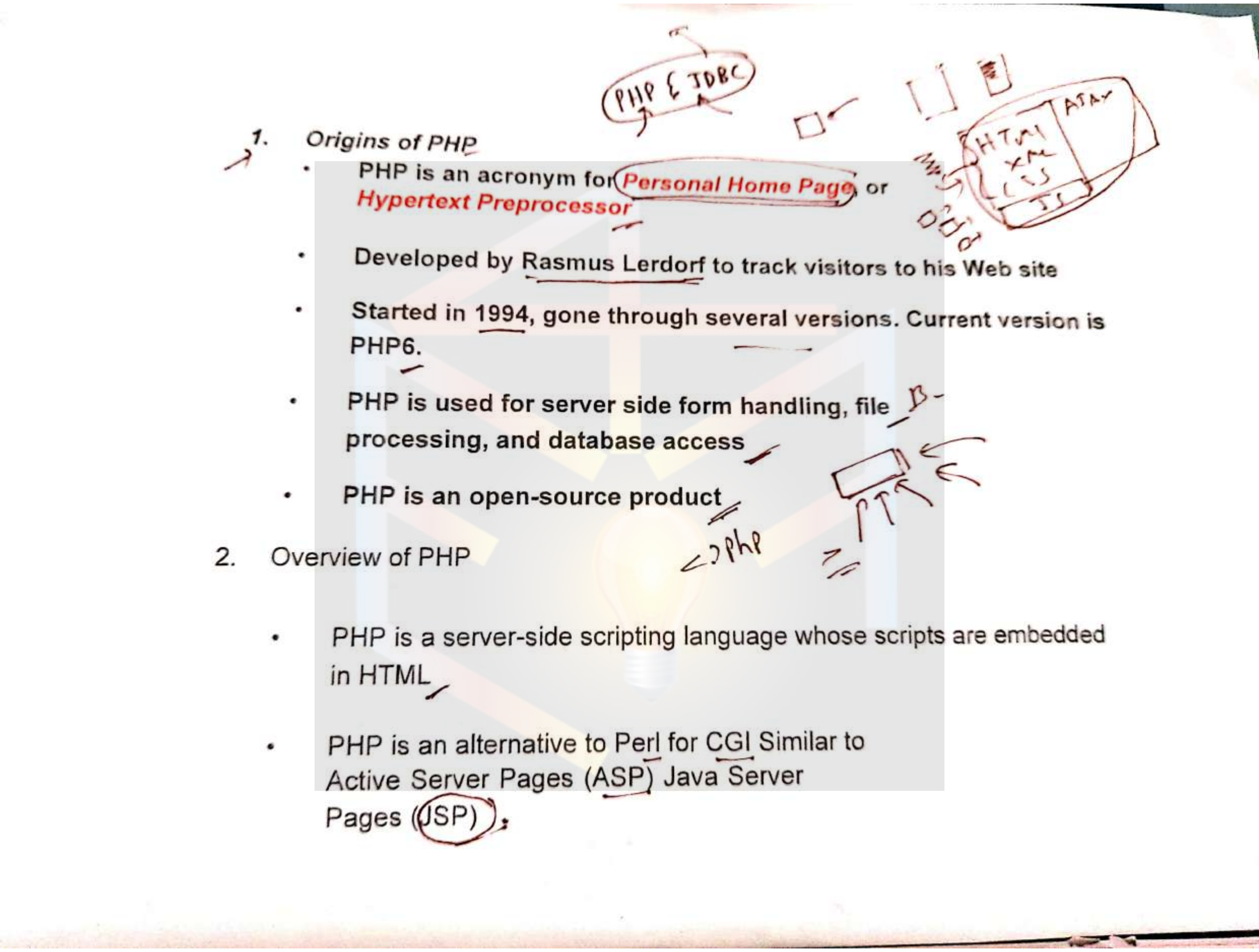


PHP & JDBC

1. *Origins of PHP*
   - PHP is an acronym for *Personal Home Page* or *Hypertext Preprocessor*
   - Developed by Rasmus Lerdorf to track visitors to his Web site
   - Started in 1994, gone through several versions. Current version is PHP6.
   - PHP is used for server side form handling, file processing, and database access
   - PHP is an open-source product

2. Overview of PHP

   <?php

   - PHP is a server-side scripting language whose scripts are embedded in HTML
   - PHP is an alternative to Perl for CGI Similar to Active Server Pages (ASP) Java Server Pages (JSP).

- As programming language, PHP has a lot of similarity to that of Perl
  e.g. a variable starts with $, and dynamic type int-8a
  $

- The PHP processor has two modes: copy HTML and interpret PHP ↔ indxh.php

1. Browser makes a request to a web server for a PHP document

2. Web server, call the PHP processor and inputs the PHP document to the PHP processor. A traditional CGI approach.

3. When PHP processor processes the PHP document, it copies the html portion to an output HTML/XML document, and interprets the PHP script and executes the scripts and then insert the results to the HTML/XML document, finally output the HTML/XML document

4. When it is done, the HTML document is handed to the Web server and it is then sent to the browser

- PHP script can be executed by call: php

  file_name.php

Run PHP as Apache module

PHP can be installed on Apache as a module. i.e. the loaded PHP interpreter with Apache

Such PHP interpreter runs in part of Apache process, and PHP interpreter functions can be called directly

Gain a bit performance, but less secured than the traditional CGI approach.

- **The example of Hello World in HTML**

```
<html>
 <head><title> Hello World </title>
 </head>
 <body>
<?PHP
 print("Hello World!"); //or print "Hello World!"; ?>
</body>
</html>
```

?php

print(" (—) ");

echo

? >

2

3. General Syntactic Characteristics

- PHP code can be specified in an HTML document internally or externally: Internally:

```
<?php
   //... PHP code here
?>
```

Externally:

```
<?php include("myScript.php");
   //or require("myScript.php");
?>
```

The differences error handling.

The file can have both PHP and HTML. If the file has PHP, the PHP must be in <?php .. ?>, even if the include is already in <?php .. ?>

<?php (..) ?>

- A document contain HTML and PHP script should have extension .PHP, PHP3, or .phtml

See example

- *Comments* - three different kinds (Java and Perl) <!- == -?

```
//  ...
#  ...
/*  ...  */
```

// # /* */ // #

- PHP statements are terminated with semicolons
- Compound statements are formed with braces (;)
- Variables are case sensitive.
- There are 31 reserved words.

- Output

- Output from a PHP script is HTML that is sent to the browser

- HTML is sent to the browser through standard output

There are three ways to produce output:

echo can have one or more parameters print similar to echo, but only allow on parameter printf is similar to that of C language

echo and print take a string, but will coerce other values to strings echo "whatever";

echo("first <br />", $sum)

print "Welcome to my site!";

4. Primitives, Operations and Expressions

- *Variables*
  - No type declarations, i.e., dynamic type
  - An unassigned variable has the value, NULL
  - The *unset* function sets a variable to NULL
  - The *IsSet* function is used to determine whether a variable is NULL

•PHP has predefined variables, including the environment variables. The list of the predefined variables can be obtained by calling *phpinfo()*

- Eight primitive types of variables
  - Four scalar types

    Boolean, integer, double, and string
  - Two compound types

    array and object
  - Two special types

    resource and `NULL`
- Boolean

  values are true and false

  0 and "" and "0" are false;
  others are true

- Integer and double

Arithmetic Operators and Expressions

– If the result of integer division is not an integer, a double is returned

– Any integer operation that results in overflow produces a double

– The modulus operator coerces its operands to integer, if necessary

– When a double is rounded to an integer, the rounding is always towards zero

- ***Arithmetic functions***

`floor, ceil, round, abs, min, max, rand`, etc.

1.1 1 $\frac{1.1}{2}$ 1.1 ↓ 1   -1.1 +1.1

- Strings
  - Characters are single bytes
  - String literals use single or double quotes similar to that of Perl
    - *Single-quoted string literals*
      Embedded variables are not interpolated Embedded escape sequences are not recognized
    - Embedded variables in a double-quoted string are interpolated. If there is a variable name but you don't want it interpolated, use backslash
    - Embedded escape sequences are recognized or both single- and double-quoted literal strings, embedded delimiters must be backslashed
  - *String Operations and Functions*
    - The only operator is period, for catenation
    - Indexing - $str{3} is the fourth character

{ }

0, 1, 2, 3

String functions:

`strlen, strcmp, strpos, substr, strstr,`

`substr_replace, str_replace, explode`

`chop` – remove whitespace from the right end

`trim` – remove whitespace from both ends

`ltrim` – remove whitespace from the left end

`strtolower, strtoupper`

- *Scalar Type Conversions*
  - *String to numeric*

  - If the string contains an e or an E, it is converted to double; otherwise to int
  - If the string does not begin with a sign or a digit, zero is used

- Explicit conversions – casts

- e.g., `(int)$total` or `intval($total)` or

  `settype($total, "integer")`

- The type of a variable can be determined with `gettype` or `is_type`

  `gettype($total)` - it may return `"unknown"`

  `is_integer($total)` – a predicate function

5. Control Statements

- Control Expressions

  - Relational operators , similar to Perl. Special operators (including `===`, `identical`, and `!==`, `no identical`)

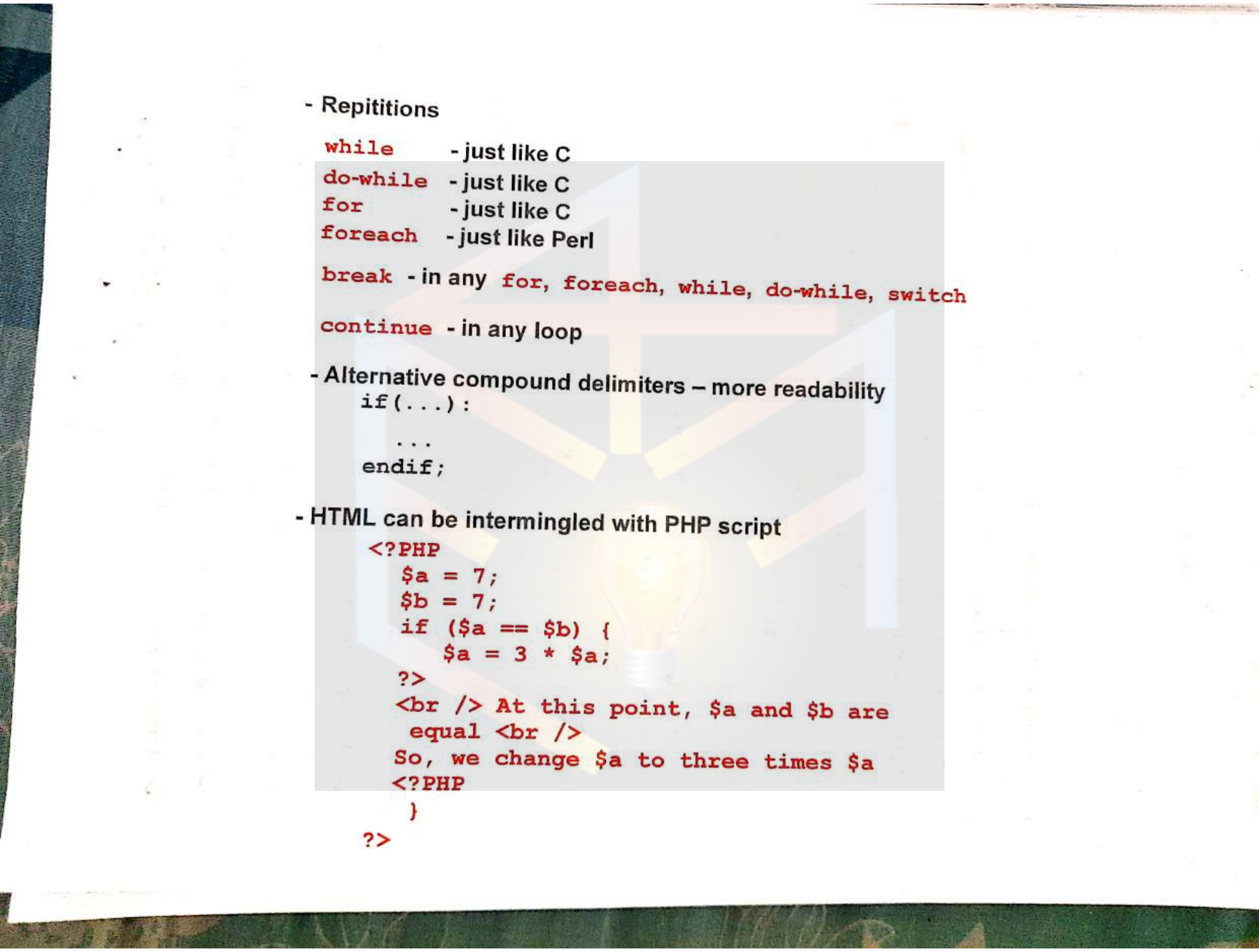
  - Boolean operators - same as Perl (two sets, `&&` and `and`, etc.)

- Selection statements

  `if, if-else, elseif`

  `switch` - as in C

  - The switch expression type must be integer, double, or string



- Repititions

```
while       - just like C
do-while    - just like C
for         - just like C
foreach     - just like Perl
```

`break` - in any `for, foreach, while, do-while, switch`

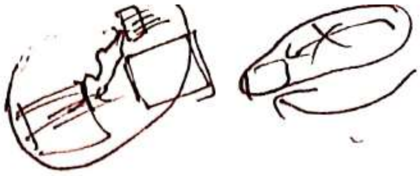
`continue` - in any loop

- Alternative compound delimiters – more readability

```
if(...):
   ...
endif;
```

- HTML can be intermingled with PHP script

```
<?PHP
   $a = 7;
   $b = 7;
   if ($a == $b) {
      $a = 3 * $a;
   ?>
   <br /> At this point, $a and $b are
    equal <br />
   So, we change $a to three times $a
   <?PHP
    }
?>
```



3

PHP - INTRODUCTION

https://www.tutorialspoint.com/php/php_introduction.htm Copyright © tutorialspoint.com

PHP started out as a small open source project that evolved as more and more people found out how useful it was.
Rasmus Lerdorf unleashed the first version of PHP way back in 1994.

PHP is a recursive acronym for "PHP: Hypertext Preprocessor".

PHP is a server side scripting language that is embedded in HTML. It is used to manage dynamic content, databases, session tracking, even build entire e-commerce sites.

