1. Write a PHP program to do the multiplication of two matrices.

```
<?php
$a = array(array(1,2,3),array(4,5,6),array(7,8,9));
$b = array(array(7,8,9),array(4,5,6),array(1,2,3));
$m=count($a);
$n=count($a[2]);
$p=count($b);
$q=count($b[2]);
echo "the first matrix :"."<br/>";
for ($row = 0; $row < $m; $row++) {
for ($col = 0; $col < $n; $col++)
echo " ".$a[$row][$col];
echo "<br/>";
}
echo "the second matrix :"."<br/>";
for ($row = 0; $row < $p; $row++) {
for ($col = 0; $col < $q; $col++)
echo " ".$b[$row][$col];
echo "<br/>";

}
echo "the transpose for the first matrix is:"."<br/>"; for ($row = 0; $row < $m; $row++) {
for ($col = 0; $col < $n; $col++)
echo " ".$a[$col][$row];
echo "<br/>";
}
if(($m===$p) and ($n===$q)) {
echo "the addition of matrices is:"."<br/>";
for ($row = 0; $row < 3; $row++) {
for ($col = 0; $col < 3; $col++)
echo " ".$a[$row][$col]+$b[$row][$col]." "; echo "<br/>";
}
}
if($n===$p){
echo " The multiplication of matrices: <br/>";
$result=array();
for ($i=0; $i< $m; $i++) {
for($j=0; $j < $q; $j++){
$result[$i][$j] = 0;
for($k=0; $k < $n; $k++)
$result[$i][$j] += $a[$i][$k] * $b[$k][$j];
}
}
for ($row = 0; $row < $m; $row++) {
for ($col = 0; $col < $q; $col++)
echo " ".$result[$row][$col];
echo "<br/>";
}
}
?>
```

2. Explain the following object-oriented principles in detail:

(a) Polymorphism (b) Interface

Polymorphism

Polymorphism is the OO concept, where the object can act differently at different times. Even ifthe same function name is used.

Overriding of methods in two or more derived classes takes place, the actual method called will depend on the type of the object. In the below example program, the display ( ) method is defined in both the derived classes. The method accessed depends on the object used to access the function.

```
<?php

Class Student
{
        private $name;
        public $USN;
        public $address;
      public $avg;

function __construct($n,$usn,$add,$avg)
{
      $this->name = $n;
      $this->USN= $usn;
      $this->address = $add;
      $this->avg = $avg;
}
protected function display()
{
   echo $this->name;
}
}

class FeeDetails extends Student
{
      priva
      te
      $spo
      rts;
      priva
      te
      $aca
      demi
      c;

      function __construct($n,$usn,$add,$avg,$s,$ac)
      {
      parent::__construct($n,$usn,$add,$avg);
```

```
        $this->sports= $s;
        $this->academic= $ac;
        }

        public function total()
        {
                $sum= ($this-
                >sports)+($this->academic);
                echo "total fee is $sum";
        }
        public function display()
        {
                parent::display();
                $this->total();
        }
}

class Scholarship extends Student
{

        function __construct($n,$usn,$add,$avg)
        {
        parent::__construct($n,$usn,$add,$avg);
        }

        public function display()
        {
                par
                ent
                ::di
                spl
                ay(
                );
                ech
                o
                "N
                o
                fee
                s";
        }
}

$f1 = new FeeDetails("Sajeev", "1VA15CS013","Bangalore",66,25000,3000);
$s1=new Scholarship("Amith", "1VA15CS013","Bangalore",66);
$f1->display();
$s1->display();

?>
```

OUTPUT:

Sajeev total fee is 28000 Amith No fees

The same display( ) method is invoked by two different objects. First the display( ) function of FeeDetails class is invoked and then the display( ) function of Scholarship class is invoked.

The run time decision of determining the function to be invoked at run time, is called **dynamic dispatching**. Just as each object can maintain its own properties, each object also manages its own table of methods. This means that two objects of the same type can have different implementations with the same name.

Object Interfaces

An object **interface** is a way of declaring a formal list of methods(only) without specifying their implementation. The class that uses the interface **must** implement the declared functions.

Interfaces provide a mechanism for defining what a class can do without specifying how it does it. Interfaces are defined using the interface keyword. It looks similar to PHP classes, except an interface contains no properties and its methods do not have method bodies defined.

Eg -

```
interface Viewable {
public function
getmessage();
public function
getData();
}
```

Notice that an interface contains only public methods, and instead of having a method body, eachmethod is terminated with a semicolon.

