**Introduction:**

**PHP OOP**

**Fundamentals & syntax**

PHP can be used both procedurally, and as an object-oriented programming language.

OOP beings with the definition of a class, which is a template for a particular type of data: an employee, a user, a page of content etc… A class definition contains both variables and functions (properties and methods in class definition). Combined, the properties and methods are the members of the class.

Note that class names conventionally begin with an uppercase letter.

What actually takes a lot of effort, at lease to do it right, is to master the design process: understanding what members to define and, more importantly, how to implement sophisticated OOP concepts such as: inheritance, access control, overriding methods, scope resolution, abstraction and so on…

\*PHP Advanced: Visual quick pro guide goes more in depth with PHP as an OOP language\*

OOP syntax in PHP

Most classes are not used directly; rather, you create an instance of that class – a specific variable of the class’s type. That instance is called an object.

$obj = new ClassName();

The new keyword is used to create an object.

Whereas the code $name = ‘Ali’; creates a variable of type string, $mine = new Car(); creates a variable of type Car. Each property and method are now embedded in $mine.

Behind the scenes (i.e. in the class definition), a special method called the constructor is automatically invoked when a new object of that type is generated. If the constructor takes arguments, like any function can, those may be provided when the object is created:

$mine = new Car(“BMW”, “3 series”, 2015);

Once you have an object, you reference its properties and methods using the syntax:

$obj.name -> member\_name;

$mine->color = “white”;

$mine->start(); //if the method takes arguments, those can be provided too

**Working with MySQL**

3 pre-defined classes:

1. MySQLi – primary class, provides a database connection, a querying method & more
2. MySQLi\_Result – Result, used to handle results of SELECT queries (among others)
3. MySQLi\_STMT – is for performing prepared statements

Creating a connection

$mysqli = new MySQLi(HOSTNAME, USERNAME, PASSWORD, DB);

If the connection could not be made: if($mysqli ->connect.error){ echo $mysqli->connect\_error; }

To establish the character set: $mysqli->set\_charset(‘utf8’);

//At this point you’re ready to execute your queries

//After executing the queries, call the close() method: $mysqli->close();

//To be extra tidy, you can delete the object too: unset($mysqli);

You can use print\_r() to learn about, & debug objects in PHP.

Executing simple queries

Once you have successfully established a connection to the MySQLserver, you can being using the MySQLi object to query the database, For that, call the query() method:

$q = “SELECT \* FROM tablename”;

$mysqli->query($q);

You can test if executed correctly using:

If($mysqli->query($q)){ //worked }

OR

If($mysqli->error){ //did not work echo $mysqli->error; }

If the query just executed was an INSERT, you can retrieve the automatically generated primary key value via the insert\_id property:

$id = mysqli->insert\_id;

If the query just executed was an UPDATE, INSERT or DELETE, you can retrieve the number of affected rows from the affected\_rows property:

Echo “$mysqli->affected\_rows rows were affected by the query”;

To sanctify data used in a query, apply the real\_escape\_method() to a string variable beforehand:

$var = $mysqli->real\_escape\_string($var);

EXAMPLE (p502-504):

If(empty($\_POST[‘first\_name’]){

$errors[] = “You forgot to enter your first name”;

}else{

$fn = $mysqli->real\_escape\_string(trim($\_POST[‘first\_name’]));

}

If(empty($errors)) {

$q = “INSERT INTO users(first\_name) VALUES(‘$fn’)”;

$mysqli->query($q);

If($mysqli->affected\_rows == 1){

Echo “Thank you for registering”;

}else{

Echo $mysqli->error;

}

Mysqli->close();

Unset($mysqli);

}else{ //report the errors }

//close database connection

Fetching results

When executing SELECT queries, the code is a bit different, as you have to handle the query’s results.

$q = “SELECT \* FROM tablename”;

$result = $mysqli->query($q);

If($result->num\_rows>0) {

//handle the results

//if you only have one row returne

$row = $result->fetch\_array();

//if multiple records

While($row->fetch\_array()){

//use $row

}

//free the resource

$result->free();

You can also fetch multiple records using OOP syntax using fetch\_object():

$q = “SELECT \* FROM users”;

$r = $mysqli->query($q);

While($row->fetch\_object()){

//use $row->user\_id

//use $row->first\_name

}

EXAMPLE (p506-507):

$q = “SELECT CONCAT(last\_name, ‘, ‘, first\_name) AS name FROM users ORDER BY name ASC”;

$r = $mysqli->query($q); //run the query

$num = $r->num\_rows;

If($num > 0){

Echo “There are currently $num registered users”;

While($row = $r->fetch\_object()){

Echo $row->name;

}

$r->free(); //free up the resources

Unset($r);

}else{ //if no record returned

Echo “There are no registered users”;

}

$mysqli->close();

Unset($mysqli);

**The DateTime class**

$dt = new DateTime(); //To create new DateTime object

If this is created without providing any arguments to the constructor, the generated DateTime argument will represent the current date & time.