It is integrated with a number of popular databases, including MySQL, PostgreSQL, Oracle, Sybase, Informix, and Microsoft SQL Server.

PHP is pleasingly zippy in its execution, especially when compiled as an Apache module on the Unix side. The MySQL server, once started, executes even very complex queries with huge result sets in record-setting time.

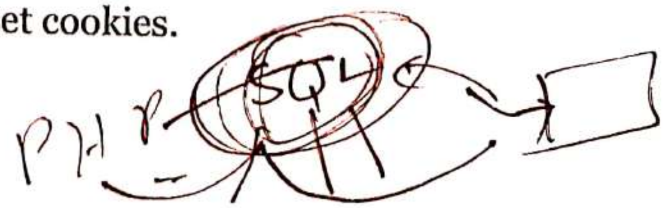
PHP supports a large number of major protocols such as POP3, IMAP, and LDAP. PHP4 added support for Java and distributed object architectures COMandCORBA, making n-tier development a possibility for the first time.

PHP is forgiving: PHP language tries to be as forgiving as possible.

PHP Syntax is C-Like.

Common uses of PHP

- ✓ PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- ✓ PHP can handle forms, i.e. gather data from files, save data to a file, through email you can send data, return data to the user.
- ✓ You add, delete, modify elements within your database through PHP.
- ✓ Access cookies variables and set cookies.



Using PHP, you can restrict users to access some pages of your website.

It can encrypt data.

Characteristics of PHP

Five important characteristics make PHP's practical nature possible −

- Simplicity
- Efficiency
- Security
- Flexibility
- Familiarity

① → PHP

⑤

"Hello World" Script in PHP

To get a feel for PHP, first start with simple PHP scripts. Since "Hello, World!" is an essential example, first we will create a friendly little "Hello, World!" script.

As mentioned earlier, PHP is embedded in HTML. That means that in amongst your normal HTML orXHTMLifyou recutting − edge you'll have PHP statements like this −

Live Demo

```
<html>

  <head>
    <title>Hello World</title>
  </head>

  <body>
    <?php echo "Hello, World!";?>
  </body>

</html>
```

<?php echo " " ;?>

print(" ");

It will produce following result −

```
Hello, World!
```

If you examine the HTML output of the above example, you'll notice that the PHP code is not present in the file sent from the server to your Web browser. All of the PHP present in the Web page is processed and stripped from the page; the only thing returned to the client from the Web server is pure HTML output.

All PHP code must be included inside one of the three special markup tags ATE are recognised by the PHP Parser.

```
<?php PHP code goes here ?>

<?  PHP code goes here ?>

<script language = "php"> PHP code goes here </script>
```

A most common tag is the <?php...?> and we will also use the same tag in our tutorial.

From the next chapter we will start with PHP Environment Setup on your machine and then we will dig out almost all concepts related to PHP to make you comfortable with the PHP language.

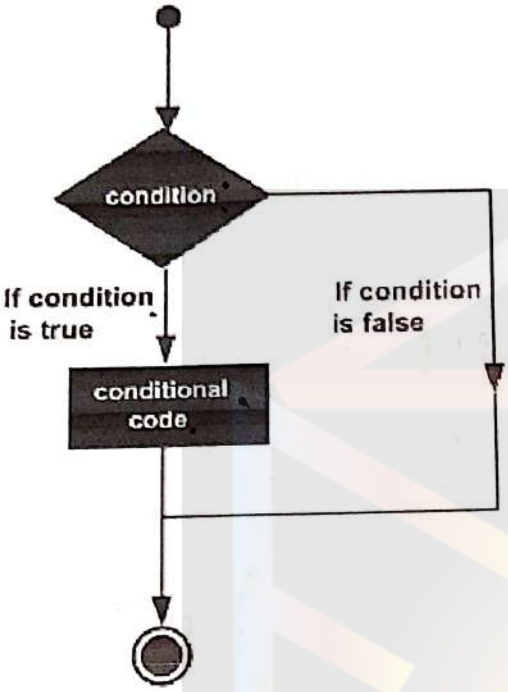
PHP - DECISION MAKING

(4)

Advertisements if if

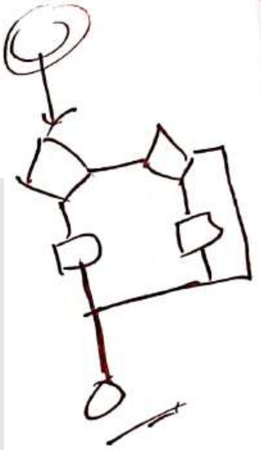
The if, elseif ...else and switch statements are used to take decision based on the different condition.

You can use conditional statements in your code to make your decisions. PHP supports following three decision making statements –



if
else

if
else if
else



if...else statement – use this statement if you want to execute a set of code when a condition is true and another if the condition is not true

elseif statement – is used with the if...else statement to execute a set of code if one of the several condition is true

switch statement – is used if you want to select one of many blocks of code to be executed, use the Switch statement. The switch statement is used to avoid long blocks of if..elseif..else code.

The If...Else Statement

If you want to execute some code if a condition is true and another code if a condition is false, use the if....else
statement.

Syntax

```
if (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, Otherwise, it will output "Have a nice day!":

Live Demo

```
<html>
  <body>

    <?php
      $d = date("D");
      if ($d == "Fri")
        echo "Have a nice weekend!";
      else
        echo "Have a nice day!";
    ?>

  </body>
</html>
```

"D" <?php

$d

$d

It will produce the following result –

```
Have a nice weekend!
```

The ElseIf Statement

If you want to execute some code if one of the several conditions are true use the elseif statement

Syntax

```
if (condition)
  code to be executed if condition is true;
elseif (condition)
  code to be executed if condition is true;
else
  code to be executed if condition is false;
```

Example

The following example will output "Have a nice weekend!" if the current day is Friday, and "Have a nice Sunday!" if the current day is Sunday. Otherwise, it will output "Have a nice day!" –

Live Demo

```
<html>
  <body>

    <?php
      $d = date("D");

      if ($d == "Fri")
        echo "Have a nice weekend!";

      elseif ($d == "Sun")
        echo "Have a nice Sunday!";

      else
        echo "Have a nice day!";
    ?>

  </body>
</html>
```

It will produce the following result –

```
Have a nice Weekend!
```

The Switch Statement

If you want to select one of many blocks of code to be executed, use the Switch statement.

The switch statement is used to avoid long blocks of if..elseif..else code.

Syntax

```
switch (expression){
  case label1:
    code to be executed if expression = label1;
    break;

  case label2:
    code to be executed if expression = label2;
    break;
    default:

 code to be executed
 if expression is different
 from both label1 and label2;
}
```

Example

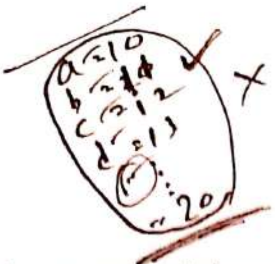
The *switch* statement works in an unusual way. First it evaluates given expression then seeks a lable to match the resulting value. If a matching value is found then the code associated with the matching label will be executed or if none of the lable matches then statement will execute any specified default code.

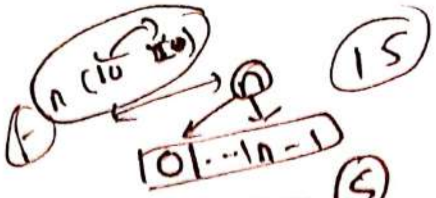
Live Demo

```
<html>
  <body>

    <?php
      $d = date("D");

      switch ($d){
        case "Mon":
          echo "Today is Monday";
          break;

        case "Tue":
          echo "Today is Tuesday";
          break;

        case "Wed":
          echo "Today is Wednesday";
          break;

        case "Thu":
          echo "Today is Thursday";
          break;

        case "Fri":
          echo "Today is Friday";
          break;

        case "Sat":
          echo "Today is Saturday";
          break;

        case "Sun":
          echo "Today is Sunday";
          break;

        default:
          echo "Wonder which day is this ?";
      }
    ?>

  </body>
</html>
```

It will produce the following result –

```
Today is Monday
```





PHP - ARRAYS

https://www.tutorialspoint.com/php/php_arrays.htm

An array is a data structure that stores one or more similar type of values in a single value. For example if you want to store 100 numbers then instead of defining 100 variables its easy to define an array of 100 length.

There are three different kind of arrays and each array value is accessed using an ID c which is called array index.

Numeric array – An array with a numeric index. Values are stored and accessed in linear fashion.

Associative array – An array with strings as index. This stores element values in association with key values rather than in a strict linear index order.

Multidimensional array – An array containing one or more arrays and values are accessed using multiple indices

NOTE – Built-in array functions is given in function reference PHP Array Functions

Numeric Array

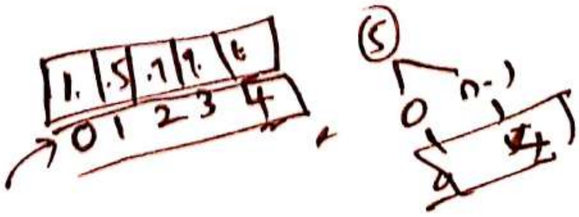
These arrays can store numbers, strings and any object but their index will be represented by numbers. By default array index starts from zero.

Example

Following is the example showing how to create and access numeric arrays.

Here we have used array function to create array. This function is explained in function reference.

Live Demo



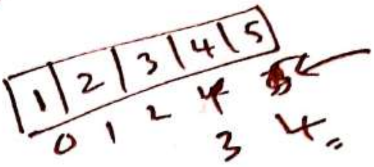
```
<html>
   <body>
```

$

```
<?php
  /* First method to create array. */
  $numbers = array( 1, 2, 3, 4, 5);

  foreach( $numbers as $value ) {
    echo "Value is $value <br />";
  }

  /* Second method to create array. */
  $numbers[0] = "one";
  $numbers[1] = "two";
  $numbers[2] = "three";
  $numbers[3] = "four";
  $numbers[4] = "five";

  foreach( $numbers as $value ) {
    echo "Value is $value <br />";
  }

?>

 </body>
</html>
```

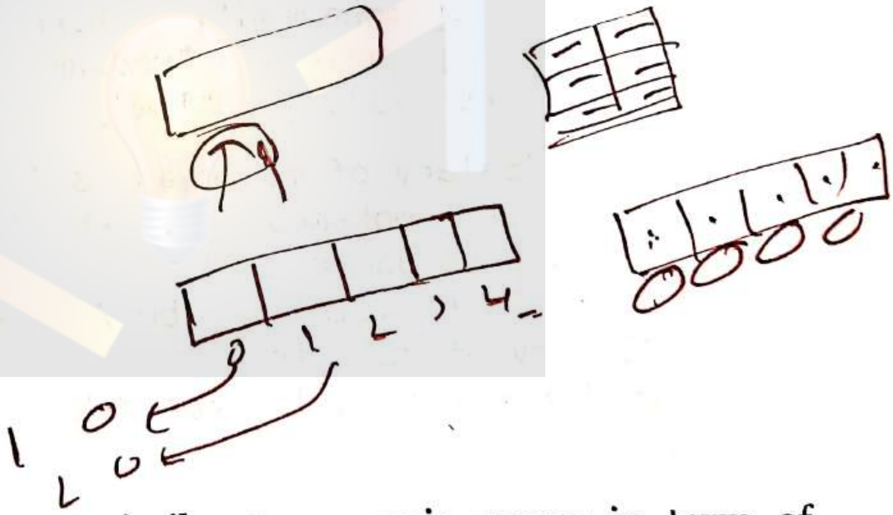
<?php

?>



This will produce the following result –

```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
Value is one
Value is two
Value is three
Value is four
Value is five
```



Associative Arrays

The associative arrays are very similar to numeric arrays in term of functionality but they are different in terms of their index. Associative array will have their index as string so that you can establish a strong association between key and values.

To store the salaries of employees in an array, a numerically indexed array would not be the best choice. Instead, we could use the employees names as the keys in our associative array, and the value would be their respective salary.

NOTE – Don't keep associative array inside double quote while printing otherwise it would not return any value.

Example

Live Demo

<?php—?>

```
<html>
  <body>

    <?php
      /* First method to associate create array. */
      $salaries = array("mohammad" => 2000, "qadir" => 1000,
      "zara" => 500);

      echo "Salary of mohammad is ".
      $salaries['mohammad'] . "<br />"; echo
      "Salary of qadir is ".
      $salaries['qadir']. "<br />"; echo
      "Salary of zara is ".
      $salaries['zara']. "<br />";

      /* Second method to create array. */
      $salaries['mohammad'] = "high";
      $salaries['qadir'] = "medium";
      $salaries['zara'] = "low";

      echo "Salary of mohammad is ".
      $salaries['mohammad'] . "<br />"; echo
      "Salary of qadir is ".
      $salaries['qadir']. "<br />"; echo
      "Salary of zara is ".
      $salaries['zara']. "<br />";
   ?>
```

'O'=> —

key

m. qu zar

```
  </body>
</html>
```

This will produce the following result –

```
Salary of mohammad is 2000
Salary of qadir is 1000
Salary of zara is 500
Salary of mohammad is high
Salary of qadir is medium
Salary of zara is low
```

x[ ][ ] x[ ][ ][ ]

Multidimensional Arrays

A multi-dimensional array each element in the main array can also be an array. And each element in the sub-array can be an array, and so on. Values in the multi-dimensional array are accessed using multiple index.

Example

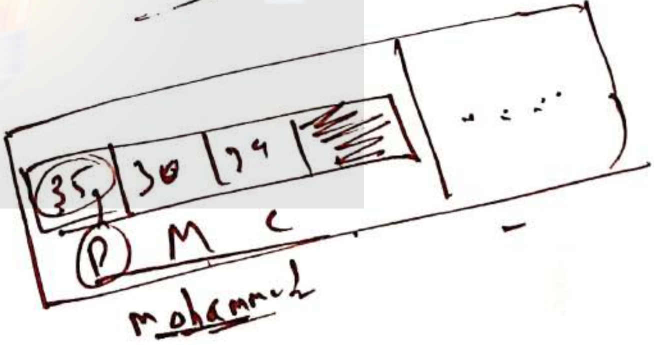
In this example we create a two dimensional array to store marks

of three students in three subjects – This example is an associative

array, you can create numeric array in the same fashion.

Live Demo

```
<html>
  <body>

    <?php
      $marks = array(
        "mohammad" => array (
          "physics" => 35,
          "maths" => 30,
          "chemistry" => 39
        ),

