF MYSQL

1. OPEN SOURCE TECHNOLOGY
2. EASE OF INTEGRATION
3. CAN HANDLE LARGE AMOUNTS OF DATA
4. BUILT-IN SECURITY FEATURES
5. EFFICIENT QUERY HANDLING

II VARIABLE SCOPES

a] LOCAL SCOPE

— A VARIABLE DECARED INSIDE A FUNCTION IS ONLY ACCESSIBLE WITHIN THAT FUNCTION

Eg.

```
function EXPIC()
{
    $x = "Local is good";
    echo $x;
}

EXPIC();
```

b] GLOBAL SCOPE

— A VARIABLE DECLARED OUTSIDE ANY FUNCTION IS CONSIDERED GLOBAL AND CAN BE ACESSED ANYWHERE WITHIN THE SCRIPT

Eg.
```
$y = "Global is me";

function EXPIC()
{
    global $y;
    echo $y;
}

EXPIC();
echo $y;
```

c] STATIC SCOPE
— A VARIABLE IN A FUNCTION RETAINS ITS VALUE BETWEEN CALLS AND LOCAL TO THE FUNCTION

Eg.
```
function EXPIC()
{
    static $2 = 0;
    echo $2;
    $2++;
}

EXPIC();    // 0
EXPIC();    // 1
EXPIC();    // 2
EXPIC();    // 3
```

d] GLOBALS ARRAY

— THESE ARE PRE DEFINED VARIABLES ARE ALWAYS ACCESSIBLE REGARDLESS OF SCOPE.

Eg.
```
$xy2 = 10;

function EXPIC)
{
   echo $GLOBALS["xy2"]; // 10
}

EXPIC);
 echo $xy2 ; // 10
```

III [ECHO AND PRINT]

a] echo

— CAN TAKE MULTIPLE PARAMETERS AND DOES NOT HAVE A RETURN VALUE

Eg.
```
$a = "Hello World";
echo $a;

echo "Hello", "World";
```

b] print

— CAN TAKE ONLY ONE PARAMETER AND RETURNS 1

Eg. $b = "Hello World";
print $b; // Hello World

print "Hello World"; // Hello World

IV OPERATORS IN PHP

a) ARITHMETIC OPERATORS

① (+) - ADDS TWO OR MORE VARIABLES
Eg. $res = $a + $b;

② (-) - SUBTRACTS RIGHT FROM LEFT VARIABLE
Eg. $res = $a - $b;

③ (*) - MULTIPLIES TWO OPERANDS
Eg. $res = $a * $b;

④ (/) - DIVIDES LEFT BY RIGH OPERAND
Eg. $res = $a / $b;

⑤ (%) - RETURNS REMAINDER OF DIVISION
Eg. $res = $a % $b;

⑥ (++) - INCREASES VALUE BY 1
Eg. $a++;

⑦ (--) - DECREASES VALUE BY 1
Eg. $a--;

b] ASSIGNMENT OPERATOR

① (=) - ASSIGNS VALUE ON RIGHT

Eg. $a = 10;

② (+=) - ADDS AND ASSIGNS

Eg. $a += 5;

③ (-=) - SUBTRACTS AND ASSIGNS

Eg. $a -= 5;

④ (*=) - MULTIPLIES AND ASSIGNS

Eg. $a *= 5;

⑤ (/=) - DIVIDES AND ASSIGNS

Eg. $a /= 5;

c] COMPARISON OPERATOR

① (==) - CHECK FOR EQUALITY

Eg. ($a == $b)

② (===) - CHECK TYPE AND VALUE

Eg. ($a === $b)

③ (!=) - CHECK IF NOT EQUAL

Eg. ($a != $b)

④ (!==) - CHECK TYPE AND NOT EQUAL

Eg. ($a !== $b)

⑤ (>) - CHECK IF GREATER

Eg. ($a > $b)

⑥ (<) - CHECK IF LESS

Eg. ($a < $b)

⑦ (>=) - CHECK IF GREATER OR EQUAL

Eg. ($a >= $b)

⑧ (<=) - CHECK IF LESS OR EQUAL

Eg. ($a <= $b)

d] LOGICAL OPERATOR

① (&&) - and - RETURNS TRUE IF ALL TRUE

Eg. ($a == $b && $a == $c)

② (||) - or - RETURNS TRUE IF EITHER TRUE

Eg. ($a == $b || $a == $c)

③ (!) - not - RETURNS TRUE IF FALSE

Eg. (!$ a)

PROCESS	PRODUCT	TOTAL	SIGN
09	15	24	8

X. I. E.
Mahim, Mumbai

EXPERIMENT - 2

Page No. :

Date :

[DECISION MAKING]

