* Object Oriented Concepts
  + **Class**
    - Class is a one type of structure is contains data member and member function
  + **Object**
    - Objects are also known as instance
  + **Member** **Variable**
    - $var one=2;
  + **Member function**
    - Function(){}
  + **Inheritance**
    - Class one extends two
  + **Parent class**
    - A class that is inherited from by another class. This is also called a base class or super class.
  + **Child Class**
    - A class that inherits from another class. This is also called a subclass or derived class.
  + **Polymorphism**
    - This is an object oriented concept where same function can be used for different purposes.
  + **Overloading**
    - a type of polymorphism in which some or all of operators have different implementations depending on the types of their arguments. Similarly functions can also be overloaded with different implementation
  + **Data Abstraction**
    - Any representation of data in which the implementation details are hidden (abstracted).
  + **Encapsulation**
    - refers to a concept where we encapsulate all the data and member functions together to form an object.
  + **Constructor**
    - refers to a special type of function which will be called automatically whenever there is an object formation from a class.
  + **Destructor**
    - refers to a special type of function which will be called automatically whenever an object is deleted or goes out of scope.
  + Example of class
    - class phpDemo

{

var $one;

var $two;

function myFunction(){

}

}

* + - The special form class, followed by the name of the class that you want to define
    - A set of braces enclosing any number of variable declarations and function definitions.
    - Variable declarations start with the special form var, which is followed by a conventional $ variable name;
    - The variable $this is a special variable and it refers to the same object ie. itself.
  + **Creating Objects in PHP**
    - Once you defined your class, then you can create as many objects as you like of that class type.
    - $physics = new Books;
    - $maths = new Books;
  + **Calling Member Functions**
    - After creating your objects, you will be able to call member functions related to that object. One member function will be able to process member variable of related object only.
    - $physics->setTitle( "Physics for High School" );
    - $physics->getTitle();
  + **Constructor** **Functions**
    - Constructor Functions are special type of functions which are called automatically whenever an object is created.
    - PHP provides a special function called \_\_construct() to define a constructor
    - function \_\_construct( $par1, $par2 ) {

$this->title = $par1;

$this->price = $par2;

}

* + **Destructor**
    - Like a constructor function you can define a destructor function using function \_\_destruct().
  + **Inheritance**
    - PHP class definitions can optionally inherit from a parent class definition by using the extends clause.
    - class Child extends Parent
    - Automatically has all the member variable declarations of the parent class.
    - Automatically has all the same member functions as the parent, which (by default) will work the same way as those functions do in the parent.
  + **Function Overriding**
    - Function definitions in child classes override definitions with the same name in parent classes. In a child class, we can modify the definition of a function inherited from parent class.
  + **Public Members**
    - Unless you specify otherwise, properties and methods of a class are public. That is to say, they may be accessed in three possible situations
    - From outside the class in which it is declared
    - From within the class in which it is declared
    - From within another class that implements the class in which it is declared
  + **Private members**
    - By designating a member private, you limit its accessibility to the class in which it is declared. The private member cannot be referred to from classes that inherit the class in which it is declared and cannot be accessed from outside the class.
    - private $car = "skoda";

private function myPrivateFunction() {

return("I'm not visible outside!");

}

* + **Protected members**
    - A protected property or method is accessible in the class in which it is declared, as well as in classes that extend that class. Protected members are not available outside of those two kinds of classes.
    - protected by using protected keyword in front of the member
    - protected $car = "skoda";
  + **Interfaces**
    - Interfaces are defined to provide a common function names to the implementers

interface Mail {

public function sendMail();

}

* + **Constants** 
    - A constant is somewhat like a variable, in that it holds a value, but is really more like a function because a constant is immutable. Once you declare a constant, it does not change.
    - Declaring one constant is easy, as is done in this version of MyClass
    - const requiredMargin = 1.7;
  + **Abstract** **Classes**
    - An abstract class is one that cannot be instantiated, only inherited. You declare an abstract class with the keyword abstract, When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child; additionally, these methods must be defined with the same visibility.
    - Sdf

abstract class MyAbstractClass {

abstract function myAbstractFunction() {

}

}

* + **Static Keyword**
    - Declaring class members or methods as static makes them accessible without needing an instantiation of the class.
    - Ds

class Foo {

public static $my\_static = 'foo';

public function staticValue() {

return self::$my\_static;

}

}

* + - print Foo::$my\_static . "\n";
    - $foo = new Foo();
  + **Final Keyword**
    - PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with final. If the class itself is being defined final then it cannot be extended.
    - final public function moreTesting() {

echo "BaseClass::moreTesting() called<br>";

}

* + **Calling parent constructors**
    - Instead of writing an entirely new constructor for the subclass, let's write it by calling the parent's constructor explicitly and then doing whatever is necessary in addition for instantiation of the subclass.
* **Basics of Namespaces**
  + As the size of your PHP code library increases, the more likely you will accidentally reuse a function or class name that has been declared before
  + By default, all constant, class and function names are placed in a global space — like they were before namespaces were supported.
  + Namespaced code is defined using a single namespace keyword at the top of your PHP file. It must be the first command (with the exception of declare) and no non-PHP code or white-space can precede the command,
  + namespace MyProject;
* **Sub-namespaces**
  + PHP allows you to define a hierarchy of namespace names so libraries can be sub-divided. Sub-namespaces are separated using a backslash (\) character,
  + MyProject\SubName
* **Calling Namespaced Code**
* **Qualified name**
  + An identifier separator, e.g. Lib1MyFunction()
* **Unqualified name** 
  + An identifier without a namespace separator, e.g. MyFunction().
  + function \_\_autoload($class\_name) {
  + require\_once("classes/$class\_name.php");
  + }Namespaces can be imported with the use operator
* **Namespace Aliases**
  + Namespace aliases are perhaps the most useful construct. Aliases allow us to reference long namespaces using a shorter name
  + use App\Lib1 as L;
* **PHP Name Resolution Rules** 
  + Calls to fully-qualified functions, classes or constants are resolved at compile-time.
  + Unqualified and qualified names are translated according to the import rules,
* + Any number of use statements can be defined or you can separate individual namespaces with a comma.
* **Namespace Aliases**
  + Namespace aliases are perhaps the most useful construct. Aliases allow us to reference long namespaces using a shorter name
  + use App\Lib1 as L;
* **The namespace Keyword**
  + The namespace keyword can be used to explicitly reference an item within the current namespace or a sub-namespace. It is the namespace equivalent of self within classes:
* **Autoloading Namespaced Classes**
  + One of the best time-saving features of PHP 5 is autoloading. In global (non-namespaced) PHP code, a standard autoload function
  + function \_\_autoload($class\_name) {
  + require\_once("classes/$class\_name.php");
  + }