        "qadir" => array (
          "physics" => 30,
          "maths" => 32,
```



```
            "chemistry" => 29
         ),

         "zara" => array (
            "physics" => 31,
            "maths" => 22,
            "chemistry" => 39
         )
      );

      /* Accessing multi-dimensional array values */
      echo "Marks for mohammad in physics : " ;
      echo $marks['mohammad']['physics'] . "<br />";

      echo "Marks for qadir in maths : ";
      echo $marks['qadir']['maths'] . "<br />";

      echo "Marks for zara in chemistry : " ;
      echo $marks['zara']['chemistry'] . "<br />";
   ?>

   </body>
</html>
```

This will produce the following result –

```
Marks for mohammad in physics : 35
Marks for qadir in maths : 32
Marks for zara in chemistry : 39
```

1 100

PHP - LOOP TYPES

6

https://www.tutorialspoint.com/php/php_loop_types.htm Copyright © tutorialspoint.com

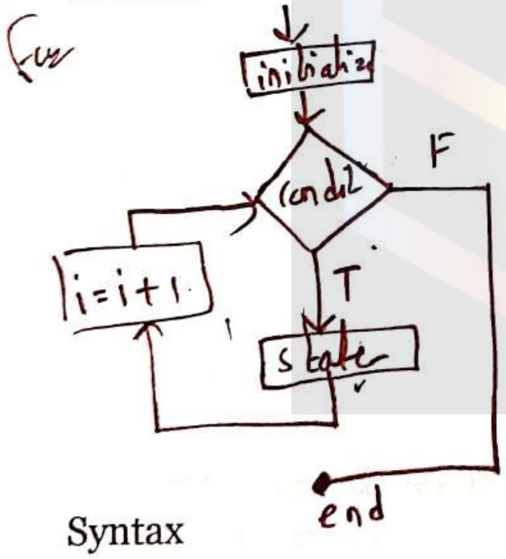
Loops in PHP are used to execute the same block of code a specified number of times. PHP supports following four loop types.

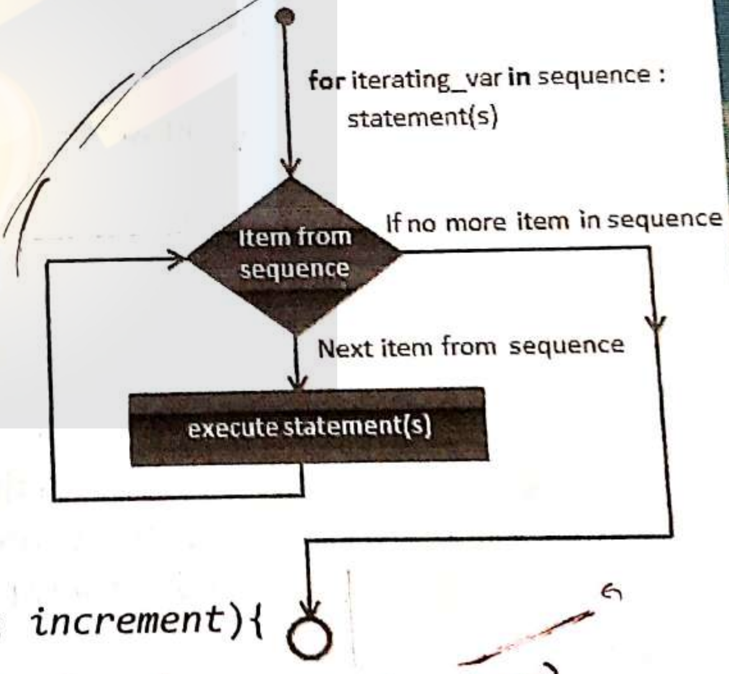
- for – loops through a block of code a specified number of times.
- while – loops through a block of code if and as long as a specified condition is true.
- do...while – loops through a block of code once, and then repeats the loop as long as a special condition is true.
- foreach – loops through a block of code for each element in an array.

We will discuss about continue and break keywords used to control the loops execution.

The for loop statement

The for statement is used when you know how many times you want to execute a statement or a block of statements.





Syntax

```
for (initialization; condition; increment){
  code to be executed;
}
```

for (init ;condi ;in. i++)
i=0 i<n i=i+1
{
}

The initializer is used to set the start value for the counter of the number of loop iterations. A variable may be declared here for this purpose and it is traditional to name it $i.

Example

The following example makes five iterations and changes the assigned value of two variables on each pass of the loop –

Live Demo

```
<html>
  <body>

    <?php
      $a = 0;
      $b = 0;

      for( $i = 0; $i<5; $i++ ) {
        $a += 10;
        $b += 5;
      }

      echo ("At the end of the loop a = $a and b = $b" );
   ?>

  </body>
</html>
```

50 40 30 20 10 | 50 15 10 5

0 | 0

a | b

i = 0 1 4

0 < 5 (T)

5 < 5

a = a + 10

b = b + 5

a + = 10

a = a + 10

0 1 2 3 4 5

This will produce the following result –

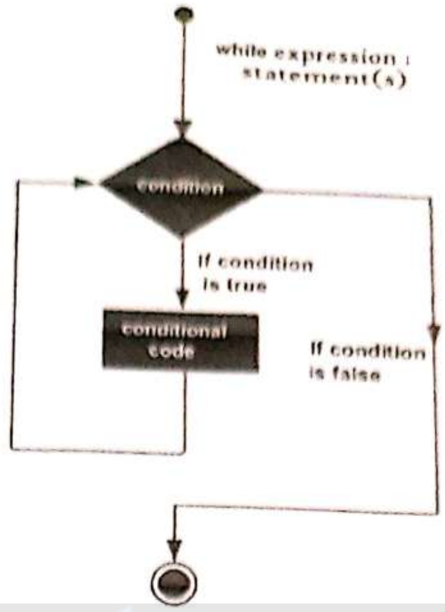
```
At the end of the loop a = 50 and b = 25
```

The while loop statement

The while statement will execute a block of code if and as long as a test expression is true.

If the test expression is true then the code block will be exeouted. After the code has executed the test expression will again be evaluated and the loop will continue until the test expression is found to be false.

while (—)
{
—
}



Syntax

```
while (condition) {
  code to be executed;
}
```

Example

This example decrements a variable value on each iteration of the loop and the counter increments until it reaches 10 when the evaluation is false and the loop ends.

Live Demo

```
<html>
  <body>

    <?php
      $i = 0;
      $num = 50;

      while( $i < 10) {
        $num--;
        $i++;
      }

      echo ("Loop stopped at i = $i and num = $num" );
    ?>
```

i = 0 1 … 10

num = ~~50~~ 49

0 … 10

10

50
-10
40

0 10 1|10

9|10

10|40

```
  </body>
</html>
```

This will produce the following result –

```
Loop stopped at i = 10 and num = 40
```

The do...while loop statement

The do...while statement will execute a block of code at least once - it then will repeat the loop as long as a condition is true.

Syntax

```
do {
  code to be executed;
}
while (condition);
```

Example

The following example will increment the value of i at least once, and it will continue incrementing the variable i as long as it has a value of less than 10 –

Live Demo

```
<html>
  <body>

    <?php
      $i = 0;
      $num = 0;

      do {
        $i++;
      }

      while( $i < 10 );
      echo ("Loop stopped at i = $i" );
    ?>
```

do
{
}
wk

```
  </body>
</html>
```

This will produce the following result –

```
Loop stopped at i = 10
```

The foreach loop statement

The foreach statement is used to loop through arrays. For each pass the value of the current array element is assigned to $value and the array pointer is moved by one and in the next pass next element will be processed.

Syntax

```
 foreach (array as
      value) {
code to be executed;
 }
```

Example

Try out following example to list out the values of an array.

Live Demo

```
<html>
  <body>

    <?php
      $array = array( 1, 2, 3, 4, 5);

      foreach( $array as $value ) {
        echo "Value is $value <br />";
      }
    ?>

  </body>
</html>
```

This will produce the following result –

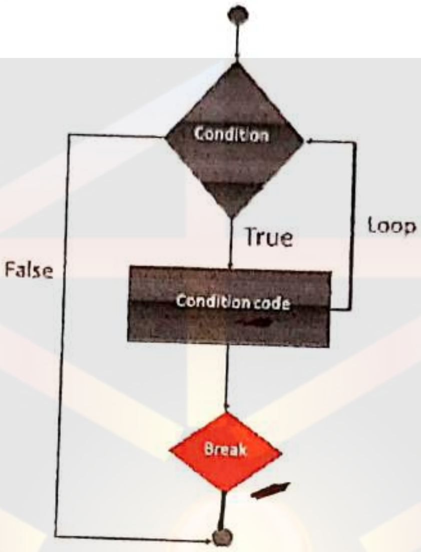
```
Value is 1
Value is 2
Value is 3
Value is 4
Value is 5
```

B T L

The break statement

The PHP break keyword is used to terminate the execution of a loop prematurely.

The break statement is situated inside the statement block. It gives you full control and whenever you want to exit from the loop you can come out. After coming out of a loop immediate statement to the loop will be executed.



Example

In the following example condition test becomes true when the counter value reaches 3 and loop terminates.

Live Demo

```
<html>
  <body>

    <?php
      $i = 0;

      while( $i < 10) {
        $i++;
        if( $i == 3 )break;
      }
      echo ("Loop stopped at i = $i" );
      ?>

   </body>
</html>
```

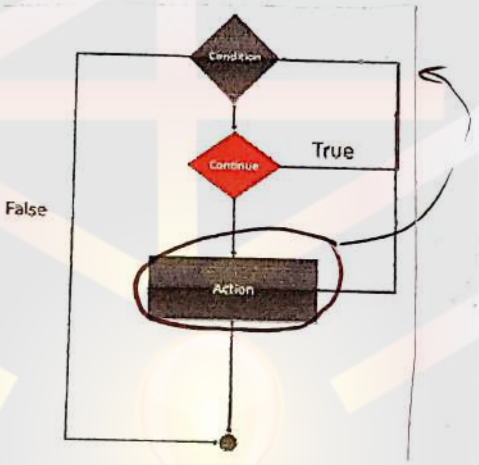
This will produce the following result –

```
Loop stopped at i = 3
```

The continue statement

The PHP continue keyword is used to halt the current iteration of a loop but it does not terminate the loop.

Just like the break statement the continue statement is situated inside the statement block containing the code that the loop executes, preceded by a conditional test. For the pass encountering continue statement, rest of the loop code is skipped and next pass starts.



1
2
~~3~~
4
5

Example

In the following example loop prints the value of array but for which condition becomes true it just skip the code and next value is printed.

Live Demo

```
<html>
   <body>

      <?php
         $array = array( 1, 2, 3, 4, 5);

         foreach( $array as $value ) {
            if( $value == 3 )continue;
            echo "Value is $value <br />";
         }
      ?>

   </body>
 </html>
```

This will produce the following result –

```
Value is 1
Value is 2
Value is 4
Value is 5
```

PHP - FILES & I/O 7

https://www.tutorialspoint.com/php/php_files.htm Copyright © tutorialspoint.com

This chapter will explain following functions related to files –

Opening a file W

R

Reading a file W

Writing a file B.

Closing a file

Opening and Closing Files

The PHP fopen function is used to open a file. It requires two arguments stating first the file name and then mode in which to operate.

Files modes can be specified as one of the six options in this table.

Sr.No Mode & Purpose

1

r

Opens the file for reading only.

Places the file pointer at the beginning of the file.

2

r+

Opens the file for reading and writing.

Places the file pointer at the beginning of the file.

3

w

Opens the file for writing only.

Places the file pointer at the beginning of the file.

and truncates the file to zero length. If files does not exist then it attempts to create a file.

4

w+

Opens the file for reading and writing only.

Places the file pointer at the beginning of the file.

and truncates the file to zero length. If files does not exist then it attempts to create a file.

5

a

0

Opens the file for writing only.

Places the file pointer at the end of the file.

If files does not exist then it attempts to create a file.

a+

Opens the file for reading and writing only.

Places the file pointer at the end of the file.

If files does not exist then it attempts to create a file.

If an attempt to open a file fails then fopen returns a value of false otherwise it returns a file pointer which is used for further reading or writing to that file.

After making a changes to the opened file it is important to close it with the fclose function. The fclose function requires a file pointer as its argument and then returns true when the closure succeeds or false if it fails.

Reading a file

Once a file is opened using fopen function it can be read with a function called fread. This function requires two arguments. These must be the file pointer and the length of the file expressed in bytes.

The files length can be found using the filesize function which takes the file name as its argument and returns the size of the file expressed in bytes.

So here are the steps required to read a file with PHP.

Open a file using fopen function. ✓

Get the file's length using filesize function.

Read the file's content using fread function.

Close the file with fclose function.

The following example assigns the content of a text file to a variable then displays those contents on the web page.

```
<html>✓
  <head> ✓
```

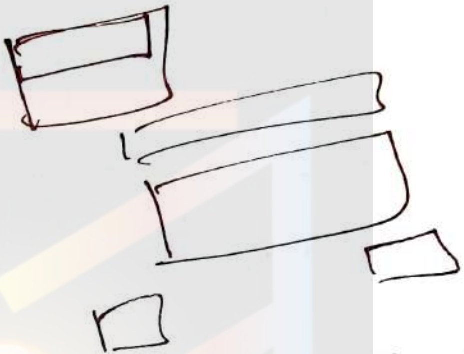
```
      <title>Reading a file using PHP</title>
   </head>

   <body>

      <?php
         $filename = "tmp.txt";
         $file = fopen( $filename, "r" );

         if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
         }

         $filesize = filesize( $filename );
         $filetext = fread( $file, $filesize );
         fclose( $file );

         echo ( "File size : $filesize bytes" );
         echo ( "<pre>$filetext</pre>" );
      ?>

   </body>
</html>
```

O

R

C

It will produce the following result –



Writing a file

A new file can be written or text can be appended to an existing file using the PHP fwrite function. This function requires two arguments specifying a file pointer and the string of data that is to be written. Optionally a third integer argument can be included to specify the length of the data to write. If the third argument is included, writing would will stop after the specified length has been reached.

The following example creates a new text file then writes a short text heading inside it. After closing this file its existence is confirmed using file_exist function which takes file name as an argument

```
<?php
   $filename = "/home/user/guest/newfile.txt";
   $file = fopen( $filename, "w" );

   if( $file == false ) {
      echo ( "Error in opening new file" );
      exit();
   }
   fwrite( $file, "This is  a simple test\n" );
   fclose( $file );
?>
<html>

   <head>
      <title>Writing a file using PHP</title>
   </head>

   <body>

      <?php
         $filename = "newfile.txt";
         $file = fopen( $filename, "r" );

         if( $file == false ) {
            echo ( "Error in opening file" );
            exit();
         }

         $filesize = filesize( $filename );
         $filetext = fread( $file, $filesize );

         fclose( $file );

         echo ( "File size : $filesize bytes" );
         echo ( "$filetext" );
         echo("file name: $filename");
      ?>

   </body>
</html>
```

It will produce the following result –

We have covered all the function related to file input and out in PHP File System Function chapter.

PHP - FORM INTRODUCTION 8

https://www.tutorialspoint.com/php/php_form_introduction.htm Copyright ©

tutorialspoint.com

Advertisements

Dynamic Websites

The Websites provide the functionalities that can use to store, update, retrieve, and delete the data in a database.

What is the Form?

A Document that containing black fields, that the user can fill the data or user can select the data.Casually the data will store in the data base

Example

Below example shows the form with some specific actions by using post method.

