

Chapter I

Lesson 1 : PHP

I. Introduction

PHP កាលពីមុនគឺមានឈ្មោះថា Personal Home page តែក្រោយមកទៀត ត្រូវបានគេដូរឈ្មោះថា HyperText Preprocessor ។ ហើយម្យ៉ាងទៀត PHP គឺជាប្រភេទ server-side script ដែលគេបង្កើតឡើងមកយ៉ាងពិសេសដើម្បីអភិវឌ្ឍន៍ Web តែម្តង។ គួរបញ្ជាក់ផងដែរថា code PHP ត្រូវបានគេសរសេរក្នុងបរិបទ HTML(HTML-embedded) តែត្រូវ Save ជា file ដែលមាន extension “.php”។

II. PHP Tags

1. Conical Style Tag

```
<?php  
    expression;  
?>
```

2. Shorten Echo Tag

```
<?= ... ?> <=> <?php echo ... ?>
```

3. Shorten Style Tag (In Case for editing short_open_tag in php.ini)

```
<?  
    expression;  
?>
```

III. Why must study PHP?

- Performance : Scripts written in PHP execute faster than those written in other scripting languages.
- Cost
- Speed
- HTML Embeddedness
- Source Code
- Portable/Cross platform : PHP is available for UNIX, Microsoft Windows, Mac OS, and OS/2, and PHP programs are portable between platforms.
- Built-in Libraries
- Ease of Learning
- OOP Support
- Database Integration : PHP Application with MYSQL database integration

IV. HTML Embedded

1. HTML in PHP script code

The way you place your HTML code inside the PHP statement.

Ex : `echo '<h1>HTML in PHP</h1>';`

2. Jump In and Out of php in HTML

some time you can open and close PHP's tag many times to met your requirement.

`<?= ... ?> <=> <?php echo ... ?>`

3. Including File

a. Include : include statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

`include('url');`

b. Include Once

`include_once('url');`

c. Require : require statement takes all the text/code/markup that exists in the specified file and copies it into the file that uses the include statement.

`require('url');`

d. Require Once

`require_once('url');`

Chapter II

Lesson 2 : Variables and Operators

I. Variables

+ What is variable?

A variable is simply a container that's used to store values. A variable represent or hold values.

ex. `$var = 5;` // `$var` is using to hold value

`$total = $var * 3;` // `$var` is using to represent value

+ How to define variable in the right ways?

- All variable in PHP are denoted with a leading dollar sign (\$)
- The value of a variable is the value of its most recent assignment
- Variables are assigned with the (=) operator
- Variable can, but do not need, to be declared before assignment
- Variables have not intrinsic(necessary) type other that the type of their current value
- Variables used before they are assigned have default values
- Must begin with a letter or underscore character

I. Variables

+ Formal style for name variable

- `$var`
- `$var_name`
- `$var1`
- `$_var`
- `$_var1`

+ Destroying Variables

use `unset()` function to destroy variable.

```
$var = 99;
```

```
echo $var;
```

```
unset($var);
```

+ Inspecting Variable Contents

The `var_dump()` function is used to display structured information (type and value or letters number) about one or more variables.

```
var_dump($var1, $var2, ...);
```


I. Variables

+ PHP's Data Types

there are free of data type:

- String
- Int(Integer)
- Float (floating point numbers - also called double)
- Boolean
- Array
- Object
- NULL
- Resource
- Callable

+ Setting and Checking Variable Data Types

use `gettype($var)` for checking variable data type.

+ Constants Variable

using `define()` function to define the constant variable in PHP.

ex. `define('EXCHANGE', 4100);`

II. Operators

+ What is operator?

is a word or symbol which use to calculation value.

+ Arithmetic Operators

1. “ + ” : Addition
2. “ - ” : Subtraction
3. “ * ” : Multiplication
4. “ / ” : Division
5. “ % ” : Modulus
6. “ ** ” : Power

+ String Operator

↗ “ . ” : Concatenation

+ Assignment Operator

↗ “ = ” : Assignment

II. Operators

+ Combination Assignment

- | | | | | |
|------------|--|-------------|-----|------------------|
| 1. “ += ” | | \$a += \$b | <=> | \$a = \$a + \$b |
| 2. “ -= ” | | \$a -= \$b | <=> | \$a = \$a - \$b |
| 3. “ *= ” | | \$a *= \$b | <=> | \$a = \$a * \$b |
| 4. “ /= ” | | \$a /= \$b | <=> | \$a = \$a / \$b |
| 5. “ %= ” | | \$a %= \$b | <=> | \$a = \$a % \$b |
| 6. “ .= ” | | \$a .= \$b | <=> | \$a = \$a . \$b |
| 7. “ **= ” | | \$a **= \$b | <=> | \$a = \$a ** \$b |

❖ Pre-Post Increment and Decrement

- | | | | | |
|-----------|-----|----------|-----|---------------|
| a. ++ \$i | <=> | \$i += 1 | <=> | \$i = \$i + 1 |
| b. \$i ++ | <=> | \$i += 1 | <=> | \$i = \$i + 1 |
| c. -- \$i | <=> | \$i -= 1 | <=> | \$i = \$i - 1 |
| d. \$i -- | <=> | \$i -= 1 | <=> | \$i = \$i - 1 |

II. Operators

+ Comparison Operators

1. “ == ” : is equal to, (Name: Equal)
2. “ === ” : strict is equal to, (Name: Identical)
3. “ != ” or “ <> ” : is not equal to, (Name: Not equal)
4. “ !== ” : strict is not equal to, (Name: Not identical)
5. “ > ” : is greater than, (Name: Greater than)
6. “ < ” : is less than, (Name: Less than)
7. “ >= ” : is greater than or equal, (Name: Greater than or equal to)
8. “ <= ” : is less than or equal , (Name: Less than or equal to)
9. “ <=> ” : Returns 0 if both operands are equal, 1 if the left is greater, and -1 if the right is greater , (Name: Spaceship)

❖ Note

Operator	<=> equivalent
(\$a < \$b)	(\$a <=> \$b) == -1
(\$a <= \$b)	(\$a <=> \$b) == -1 (\$a <=> \$b) == 0
(\$a == \$b)	(\$a <=> \$b) == 0
(\$a >= \$b)	(\$a <=> \$b) == 1 (\$a <=> \$b) == 0
(\$a > \$b)	(\$a <=> \$b) == 1

II. Operators

+ Logical Operators

1. “ ! ” : Is not true
2. “ && ” : And
3. “ || ” : Or
4. “xor ” or “^ ” : TRUE if either \$a or \$b is TRUE , but not both.

A	B	A xor B
1	1	0
1	0	1
0	1	1
0	0	0

+ Ternary Operator

(condition) ? true : false

ex: ((\$a)? 'A is true' : 'A is false')

Chapter III

Lesson 3: Controls Flow

I. Conditional Statement

A. IF

```
if(condition){  
    expression;  
}
```

B. IF/ELSE

```
if(condition){  
    expression1;  
} else {  
    expression2;  
}
```

I. Conditional Statement

C. Nested IF

```
if(condition){  
    expression 1 ;  
}else{  
    if(condition){  
        expression 2 ;  
    }else{  
        if(condition){  
            expression 3;  
        }else{  
            other expression;  
        }  
    }  
}
```

D. IF/ELSE IF

```
if(condition1){  
    expression 1;  
} else if (condition2) {  
    expression 2;  
} else if (condition3) {  
    expression 3;  
} ...  
else{  
    other expression;  
}
```


Lesson 3: Controls Flow

I. Conditional Statement

E. SWITCH/CASE

Using for a single condition with multiple value.

```
switch($var){  
    case val1:  
        expression1;  
        break;  
    .....  
    default:  
        other expression;  
        break;  
}
```

Lesson 3: Controls Flow

II. Loop Statement

A. For Loop

```
for(initial-value; termination-check; loop-end-exp){  
    expression;  
}
```

B. Foreach Loop

```
foreach ($array as $value){  
    expression;  
}
```

C. While Loop

```
initial-value;  
while(termination-check){  
    expression;  
    loop-end-exp;  
}
```

D. Do...While Loop

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

Lesson 3: Controls Flow

III. Working with String and Numeric Functions

- + Checking Variable : Is empty or not.