1] "if" STATEMENT

— EXECUTES A BLOCK OF CODE IF CONDITION IS TRUE

SYN
```
if (condition)
{
    // body
}
```

Eg.
```
$age = 19;

if ($age >18)
{
  echo "BIG HUMAN"
}
```

2] "if - else" STATEMENT

— EXECUTES ONE BLOCK IF TRUE ELSE ANOTHER

SYN
```
if (condition)
{
    // body
} else {
    // body
}
```

Eg.
```
$age = 19;

if ($age <=19)
{
    echo "Boy";
} else {
    echo "BABY";
}
```

3] "if - else if - selse" STATEMENT

S — USED TO TEST MULIPLE CONDITIONS.

SYN
```
if (con1)
{
    // body
} elseif (con2) {
    // body
} else{
    //body
}
```

Eg.
```
$s = 85;

if ($s >=90) {
    echo "A";
} else if ($s <= 80) {
    echo "B";
} else {
    echo "C";
}
```

4] "switch" STATEMENT

— USED TO PERFORM DIFF. ACTIONS BASED ON DIFF. CONDITIONS.

SYN

```
switch (x) {
    case 1:
        // body
        break;
    case 2:
        // body
        break;
    .....
    // more cases
    default:
        // body
}
```

Eg.

```
$day = "Monday";
switch ($day)
{
    case "Monday":
        echo $day;
        break;
```

```
        case "FRIDAY":
            echo $day;
            break;

        case "Saturday":
            echo $day;
            break;

        default:
            echo "Invalid";
    }
```

PROCESS	PRODUCT	TOTAL	SIGN
09	15	24	

X. I. E.
Mahim, Mumbai

EXPERIMENT - 3

[LOOPS]

Page No. :

Date :

a] "while" LOOP

— REPEATS CODE TILL TRUE ELSE DOSENT

SYN

```
while (con) {
        // body
}
```

Eg.

```
$c = 0;
while ($c c=10) {
    echo $c ;
    $c++;
}
```

b] "do-while" LOOP

— REPEATS CODE ATLEAST ONCE EVEN IF FALSE

SYN

```
$i = 1;
do {
    // body
} while (con);
```

Eg.

```
$i=1;
do {
    echo $i;
    $i++;
} while ($i <=10);
```

c] "for" - LOOP

— ADVACED LOOP

SYN

```
for ( initialization ; conditions ; iterations)
{
    // body
}
```

Eg.

```
for ($i= 0; $i <=10; $i++)
{
    echo $i;
}
```

d] "foreach" LOOP

— USED TO ITERATE OVER ARRAYS.

SYN

```
foreach ( $var_name as $var_name)
{
    // body
}
```

Eg.

```
$colors = array ("Red", "White", Pink");
for ( $colors as $c)
{
    echo $c;
}
```

c) "break"
— USED EXIT LOOP REGARDLESS CONDITION

SYN while (con)
{
if (con)
{ break;
}
// body
}

Eg. while ($i == 0)
{
if ($i == 0)
{
break;
}
}

f) "continue"

— USED TO SKIP CODE AND GO TO NEXT ITERATION

SYN

```
while (con) {      // body
    if (con)
        continue;
}
```

Eg.

```
while ($i == 1) {
    if ($i == 2) {
        continue
    }
    echo $i;
}
```

	PROCESS	PRODUCT	TOTAL	SIGN
	09	15	24.	8.

[EXPERIMENT - 4]

[ARRAYS IN PHP]

— ARRAYS IN (PHP) ARE VERSATILE DATA STRUCTURES USED TO STORE MULTIPLE VALUES UNDER A SINGLE NAME.

— THEY CAN HOLD VARIOUS TYPES OF DATA SUCH AS INTEGERS, STRINGS, OBJECTS AND EVEN OTHER ARRAYS

[TYPES OF ARRAYS]

① INDEXED ARRAYS —

→ INDEXED ARRAYS ARE WHERE EACH ELEMENT IS ASSIGNED A NUMERIC INDEX STARTING FROM 0.

SYNTAX

```
$my_array = array (
                "val_one",
                "val_two",
                ......
                "val-N"
            );
```

[OR]

```
$my_array = [ v1, v2 ..... vN];
```

(2) ASSOCIATIVE ARRAYS ——

→ ASSOCIATIVE ARRAYS ARE WHERE EACH ELEMENT IS ASSOCIATED WITH A SPECIFIC KEY, WHICH CAN BE A STRING OR AN INTEGER. KEYS ARE USED TO ACESS VALUES INSTEAD OF NUMERICAL INDEXES.