```
<html> ✓

  <head> ✓
    <title>PHP Form Validation</title> ✓
  </head> ✓

  <body> ✓          <?php —?>
    <?php

      // define variables and set to empty values
      $name = $email = $gender = $comment = $website = "";

      if ($_SERVER["REQUEST_METHOD"] == "POST") {
        $name = test_input($_POST["name"]);
        $email = test_input($_POST["email"]);
        $website = test_input($_POST["website"]);
        $comment = test_input($_POST["comment"]);
        $gender = test_input($_POST["gender"]);
```

_POST[" "]

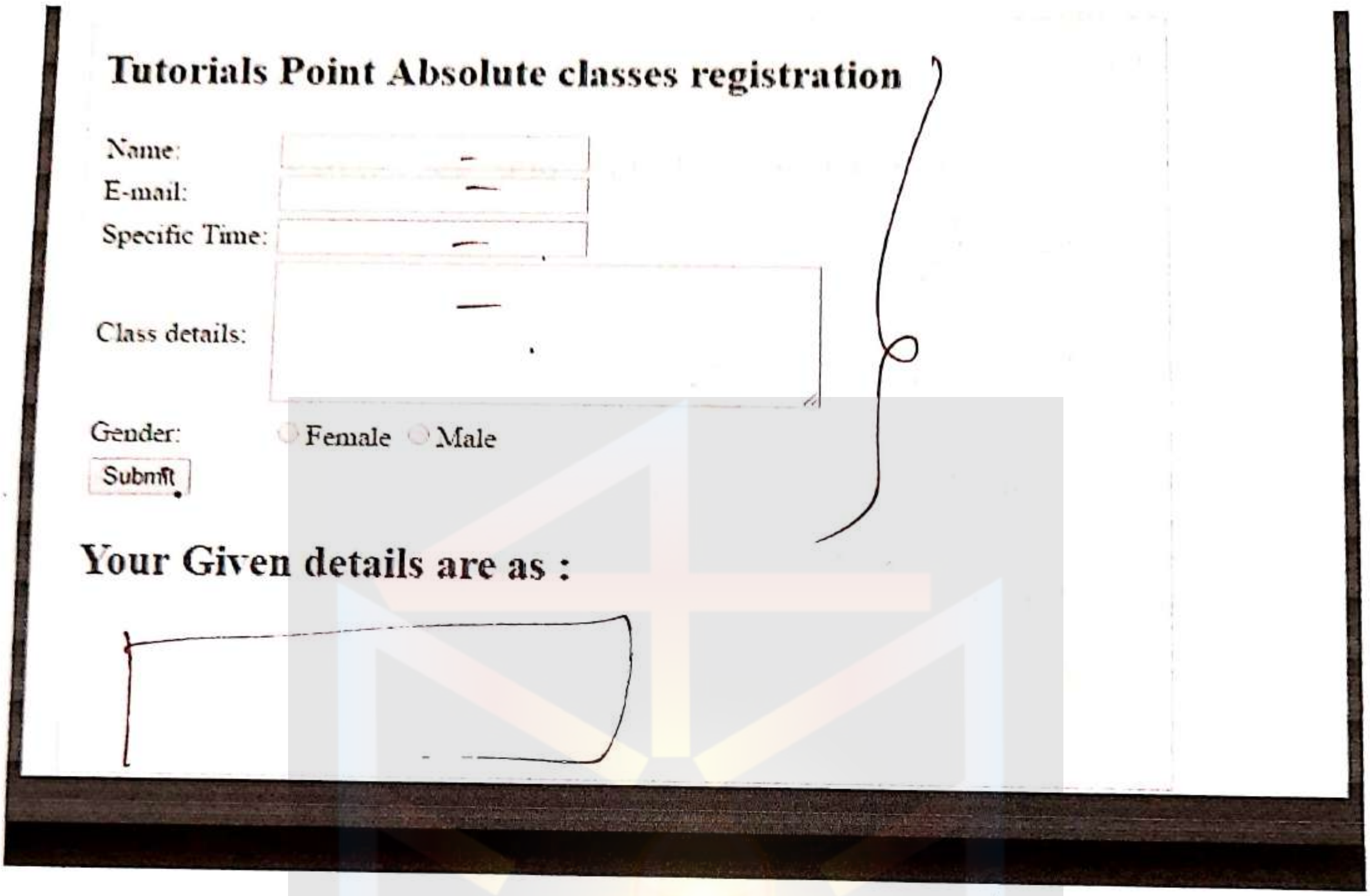
(JDBC)

```
   }
   function test_input($data) {
     $data = trim($data);
     $data = stripslashes($data);
     $data = htmlspecialchars($data);
     return $data;
   }
?>

<h2>Tutorials Point Absolute classes registration</h2>

<form method = "post" action = "">
  <table>
    <tr>
      <td>Name:</td>
      <td><input type = "text" name = "name"></td>
    </tr>

    <tr>
      <td>E-mail:</td>
      <td><input type = "text" name = "email"></td>
    </tr>

    <tr>
      <td>Specific Time:</td>
      <td><input type = "text" name = "website"></td>
    </tr>

    <tr>
      <td>Class details:</td>
      <td><textarea name = "comment" rows = "5"
  cols = "40"></textarea></td> </tr>

    <tr>
      <td>Gender:</td>
      <td>
        <input type = "radio" name = "gender"
        value = "female">Female <input type =
        "radio" name = "gender" value =
        "male">Male
      </td>
    </tr>

    <tr>
```

email @

```
          <td>
            <input type = "submit" name =
          "submit" value = "Submit"> </td>
        </tr>
      </table>
    </form>

    <?php
      echo "<h2>Your Given details are as :</h2>";
      echo $name;
      echo "<br>";

      echo $email;
      echo "<br>";

      echo $website;
      echo "<br>";

      echo $comment;
      echo "<br>";

      echo $gender;
    ?>

  </body>
</html>
```

It will produce the following result –



Tutorials Point Absolute classes registration

Name:

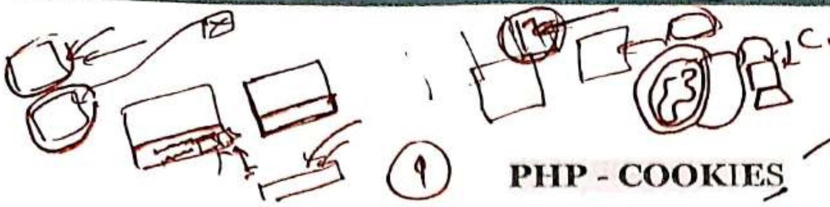
E-mail:

Specific Time:

Class details:

Gender: ○ Female ○ Male

Your Given details are as :



→ Track user browsing activities
→ remembering login details
→ Track site visitors count

9

PHP - COOKIES

https://www.tutorialspoint.com/php/php_cookies.htm Copyright © tutorialspoint.com

Cookies are text files stored on the client computer and they are kept of use tracking purpose. PHP transparently supports HTTP cookies.

There are three steps involved in identifying returning users –

- Server script sends a set of cookies to the browser. For example name, age, or identification number etc.
- Browser stores this information on local machine for future use.
- When next time browser sends any request to web server then it sends those cookies information to the server and server uses that information to identify the user.

This chapter will teach you how to set cookies, how to access them and how to delete them.

The Anatomy of a Cookie

Cookies are usually set in an HTTP header **althoughJavaScriptcanalsosetacookiedirectlyonabrowser**. A PHP script that sets a cookie might send headers that look something like this –

```
HTTP/1.1 200 OK
Date: Fri, 04 Feb 2000 21:03:38 GMT
Server: Apache/1.3.9 (UNIX) PHP/4.0b3
Set-Cookie: name=xyz; expires=Friday,
           04-Feb-07 22:03:38 GMT;
           path=/;
           domain=tutorialspoint.com
Connection: close
Content-Type: text/html
```

1 *

As you can see, the Set-Cookie header contains a name value pair, a GMT date, a path and a domain. The name and value will be URL encoded. The expires field is an instruction to the browser to "forget" the cookie after the given time and date.

If the browser is configured to store cookies, it will then keep this information until the expiry date. If the user points the browser at any page that matches the path and domain of the cookie, it will resend the cookie to the server.The browser's headers might look something like this –

```
GET / HTTP/1.0
Connection: Keep-Alive
User-Agent: Mozilla/4.6 (X11; I; Linux 2.2.6-15apmac ppc)
Host: zink.demon.co.uk:1126
Accept: image/gif, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: name=xyz
```

A PHP script will then have access to the cookie in the environmental variables
c **OOKIEor**HTTP_COOKIE_VARS[] which holds all cookie names and values. Above cookie can be accessed

using $HTTP_COOKIE_VARS["name"].

Setting Cookies with PHP

PHP provided setcookie function to set a cookie. This function requires upto six arguments and should be called before <html> tag. For each cookie this function has to be called separately.

```
setcookie(name, value, expire, path, domain, security);
```

Here is the detail of all the arguments –

Name – This sets the name of the cookie and is stored in an environment variable called
HTTP_COOKIE_VARS. This variable is used while accessing cookies.

Value – This sets the value of the named variable and is the content that you actually want to store.

Expiry – This specify a future time in seconds since 00:00:00 GMT on 1st Jan 1970. After this time cookie will become inaccessible. If this parameter is not set then cookie will automatically expire when the Web Browser is closed.

Path – This specifies the directories for which the cookie is valid. A single forward slash character permits the cookie to be valid for all directories.

Domain – This can be used to specify the domain name in very large domains and must contain at least two periods to be valid. All cookies are only valid for the host and domain which created them.

Security – This can be set to 1 to specify that the cookie should only be sent by secure transmission using HTTPS otherwise set to 0 which mean cookie can be sent by regular HTTP.

Following example will create two cookies name and age these cookies will be expired after one hour.

```
<?php
  setcookie("name", "John Watkin",
  time()+3600, "/","", 0);
  setcookie("age", "36",
  time()+3600, "/", "", 0);
?>
<html>

  <head>
    <title>Setting Cookies with PHP</title>
  </head>

  <body>
    <?php echo "Set Cookies"?>
  </body>

</html>
```

Accessing Cookies with PHP

PHP provides many ways to access cookies. Simplest way is to use either C OOKIEorHTTP_COOKIE_VARS variables. Following example will access all the cookies set in above example.

```
<html>

  <head>
    <title>Accessing Cookies with PHP</title>
  </head>

  <body>
```

```
      <?php
        echo $_COOKIE["name"]. "<br />";
        
        /* is equivalent to */
        echo $HTTP_COOKIE_VARS["name"]. "<br />";
        
        echo $_COOKIE["age"] . "<br />";
        
        /* is equivalent to */
        echo $HTTP_COOKIE_VARS["age"] . "<br />";
      ?>
      
   </body>
</html>
```

You can use isset function to check if a cookie is set or not.

```
<html>
   
   <head>
      <title>Accessing Cookies with PHP</title>
   </head>
   
   <body>
      
      <?php
         if( isset($_COOKIE["name"]))
            echo "Welcome " . $_COOKIE["name"] . "<br />";
         
         else
            echo "Sorry... Not recognized" . "<br />";
      ?>
      
   </body>
</html>
```

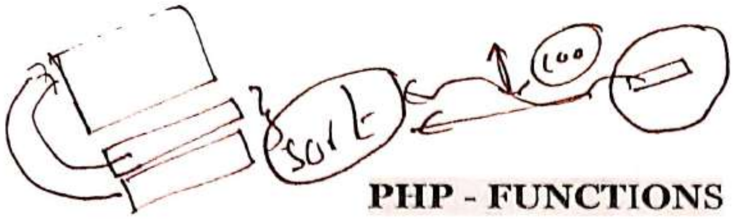
Deleting Cookie with PHP

Officially, to delete a cookie you should call setcookie with the name argument only but this does not always work well, however, and should not be relied on.

It is safest to set the cookie with a date that has already expired –

```
<?php
   setcookie( "name", "", time()- 60, "/","", 0);
```

```
  setcookie( "age", "", time()- 60, "/","", 0);
?>
<html>

  <head>
    <title>Deleting Cookies with PHP</title>
  </head>

  <body>
    <?php echo "Deleted Cookies" ?>
  </body>

</html>
```



PHP - FUNCTIONS

10

https://www.tutorialspoint.com/php/php_functions.htm Copyright © tutorialspoint.com

Advertisements

PHP functions are similar to other programming languages. A function is a piece of code which takes one more input in the form of parameter and does some processing and returns a value.

You already have seen many functions like fopen and fread etc. They are built-in functions but PHP gives you option to create your own functions as well.

There are two parts which should be clear to you –

- Creating a PHP Function ✓
- Calling a PHP Function ✓

2

In fact you hardly need to create your own PHP function because there are already more than 1000 of built-in library functions created for different area and you just need to call them according to your requirement.

Please refer to PHP Function Reference for a complete set of useful functions.

Creating PHP Function

Its very easy to create your own PHP function. Suppose you want to create a PHP function which will simply write a simple message on your browser when you will call it. Following example creates a function called writeMessage and then calls it just after creating it.

Note that while creating a function its name should start with keyword function and all the PHP code should be put inside { and } braces as shown in the following example below –

Live Demo

```
<html> ✓
   <head>✓
```

```
      <title>Writing PHP Function</title>
   </head>

   <body>                    <?php - - - ?>

     <?php
       /* Defining a PHP Function */
       function writeMessage() {
         echo "You are really a nice person, Have a nice
         time!";
       }