In PHP, a class can be said to *implement* an interface, using the

implements keyword:class FeeDetails extends Student implements

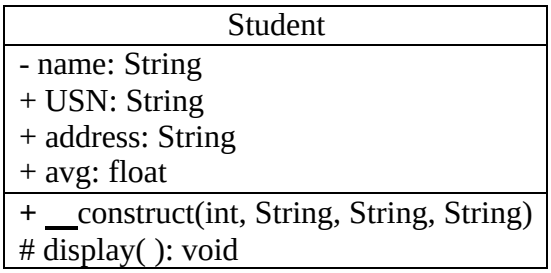
Viewable
{ ... }

This means then that the class ‘FeeDetails’ must provide implementations for the getmessage()and getData() methods.

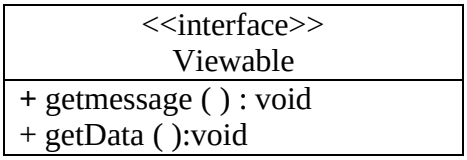
Implementing an interface allows a class to become more formal about the behavior it promises to provide. The same interface can be inherited by more than one class. It helps in the implementation of multiple inheritance.

PHP allows implementing two or more interfaces. The UML diagram to denote the interfaces include the <<interface>> stereotype. Classes that implement an interface are shown to implement using the same hollow triangles as inheritance but with dotted lines.

Base Class



Interface



3. Explain three types of errors and PHP error reporting flags. How are errors different from exceptions?

Types of Errors

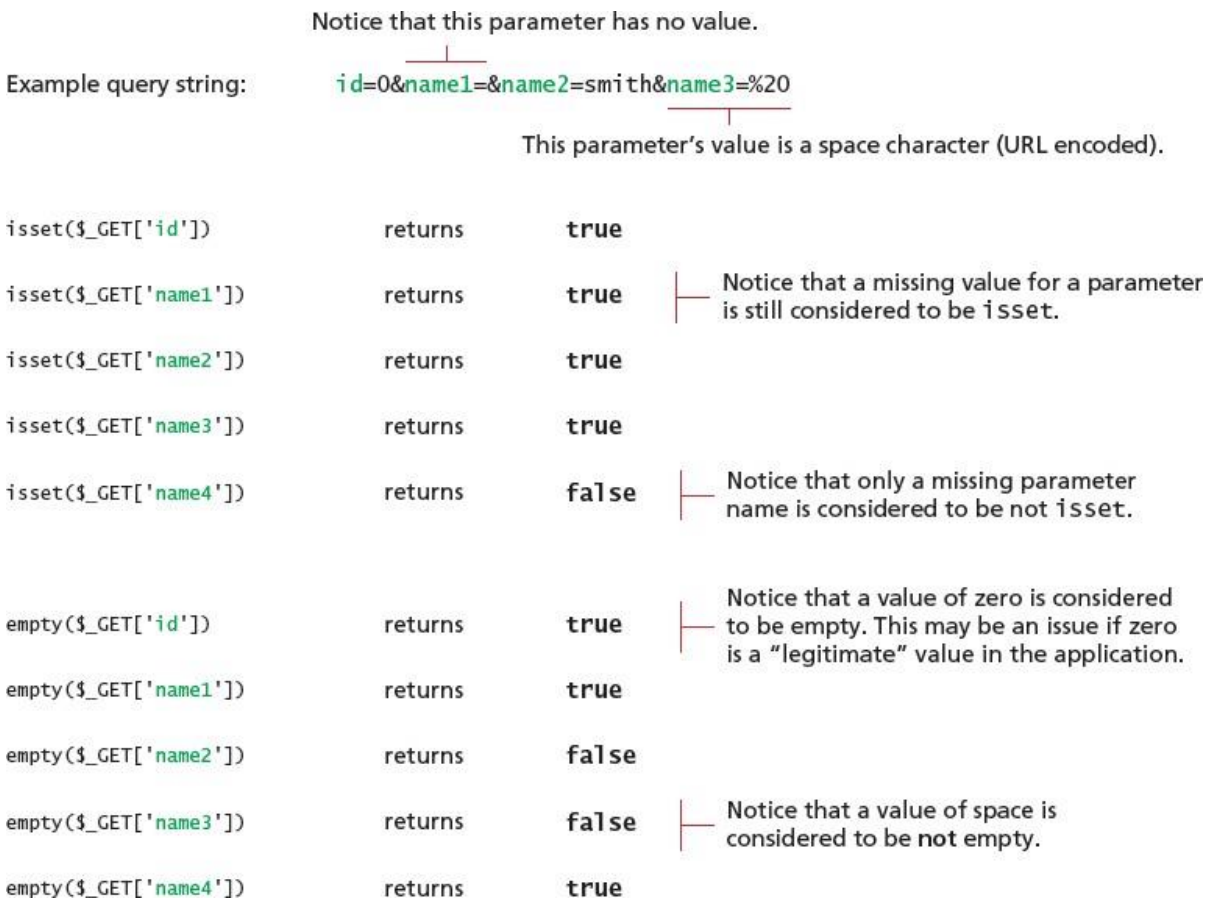
There are three different types of website problems:

- Expected errors
- Warnings
- Fatal errors

An **expected error** is an error that routinely occurs during an application. The most common example of this type of error is as a result of user inputs, ie. , entering letters when numbers were expected. The web application should be developed such that, we expect the user to not always enter expected values. Users will leave fields blank, enter text when numbers were expected, type in too much or too little text, forget to click certain things, and click things they should not etc.

