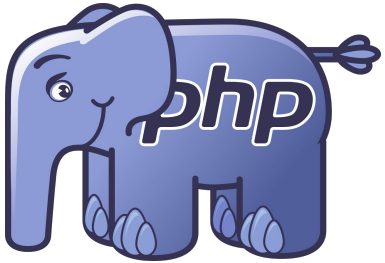
Information Technology Institute



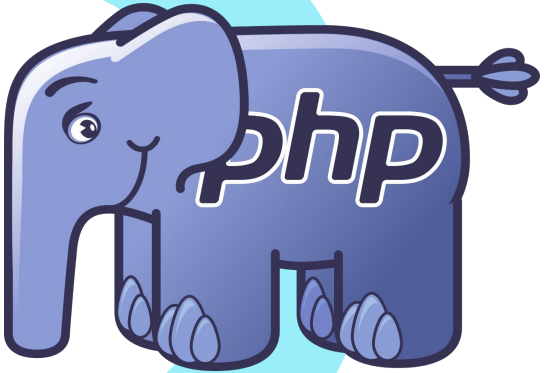
Day05

Prepared by:
Noha Shehab
Teaching Assistant
Information Technology Institute (ITI)

Agenda

- Functions
- Closures
- Generators
- Traits
- Object Oriented

Functions and closures



PHP and OOP

- PHP supports OOP concepts in its structure
- You don't need to repeat yourself just create your code one time and apply the OOP concepts to it and reuse it when you need
- PHP supports creating classes and functions that reach the OOP encapsulation
- Classes, Functions, Closures, Traits, generators and OOP principles are discussed later.
- XML DOMDocument.

Functions

- Define functions
  - function name are not case sensitive
  - you cannot define function with name starts with digit

```php
getSum(55,55);
function getSum($x,$y){
    $z=$x+$y;
    echo 'Sum of x and y is'.$z."<br>";
}
getsum(5,10)
```

  - Delete function

```php
unset($function_name);
```

Functions Parameters & return

- Functions can have optional parameters.

```php
function add($x,$y=1){
    $z=$x+$y;
    echo 'Sum of x and y is'.$z."<br>";
}
add(6);
add(6,7);
```

Variable length argument lists

- PHP 5.6 introduced variable length argument lists, using the ... token before the argument name to indicate that theparameter is variadic..

```php
function variadic_func($nonVariadic, ...$args ){
    echo
    json_encode ($args);
}
variadic_func("hello",'rr',20,55,True);
```

```
["rr",20,55,true]
```

Return keyword

- The keyword return **stops the execution of a function.**

```php
function mul($x,$y){
    return $x*$y;
    echo 'This line will never executed';
}
$res=mul(5,6);
echo $res."<br>";
```

30

Call by value vs Call by reference

- Call by value:

```php
function incrementFun($value, $amount = 1) {
    $value = $value +$amount;
}
$value=10;
incrementFun($value);
var_dump($value); // 10
```

- Call by reference

```php
function incrementFun(&$value, $amount = 1) {
    $value = $value +$amount;
}
$value=10;
incrementFun($value);
var_dump($value); // 11
```

Closures

- A closure is the PHP equivalent of an anonymous function, eg . A function that does not have a name.
- The behavior of a closure remains the same as a function's, with a few extra features.
- A closure is nothing but an object of the Closure class which is created by declaring a function without a name.

```php
$greet = function($name)
{
    printf("Hello %s\r\n", $name);
};  // ; is a must here
$greet('World');
```

Hello World

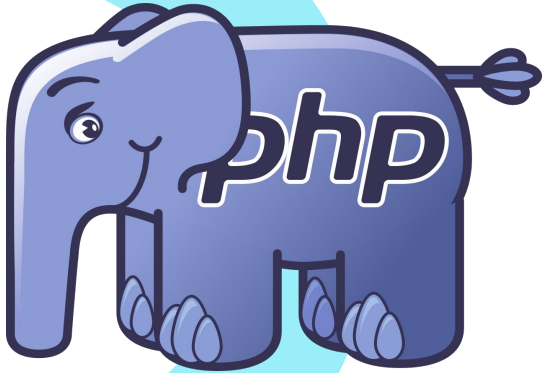
```php
var_dump(is_callable($greet)); // true
```

Closures and external variables.