       /* Calling a PHP Function */
       writeMessage();
     ?>

   </body>
</html>
```

This will display following result –

```
You are really a nice person, Have a nice time!
```

PHP Functions with Parameters

PHP gives you option to pass your parameters inside a function. You can pass as many as parameters your like. These parameters work like variables inside your function. Following example takes two integer parameters and add them together and then print them.

Live Demo

```
<html>

  <head>
    <title>Writing PHP Function
  with Parameters</title> </head>

  <body>

    <?php
      function addFunction($num1, $num2) {
        $sum = $num1 + $num2;
        echo "Sum of the two numbers is : $sum";
      }

      addFunction(10, 20);
    ?>

  </body>
</html>
```

This will display following result –

```
Sum of the two numbers is : 30
```

Passing Arguments by Reference

It is possible to pass arguments to functions by reference. This means that a reference to the variable is manipulated by the function rather than a copy of the variable's value.

Any changes made to an argument in these cases will change the value of the original variable. You can pass an argument by reference by adding an ampersand to the variable name in either the function call or the function definition.

Following example depicts both the cases.

Live Demo

```
<html>

  <head>
    <title>Passing Argument by Reference</title>
  </head>

  <body>

    <?php

      function addFive($num) {
        $num += 5;
      }

      function addSix(&$num) {
        $num += 6;
      }

      $orignum = 10;
      addFive( $orignum );

      echo "Original Value is $orignum<br />";

      addSix( $orignum );
```

&

```
         echo "Original Value is $orignum<br />";
      ?>

   </body>
</html>
```

This will display following result –

```
Original Value is 10
Original Value is 16
```

PHP Functions returning value

A function can return a value using the return statement in conjunction with a value or object. return stops the execution of the function and sends the value back to the calling code.
You can return more than one value from a function using return array1, 2, 3, 4
.

Following example takes two integer parameters and add them together and then returns their sum to the calling program. Note that return keyword is used to return a value from a function.

Live Demo

```
<html>

   <head>
      <title>Writing PHP Function which
   returns value</title> </head>

   <body>

      <?php
         function addFunction($num1, $num2) {
            $sum = $num1 + $num2;
            return $sum;
         }
         $return_value = addFunction(10, 20);

         echo "Returned value from the function :
         $return_value";
      ?>
```

```
  </body>
</html>
```

This will display following result –

```
Returned value from the function : 30
```

Setting Default Values for Function Parameters

You can set a parameter to have a default value if the function's caller doesn't pass it.

Following function prints NULL in case use does not pass any value to this function.

Live Demo

```
<html>

   <head>
      <title>Writing PHP Function which
   returns value</title> </head>

   <body>

      <?php
         function printMe($param = NULL) {
            print $param;
         }

         printMe("This is test");
         printMe();
      ?>

   </body>
</html>
```

This will produce following result –

```
This is test
```

Dynamic Function Calls

It is possible to assign function names as strings to variables and then treat these variables exactly as you would the function name itself. Following example depicts this behaviour.

Live Demo

```
<html>

   <head>
      <title>Dynamic Function Calls</title>
   </head>

   <body>

      <?php
         function sayHello() {
            echo "Hello<br />";
         }

         $function_holder = "sayHello";
         $function_holder();
      ?>

   </body>
</html>
```

This will display following result –

```
Hello
```

PHP - OPERATOR TYPES

https://www.tutorialspoint.com/php/php_operator_types.htm Copyright © tutorialspoint.com

What is Operator? Simple answer can be given using expression *4 + 5 is equal to 9*. Here 4 and 5 are called operands and + is called operator. PHP language supports following type of operators.

Arithmetic Operators

Comparison Operators

Logical orRelational Operators

Assignment Operators

Conditional orternary Operators

+
-
*
%
/

Lets have a look on all operators one by one.

Arithmetic Operators

There are following arithmetic operators supported by PHP language – Assume variable A holds 10 and variable B holds 20 then –

Show Examples

Operator	Description	Example
+	Adds two operands	A + B will give 30
-	Subtracts second operand from the first	A - B will give -10
*	Multiply both operands	A * B will give 200
/	Divide numerator by de-numerator	B / A will give

/ → Q = % → R

2

%	Modulus Operator and remainder of after an integer division	B % A will give 0
x+1 ++ x = x+1	Increment operator, increases integer value by one	A++ will give 11
--	Decrement operator, decreases integer value by one	A-- will give 9

Comparison Operators

There are following comparison operators (x = = y)

supported by PHP language Assume variable Tr

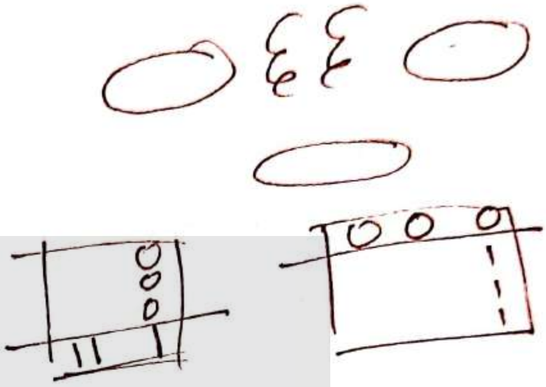
A holds 10 and variable B holds 20 then –

Show Examples

Operator	Description	Example
==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	A == B is not true.
!= !=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	A! = B is true.
> >	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	A > B is not true.
< <	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	A < B is true.

>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	**A >= B** is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	**A <= B** is true.

Logical Operators



There are following logical operators supported by PHP language Assume variable A holds 10 and variable B holds 20 then –

Show Examples

Operator	Description	Example
and	Called Logical AND operator. If both the operands are true then condition becomes true.	**AandB** is true.
or	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	**AorB** is true.
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	A && B is true.
||	Called Logical OR Operator. If any of the two operands are non zero then condition becomes true.	**A||B** is true.

~~x+=v~~ a += b

a = a+b

!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	A && B is false.

x += y ~~x=~~ y+x

Assignment Operators

There are following assignment operators supported by PHP language –

Show Examples

x += y

x = x+y

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assign the result to left	C /= A is equivalent to C = C / A

	operand	
%=	Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand	C %= A is equivalent to C = C % A

Conditional Operator

There is one more operator called conditional operator. This first evaluates an expression for a true or false value and then execute one of the two given statements depending upon the result of the evaluation. The conditional operator has this syntax –

Show Examples

Ter— ⟷ ? ⟷ : ⟷

Operator	Description	Example
? :	Conditional Expression	If Condition is true ? Then value X : Otherwise value Y

Operators Categories

All the operators we have discussed above can be categorised into following categories –

Unary prefix operators, which precede a single operand.

Binary operators, which take two operands and perform a variety of arithmetic and logical operations.

The conditional operator aternaryoperator, which takes three operands and evaluates either the second or third expression, depending on the evaluation of the first expression.

Assignment operators, which assign a value to a variable.

Precedence of PHP Operators

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator –

For example x = 7 + 3 * 2; Here x is assigned 13, not 20 because operator * has higher precedence than + so it first get multiplied with 3*2 and then adds into 7.

Here operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

	Category	Operator	Associativity
·1° ←	Unary	! ++ --	Right to left
8	Multiplicative	* / %	Left to right
7	Additive	+ -	Left to right
	Relational	< <= > >=	Left to right
	Equality	== !=	Left to right
12	Logical AND	&&	Left to right
	Logical OR	||	Left to right
	Conditional	?:	Right to left
	Assignment	= += -= *= /= %=	Right to left

$$\underline{x * y} + \underline{2 * x}$$

Op
A

ALC

12

PHP - REGULAR EXPRESSIONS

https://www.tutorialspoint.com/php/php_regular_expression.htm

01

Regular expressions are nothing more than a sequence or pattern of characters itself. They provide the foundation for pattern-matching functionality.

Using regular expression you can search a particular string inside a another string, you can replace one string by another string and you can split a string into many chunks.

01101100

PHP offers functions specific to two sets of regular expression functions, each corresponding to a certain type of regular expression. You can use any of them based on your comfort.

- POSIX Regular Expressions
- PERL Style Regular Expressions

POSIX Regular Expressions

The structure of a POSIX regular expression is not dissimilar to that of a typical arithmetic expression: various elements operators are combined to form more complex expressions.

The simplest regular expression is one that matches a single character, such as g, inside strings such as g, haggle, or bag.

Lets give explanation for few concepts being used in POSIX regular expression. After that we will introduce you with regular expression related functions.

Brackets

Brackets [] have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Sr.No	Expression & Description

1	[0-9] It matches any decimal digit from 0 through 9.
2	[a-z] It matches any character from lower-case a through lowercase z.
3	[A-Z] It matches any character from uppercase A through uppercase Z.
4	[a-Z] It matches any character from lowercase a through uppercase Z.

The ranges shown above are general; you could also use the range [0-3] to match any decimal digit ranging from 0 through 3, or the range [b-v] to match any lowercase character ranging from b through v.

Quantifiers

The frequency or position of bracketed character sequences and single characters can be denoted by a special character. Each special character having a specific connotation. The +, *, ?, {int. range}, and $ flags all follow a character sequence.

Sr.No	Expression & Description
1	p+ p(+)

It matches any string containing at least one p. x p x

2 p* 0 or more +, *

It matches any string containing zero or more p's.

3 p? ?

It matches any string containing zero or one p's.

4 p{N} ✓

It matches any string containing a sequence of N p's

5 p{2,3} ✓ ✓

It matches any string containing a sequence of two or three p's.

6 p{2, } ✓

It matches any string containing a sequence of at least two p's.

7 p$ x p p

It matches any string with p at the end of it.

8 ^p p x

It matches any string with p at the beginning of it.

Examples

Following examples will clear your concepts about matching characters.

Sr.No Expression & Description

1

[^a-zA-Z]

It matches any string not containing any of the characters ranging from a through z and A through Z.

2

p.p

It matches any string containing p, followed by any character, in turn followed by another p.

3

^.{2}$

It matches any string containing exactly two characters.

4

<b>. *</b>

It matches any string enclosed within <b> and </b>.

5

php*

It matches any string containing a p followed by zero or more instances of the sequence php.

Predefined Character Ranges

For your programming convenience several predefined character ranges, also known as character classes, are available. Character classes specify an entire range of characters, for example, the alphabet or an integer set –

Sr.No Expression & Description

1

[[:alpha:]]

It matches any string containing alphabetic characters aA through zZ.

2

[[:digit:]]

It matches any string containing numerical digits 0 through 9.

3

[[:alnum:]]

It matches any string containing alphanumeric characters aA through zZ and 0 through 9.

4

[[:space:]]

It matches any string containing a space.

PHP's Regexp POSIX Functions

PHP currently offers seven functions for searching strings using POSIX-style regular expressions –

Sr.No Function & Description

1 ereg

The ereg function searches a string specified by string for a string specified by pattern, returning true if the pattern is found, and false otherwise.

2 ereg_replace

The ereg_replace function searches for string specified by pattern and replaces pattern with
replacement if found.

ere
3 gi
The eregi function searches throughout a string specified by pattern for a string specified by string.
The search is not case sensitive.

eregi_re
4 place
The eregi_replace function operates exactly like ereg_replace, except that the search for pattern in
string is not case sensitive.

spl
5 it
The split function will divide a string into various elements, the boundaries of each element based on
the occurrence of pattern in string.

6 spliti
The spliti function operates exactly in the same manner as its sibling split, except that it is not case
sensitive.

7 sql_regcase
The sql_regcase function can be thought of as a utility function, converting each character in the
input parameter string into a bracketed expression containing two characters.

PERL Style Regular Expressions

Perl-style regular expressions are similar to their POSIX counterparts. The POSIX syntax can be used almost interchangeably with the Perl-style regular expression functions. In fact, you can use any of the quantifiers introduced in the previous POSIX section.

Lets give explanation for few concepts being used in PERL regular expressions. After that we will introduce you wih regular expression related functions.

Meta characters

A meta character is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.

For instance, you can search for large money sums using the '\d' meta character: /[\d]+000/, Here \d will search for any string of numerical character.

Following is the list of meta characters which can be used in PERL Style Regular Expressions.

Character	Description
.	a single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo|bar|baz)	matches any of the alternatives specified

Modifiers

Several modifiers are available that can make your work with regexps much easier, like case sensitivity, searching in multiple lines etc.

Modifier	Description
i	Makes the match case insensitive
	Specifies that it the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary
o	Evaluates the expression only once
s	Allows use of . to match a newline character

x	Allows you to use white space in	the expression for clarity
g	Globally finds all matches	
cg	Allows a search to continue even	after a global match fails

PHP's Regexp PERL Compatible Functions

PHP offers following functions for searching strings using Perl-compatible regular expressions –

Sr.No Function & Description

00 11 00

1 preg_match

The preg_match function searches string for pattern, returning true if pattern exists, and false otherwise.

2 preg_match_all

The preg_match_all function matches all occurrences of pattern in string.

3 preg_replace

The preg_replace function operates just like ereg_replace, except that regular expressions can be used in the pattern and replacement input parameters.

4 preg_split

The preg_split function operates exactly like split, except that regular expressions are accepted as input parameters for pattern.

5 preg_grep

The preg_grep function searches all elements of input_array, returning all elements matching the regexp pattern.

6 preg_quote

Quote regular expression characters

PHP - SESSIONS

13

https://www.tutorialspoint.com/php/php_sessions.htm Copyright © tutorialspoint.com

An alternative way to make data accessible across the various pages of an entire website is to use a PHP Session.

A session creates a file in a temporary directory on the server where registered session variables and their values are stored. This data will be available to all pages on the site during that visit.

XMAP

C://.

The location of the temporary file is determined by a setting in the php.ini file called session.save_path.
Before using any session variable make sure you have setup this path.

When a session is started following things happen –

PHP first creates a unique identifier for that particular session which is a random string of 32 hexadecimal numbers such as 3c7foj34c3jj973hjkop2fc937e3443. 32

A cookie called PHPSESSID is automatically sent to the user's computer to store unique session identification string.

A file is automatically created on the server in the designated temporary directory and bears the name of the unique identifier prefixed by sess_ ie sess_3c7foj34c3jj973hjkop2fc937e3443.

When a PHP script wants to retrieve the value from a session variable, PHP automatically gets the unique session identifier string from the PHPSESSID cookie and then looks in its temporary directory for the file bearing that name and a validation can be done by comparing both values.

A session ends when the user loses the browser or after leaving the site, the server will terminate the session after a predetermined period of time, commonly 30 minutes duration.

Starting a PHP Session

A PHP session is easily started by making a call to the session_start function.This function first checks if a session is already started and if none is started then it starts one. It is recommended to put the call to session_start at the beginning of the page.

Session variables are stored in associative array called $_SESSION[]. These variables can be accessed during lifetime of a session.

The following example starts a session then register a variable called counter that is incremented each time the page is visited during the session.

Make use of isset function to check if session

variable is already set or not. Put this code in a

test.php file and load this file many times to see

the result –

Live Demo