`empty(Variable);`

IV. Method GET and POST

- + PHP `$_GET` is a PHP super global variable which is used to collect form data after submitting an HTML form with `method="get"`.
- + PHP `$_POST` is a PHP super global variable which is used to collect form data after submitting an HTML form with `method="post"`.

Chapter IV

Lesson 4: Arrays

I. Introduction

An array is a special variable, which can hold more than one value at a time.

Ex: Array B មាន ៤ ធាតុគឺ 1, 6, 4, -4

កូដសរសេរ៖ \$B = array(1,6,4,-4);

*** Count Array : Count elements of an array.

Ex: count(\$B)

II. How to create an array?

- + There are 4 ways to create an array :
 - Direct assignment
 - Array construct
 - Specific Indexed or Associative Arrays
 - Function range

II. How to define an array?

1. Direct assignment

```
$fruit = array();
```

```
$fruit[] = 'Apple';
```

```
$fruit[] = 'Banana';
```

```
$fruit[] = 'Coconut';
```

```
Or $fruit=['Apple', 'Banana', 'Coconut'];
```

2. Array construct

```
$fruit = array('Apple', 'Banana', 'Coconut');
```

3. Specific Indexed or Associative Arrays

```
$fruit = array('a' => 'Apple', 'b' => 'Banana', 'c' => 'Coconut');
```

4. Function range

Syntax: range(low, high, step)

a) low : បញ្ជាក់ពីតម្លៃតូចបំផុតនៃ Array ។

b) high : បញ្ជាក់ពីតម្លៃខ្ពស់បំផុតនៃ Array ។

c) step : បញ្ជាក់ពីតម្លៃកើនក្នុងមួយជំហានទៅជិតស្មើ ឬក៏ស្មើនៃតម្លៃ high នៃ Array ។

```
$num = range(2, 9);
```

III. How to get any elements from array?

1. Array key (associate key)

ex: <?php

```
// define array
```

```
$fruits = array('a' => 'apple', 'b' => 'banana',  
                'p' => 'pineapple', 'g' => 'grape'  
                );
```

?>

គេទទួលបាន៖ \$fruits ['p']='pineapple'.

2. print_r function : បង្ហាញព័ត៌មាននៃ **variables** សម្រាប់ការលំអិត ដើម្បីងាយស្រួលយល់។

ex: <?php

```
// define array
```

```
$data = array('Monday', 'Tuesday', 'Wednesday');  
print_r($data);
```

?>

គេទទួលបាន៖ Array ([0] => Monday [1] => Tuesday [2] => Wednesday)

III. How to get any elements from array?

3. Foreach

a. only value

+ foreach(array as value){ echo value }

ex: <?php // define array

```
$cities = array('London', 'Paris', 'Madrid', 'Jakarta');
```

```
foreach ($cities as $c) { echo "$c \r\n"; } ?>
```

b. Keys and Values

+ foreach(array as key => value){ echo key ." => ". value; }

ex: <?php // define array

```
$cities = array( "United Kingdom" => "London", "United States" =>
```

```
"Washington", "France" => "Paris", "India" => "Delhi" );
```

```
foreach ($cities as $key => $value) { echo "$value is in $key. \r\n"; }
```

```
?>
```


IV. Working with Array Functions

PHP មាន Functions មួយចំនួន ដែលមានស្រាប់ ដើម្បីបំពេញការងារប្រតិបត្តិការជាមួយ Array ។ ហើយ Functions ទាំងអស់នោះ រួមមាន៖

- explode() : បំបែក string មួយ ឱ្យទៅជាធាតុរបស់ Array តាមរយៈលុបបំបាត់ argument ទីមួយចោល។ ហើយ Functions មួយនេះ មានពីរ Arguments ។

ex: <?php \$str = 'tinker,tailor,soldier,spy';

\$arr = explode(',', \$str);

print_r(\$arr); ?>

គេទទួលបាន៖ Array នៃ \$arr មានធាតុ ('tinker', 'tailor', 'soldier', 'spy') ;

- Implode() : បង្រួមធាតុរបស់ Array ឱ្យទៅជា String មួយ ដែលមាន argument ទីមួយ ជាឈ្មោះ។ ហើយ Functions មួយនេះ មានពីរ Arguments ។

ex: <?php

\$arr = array('one', 'two', 'three', 'four');

// output: 'one and two and three and four'

\$str = implode(' and ', \$arr); print_r(\$str); ?>

IV. Working with Array Functions

- min() និង max() : រកធាតុដែលតូចបំផុត និងធំបំផុតក្នុង Array មួយ។

ex: <?php // define array

```
$arr = array(7, 36, 5, 48, 28, 90, 91, 3, 67, 42);
```

```
// get min and max
```

```
// output: 'Minimum is 3 and maximum is 91' echo 'Minimum is ' . min($arr) . ' and  
maximum is ' . max($arr);    ?>
```

- array_unique() : លុបធាតុបន្តបន្ទាប់រហស៍ Array ដែលជាន់នឹងធាតុដែលមានស្រាប់។

ex: <?php // define array \$duplicates = array('a', 'b', 'a', 'c', 'e', 'd', 'e');

```
// output: ('a', 'b', 'c', 'e', 'd')
```

```
$uniques = array_unique($duplicates);
```

```
print_r($uniques); ?>
```

IV. Working with Array Functions

- `array_reverse()` : បញ្ជ្រាស់លំដាប់លំដោយនៃធាតុរបស់ Array ពីដើមទៅចុង ពីចុងទៅដើម។

ex: `<?php // define array`

```
$rainbow = array('violet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red');  
// reverse array  
// output: ('red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet');  
$arr = array_reverse($rainbow); ?>
```

- `in_array()` : ស្វែងរកធាតុនៃ Array មួយ ដោយ Function នេះ មានពីរ Arguments ។ ពេលរកឃើញនឹងចេញលេខមួយ បញ្ជាក់ថា ក្នុង Array មួយនោះ ពិតជាមានធាតុដែលត្រូវរកមែន។

ex: `<?php $cities = array('London', 'Paris', 'Barcelona', 'Lisbon', 'Zurich');`

```
// output : 1
```

```
echo in_array('Barcelona', $cities); ?>
```

IV. Working with Array Functions

- `array_merge()` : ប្រើដើម្បីបង្រួមធាតុចូលគ្នារវាង Array មួយ ទៅ Array ជាច្រើនទៀត។

ex: <?php

```
$a1=array("red","green");  
$a2=array("blue","yellow");  
$a3=array("pink","cyan");  
//output: ('red','green','blue','yellow','pink','cyan')  
print_r(array_merge($a1,$a2,$a3)); ?>
```

សម្គាល់៖ **Spread Operator in array expression** instead of using `array_merge()`