Not every expected error is the result of user input. Web applications that rely on connections to externalities such as database management systems, legacy software systems, or web services is expected to occasionally fail to connect. So there should be some type of logic that verifies the errors in code, check the user inputs and check if it contains the expected values.

PHP provides two functions for testing the value
of a variable.isset() function - which returns true if
a variable is not null.
empty() function - which returns true if a variable is null, false, zero, or an empty string.



To check a numeric value, use the is_numeric() function, as shown

```
$id = $_GET['id'];
if (!empty($id) && is_numeric($id) ) {
   // use the query string since it exists and is a numeric value

...
}
```

Another type of error is **warnings**, which are problems that generate a PHP warning message (which may or may not be displayed) but will not halt the execution of the page. For instance, calling a function without a required parameter will generate a warning message but not stop execution. While not as serious as expected errors, these types of incidental errors should be eliminated by the programmer.

The **fatal errors**, which are serious in that the execution of the page will terminate unless handled in some way. These types of errors are exceptional and unexpected, such as a required input file being missing or a database table or field disappearing. These types of errors need to bereported so that the developer can try to fix the problem, and also the page needs to recover gracefully from the error so that the user is not excessively puzzled or frustrated.

Exceptions

In the context of PHP, error and exception is not the same. An **error** is some type of

problem that generates a nonfatal warning message or that generates an error message that terminates the program's execution. An **exception** refers to objects that are of type Exception and which are used in conjunction with the object-oriented try . . . catch language construct for dealing with runtime errors.

PHP Error Reporting

PHP has a flexible and customizable system for reporting warnings and errors that can be set programmatically at runtime or declaratively at design-time within the **php.ini** file. There are three main error reporting flags:

- error_reporting
- display_errors
- log_errors

Note: PHP.ini is a configuration file, that is used to customize behavior of PHP at runtime. It consists of settings for different values such as register global variables, display errors, log errors, max uploading size setting, maximum time to execute a script and other configurations. When PHP Server starts up it looks for PHP.ini file first to load various values for settings.

The error_reporting Setting

The error_reporting setting specifies which type of errors are to be reported. It can be setprogrammatically inside any PHP file by using the error_reporting() function: error_reporting(E_ALL);

It can also be set within the **php.ini** file:
error_reporting = E_ALL

The possible levels for error_reporting are defined by predefined constants(the default setting iszero, that is, no reporting).

Constant Name	Value	Description
E_ALL	8191	Report all errors and warnings
E_ERROR	1	Report all fatal runtime errors
E_WARNING	2	Report all nonfatal runtime errors (i.e., warnings)
	0	No reporting

The display_errors Setting

The display_error setting specifies whether error messages should or should not be displayed inthe browser. It can be set programmatically via the ini_set() function:
ini_set('display_errors','0');
It can also be set within the **php.ini** file:
display_errors = Off

The log_error Setting

The log_error setting specifies whether error messages should or should not be sent to the servererror log. It can be set programmatically via the ini_set() function:
ini_set('log_errors','1');

4. Describe how to modify DOM in jQuery.

Modifying the DOM

jQuery comes with several useful methods to manipulate the DOM elements themselves.

Creating DOM and textNodes

If you decide to think about your page as a DOM object, then you will want to manipulate the tree structure rather than merely manipulate strings. jQuery is able to convert strings containing valid DOM syntax into DOM objects automatically.

The jQuery methods to manipulate the DOM take an HTML string, jQuery objects, or DOM objects as parameters, you might prefer to define your element as

```
var element = $("<div></div>"); //create new DOM node based on html
```

This way you can apply all the jQuery functions to the object, rather than rely on pure JavaScript, which has fewer shortcuts. If we consider creation of a simple <a> element with multiple attributes, you can see the comparison of the JavaScript and jQuery techniques.

Prepending and Appending DOM Elements

When an element is defined, it must be inserted into the existing DOM tree. You can also insert the element into several places at once if you desire, since selectors can return an array of DOM elements.