```
<?php
  session_start();

  if( isset( $_SESSION['counter'] ) ) {
    $_SESSION['counter'] += 1;


  }else {
    $_SESSION['counter'] = 1;
  }

  $msg = "You have visited this page
  ". $_SESSION['counter']; $msg .= "in
  this session.";
?>

<html>

  <head>
    <title>Setting up a PHP session</title>
  </head>

  <body>
    <?php echo ( $msg ); ?>
  </body>
```

```
</html>
```

It will produce the following result –

```
You have visited this page 1in this session.
```

Destroying a PHP Session

A PHP session can be destroyed by session_destroy function. This function does not need any argument and a single call can destroy all the session variables. If you want to destroy a single session variable then you can use unset function to unset a session variable.

Here is the example to unset a single variable –

```
<?php
  unset($_SESSION['counter']);
?>
```

Here is the call which will destroy all the session variables –

```
<?php
  session_destroy();
?>
```

Turning on Auto Session

You don't need to call start_session function to start a session when a user visits your site if you can set session.auto_start variable to 1 in php.ini file.

Sessions without cookies

There may be a case when a user does not allow to store cookies on their machine. So there is another method to send session ID to the browser.

Alternatively, you can use the constant SID which is defined if the session started. If the client did not send an appropriate session cookie, it has the form session_name=session_id. Otherwise, it expands to an empty string.

Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

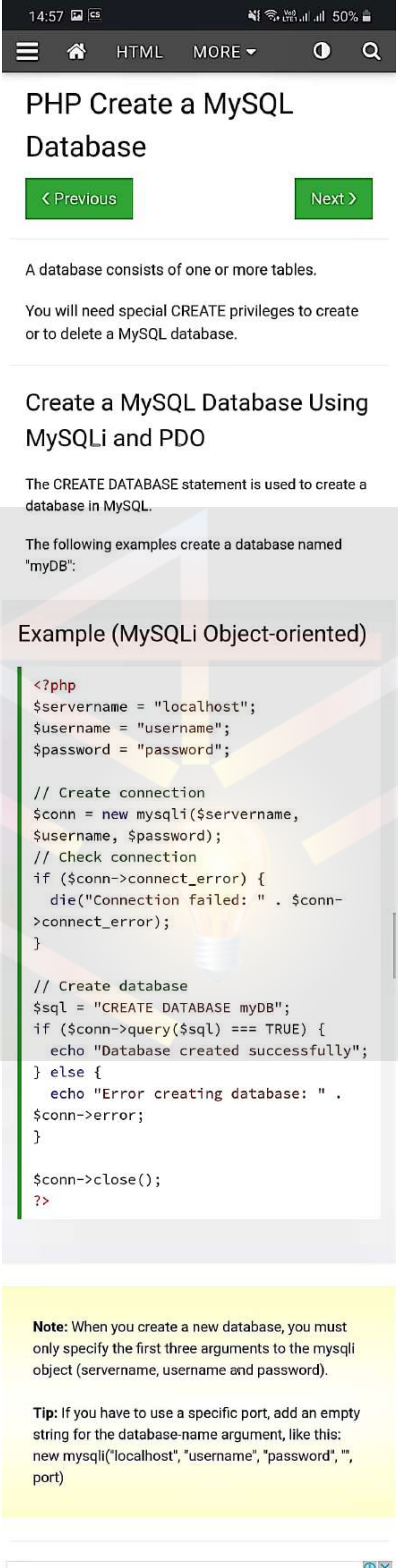
Live Demo

```
<?php
  session_start();

  if (isset($_SESSION['counter'])) {
    $_SESSION['counter'] = 1;
  }else {
    $_SESSION['counter']++;
  }

  $msg = "You have visited this page
  ". $_SESSION['counter']; $msg .= "in
  this session.";

  echo ( $msg );
?>

<p>
  To continue click following link <br />

  <a href = "nextpage.php?<?php echo
htmlspecialchars(SID); ?>"> </p>
```

It will produce the following result −

```
You have visited this page 1in this session.
To continue click following link
```

The htmlspecialchars may be used when printing the SID in order to prevent XSS related attacks.



PHP Create a MySQL Database

A database consists of one or more tables.

You will need special CREATE privileges to create or to delete a MySQL database.

Create a MySQL Database Using MySQLi and PDO

The CREATE DATABASE statement is used to create a database in MySQL.

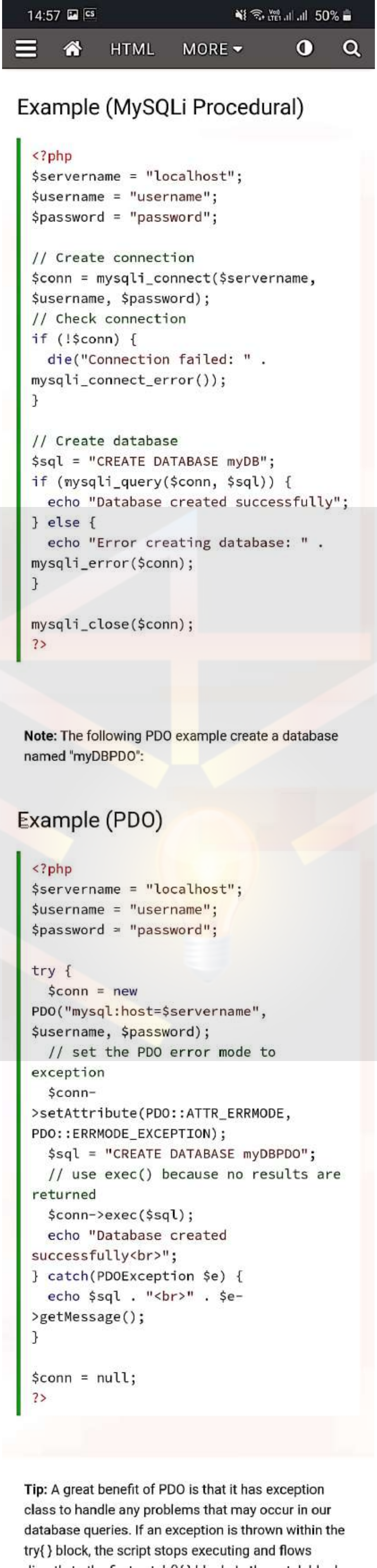
The following examples create a database named "myDB":

Example (MySQLi Object-oriented)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername,
$username, $password);
// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn-
>connect_error);
}

// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " .
$conn->error;
}

$conn->close();
?>
```

Note: When you create a new database, you must only specify the first three arguments to the mysqli object (servername, username and password).

Tip: If you have to use a specific port, add an empty string for the database-name argument, like this: new mysqli("localhost", "username", "password", "", port)



Example (MySQLi Procedural)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = mysqli_connect($servername,
$username, $password);
// Check connection
if (!$conn) {
  die("Connection failed: " .
mysqli_connect_error());
}

// Create database
$sql = "CREATE DATABASE myDB";
if (mysqli_query($conn, $sql)) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " .
mysqli_error($conn);
}

mysqli_close($conn);
?>
```

Note: The following PDO example create a database named "myDBPDO":

Example (PDO)

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

try {
  $conn = new
PDO("mysql:host=$servername",
$username, $password);
  // set the PDO error mode to
exception
  $conn-
>setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
  $sql = "CREATE DATABASE myDBPDO";
  // use exec() because no results are
returned
  $conn->exec($sql);
  echo "Database created
successfully<br>";
} catch(PDOException $e) {
  echo $sql . "<br>" . $e-
>getMessage();
}

$conn = null;
?>
```

Tip: A great benefit of PDO is that it has exception class to handle any problems that may occur in our database queries. If an exception is thrown within the try{ } block, the script stops executing and flows

.Exception handling improvements.

•Enhanced BLOB/CLOB functionality.

Connection and statement interface enhancements.

National character set support.
SQL ROWID access.

SQL 2003 XML data type support.

Annotations.
Loading [MathJax]/jax/output/HTML-CSS/jax.js

PERL - DATABASE ACCESS

15

https://www.tutorialspoint.com/perl/perl_database_access.htm

This chapter teaches you how to access a database inside your Perl script. Starting from Perl 5 has become very easy to write database applications using DBI module. DBI stands for Database Independent Interface for Perl, which means DBI provides an abstraction layer between the Perl code and the underlying database, allowing you to switch database implementations really easily.

The DBI is a database access module for the Perl programming language. It provides a set of methods, variables, and conventions that provide a consistent database interface, independent of the actual database being used.

Architecture of a DBI Application

DBI is independent of any database available in backend. You can use DBI whether you are working with Oracle, MySQL or Informix, etc. This is clear from the following architure diagram.

Here DBI is responsible of taking all SQL commands through the API, i. e. , ApplicationProgrammingInterface and to dispatch them to the appropriate driver for actual execution. And finally, DBI is responsible of taking results from the driver and giving back it to the calling scritp.

Notation and Conventions

Throughout this chapter following notations will be used and it is recommended that you should also follow the same convention.

$dsn	Database source name
$dbh	Database handle object
$sth	Statement handle object
$h	Any of the handle types above ($dbh, $sth, or $drh)
$rc	General Return Code (boolean: true=ok, false=error)
$rv	General Return Value (typically an integer)
@ary	List of values returned from the database.
$rows	Number of rows processed (if available, else -1)
$fh	A filehandle
undef	NULL values are represented by undefined values in Perl
\%attr	Reference to a hash of attribute values passed to methods

Database Connection

Assuming we are going to work with MySQL database. Before connecting to a database make sure of the followings. You can take help of our MySQL tutorial in case you are not aware about how to create database and tables in MySQL database.

You have created a database with a name TESTDB.

You have created a table with a name TEST_TABLE in TESTDB.

This table is having fields FIRST_NAME, LAST_NAME, AGE, SEX and INCOME.

User ID "testuser" and password "test123" are set to access TESTDB.

Perl Module DBI is installed properly on your machine.

You have gone through MySQL tutorial to understand MySQL Basics.

Following is the example of connecting with MySQL database "TESTDB" –

```
#!/usr/bin/perl
```

```
use DBI
use strict;

my $driver = "mysql";
my $database = "TESTDB";
my $dsn = "DBI:$driver:database=$database";
my $userid = "testuser";
my $password = "test123";

my $dbh = DBI->connect($dsn, $userid, $password ) or die
$DBI::errstr;
```

If a connection is established with the datasource then a Database Handle is returned and saved into dbhforfurtheruseotherwisedbh is set to *undef* value and $DBI::errstr returns an error string.

INSERT Operation

DBI :- database independent Interface.

INSERT operation is required when you want to create some records into a table. Here we are using table TEST_TABLE to create our records. So once our database connection is established, we are ready to create records into TEST_TABLE. Following is the procedure to create single record into TEST_TABLE. You can create as many as records you like using the same concept.

Record creation takes the following steps –

Preparing SQL statement with INSERT statement. This will be done using prepare API.

Executing SQL query to select all the results from the database. This will be done using execute API.

Releasing Stattement handle. This will be done using finish API.

If everything goes fine then commit this operation otherwise you can rollback complete transaction.
Commit and Rollback are explained in next sections.

```
my $sth = $dbh->prepare("INSERT INTO TEST_TABLE
                (FIRST_NAME, LAST_NAME, SEX, AGE, INCOME )
                 values
                ('john', 'poul', 'M', 30, 13000)");
$sth->execute() or die $DBI::errstr;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

this sth→ext

Using Bind Values

There may be a case when values to be entered is not given in advance. So you can use bind variables which will take the required values at run time. Perl DBI modules make use of a question mark in place of actual value and then actual values are passed through execute API at the run time. Following is the example –

```
my $first_name = "john";
my $last_name = "poul";
my $sex = "M";
my $income = 13000;
my $age = 30;
my $sth = $dbh->prepare("INSERT INTO TEST_TABLE
                (FIRST_NAME, LAST_NAME, SEX, AGE, INCOME )
                 values
                (?,?,?,?)");
$sth->execute($first_name,$last_name,$sex, $age, $income)
      or die $DBI::errstr;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

READ Operation

READ Operation on any databasse means to fetch some useful information from the database, i.e., one or more records from one or more tables. So once our database connection is established, we are ready to make a query into this database. Following is the procedure to query all the records having AGE greater than 20. This will take four steps –

Preparing SQL SELECT query based on required conditions. This will be done using prepare API.

Executing SQL query to select all the results from the database. This will be done using execute API.

Fetching all the results one by one and printing those results.This will be done using fetchrow_array API.

Releasing Stattement handle. This will be done using finish API.

```
my $sth = $dbh->prepare("SELECT
                FIRST_NAME,
                LAST_NAME FROM
                TEST_TABLE
                WHERE AGE > 20");
```

Select

```
$sth->execute() or die $DBI::errstr;
print "Number of rows found :" + $sth->rows;
while (my @row = $sth->fetchrow_array()) {
  my ($first_name, $last_name ) = @row;
  print "First Name = $first_name, Last Name = $last_name\n";
}
$sth->finish();
```

Using Bind Values

There may be a case when condition is not given in advance. So you can use bind variables, which will take the required values at run time. Perl DBI modules makes use of a question mark in place of actual value and then the actual values are passed through execute API at the run time. Following is the example –

```
$age = 20;
my $sth = $dbh->prepare("SELECT
                FIRST_NAME,
                LAST_NAME FROM
                TEST_TABLE
                WHERE AGE > ?");
$sth->execute( $age ) or die $DBI::errstr;
print "Number of rows found :" + $sth->rows;
while (my @row = $sth->fetchrow_array()) {
  my ($first_name, $last_name ) = @row;
  print "First Name = $first_name, Last Name = $last_name\n";
}
$sth->finish();
```

UPDATE Operation

UPDATE Operation on any database means to update one or more records already available in the database tables. Following is the procedure to update all the records having SEX as 'M'. Here we will increase AGE of all the males by one year. This will take three steps –

Preparing SQL query based on required conditions. This will be done using prepare API.

Executing SQL query to select all the results from the database. This will be done using execute API.

Releasing Stattement handle. This will be done using finish API.