+ **(...)** និង **(&)** : សញ្ញា **(...)** នេះ ប្រើសម្រាប់ពន្លាត Array។ រីឯសញ្ញា **(&)** នេះវិញ ប្រើដើម្បីភ្ជាប់ variable by

reference for getting data from variable ។

ex: <?php

```
$ar1 = 'red';  
$ar2 = [&$ar1, 'green', 'blue', 'yellow'];  
$ar3 = [... $ar2, 'pink', 'cyan']; print_r($ar3); ?>
```

IV. Working with Array Functions

- `array_map()` : ប្រើដើម្បីបញ្ជូនធាតុនីមួយៗនៃ Array មួយទៅអោយ Function មួយ ដែលបង្កើតដោយ user ហើយបញ្ជូនត្រឡប់មកវិញនូវ Array ថ្មីមួយ ដែលមានធាតុថ្មីៗចេញពី Function

នេះ៖ ។ syntax: `array_map(myfunction, array1, array2, array3, ...)`

```
ex1: <?php function myfunction($v){ if ($v=="Dog"){ return "Fido";}  
      return $v; }
```

```
$a=array("Horse","Dog","Cat");
```

```
print_r(array_map("myfunction",$a));    ?>
```

```
ex2: <?php $a = [1, 2, 3, 4, 5];
```

```
$b = array_map(fn($n) => $n * $n * $n, $a);
```

```
print_r($b); ?>
```

*** The `fn` keyword is used to create arrow functions. Arrow functions are only available in PHP versions 7.4 and up. Syntax: `fn(arguments) => expression to be returned;`

V. Multi Dimensional Array

1. Two Dimensional

Ex: <?php

```
$arr=array(  
    'fruit'=>array('red'=>'Apple','yellow'=>'Banana'),  
    'flower'=>array('red'=>'Rose','yellow'=>'Sun Flower'),  
); ?>
```

គេទទួលបាន៖

```
$arr['fruit']['red'] = 'Apple';  
$arr['fruit']['yellow'] = 'Banana';  
$arr['flower']['red'] = 'Rose';  
$arr['flower']['yellow'] = 'Sun Flower';
```

VI. Working with Dates and Times

ក្រៅពីអនុញ្ញាតឱ្យអ្នករាល់គ្នាអាចដាក់តម្លៃផ្សេងៗក្នុង Array បានហើយ PHP ក៏នៅមាន Functions ពេញលេញមួយចំនួនដែលធ្វើការជាមួយនឹងតម្លៃរបស់ Date and Time ផងដែរ។ បើទូសបីជា ភាសា PHP មិនបានបញ្ជាក់ពី Date type ចំពោះតម្លៃរបស់ Date and Time ក៏ដោយ ក៏ភាសាមួយនេះ បានអនុញ្ញាតឱ្យអ្នកសរសេរកម្មវិធី មានភាពបត់បែនក្នុងការដោះស្រាយបញ្ហាទាំងអស់នេះ នៅពេលដែលចង់បង្កើត និងរៀបចំវា។

លើសពីនេះទៅទៀត PHP បានតាងតម្លៃនៃ Date/time ទៅជា UNIX timestamp format ។

- + What is the UNIX time stamp?

Unix time stamp គឺជាមធ្យោបាយមួយ ដែលសម្រាប់តាមដានពេលវេលាសរុបដែលគិតជាវិនាទី។ ការដែលគេអនុវត្តដូច្នេះ គឺគេបានធ្វើចាប់តាំងពីថ្ងៃទី១ ខែមករា ក្នុងឆ្នាំ១៩៧០ តាមបទដ្ឋានស្តង់ដាមួយ ដែលមានឈ្មោះថា UTC (Coordinated Universal Time)។

VI. Working with Dates and Times

- ❖ បំលែងតម្លៃ Date/time ឱ្យទៅជា Unix timestamp គេត្រូវប្រើ mktime() ។

syntax: `mktime(hour, minute, second, month, day, year, is_dst)`

EX: `<?php // return timestamp for Jan 5 2008 10:15`

`// output: 1199508300`

`echo mktime(10,15,00,1,5,2008);`

`?>`

- ❖ ដើម្បីទទួលបានកាលបរិច្ឆេទ(Date) មួយ គេប្រើ function មួយឈ្មោះថា `getdate()`។ គេដឹងហើយថា `getdate()` បាន return value ជាតម្លៃ Array។

ex: `Array ([seconds] => 33, [minutes] => 27, [hours] => 19, [mday] => 12, [wday] => 1, [mon] => 11,`

`[year] => 2021, [yday] => 315, [weekday] => Monday, [month] => November, [0] => 1194875853);`

`<?php // get current date and time as array`

`$now = getdate();`

`// output: 'It is now 19:26:23 on 12-11-2021'`

`echo 'It is now ' . $now['hours'] . ':' . $now['minutes'] . ':' . $now['seconds'] . ' on '`

`. $now['mday'] . '-' . $now['mon'] . '-' . $now['year'];`

`?>`

VI. Working with Dates and Times

❖ បំប្លែងតម្លៃ Date/time ឱ្យទៅជា Unix timestamp គេត្រូវប្រើ mktime() ។

syntax: mktime(hour, minute, second, month, day, year, is_dst)

Chapter V

Lesson 5: Functions and Classes

I. Functions

+ How many types of function?

1. Built-in Function

2. User Define Function

- Without Parameter (Static)
 - a. Return Value (later calculation)
 - b. None Return Value (finish at that time)
- With Parameters (Dynamic)
 - a. Return Value (later calculation)
 - b. Non Return Value (finish at that time)

Without Parameter (Static)

- a. Return Value (later calculation)

ឧទាហរណ៍:

```
<?php
function total_s(){
    $a=12;$b=36;
    $t=$a+$b;
    return $t;
}
echo total_s();
?>
```

- b. Non Return Value (finish at that time)

ឧទាហរណ៍:

```
<?php
function writeMsg() {
    echo "Hello world!";
}
writeMsg();
?>
```

With Parameter (Dynamic)

- a. Return Value (later calculation)

ឧទាហរណ៍:

```
<?php
function total_s(int $a, int $b):int {
    $t=$a+$b;
    return $t;
}
$a=13;
$b=35;
echo total_s($a,$b);
?>
```

- b. None Return Value (finish at that time)

ឧទាហរណ៍:

```
<?php
function writeMsg(string $msg_text) {
    echo $msg_text;
}
$msg_text="ស្វាគមន៍ការមកដល់ NIEI";
writeMsg($msg_text);
?>
```

Lesson 5: Functions and Classes

I. Functions

+ Function Structure

```
function function_name(arg1, ...){  
    code to be executed;  
}
```

+ Dynamic Argument Lists

```
function calcAverage() {  
    $args = func_get_args();  
    $count = func_num_args();  
    $sum = array_sum($args);  
    $avg = $sum / $count;  
    return $avg;  
}
```

Lesson 5: Functions and Classes

I. Functions

+ Setting Default Argument Values

```
function buildAddress($username, $domain = 'mydomain.info') {  
    return $username . '@' . $domain;  
}
```

+ Variable Scope

- Local Variable (any variables inside the function);
- Global Variable
- Constant Variable

- by using define function

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

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

- must be in uppercase letter (needing don't confuse!) or not.
- cannot reassign
- without dollar sign

- Reference Variable

Lesson 5: Functions and Classes

I. Functions

+ Recursive Functions

PHP supports recursion, or in simple words, we can call a PHP function within another function without any argument, just like in C++.