- You can go further by creating "dynamic" closures via the use keyword to call the external variables, see the example..

```php
$quantity = 1;
# use special keyword to use the variable form outside
$calculator = function($number) use($quantity) {
    return $number + $quantity;
};
var_dump($calculator(7)); // 8
```

Object Oriented with



OOP Principles

- **Encapsulation** : binds together the data and functions that manipulate the data, and that keeps both safe from outside. Data encapsulation led to the important OOP concept of data hiding.
- **Inheritance**: This allows classes to be arranged in a hierarchy that represents "is-a-type-of" relationships.
- **Polymorphism**: Ability to take more than one form
- **Abstraction**: means ignoring irrelevant features, properties, or functions and emphasizing the relevant ones.
- **Reflection**: allows inspection of classes, interfaces, fields and methods at runtime

Classes

- Use the class keyword to define a class.
- Class can have methods, attributes
- Default access modifier for function is public
- You must define the access modifiers for the attributes
- Access modifiers
  - Public
  - Private
  - Protected
  - Static

```php
class Person{
   public $name;
    function seyHello(){
        echo "Hello ". $this->name;
    }
}
$p= new Person();
$p->name="Noha";
$p->seyHello();
```

Access modifiers

- The default option is public , meaning that if you do not specify an access modifier for an attribute or method, it will be public.

- Items that are public can be accessed from inside or outside the class.

- The private access modifier means that the marked item can be accessed only from inside the class. You may also choose to make some methods private, for example, if they are utility functions for use inside the class only. Items that are private will not be inherited.

Access modifiers

- The protected access modifier means that the marked item can be accessed only from inside the class. It also exists in any subclasses; again, you can think of protected as being halfway in between private and public.

```
class Person{
    public $name;
    private $first_name;
    protected $last_name;
    private function sayHello(){
     echo "Hello ".$this->first_name;}
}
```

Class constructor, destructor

- Classes can define a special **__construct()** method, which is executed as part of object creation. This is often used to specify the initial values of an object.

```php
class Person{
    private $first_name;
    private $last_name;
    function __construct($first_name,$last_name){
        $this->first_name=$first_name;
        $this->last_name=$last_name;
    }
}
$p= new Person("Noha","Shehab");
```

- __destruct(): to unset variables before the object goes out of scope.

```php
function __destruct(){
    echo "<br> destructing object";
}
```

```php
unset($p);
```

Dynamic setter and getter

- Classes can define a special **__get()**, **__set()** method, that modify the class properties in run time, they are kind of magic functions

```php
class Person{

    function __set($name,$value){
        $this->$name=$value;
    }
    function __get($name){
        return $this->$name;
    }
}
```

```php
$p = new Person();
$p->__set("track","opensource");
echo $p->__get("track"); // opensource
```

Static methods

- Access class members using the class name
- Constants are static members by default.

```php
class Math
{
    const pi = 3.14159;
    static function squared($input)
    {
        return $input*$input;
    }
}
echo Math::squared(8);
```

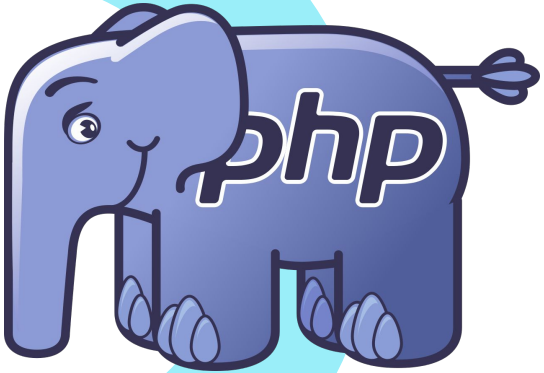
$this and self

- Use $this to refer to the current object.
- Use self to refer to the current class.

In other words, use $this->member for non-static members, use self::$member for static members

```php
class Math
{
    const pi = 3.14159;
    static $mul=1;
    function testSelf(){
        echo self::pi;
    }
```

Closures and classes



Bind a closure to a class

- You can bind a closure to a class using bindto

```php
$myclousre=function(){
    echo $this->property."<br>";
};
class MyClass{
    public $property;
    function __construct($propertyValue){
        $this->property=$propertyValue;
    }
}
```

```php
$objClass= new MyClass("Hello from clousre");
$myBoundClousre= $myclousre->bindTo($objClass);
var_dump($myBoundClousre);
$myBoundClousre();
```

```
object(Closure)[3]
  public 'this' =>
    object(MyClass)[2]
      public 'property' => string 'Hello from clousre' (length=18)

Hello from clousre
```