$dt->setDate(2001,4,20); //year, month, day (in that order)

$dt->setTime(11,15); //hours, min, secs (in that order)

Once you have a DateTime object, you can manipulate its value by adding and subtracting time periods. One way to do so if the modify() method:

$dt->modify(‘+1 day’);

$dt->modify(‘-1 month’);

$dt->modify(‘next Thursday’);

The add() method is used to add a time period to the represented date & time. It takes as its lone argument an object of DateInterval:

$di = new DateInterval(*interval*);

$dt->add($di);

Theres a specific notation used to set the interval, always starting with the letter P, for *period*. After that, add an integer & a period designator:

Y – years

M – months

D – days

W – weeks

H – hours

M – minutes

S – seconds

The time period designators are preceded by a T, for *time*.

\*These characters should be combined in order\*

P3W = 3 weeks

P2Y3M = 2 years & 3 months

P2M3DT4H18M43S = 2 months, 3 days, 4 hours, 18 minutes & 43 seconds

The sub() method functions just the same as add(), but subtracts the time period from the object:

$di = new DateInterval(‘P2W’);

$dt->sub($di);

The diff() method returns a DateInterval object that reflects the amount of time between 2 DateTime objects:

$diff = $dt->diff($dt2);

Format() returns the represented date format as you want it.

EXAMPLE (p513-517):

//set the start and end date as today and tomorrow

$start = new DateTime();

$end = new DateTime();

$end->modify(‘+1 day’);

//default format for displaying dates

$format = ‘m/d/Y’;

//this function validates a provided date string. The function returns an array—month, day, year—if valid

Function validate\_date($date) {

//break up string in to its parts

$date\_array = explode(‘/’,$date);

//return false if there aren’t 3 items

If(count($date\_array) !=3) return false;

//return false if its not a valid date: checkdate() checks the date provided actually exists

If(!checkdate($date\_array[0], $date\_array[1], $date\_array[2])) return false;

//return the array

Return $date\_array;

} //end of validate\_date() function

//check for form submission

If(isset($\_POST[‘start’], $\_POST[‘end’])) {

//call the validation function on both dates:

If( (list($sm, $sd, $sy) = validate\_date($\_POST[‘start’])) && (list($em, $ed, $ey) = validate\_date($\_POST[‘end’]) ) {

//if its ok, adjust the DateTime objects

$start->setDate($sy, $sm, $sd);

$end->setDate($ey, $em, $ed);

//the start date must come first:

If($start < $end){

//determine the interval

$interval = $start->diff($end);

//print the results

Echo “The event has been planned starting on {$start- >format($format)} and ending on {$end->format($format)}, which is a period of $interval->days day(s).”;

}else{ //end date must be later!

Echo “The starting date must precede the ending date”;

}

}else{ //an invalid date

Echo “One or both of the submitted dates was invalid”;

}

}//end of form submission

//show the form

<form action=”datetime.php” method=”post”>

Start Date: <input type=”text” name=”start\_date” value=” <?php echo $start- >format($format); ?>” /> (MM/DD/YYYY) <br/>

End Date: <input type=”text” name=”start\_date” value=” <?php echo $end- >format($format); ?>” /> (MM/DD/YYYY) <br/>

<input type=”submit” value=”Submit”/>

</form>

**TEST**

**Intro**

1. What is meant by *members* of the class?
2. How do you create an object / AKA an instance of a class?
3. What syntax is used to reference members of an object?

**SQL**

1. What are the 3 pre-defined classes associated with SQL?
2. Create a connection to a database using OOP syntax – include code if connection failed
3. How do you set character set?
4. Use print\_r on an object!
5. Give example of executing an UPDATE, INSERT and DELETE query
6. Give example of executing a SELECT query, and print records on the page

**DateTime**

1. How do you create a new DateTime object (with and without parameters)
2. Give an example of: modify(), add() & subtract()
3. Follow through the example until understood