ex:

```
<?php

function NaturalNumbers($number) {
    if($number<=10){
        echo "$number <br/>";
        NaturalNumbers($number+1);
    }
}

NaturalNumbers(1);

?>
```


Lesson 5: Functions and Classes

II. Classes

មិនត្រឹមតែអាចអោយអ្នករាល់គ្នាបង្កើត **Functions** ដោយខ្លួនឯងបានទេ **PHP** ក៏អាចបញ្ចូល **Functions** ជាច្រើនដែលមានទំនាក់ទំនងគ្នាអោយទៅជាក្រុមមួយ ដែលគេអោយឈ្មោះថា **Class**. **Classes** ត្រូវបានជាចាត់ទុកថា ជាការសាងសង់មូលដ្ឋានគ្រឹះមួយដែលស្ថិតនៅពីក្រោយ **OOP (Object-oriented programming)** ព្រោះ **OOP** ជាគំរូនៃការសរសេរកម្មវិធីមួយ ដែលមានទំនាក់ទំនងនឹង **Object** ហើយការដែលប្រើប្រាស់ **Objects** ទាំងអស់នោះ អាចជាមូលដ្ឋានដើម្បីឈានទៅរកការបង្កើតកម្មវិធីមួយ។

+ Classes and Objects

បើគិតពី **Class** មួយ វាដូចទៅនឹង **ecosystem** អញ្ចឹង ព្រោះតែ **Class** វាមានភាពឯករាជ្យក្នុងការប្រមូល **Variables** ហើយនិង **Functions** ដើម្បីអោយធ្វើការរួមគ្នា ក្នុងការបង្កើតការងារចាំបាច់មួយ ឬក៏ច្រើន។ **Variables** នៅក្នុង **Class** ត្រូវបានដាក់ឈ្មោះថា **Properties** រីឯ **Functions** វិញ ត្រូវគេហៅថា **Methods**. **Classes** វាដើរតួដូចជាពុម្ពគំរូសម្រាប់ **Objects** អញ្ចឹង។ ម្យ៉ាងទៀតរាល់ **Object** តែងតែមាន **properties** និង **Functions** ដើម្បីឆ្លើយតបនឹងគំរូនៃ **Class** ។ មួយវិញទៀត **Objects** ទាំងអស់គឺឯករាជ្យទាំងស្រុង ជាមួយនឹង **Properties** និង **Functions** ផ្ទាល់ខ្លួនរបស់ពួកវា។ ដូច្នេះហើយបើទូលក្នុង **Class** ដូចគ្នាមែន ក៏ត្រូវរៀបចំ **Objects** ទាំងអស់នោះ អោយមានភាពឯករាជ្យដែរ។

Lesson 5: Functions and Classes

II. Classes

+ Defining and Using Classes

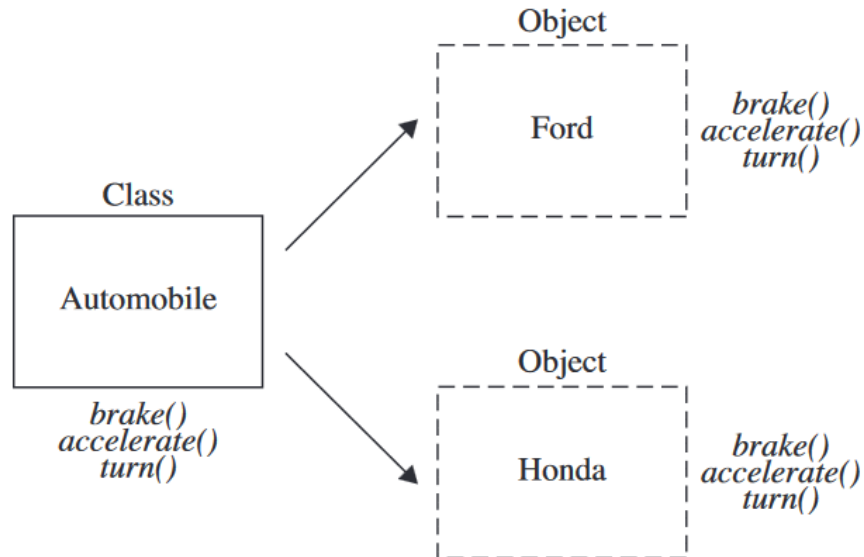
In PHP, classes are defined much like functions: a class definition begins with the class keyword, which is followed by the class name and a pair of curly braces. The complete class definition must be enclosed within these braces; in most cases, this definition consists of property (variable) definitions followed by method (function) definitions.

Class Structure

```
class class_name{  
    // properties  
    __constructor()  
    __destructor()  
    // methods  
}
```

Lesson 5: Functions and Classes

II. Classes



```
<?php
// class definition
class Automobile {
    // properties
    public $color;
    public $make;
    // methods
    public function accelerate() {
        echo 'Accelerating...';
    }
    public function brake() {
        echo 'Slowing down...';
    }
    public function turn() {
        echo 'Turning...';
    }
}

?>
```

Lesson 5: Functions and Classes

II. Classes

+ Defining and Using Classes

Once a class has been defined, objects can be created from the class with the new keyword. Class methods and properties can directly be accessed through this object instance. Here's an example, which creates an instance of the Automobile class and assigns it to \$car, and then sets the object's properties and invokes object methods (note the -> symbol used to connect objects to their properties or methods):

```
<?php
    // instantiate object
    $car = new Automobile;
    // set object properties
    $car->color = 'red';
    $car->make = 'Ford Taurus';
    // invoke object methods
    $car->accelerate();
    $car->turn();
?>
```

```
<?php
    // class definition
    class Automobile {
        // properties
        public $color;
        public $make;
        // methods
        public function accelerate() {
            echo 'Accelerating...';
        }
        public function brake() {
            echo 'Slowing down...';
        }
        public function turn() {
            echo 'Turning...';
        }
    }
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Defining and Using Classes

To access or change a class method or property from within the class itself, it's necessary to prefix the corresponding method or property name with `$this`, which refers to "this" class. To see how this works, consider this revision of the preceding example, which sets a class property named `$speed` and then modifies this property from within the `accelerate()` and `brake()` functions:

```
<?php    // class definition
          class Automobile {
              // properties
              public $color; public $make; public $speed = 55;
              // methods
              public function accelerate() {
                  $this->speed += 10;
                  echo 'Accelerating to ' . $this->speed . '...';
              }
              public function brake() {
                  $this->speed -= 10;
                  echo 'Slowing down to ' . $this->speed . '...';
              }
              public function turn() {
                  $this->brake();
                  echo 'Turning...'; $this->accelerate();
              }
          }

?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

PHP's object model also supports many more advanced features, giving developers a great deal of power and flexibility in building OOP-driven applications. This section illustrates six such features:

1. **Using Constructors and Destructors**
2. **Extending Classes**
3. **Adjusting Visibility Settings**
4. **Class Constants**
5. **Abstract Classes**
6. **Interfaces**

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

1. Using Constructors and Destructors

PHP makes it possible to automatically execute code when a new instance of a class is created, using a special class method called a constructor. You can also run code when a class instance ends using a so-called destructor. Constructors and destructors can be implemented by defining functions named `__construct()` and `__destruct()` within the class, and placing object (de)initialization code within them. Here's a simple example illustrating how this works:

```
<?php

// define class
class Machine {
    // constructor
    function __construct() {
        echo "Starting up...\n";
    }
    // destructor
    function __destruct() {
        echo "Shutting down...\n";
    }
}

// create an object
// output: "Starting up..."
$m = new Machine();

// then destroy it
// output: "Shutting down..."
unset($m);

?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

2. Extending Classes

For most developers, extensibility is the most powerful reason for using the OOP paradigm. Put very simply, extensibility implies that a new class can be derived from an existing one, inheriting all the properties and methods of the parent class and adding its own new properties and methods as needed. An inherited class is defined by using the extends keyword.