If everything goes fine then commit this operation otherwise you can rollback complete transaction. See next section for commit and rollback APIs.

```
my $sth = $dbh->prepare("UPDATE TEST_TABLE
                SET AGE = AGE + 1
                WHERE SEX = 'M'");
$sth->execute() or die $DBI::errstr;
print "Number of rows updated :" + $sth->rows;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

Using Bind Values

There may be a case when condition is not given in advance. So you can use bind variables, which will take required values at run time. Perl DBI modules make use of a question mark in place of actual value and then the actual values are passed through execute API at the run time. Following is the example –

```
$sex = 'M';
my $sth = $dbh->prepare("UPDATE TEST_TABLE
                SET AGE = AGE + 1
                WHERE SEX = ?");
$sth->execute('$sex') or die $DBI::errstr;
print "Number of rows updated :" + $sth->rows;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

In some case you would like to set a value, which is not given in advance so you can use binding value as follows.
In this example income of all males will be set to 10000.

```
$sex = 'M';
$income = 10000;
my $sth = $dbh->prepare("UPDATE TEST_TABLE
                SET INCOME = ?
                WHERE SEX = ?");
$sth->execute( $income, '$sex') or die $DBI::errstr;
print "Number of rows updated :" + $sth->rows;
$sth->finish();
```

DELETE Operation

DELETE operation is required when you want to delete some records from your database. Following is the procedure to delete all the records from TEST_TABLE where AGE is equal to 30. This operation will take the following steps.

Preparing SQL query based on required conditions. This will be done using prepare API.

Executing SQL query to delete required records from the database. This will be done using execute API.

Releasing Stattement handle. This will be done using finish API.

If everything goes fine then commit this operation otherwise you can rollback complete transaction.

```
$age = 30;
my $sth = $dbh->prepare("DELETE FROM TEST_TABLE
                        WHERE AGE = ?");
$sth->execute( $age ) or die $DBI::errstr;
print "Number of rows deleted :" + $sth->rows;
$sth->finish();
$dbh->commit or die $DBI::errstr;
```

Using do Statement

If you're doing an UPDATE, INSERT, or DELETE there is no data that comes back from the database, so there is a short cut to perform this operation. You can use do statement to execute any of the command as follows.

```
$dbh->do('DELETE FROM TEST_TABLE WHERE age =30');
```

do returns a true value if it succeeded, and a false value if it failed. Actually, if it succeeds it returns the number of affected rows. In the example it would return the number of rows that were actually deleted.

COMMIT Operation

Commit is the operation which gives a green signal to database to finalize the changes and after this operation no change can be reverted to its orignal position.

Here is a simple example to call commit API.

```
$dbh->commit or die $dbh->errstr;
```

ROLLBACK Operation

If you are not satisfied with all the changes or you encounter an error in between of any operation , you can revert those changes to use rollback API.

Here is a simple example to call rollback API.

Ctrl + 2

```
$dbh->rollback or die $dbh->errstr;
```

Begin Transaction

Many databases support transactions. This means that you can make a whole bunch of queries which would modify the databases, but none of the changes are actually made. Then at the end, you issue the special SQL query COMMIT, and all the changes are made simultaneously. Alternatively, you can issue the query ROLLBACK, in which case all the changes are thrown away and database remains unchanged. Perl DBI module provided begin_work API, which enables transactions byturningAutoCommitoff until the next call to commit or rollback. After the next commit or rollback, AutoCommit will automatically be turned on again.

```
$rc                = $dbh->begin_work  or die $dbh->errstr;
```

AutoCommit Option

If your transactions are simple, you can save yourself the trouble of having to issue a lot of commits. When you make the connect call, you can specify an AutoCommit option which will perform an automatic commit operation after every successful query. Here's what it looks like

–

```
my $dbh = DBI->connect($dsn, $userid, $password,
             {AutoCommit => 1})
             or die $DBI::errstr;
```

Here AutoCommit can take value 1 or 0, where 1 means AutoCommit is on and 0 means AutoCommit is off.

Automatic Error Handling

When you make the connect call, you can specify a RaiseErrors option that handles errors for you automatically. When an error occurs, DBI will abort your program instead of returning a failure code. If all you want is to abort the program on an error, this can be convenient. Here's what it looks like –

```
my $dbh = DBI->connect($dsn, $userid, $password,
        {RaiseError => 1})
        or die $DBI::errstr;
```

Here RaiseError can take value 1 or 0.

Disconnecting Database

To disconnect Database connection, use disconnect API as follows –

```
$rc = $dbh->disconnector warn $dbh->errstr;
```

The transaction behaviour of the disconnect method is, sadly, undefined. Some database systems suchasOracleandIngres will automatically commit any outstanding changes, but others suchasInformix will rollback any outstanding changes. Applications not using AutoCommit should explicitly call commit or rollback before calling disconnect.

Using NULL Values

Undefined values, or undef, are used to indicate NULL values. You can insert and update columns with a NULL value as you would a non-NULL value. These examples insert and update the column age with a NULL value –

```
$sth = $dbh->prepare(qq {
      INSERT INTO TEST_TABLE (FIRST_NAME, AGE) VALUES (?, ?)
    });
$sth->execute("Joe", undef);
```

Here qq{} is used to return a quoted string to prepare API. However, care must be taken when trying to use NULL values in a WHERE clause. Consider –

```
SELECT FIRST_NAME FROM TEST_TABLE WHERE age = ?
```

Binding an undef NULL to the placeholder will not select rows, which have a NULL age! At least for database engines that conform to the SQL standard. Refer to the SQL manual for your database engine or any SQL book for the reasons for this. To explicitly select NULLs you have to say "WHERE age IS NULL".

A common issue is to have a code fragment handle a value that could be either defined or undef
non - NULLorNULL at runtime. A simple technique is to prepare the appropriate statement as needed, and substitute the placeholder for non-NULL cases –

```
$sql_clause = defined $age? "age
= ?" : "age IS NULL"; $sth =
$dbh->prepare(qq {
     SELECT FIRST_NAME FROM
    TEST_TABLE WHERE $sql_clause });
$sth->execute(defined $age ? $age : ());
```

Some Other DBI Functions

available_drivers

```
@ary = DBI->available_drivers;
@ary = DBI->available_drivers($quiet);
```

Returns a list of all available drivers by searching for DBD::* modules through the directories in @INC. By default, a warning is given if some drivers are hidden by others of the same name in earlier directories. Passing a true value for $quiet will inhibit the warning.

installed_drivers

```
%drivers = DBI->installed_drivers();
```

Returns a list of driver name and driver handle pairs for all drivers 'installed' **loaded** into the current process. The driver name does not include the 'DBD::' prefix.

data_sources

```
@ary = DBI->data_sources($driver);
```

Returns a list of data sources **databases** available via the named driver. If $driver is empty or undef, then the value of the DBI_DRIVER environment variable is used.

quote

```
$sql = $dbh->quote($value);
$sql = $dbh->quote($value, $data_type);
```

Quote a string literal for use as a literal value in an SQL statement, by escaping any special characters suchasquotationmarks contained within the string and adding the required type of outer quotation marks.

```
$sql = sprintf "SELECT foo FROM bar
          WHERE baz = %s",
        $dbh->quote("Don't");
```

For most database types, quote would return 'Don''t'

includingtheouterquotationmarks. It is valid for the quote method to return an SQL expression that evaluates to the desired string. For example –

```
$quoted = $dbh->quote("one\ntwo\0three")
```

```
may produce results which will be equivalent to
```

```
CONCAT('one', CHAR(12), 'two', CHAR(0), 'three')
```

Methods Common to All Handles

err

```
$rv = $h->err;
or
$rv = $DBI::err
or
$rv = $h->err
```

Returns the native database engine error code from the last driver method called. The code is typically an integer but you should not assume that. This is equivalent to DBI :: errorh->err.

errstr

```
$str = $h->errstr;
or
$str = $DBI::errstr
or
$str = $h->errstr
```

Returns the native database engine error message from the last DBI method called. This has the same lifespan issues as the "err" method described above. This is equivalent to **DBI** :: **errstror** h->errstr.

rows

```
$rv = $h->rows;
or
$rv = $DBI::rows
```

This returns the number of rows effected by previous SQL statement and equivalent to $DBI::rows.

trace

```
$h->trace($trace_settings);
```

DBI sports an extremely useful ability to generate runtime tracing information of what it's doing, which can be a huge time-saver when trying to track down strange problems in your DBI programs. You can use different values to set trace level. These values varies from 0 to 4. The value 0 means disable trace and 4 means generate complete trace.

Interpolated Statements are Prohibited

It is highly recommended not to use interpolated statements as follows –

```
while ($first_name = <>) {
  my $sth = $dbh->prepare("SELECT *
                 FROM TEST_TABLE
                 WHERE FIRST_NAME = '$first_name'");

  $sth->execute();
  # and so on ...
}
```

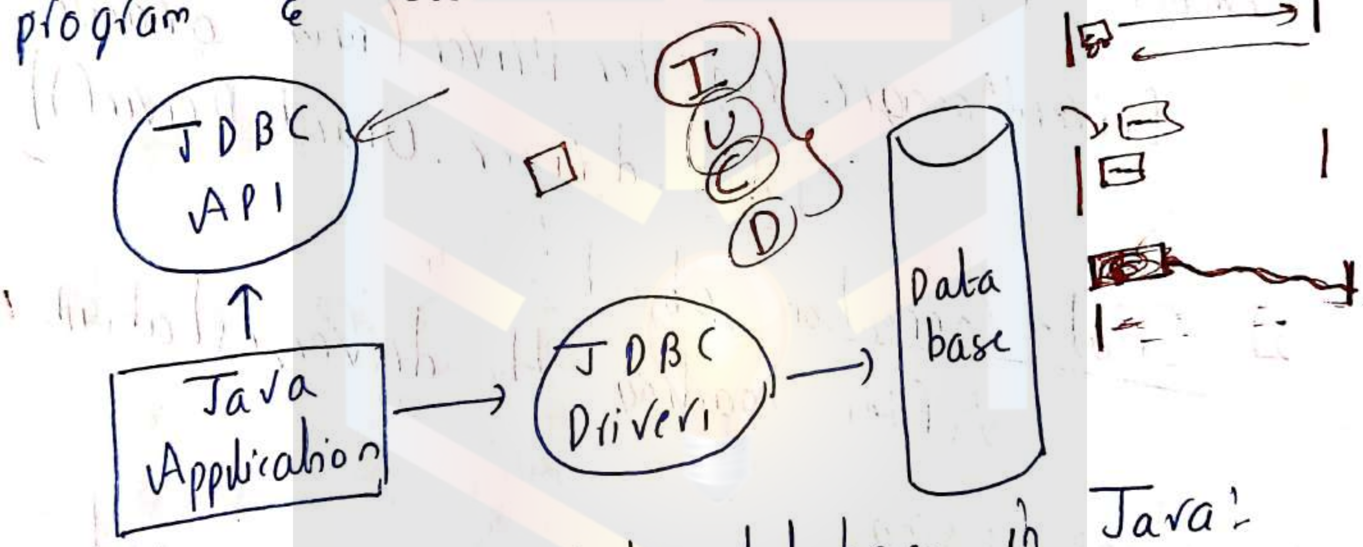
Thus don't use interpolated statement instead use bind value to prepare dynamic SQL statement.

→ JDBC :- (Java Database Connectivity)

→ JDBC is an standard API specification developed to move data from Frontend to backend(database).

→ This API consists of classes & interfaces written in java.

→ It acts as a channel b/n java program & database (to send data).



→ Steps to connect to database in Java:-

1) Loading/registering the Driver

→ Load/register the driver before starting using it

→ only 1 time process.

J
J

a) class.ForName() :- here we load the driver class file into memory at runtime.

Class.ForName("oracle.jdbc.driver.OracleDriver");

(commonly used) jar

b) DriverManager.registerDriver() :- Driver manager is a Java inbuilt class with a static member register

DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver())

2) Create Connection object :- After loading the driver, establish connections using.

Connection con = DriverManager. ~~get~~ getConnection(url, user, password)

url ↓ for databan

user ↓ username of databan

password ↓ pass of databan

con

Conn

3) Create a Statement:-

→ Now as connection is established you can interact with the database.

→ It enables you to send SQL commands & receive data from your database

Statement stmt = con.createStatement();

st sql = " "

4) Execute the Query:-

→ most important part executing the query here ~~it is~~ we use SQL Query

C	I	U	D

1) ✓ Select 2 types

Query for updating/inserting table in a database (executeUpdate() sql query method which returns status.) C d

```
String sql = "any update/insert query"
int m = ~~st~~ stmt.executeUpdate(sql)
if (m==1) SOP("inserted succ");
else SOP("Failed");
```

Query for retriving data. (executeQuery() SQL query method to return the result set)

```
ResultSet rs = stmt.executeQuery(SQL);
while(rs.next())
{ ~~sh o = rs.next~~      } to print data
}
```

quote

5) Close the connection:

Once our task is completed we can close the connection

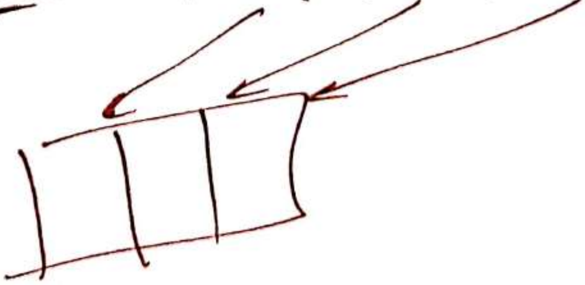
con.close();

example printed

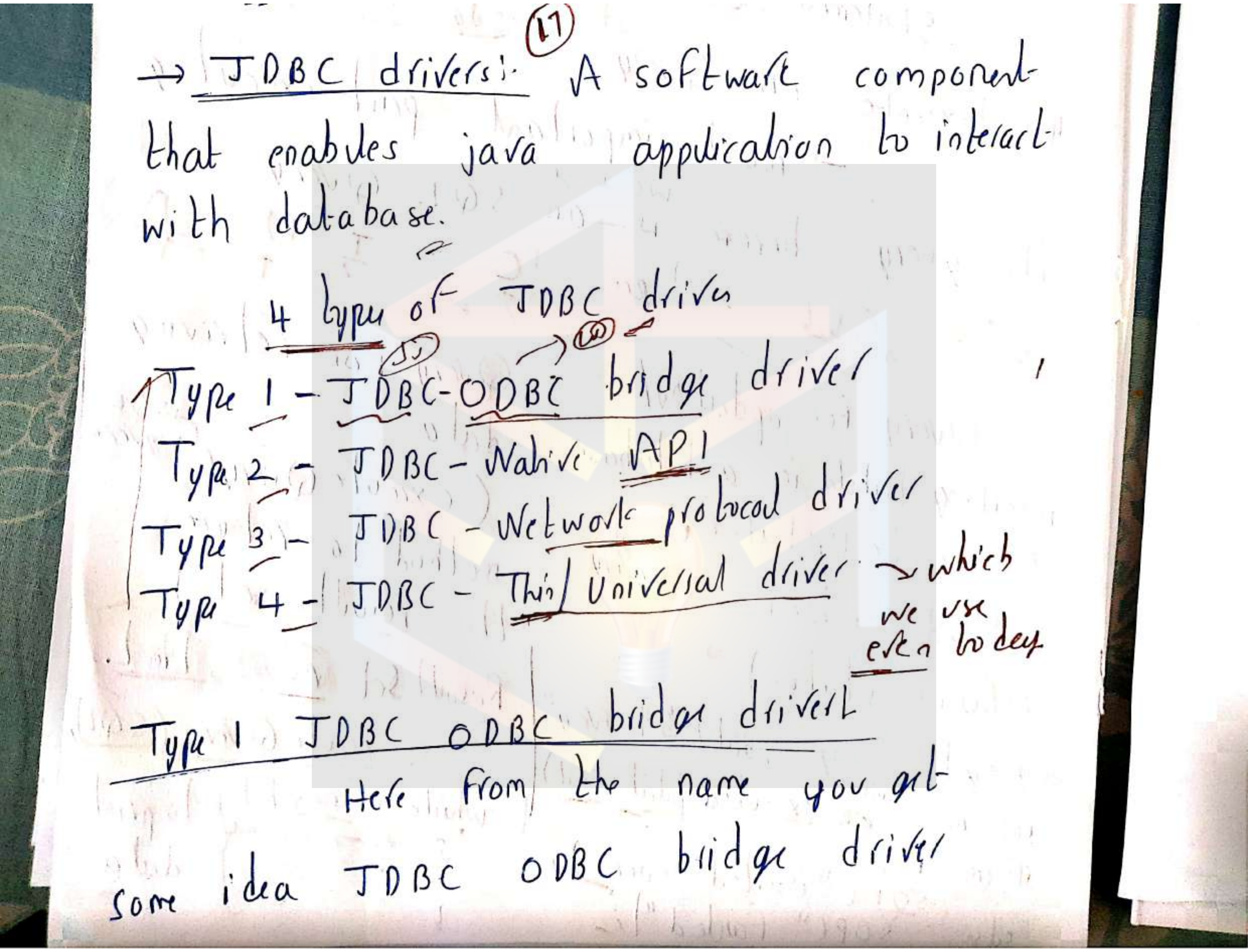
17

→ JDBC drivers: A software component that enables java application to interact with database

```
importjava.sql.*;
importjava.util.*;
class Main
{
   public static void main(String a[])
   {
        //Creating the connection
        String url = "jdbc:oracle:thin:@localhost:1521:xe";
        String user = "system";
        String pass = "12345";

        //Entering the data
        Scanner k = new Scanner(System.in);
        System.out.println("enter name");
        String name = k.next();
        System.out.println("enter roll no");
        int roll = k.nextInt();
        System.out.println("enter class");
        String cls = k.next();

        //Inserting data using SQL query
        String sql = "insert into student1 values('"+name+"',"+roll+",'"+cls+"')";
        Connection con=null;
```



```
        try
        {
            DriverManager.registerDriver(new oracle.jdbc.OracleDriver());

            //Reference to connection interface
            con = DriverManager.getConnection(url,user,pass);

            Statement st = con.createStatement();
            int m = st.executeUpdate(sql);
            if (m == 1)
                System.out.println("inserted successfully : "+sql);
            else
                System.out.println("insertion failed");
            con.close();
        }
        catch(Exception ex)
        {
            System.err.println(ex);
        }
    }
}
```



→ JDBC drivers:- A software component that enables java application to interact with database.

4 types of JDBC drivers

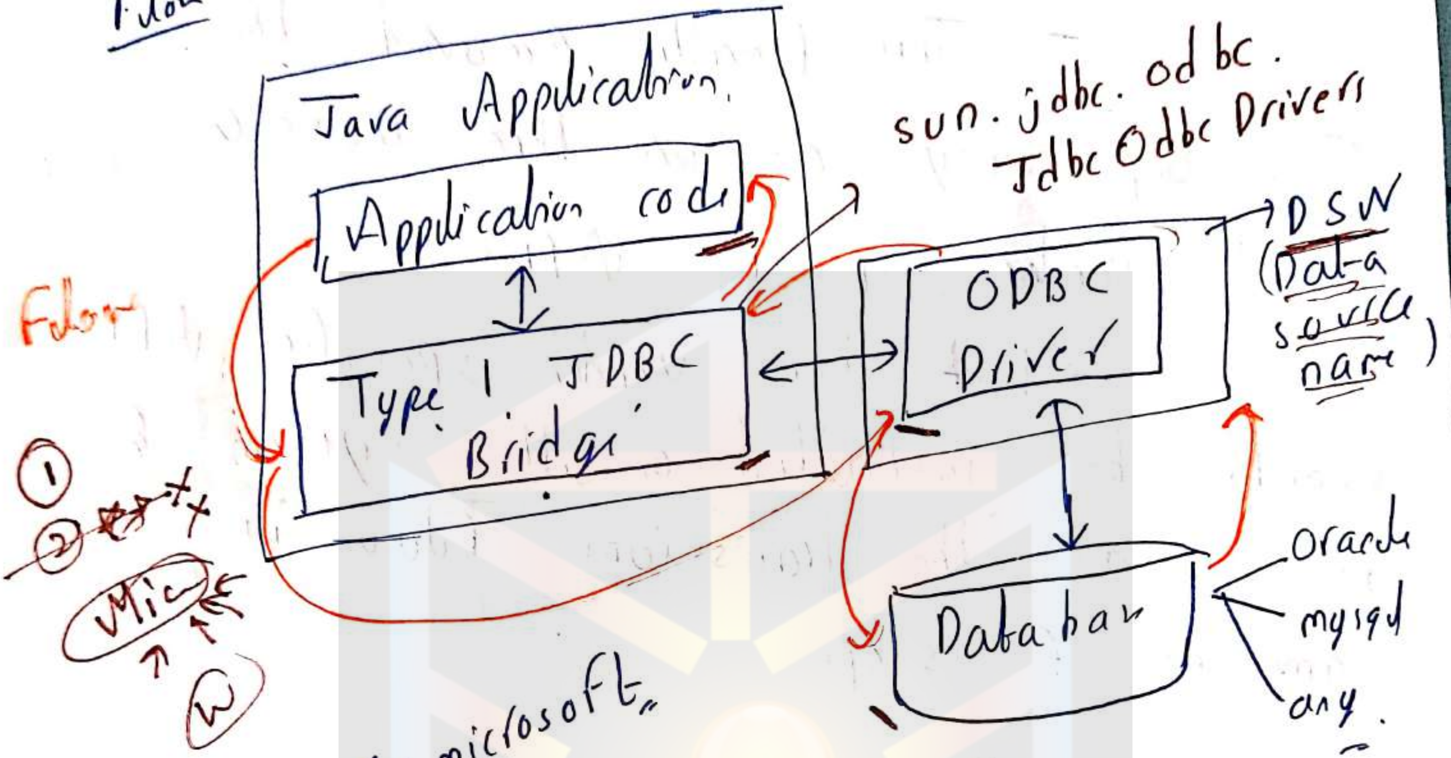
Type 1 - JDBC-ODBC bridge driver
Type 2 - JDBC-Native API
Type 3 - JDBC - Network protocol driver
Type 4 - JDBC - Thin/ Universal driver → which we use even today

Type 1 JDBC ODBC bridge driver

Here from the name you get some idea JDBC ODBC bridge driver

→ For JDBC, ODBC is acting as a bridge (Open data base connectivity)

Flow



By microsoft

→ ODBC is windows specific.

→ Hence this driver only works for windows OS.

→ After Java 7 this driver is completely discontinued.

→ multiple flow transfers are done & OS specific hence not used now.

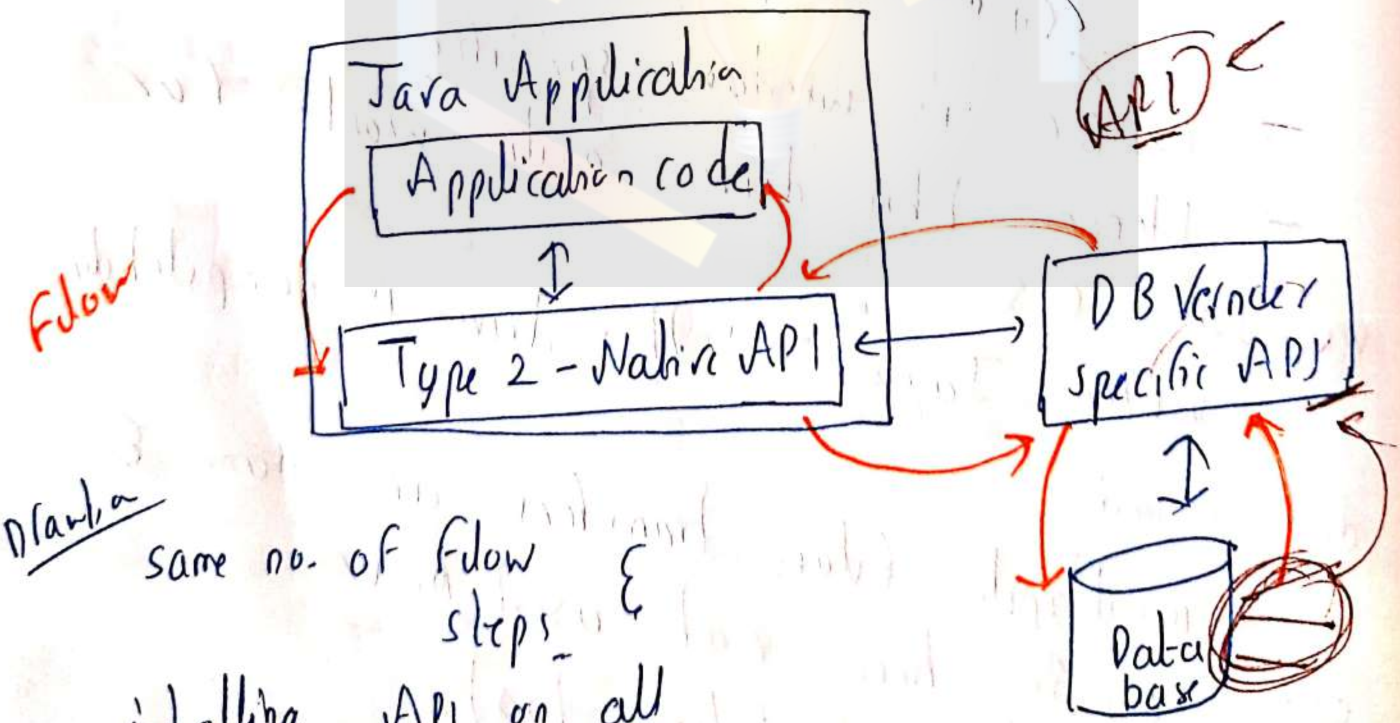
(no exceptional handeling tool).

Type 2 :- JDBC Native API

→ Here to avoid ODBC we use vender specific Database API's

→ This finally removed the OS dependency as we are using direct vender specific API's

→ But this vender specific API should be installed on all system & as well as the transsions flow is same no. of steps



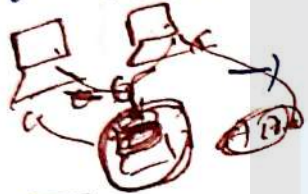
Drawback
same no. of flow & steps
installing API on all systems,

Type 3 :- Network protocol driver

→ Just a small modification to Type 2 (avoiding multiple installation)

→ here we maint a server through which we run type 2

→ hence as it's shared by server it called Network protocol driver.



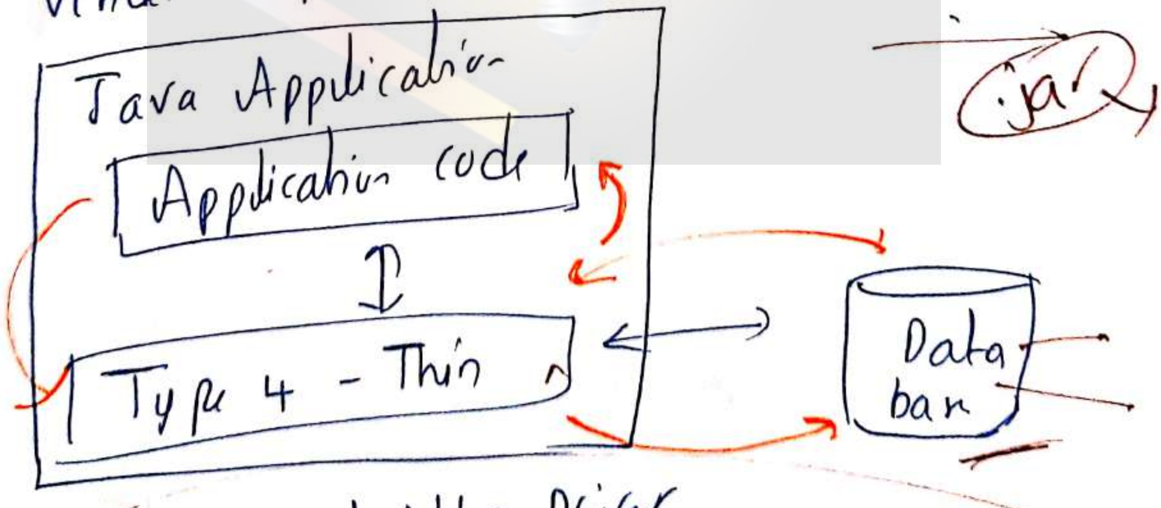
→ expensive hence not used

Type 4 :- Thin/ Universal driver

→ Even we use this driver now

→ here no middleware (ODBC or DB vender specific API).

OD



MySQL:- com.msql.jdbc.Driver

oracle: oracle.jdbc.driver.OracleDriver