SYNTAX

```
$my_array = array (
            "key_a" => "val_one",
            "key_b" => "val_two",
            ..................
            "key_N" => "val_N",
            );
```

[OR]

```
$my_away = [
           "key_a" => "val_one",
           "key_b" => "val_two",
           ..................
           "key_N" => "val_N",
           ];
```

(3) MULTI-DIMENSIONAL ARRAYS ——

→ MULTI-DIMENSIONAL ARRAYS ARE ARRAYS CONTAINING ONE OR MORE ARRAYS AS ELEMENTS THESE ARRAYS ARE FORMED IN A GRID LIKE STRUCTURE.

SYNTAX

```
$my_array = array (
            array ("val_1a", "val_1b", ...., "val_1N"),
            array ("val_2a", "val_2b", ...., "val_2N"),
            array ("val_3a", "val_3b", ...., "val_3N"),
            );
```

[$my_array[-][-];]

[OR]

```
$my_array = array (
            "A" => array ( 34, 85 ..... ),
            "B" => array ( 71, 23 ..... ),
            "C" => array ( 95, 15 ..... ),
            );
```

[$my_array["A"][-];]

9] implode()

→ USED TO JOIN ARRAY ELEMENTS WITH A STRING. IT TAKES AN ARRAY AND A SEPERATOR. IT RETURNS A STRING CONTAINING ALL ARRAY ELEMENTS JOINED BY THE SEPERATOR.

SYNTAX

```
implode (seperator, $arr_name);
```

b] explode()

→ IT IS THE OPPOSITE OF "implode()". IT SPLITS A STRING INTO AN ARRAY OF SUBSTRINGS BASED ON A DELIMETER. IT TAKE A DELIMETER AND THE STRING TO SPLIT.

SYNTAX

```
explode ("," $ str);
$str = "app, mobile, home";
```

c] array_flip()

→ EXCHANGES ALL KEYS WITH THEIR ASSOCIATED VALUES IN AN ARRAY.

SYNTAX

```
$og_ar = ("a" => 1,
          "b" => 2
         );
$flip_ar = $array_flip ($og_ar);
```

ARRAY TRAVERSING

a) VIA "FOR LOOP"

→
```
$my_arr = array (1,2,3,4);

for ($i=0 ; $i <count($my_arr); $i++)
{
    echo $my_arr[$i] . "<br>";
}
```

b) VIA "FOREACH LOOP"

→
```
$my_arr = array (5, 13, 18, 21, 29);

foreach ( $my_arr as $val)
{
    echo $val . "<br>";
}
```

[or]

```
$my_arr = array("a" => 1, "b" => 2, "c" => 3);

foreach ( $my_arr as $key => $val)
{
    echo $key . ":" . $val . "<br>";
}
```

c] array_map()

→ USES A CALLBACK FUNCTION TO EACH ELEMENT OF AN ARRAY AND RETURNS A NEW ARRAY WHICH THE MODIFIED ELEMENTS.

Eg

```
function sqr ($n)
{
    return $n * $n;
}

$ar = [1,2,3,4,5];

$sqr_ar = array_map('sqr', $ar);
```

d] array_walk()

→ USES A CALLBACK FUNCTION TO EACH ELEMENT OF AN ARRAY. IT DOES NOT CREATE A NEW ARRAY.

Eg

```
function sqr ($n) {
    $ans = $n * $n;
}
$ar = [2, 5, 8];
array
```

Eg.

```
function sqr ( $n)
{
    $val = $n * $n;
}

$ar = array ( 2, 5, 8, 11);
array_walk('sqr', $ar);
```

	PRODUCT	PROCESS	TOTAL	SIGN
	09	15	24.	

[EXPERIMENT - 5]

Page No.
Date

- IN (PHP) THERE ARE NUMEROUS STRING RELATED FUNCTIONS TO CHANGE AND MANIPULATE STRINGS EFFECTIVELY.

a) strlen()

→ RETURNS THE LENGTH OF A GIVEN STRING

SYNTAX

- int strlen (string $string)

Eg
```
$ str = "Hello World";
echo strlen ($str); // 13
```

b) strtolower()

→ CONVERTS A STRING TO LOWERCASE

SYNTAX

- string strtolower (string $string)

Eg.
```
$str = "Hello Mr. White";
echo strtolower ($str); // hello mr. white
```

c] strtoupper()

→ CONVERTS A STRING TO UPPERCASE

SYNTAX

— string strtoupper (string $string)

Eg. $str = "Hello"
echo strtoupper ($str); // HELLO

d] substr()

→ RETURNS A PART OF A STRING

SYNTAX

— string substr (string $string ,
int $start ,
[int $length])

$string — main string
$start — replace / return from index
$length — OPTIONAL, how long from $start

Eg.