```
<!DOCTYPE html>
<html>
<body>
    <?php
        class MyClass {
            public function hello() {
                echo "Hello World!";
            }
        }

        class AnotherClass extends MyClass {
        }

        $obj = new AnotherClass();
        $obj->hello();
    ?>
</body>
</html>
```


Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

3. Adjusting Visibility Settings :

In the first part of this chapter, you learned the difference between “local” and “global” scope, and you saw how variables used inside a class are invisible to the main program. So that you should know **Access Modifiers**.

+ **Access Modifier** : Properties and methods can have access modifiers which control where they can be accessed.

There are three access modifiers:

- 1) public - the property or method can be accessed from everywhere. This is default.
- 2) protected - the property or method can be accessed within the class and by classes derived from that class.
- 3) private - the property or method can ONLY be accessed within the class

Ex:

```
<?php
    class Fruit {
        public $name;
        protected $color;
        private $weight;
    }

    $mango = new Fruit();
    $mango->name = 'Mango'; // OK
    $mango->color = 'Yellow'; // ERROR
    $mango->weight = '300'; // ERROR
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

4. **Class Constants**

Constants cannot be changed once it is declared. Class constants can be useful if you need to define some constant data within a class. A class constant is declared inside a class with the **const** keyword. Class constants are case-sensitive. However, it is recommended to name the constants in all uppercase letters. We can access a constant from outside the class by using the class name followed by the scope resolution operator (**::**) followed by the constant name, like here:

```
<?php
class Welcome_Alert {
    const SHOW_MESSAGE = "Welcome to NIEI !!!";
}
echo Welcome_Alert :: SHOW_MESSAGE;
?>
```

Or, we can access a constant from inside the class by using the self keyword followed by the scope resolution operator (**::**) followed by the constant name, like here:

```
<?php
class Sh_Message {
    const SHOW_MESSAGE = "Welcome
for visiting NIEI!";
    public function Welcome_Show() {
        echo self::SHOW_MESSAGE;
    }
}
$mess = new Sh_Message();
$mess->Welcome_Show();
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

5. **Abstract Classes**

Abstract classes and methods are when the parent class has a named method, but need its child class(es) to fill out the tasks. An abstract class is a class that contains at least one abstract method. An abstract method is a method that is declared, but not implemented in the code. An abstract class or method is defined with the **abstract** keyword:

Syntax:

```
<?php
    abstract class ParentClass {
        abstract public function someMethod1();
        abstract public function someMethod2($name,$color);
    }
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

5. Abstract Classes

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method.
- The child class method must be defined with the same or a less restricted access modifier.
- The number of required arguments must be the same. However, the child class may have optional arguments in addition

Ex: កំណត់ទី១

```
<?php
// Parent class
abstract class Car {
    public $name;
    public function __construct($name) {
        $this->name = $name;
    }
    abstract protected function intro() : string;
}
// Child classes
class Audi extends Car {
    protected function intro() : string {
        return "Choose German quality! I'm an $this->name!";
    }
}
// Create objects from the child classes
$audi = new audi("Audi");
echo $audi->intro();
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

5. **Abstract Classes**

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method.
- The child class method must be defined with the same or a less restricted access modifier.
- The number of required arguments must be the same. However, the child class may have optional arguments in addition

Ex: ករណីទី២

```
<?php
abstract class ParentClass {
    // Abstract method with an argument
    abstract protected function prefixName($name);
}

class ChildClass extends ParentClass {
    public function prefixName($name) {
        if ($name == "John Doe") {
            $prefix = "Mr.";
        } elseif ($name == "Jane Doe") {
            $prefix = "Mrs.";
        } else {
            $prefix = "";
        }
        return "{$prefix} {$name}";
    }
}

$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

5. Abstract Classes

When inheriting from an abstract class, the child class method must be defined with the same name, and the same or a less restricted access modifier. So, if the abstract method is defined as protected, the child class method must be defined as either protected or public, but not private. Also, the type and number of required arguments must be the same. However, the child classes may have optional arguments in addition.

So, when a child class is inherited from an abstract class, we have the following rules:

- The child class method must be defined with the same name and it redeclares the parent abstract method.
- The child class method must be defined with the same or a less restricted access modifier.
- The number of required arguments must be the same. However, the child class may have optional arguments in addition

Ex: ករណីទី៣

```
<?php
abstract class ParentClass {
    // Abstract method with an argument
    abstract protected function prefixName($name);
}
class ChildClass extends ParentClass {
    // The child class may define optional arguments that is not in the parent's abstract method
    public function prefixName($name, $separator = ".", $greet = "Dear") {
        if ($name == "John Doe") {
            $prefix = "Mr";
        } elseif ($name == "Jane Doe") {
            $prefix = "Mrs";
        } else {
            $prefix = "";
        }
        return "{$greet} {$prefix}{$separator} {$name}";
    }
}
$class = new ChildClass;
echo $class->prefixName("John Doe");
echo "<br>";
echo $class->prefixName("Jane Doe");
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

6. Interfaces

Interfaces allow you to specify what methods a class should implement. Interfaces make it easy to use a variety of different classes in the same way. When one or more classes use the same interface, it is referred to as "polymorphism". Interfaces are declared with the interface keyword:

Syntax:

```
<?php
    interface InterfaceName {
        public function someMethod1();
        public function someMethod2($name, $color);
        public function someMethod3() : string;
    }
?>
```

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

6. Interfaces

❖ Interfaces vs. Abstract Classes

Interface are similar to abstract classes. The difference between interfaces and abstract classes are:

- Interfaces cannot have properties, while abstract classes can.
- All interface methods must be public, while abstract class methods is public or protected.
- All methods in an interface are abstract, so they cannot be implemented in code and the abstract keyword is not necessary.
- Classes can implement an interface while inheriting from another class at the same time.

Lesson 5: Functions and Classes

II. Classes

+ Using Advanced OOP Concepts

6. Interfaces

❖ Using Interfaces

To implement an interface, a class must use the implements keyword. A class that implements an interface must implement all of the interface's methods.

Ex:

```
<?php
    interface Animal {
        public function makeSound();
    }
    class Cat implements Animal {
        public function makeSound() {
            echo "Meow";
        }
    }
    $animal = new Cat();
    $animal->makeSound();
?>
```

Chapter VI

Lesson 6: Working with Databases and SQL

I. *Introducing Databases and SQL*

នៅក្នុងយុគសម័យអ៊ីនធឺណិត ព័ត៌មានមួយចំនួនធំ មិនត្រូវបានគេ ដាក់បង្ហាញនៅក្នុងទូរទស្សន៍ដាច់ដោយឡែកទេ ដោយជំនួសមកវិញ គឺត្រូវដាក់ចែករៀបចំជាលក្ខណៈ ឌីជីថល គឺលេខម៉ាស៊ីន ១ និង 0 អោយស្ថិតក្នុង electronic database ដែលជាកន្លែងផ្ទុកទិន្នន័យ ដែលមានលក្ខណៈដូចនឹង កុងតឺន័រ(Containers) អញ្ចឹង។ ហើយ containers ទាំងនោះ មានរចនាសម្ព័ន្ធជាក់លាក់មួយក្នុងការរៀបចំព័ត៌មានអោយមានសណ្តាប់ធ្នាប់ ។ ម្យ៉ាងវិញទៀត ការរក្សាទិន្នន័យក្នុង electronic database ត្រូវបានគេបែងចែកជាមួយនឹង Tools មួយចំនួន ដែលអាចអោយ អ្នកប្រើប្រាស់ អាចបន្សុត ហើយនឹងទាញទិន្នន័យមកប្រើប្រាស់ បានយ៉ាងងាយ តាមរយៈនៃការសរសេរ សំណើ (criteria) ផ្សេងៗ។

គួរកត់សម្គាល់ថា electronic databases សព្វថ្ងៃ គឺជា relational databases ដែលអាចអោយ អ្នកប្រើប្រាស់ បញ្ជាក់ពី relationships រវាង table មួយ ទៅ table មួយទៀត ក្នុង database ដើម្បីតភ្ជាប់គ្នា ងាយស្រួលក្នុងការគ្រប់គ្រងទិន្នន័យ។ មែនហើយ មានប្រព័ន្ធគ្រប់គ្រង មូលដ្ឋានទិន្នន័យ(database management systems) មួយចំនួនដែលអាចប្រើបាន នាពេលបច្ចុប្បន្ននេះ ខ្លះជាពាណិជ្ជកម្ម និងខ្លះឥតគិតថ្លៃ។ អ្នកទាំងអស់គ្នា ប្រហែលជាធ្លាប់បានឮហើយអំពីពួកវា មួយចំនួននោះរួមមាន៖ Oracle, Microsoft Access, MySQL និង PostgreSQL ជាដើម។

ប្រព័ន្ធមូលដ្ឋានទិន្នន័យទាំងនេះមានអនុភាពណាស់ កម្មវិធីដែលសំបូរទៅដោយលក្ខណៈពិសេស មានសមត្ថភាពរៀបចំ និងស្វែងរក records រាប់លានក្នុងល្បឿនលឿនបំផុត។ ជាក់ស្តែង ពួកវាត្រូវបានគេ ប្រើប្រាស់យ៉ាងទូលំទូលាយដោយអាជីវកម្ម និងការិយាល័យរដ្ឋាភិបាល ជាញឹកញាប់សម្រាប់គោលបំណងសំខាន់នៃការគ្រប់គ្រងទិន្នន័យ។ បើក្រឡេកមើលអក្សរកាត់ SQL វិញ គឺមកពីពាក្យ Structured Query Language។ ហើយ SQL បានក្លាយជាស្តង់ដារនៃវិទ្យាស្ថានស្តង់ដារជាតិអាមេរិក (ANSI) ក្នុងឆ្នាំ

Lesson 6: Working with Databases and SQL

I. *Introducing Databases and SQL*

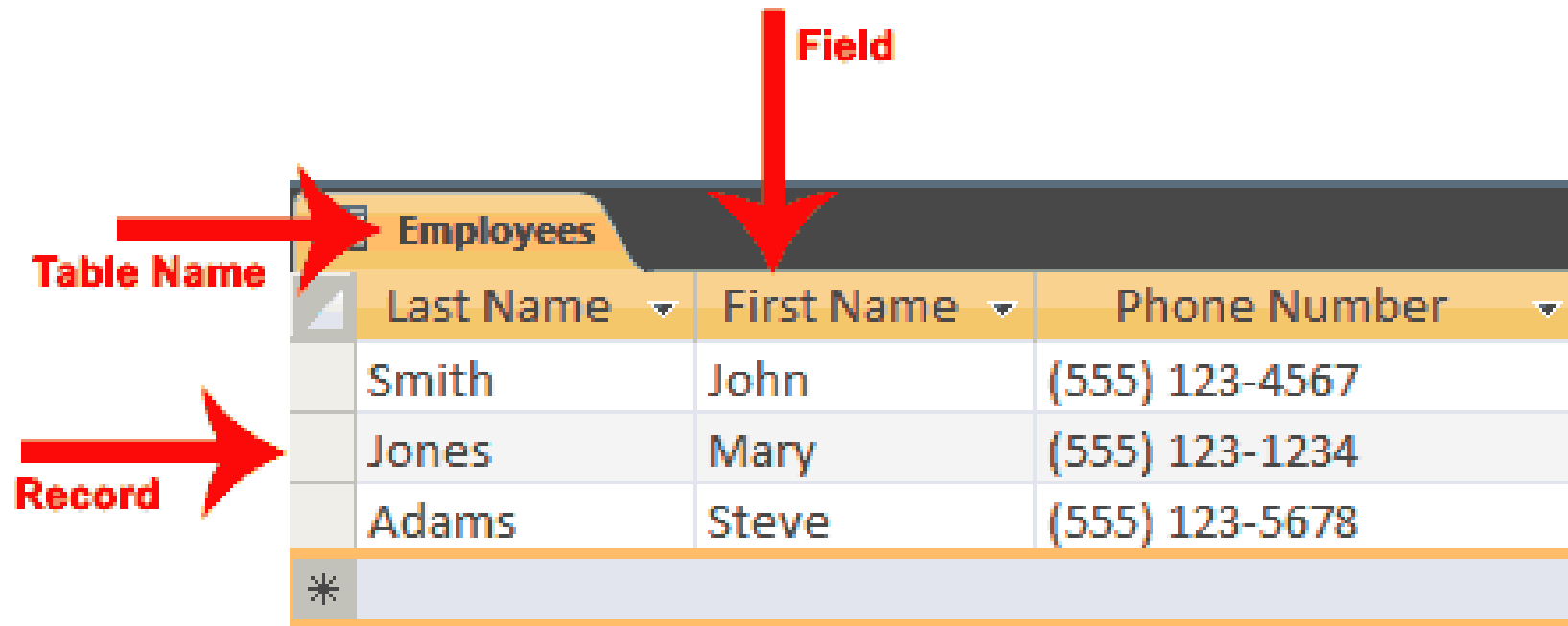
SQL មានតួនាទីអាចធ្វើការបានមួយចំនួនដូចខាងក្រោម៖

1. SQL can retrieve data from a database
2. SQL can insert records in a database
3. SQL can update records in a database
4. SQL can delete records from a database
5. SQL can create new databases
6. SQL can create new tables in a database
7. SQL can create stored procedures in a database
8. SQL can create views in a database
9. SQL can set permissions on tables, procedures, and views

Lesson 6: Working with Databases and SQL

II. Understanding Databases, Records, and Primary Keys

រាល់ database សុទ្ធតែមាន Table មួយ ឬច្រើន។ ដែល Tables ទាំងអស់នោះ បានរៀបចំទិន្នន័យ ដោយបានដាក់ជា Rows ហើយនិង Columns. រាល់ Column នីមួយៗ សុទ្ធមានចំណងជើង Column ដែលគេអោយឈ្មោះថា field ។ មួយវិញទៀត ជួរដេកនៃ tables ត្រូវបានគេអោយឈ្មោះថា Record ។ ម្យ៉ាងវិញទៀត Primary Key គឺជាការកំណត់ ID ជាកំណត់មួយ (Column ID ហាមមានលេខ ជាន់គ្នា) នៃ Record នីមួយៗនៅក្នុង table ។ Table មួយអាចមាន Primary Key តែមួយ។ ហើយនៅក្នុង Table, Primary Key នេះអាចមានជួរឈរតែមួយ។



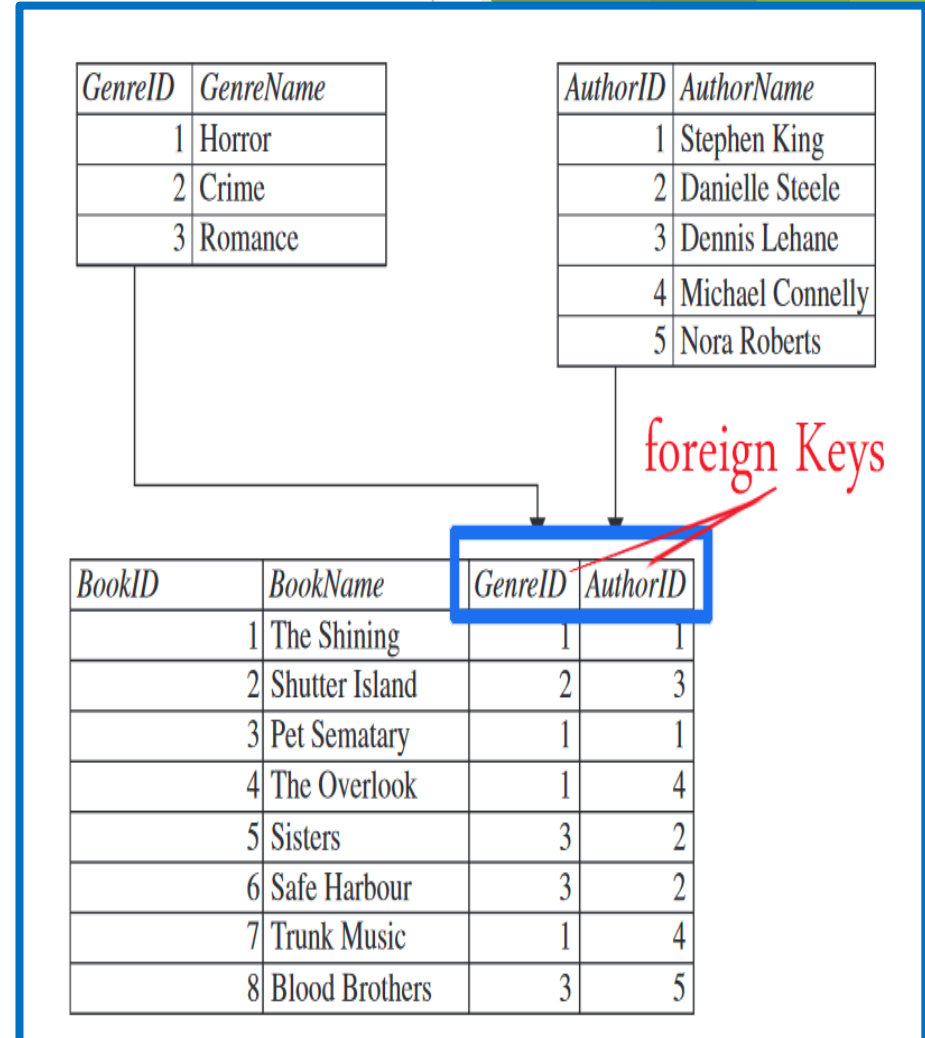
The diagram illustrates a database table structure. A red arrow labeled "Table Name" points to the table name "Employees". A red arrow labeled "Field" points to the column headers "Last Name", "First Name", and "Phone Number". A red arrow labeled "Record" points to the first row of data, which contains the values "Smith", "John", and "(555) 123-4567". The table has a footer row with an asterisk (*) in the first column.

Employees		
Last Name	First Name	Phone Number
Smith	John	(555) 123-4567
Jones	Mary	(555) 123-1234
Adams	Steve	(555) 123-5678
*		

Lesson 6: Working with Databases and SQL

III. Understanding Relationships and Foreign Keys