The append() method takes as a parameter an HTML string, a DOM object, or a jQuery object. That object is then added as the last child to the element(s) being selected. In below figure we can see the effect of an append() method call. Each element with a class of linkOut has the jsLink element appended to it.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
         funwebdev.com
     </div>
     <div class="linkIn">
         /localpage.html
     </div>
     <div class="linkOut">
         pearson.com
     </div>
<div>
```

jQuery append

```
$(".linkOut").append(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
         funwebdev.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
     </div>
     <div class="linkIn">
         /localpage.html
     </div>
     <div class="linkOut">
         pearson.com
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
     </div>
<div>
```

The appendTo() method is similar to append() but is used in the syntactically converse way. If we were to use appendTo(), we would have to switch the object making the call and theparameter to have the same effect as the previous code:

jsLink.appendTo($(".linkOut"));

The prepend() and prependTo() methods operate in a similar manner except that they add the new element as the first child rather than the last.

HTML Before

```
<div class="external-links">
     <div class="linkOut">
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
        pearson.com
    </div>
<div>
```

jQuery append

```
$(".linkOut").prepend(jsLink);
```

HTML After

```
<div class="external-links">
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        funwebdev.com
    </div>
     <div class="linkIn">
         /localpage.html
    </div>
     <div class="linkOut">
<a href='http://funwebdev.com'
title='jQuery'>Visit Us</a>
        pearson.com
    </div>
<div>
```

5. Explain JavaScript pseudo classes such as prototypes, emulating classes with functions, object literals.

JavaScript Pseudo-Classes

Although JavaScript has no formal class mechanism, it does support objects (such as the DOM). While most object-oriented languages that support objects also support classes formally, JavaScript does not. Instead, you define **pseudo-classes** through a variety of interesting and nonintuitive syntax constructs. Many common features of object-oriented programming, such as inheritance and even simple methods, must be arrived at through these nonintuitive means. Benefits of using object-oriented design in your JavaScript include increased code reuse, better memory management, and easier maintenance.

Almost all modern frameworks (such as jQuery and the Google Maps API) use prototypes to simulate classes, so understanding the mechanism is essential to apply those APIs in applications.
An object can be instantiated using an object represented by the list of key-value pairs with colons between the key and value with commas separating key-value pairs.
A dice object, with a string to hold the color and an array containing the values representing eachside (face), could be defined all at once using object literals as follows:

```
var oneDie = { color : "FF0000", faces : [1,2,3,4,5,6] };
```

Once defined, these elements can be accessed using dot notation. For instance, one could changethe color to blue by writing:
oneDie.color="0000FF";

Emulate Classes through Functions

Although a formal *class* mechanism is not available to us in JavaScript, it is possibleto get close by using functions to encapsulate variables and methods together, as shown below.

```
function
Die(col) {
this.color=
col;
this.faces=[
1,2,3,4,5,6]
;
}
```

The ‘this’ keyword inside of a function refers to the instance, so that every reference to internal properties or methods manages its own variables, as is the case with PHP.
One can create an instance of the object as follows, very similar to PHP.
var oneDie = new Die("0000FF");

Using Prototypes

Prototypes are an essential syntax mechanism in JavaScript, and are used to make JavaScript behave more like an object-oriented language. The prototype properties and methods are defined *once* for all instances of an *object*. So modification of the definition of the randomRoll() method, is done as shown below by moving the randomRoll() method into the prototype.

```
// Start
Die Class
function
Die(col) {
this.color
=col;
this.faces
=[1,2,3,4,
5,6];
}
Die.prototype.randomRoll = function() {
var randNum = Math.floor((Math.random() *
this.faces.length) + 1);return faces[randNum-1];
};
// End Die Class
```

This definition is better because it defines the method only once, no matter how many instances of Die are created. The below figure shows how the prototype object (not class) is updated to contain the method so that subsequent instantiations (x and y) refers that one-method definition. Since all instances of a Die share the same prototype object, the function declaration only happens one time and is shared with all Die instances.

A prototype is an object from which other objects inherit.

6. Define Caching and explain page output caching in detail.

   Caching is storing the information. Browser uses caching to speed up the user experience by using locally stored versions of images and other files rather than re-requesting the files from the server.

Page Output Caching

In this type of caching, the contents of the rendered PHP page(page ready to display on screen) are written to disk for fast retrieval. This can be particularly helpful because it allows PHP to send a page response to a client without going through the entire page processing life cycle again.

Page output caching is especially useful for pages whose content does not change frequently but which require significant processing to create. There are two models for page caching: full page caching and partial page caching.

In full page caching, the entire contents of a page are cached. In partial page caching, only specific parts of a page are cached while the other parts are dynamically generated in the normal manner.

Page caching is not included in PHP by default, it has to be attached by using add-ons such as Alternative PHP Cache (open source) and Zend (commercial).