**OOP(Object Oriented Programming)**

* Advance php features like design patterns,
  + OOPS,
  + Namespace,
  + Traits
* Object Oriented Programming
  + It is the core of any framework and if trainee is clear with OOPs they can easily jump or switch to any new framework and technology.
* Refence links
  + <https://www.youtube.com/watch?v=7GWwjn0bs5g&list=PLJ49JaW9k5klgfZ22BlgZiElIz3cz08ta&index=7>
* The fundamentals of OOPs
  + Traits
    - Traits is a special class introduce in php7 to overcome limitation of multiple inheritance
    - https://www.php.net/manual/en/language.oop5.traits.php
  + Namespace
    - Namespace is useful to understand the directory structure of any framework
    - https://www.sitepoint.com/php-53-namespaces-basics/
  + Standards
    - Standards will teach the trainee how to write the code?
    - <https://www.php-g.org/psr/>
    - Basic Coding Standard
    - Logger Interface
    - Autoloading Standard,
* **Composer**
  + Download composer.phar file and install one php library using php composer.phar
  + <https://getcomposer.org/>
  + Autoload that installed php libraray by doing practial.
  + Autoload understanding : <https://www.brainbell.com/php/auto-loading.html>
* **PHP design patterns**
  + Design patterns are necessary to understand core of most of the frameworks have code written using various design patterns. To understand the basics of design patter go through following links
  + <https://phptherightway.com/pages/Design-Patterns.html>
* **Reference**:
  + PHP design patterns
    - https://www.ibm.com/developerworks/library/os-php-designptrns <http://www.phptherightway.com/pages/DesignPatterns>.
    - html OOPS Basics
    - <https://www.tutorialspoint.com/php/php_object_oriented.htm>
    - Basics of Namespaces
    - <https://www.sitepoint.com/php-53-namespaces-basics/>
* **Type Hint**
  + With Type hinting we can specify the expected data type (arrays, objects, interface, etc.) for an argument in a function declaration. This practice can be most advantageous because it results in better code organization and improved error messages.
  + When we would like to force a function to get only arguments of the type array, we can put the keyword array in front of the argument name,
    - function functionName (array $argumentName)
    - {
    - //code
    - }
  + Type hinting can also be used to force a function to get an argument of type Object
    - class Car {
    - protected $driver;
    - // The constructor can only get Driver objects as arguments.
    - public function \_\_construct(Driver $driver)
    - {
    - $this -> driver = $driver;
    - }
    - }
    - class Driver {}

    - $driver1 = new Driver();
    - $car1 = new Car($driver1);
* **Namespace**
  + Namespace is useful to understand the directory structure of any framework
  + 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
  + 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
  + <?php
  + // define this code in the MyProject namespace
  + namespace MyProject;
  + It is not possible to nest namespaces or define two or more namespaces for the same code block
  + However, you can define different namespaced code in the same file, e.g.
  + <?php
  + namespace MyProject1;
  + // PHP code for the MyProject1 namespace
  + namespace MyProject2;
  + **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
    - MyProject\Database\MySQL
    - Example
      * In a file named lib1.php, we will define a constant, function, and class with the Html namespace:
      * namespace Html;
      * class Table {
      * public $title = "";
      * public $numRows = 0;
      * public function message() {
      * echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
      * }
      * }
      * $table = new Table();
      * $table->title = "My table";
      * $table->numRows = 5;
  + **Importing, Aliases, and Name Resolution**
    - Namespace Importing
      * Namespaces can be imported with the use operator,
      * use App\Lib1;
      * echo \App\Lib1\MYCONST . "\n";
      * echo \App\Lib1\MyFunction() . "\n";
      * echo \App\Lib1\MyClass::WhoAmI() . "\n";
    - Namespace Aliases
      * Namespace aliases are perhaps the most useful construct. Aliases allow us to reference long namespaces using a shorter name..
      * use App\Lib1 as L;
      * use App\Lib2\MyClass as Obj;
      * The first use statement defines AppLib1 as ‘L’. Any qualified names using ‘L’ will be translated to ‘AppLib1’ at compile-time
    - PHP Name Resolution Rules
      * PHP identifier names are resolved using the following namespace rules
      * Calls to fully-qualified functions, classes or constants are resolved at compile-time.
      * Unqualified and qualified names are translated according to the import rules
      * inside a namespace, all qualified names not already translated according to import rules have the current namespace prepended
      * Unqualified class names are translated according to current import rules and the full name is substituted for short imported name
      * Unqualified function calls within a namespace are resolved at run-time
      * Calls to unqualified or qualified class names are resolved at run-time.
  + **The namespace Keyword**
    - The namespace keyword can be used to explicitly reference an item within the current namespace or a sub-namespace. It is the namespace equivalent of self within classes:
    - Autoloading Namespaced Classes
      * One of the best time-saving features of PHP 5 is autoloading.
      * function \_\_autoload($class\_name) {
      * require\_once("classes/$class\_name.php");
      * }
      * the fully-qualified namespace and class name is passed to the \_\_autoload function
* **Traits**
  + Traits is a special class introduce in php7 to overcome limitation of multiple inheritance. It is widely used by us, please study it carefully
  + PHP implements a way to reuse code called Traits.
  + Traits are a mechanism for code reuse in single inheritance languages such as PHP. A Trait is intended to reduce some limitations of single inheritance by enabling a developer to reuse sets of methods freely in several independent classes living in different class hierarchies
  + The semantics of the combination of Traits and classes is defined in a way which reduces complexity, and avoids the typical problems associated with multiple inheritance and Mixins
  + A Trait is similar to a class,
    - trait ezcReflectionReturnInfo {
    - function getReturnType() { /\*1\*/ }
    - function getReturnDescription() { /\*2\*/ }
    - }
    - class ezcReflectionMethod extends ReflectionMethod {
    - **use ezcReflectionReturnInfo;**
    - /\* ... \*/
    - }
  + Traits are declared with the trait keyword:
  + trait message1 {
  + public function msg1() {
  + echo "OOP is fun! ";
  + }
  + }
  + Sometimes you may want to inherit from more than one class. Since this is not possible in PHP and other single inheritance languages like it, we have something called traits
  + Multiple Traits
    - Multiple Traits can be inserted into a class by listing them in the use statement, separated by commas.
    - use Hello, World;
    - as classes can make use of traits, so can other traits. By using one or more traits in a trait definition, it can be composed partially or entirely of the members defined in those other traits.
    - trait Hello {
    - public function sayHello() {
    - echo 'Hello ';
    - }
    - }
    - trait World {
    - public function sayWorld() {
    - echo 'World!';
    - }
    - }
    - trait HelloWorld {
    - use Hello, World;
    - }
    - class MyHelloWorld {
    - use HelloWorld;
    - }
  + Abstract Trait Members
    - Traits support the use of abstract methods in order to impose requirements upon the exhibiting class. Public, protected, and private methods are supported
    - trait Hello {
    - public function sayHelloWorld() {
    - echo 'Hello'.$this->getWorld();
    - }
    - abstract public function getWorld();
    - }
  + Static Trait members
    - Traits can define static variables, static methods and static properties.
    - trait StaticExample {
    - public static function doSomething() {
    - return 'Doing something';
    - }
    - }m
    - Example::doSomething();
  + **Constants**
    - Traits can, as of PHP 8.2.0, also define constants.
    - trait ConstantsTrait {
    - public const FLAG\_MUTABLE = 1;
    - final public const FLAG\_IMMUTABLE = 5;
    - }