អ្នកប្រាកដជាដឹងហើយថា database មួយ ផ្ទុកនូវ tables ផ្សេងគ្នាជាច្រើន។ ហើយ Tables នៅក្នុងប្រព័ន្ធ relational database បានតភ្ជាប់គ្នាទៅវិញទៅមកតាមរយៈ fields មួយ ឬច្រើន ដែល fields នោះ ត្រូវបានគេអោយឈ្មោះថា foreign keys។ foreign keys ទាំងអស់នេះហើយ ដែលជាបង្កើត relationships បែប one-to-one, one-to-many or many-to-many ដើម្បីងាយស្រួលក្នុងការប្រមូលផ្តុំព័ត៌មានរួមសំខាន់ៗចេញពី Tables ទាំងអស់នោះ។



Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

Structured Query Language(SQL) គឺជាភាសាកូដស្តង់ដា មួយ ដែលមានទំនាក់ទំនងជាមួយ database ក្នុងការបញ្ចូល លុប កែ ទិន្នន័យ និងផ្សេងៗ មួយចំនួនទៀត។ ម្យ៉ាងវិញទៀត ភាសានេះ ត្រូវបានគេប្រើប្រាស់ក្នុងអាជីវកម្មធំៗ នាពេលបច្ចុប្បន្ននេះ។ មែនហើយ បញ្ហារបស់ SQL វាស្រដៀងនឹង ការនិយាយភាសាអង់គ្លេសអញ្ចឹង ហេតុនេះហើយ ទើប SQL ពិតជាងាយស្រួលសិក្សាមែនទែន។ រាល់ការសរសេរឃ្លា របស់កូដ SQL វាតែងតែចាប់ផ្តើមដោយពាក្យសកម្មៗដូចជា៖ DELETE, INSERT, ALTER or DESCRIBE និងបញ្ចប់ ឃ្លាវិញ ដោយសញ្ញា " ; " ។

ឧទាហរណ៍៖

- CREATE DATABASE library;
- SELECT movie FROM movies WHERE rating > 4;
- DELETE FROM cars WHERE year_of_manufacture < 1980;

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

Some of The Most Important SQL Commands

1. **SELECT** - extracts data from a database
2. **UPDATE** - updates data in a database
3. **DELETE** - deletes data from a database
4. **INSERT INTO** - inserts new data into a database
5. **CREATE DATABASE** - creates a new database
6. **ALTER DATABASE** - modifies a database
7. **CREATE TABLE** - creates a new table
8. **ALTER TABLE** - modifies a table
9. **DROP TABLE** - deletes a table
10. **CREATE INDEX** - creates an index (search key)
11. **DROP INDEX** - deletes an index

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1. The SQL SELECT Statement

The SELECT statement is used to select data from a database.

syntax: `SELECT column1, column2, ...FROM table_name;`

Here, column1, column2, ... are the field names of the table you want to select data from. *ex1: SELECT CustomerName, City FROM Customers;*

If you want to select all the fields available in the table, use the following syntax:

syntax: `SELECT * FROM table_name;`

*ex2: SELECT * FROM Customers;*

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.1. The SQL SELECT DISTINCT Statement

The SELECT DISTINCT statement is used to return only distinct (different) values. Inside a table, a column often contains many duplicate values; and sometimes you only want to list the different (distinct) values.

syntax: `SELECT DISTINCT column1, column2, ...FROM table_name;`

ex: `SELECT DISTINCT Country FROM Customers;`

1.2. SQL WHERE Clause

The WHERE clause is used to filter records. It is used to extract only those records that fulfill a specified condition.

syntax: `SELECT column1, column2, ...FROM table_name WHERE condition;`

ex1: `SELECT * FROM Customers WHERE Country='Mexico';`

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.2. SQL WHERE Clause

ex2: `SELECT * FROM Customers WHERE CustomerID=1;`

Operators in The WHERE Clause

The following operators can be used in the WHERE clause:

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal. Note: In some versions of SQL this operator may be written as !=
BETWEEN	Between a certain range
LIKE	Search for a pattern
IN	To specify multiple possible values for a column

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

1.2. SQL WHERE Clause

1.2.1. AND, OR and NOT

- ❖ The AND and OR operators are used to filter records based on more than one condition:
 - The AND operator displays a record if all the conditions separated by AND are TRUE.
 - The OR operator displays a record if any of the conditions separated by OR is TRUE.
- ❖ The NOT operator displays a record if the condition(s) is NOT TRUE.

Syntax: **AND**

```
SELECT column1, column2, ...FROM table_name WHERE condition1 AND condition2 AND  
condition3 ...;
```

Syntax: **OR**

```
SELECT column1, column2, ...FROM table_name WHERE condition1 OR condition2 OR  
condition3 ...;
```

Syntax: **NOT**

```
SELECT column1, column2, ...FROM table_name WHERE NOT condition;
```

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

1.2. SQL WHERE Clause

1.2.2. NULL Values

A field with a NULL value is a field with no value. If a field in a table is optional, it is possible to insert a new record or update a record without adding a value to this field. Then, the field will be saved with a NULL value. We will have to use the **IS NULL** and **IS NOT NULL** operators.

IS NULL Syntax:

```
SELECT column_names FROM table_name WHERE column_name IS NULL;
```

IS NOT NULL Syntax:

```
SELECT column_names FROM table_name WHERE column_name IS NOT NULL;
```

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

1.2. SQL WHERE Clause

1.2.3. COUNT()

The COUNT() function returns the number of rows that matches a specified criterion.

COUNT() Syntax:

```
SELECT COUNT(column_name) FROM table_name WHERE condition;
```

1.2.4. AVG()

The AVG() function returns the average value of a numeric column.

AVG() Syntax:

```
SELECT AVG(column_name) FROM table_name WHERE condition;
```

1.2.5. SUM()

The SUM() function returns the total sum of a numeric column.

SUM() Syntax:

```
SELECT SUM(column_name) FROM table_name WHERE condition;
```

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.2. SQL WHERE Clause

1.2.6. LIKE Operator

The LIKE operator is used in a WHERE clause to search for a specified pattern in a column. There are two wildcards often used in conjunction with the LIKE operator:

- The percent sign (%) represents zero, one, or multiple characters.
- The underscore sign (_) represents one, single character.

Note: MS Access uses an asterisk (*) instead of the percent sign (%), and a question mark (?) instead of the underscore (_).

LIKE Syntax:

```
SELECT column1, column2, ...FROM table_name WHERE columnN LIKE pattern;
```

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.2. SQL WHERE Clause

1.2.6. LIKE Operator

Here are some examples showing different LIKE operators with '%' and '_' wildcards:

LIKE Operator	Description
WHERE CustomerName LIKE 'a%'	Finds any values that start with "a"
WHERE CustomerName LIKE '%a'	Finds any values that end with "a"
WHERE CustomerName LIKE '%or%'	Finds any values that have "or" in any position
WHERE CustomerName LIKE '_r%'	Finds any values that have "r" in the second position
WHERE CustomerName LIKE 'a_%'	Finds any values that start with "a" and are at least 2 characters in length
WHERE CustomerName LIKE 'a__%'	Finds any values that start with "a" and are at least 3 characters in length
WHERE ContactName LIKE 'a%o'	Finds any values that start with "a" and ends with "o"

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.2. SQL WHERE Clause

1.2.7. Wildcard Characters

A wildcard character is used to substitute one or more characters in a string. Wildcard characters are used with the LIKE operator. The LIKE operator is used in a WHERE clause to search for a specified pattern in a column.

Wildcard Characters in MS Access:

Symbol	Description	Example
*	Represents zero or more characters	bl* finds bl, black, blue, and blob
?	Represents a single character	h?t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
!	Represents any character not in the brackets	h[!oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
#	Represents any single numeric character	2#5 finds 205, 215, 225, 235, 245, 255, 265, 275, 285, and 295

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

1.2. SQL WHERE Clause

1.2.7. Wildcard Characters

Wildcard Characters in SQL Server

Symbol	Description	Example
%	Represents zero or more characters	bl% finds bl, black, blue, and blob
_	Represents a single character	h_t finds hot, hat, and hit
[]	Represents any single character within the brackets	h[oa]t finds hot and hat, but not hit
^	Represents any character not in the brackets	h[^oa]t finds hit, but not hot and hat
-	Represents any single character within the specified range	c[a-b]t finds cat and cbt
%	Represents zero or more characters	bl% finds bl, black, blue, and blob

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

1.2. SQL WHERE Clause

1.2.8. IN Operator

The IN operator allows you to specify multiple values in a WHERE clause.

IN Syntax:

```
SELECT column_name(s) FROM table_name WHERE column_name IN (value1, value2, ...);
```

or

```
SELECT column_name(s) FROM table_name WHERE column_name IN (SELECT STATEMENT);
```

1.2.9. BETWEEN Operator

The BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.

BETWEEN Syntax:

```
SELECT column_name(s) FROM table_name WHERE column_name BETWEEN value1 AND value2;
```

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.3. ORDER BY Keyword

The ORDER BY keyword is used to sort the result-set in ascending or descending order. The ORDER BY keyword sorts the records in ascending order(ASC) by default. To sort the records in descending order, use the DESC keyword.

Syntax:

```
SELECT column1, column2, ...FROM table_name ORDER BY column1, column2, ... ASC | DESC;
```

1.4. INSERT INTO Statement

The INSERT INTO statement is used to insert new records in a table.

INSERT INTO Syntax:

It is possible to write the INSERT INTO statement in two ways:

- a. Specify both the column names and the values to be inserted:

```
INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3,...);
```

- b. If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query. However, make sure the order of the values is in the same order as the columns in the table. Here, the INSERT INTO syntax would be as follows:

```
INSERT INTO table_name VALUES (value1, value2, value3, ...);
```

Lesson 6: Working with Databases and SQL

IV. *Understanding SQL Statements*

1.5. UPDATE Statement

The UPDATE statement is used to modify the existing records in a table.

Syntax:

```
UPDATE table_name SET column1 = value1, column2 = value2, ... WHERE condition;
```

1.6. DELETE Statement

The DELETE statement is used to delete existing records in a table.

Syntax:

```
DELETE FROM table_name WHERE condition;
```

Notice: Delete All Records

```
DELETE FROM table_name;
```

Lesson 6: Working with Databases and SQL

IV. Understanding SQL Statements

1.7. SELECT TOP Clause

The SELECT TOP clause is used to specify the number of records to return. The SELECT TOP clause is useful on large tables with thousands of records. Returning a large number of records can impact performance.

Syntax:

```
SELECT TOP number|percent column_name(s) FROM table_name WHERE condition;
```

1.8. MIN() and MAX() Functions

The MIN() function returns the smallest value of the selected column. The MAX() function returns the largest value of the selected column.

MIN Syntax:

```
SELECT MIN(column_name) FROM table_name WHERE condition;
```

MAX Syntax:

```
SELECT MAX(column_name) FROM table_name WHERE condition;
```