- Closure can access public properties only

Bind a closure to a class

- The only way to access the private members inside a class using the closure is to access it across the class scope itself, see this

```php
$myclousre=function(){
    echo $this->property."<br>";
};
class MyClass{
    private $property;
    function __construct($propertyValue){
        $this->property=$propertyValue;
    }
}
```

```
object(Closure)[3]
  public 'this' =>
    object(MyClass)[2]
      private 'property' => string 'Hello from clousre' (length=18)

Hello from clousre
```

```php
$objClass= new MyClass("Hello from clousre");
$myBoundClousre= $myclousre->bindTo($objClass,MyClass::class);
//MyClass::Class -> class constant reference , bindto takes the scope
$myBoundClousre();
```

Bind a closure to a class

- From PHP07 you can call the closure directly with the object
- As opposed to the bindTo method, there is no scope to worry about.
- The scope used for this call is the same as the one used when accessing or invoking a property of the instance.

```php
$myclousre=function(){
    echo $this->property."<br>";
};
class MyClass{
    private $property;
    function __construct($propertyValue){
        $this->property=$propertyValue;
    }
}
$objClass= new MyClass("Hello from clousre using call");
$myclousre->call($objClass);
```

Hello from clousre using call

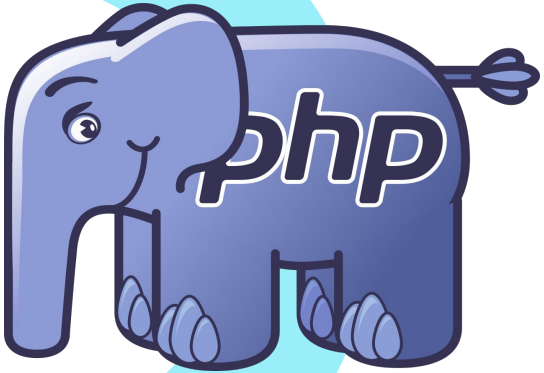
Define a closure inside a class

- A closure created inside a method's class which is invoked in an
- object context will have the same scope as the method's/

```php
class NewClass{
    private $prop;
    function __construct($propertyValue){
        $this->prop=$propertyValue;
    }
    function display(){
        //  a clousure here....
        return function(){
            echo $this->prop."<br>";
        };
    }
}
```

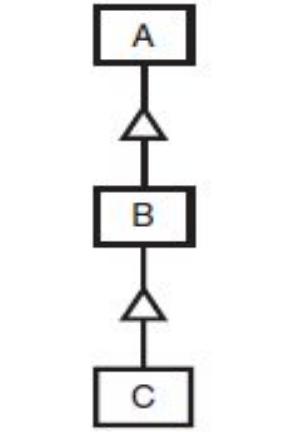
```php
$obj= new NewClass("Hello World");
$display= $obj->display();
$display();  // Hello world
```

Inheritance

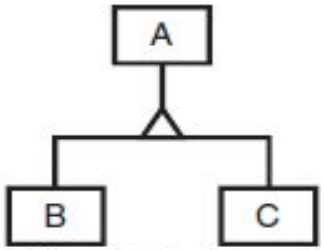


Inheritance

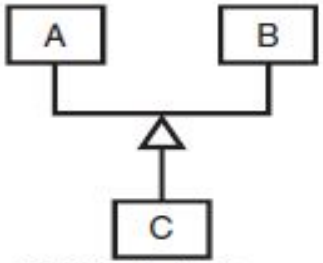
- PHP supports single inheritance.



Single Inheritance



Single Inheritance



Multiple Inheritance

Inheritance

- Check....

```php
class Person{
    public $name;
    public function callingPerson(){
        echo "I am a person";}
}
class Studnet extends Person{
    public $level;
    public function callingStudent(){
        echo "I am a person";}
}
```

```php
$std= new Studnet();
$std->name="Omar";
$std->level=1;
$std->callingPerson();
$std->callingStudent();
```

Inheritance and parent constructor

- Check....

```php
class Person{
    public $name;
    public function __construct(){
        echo "creating a person ";}
}
class Studnet extends Person{
    public $level;
    public function __construct(){
          parent::__construct();
        echo "creating a student";}
}

$std= new Studnet();
```

Inheritance and overriding

- Check....

```php
class Person{
    public $name;
    public function say_hello(){
        echo "Hello person";}
}
class Studnet extends Person{
    public $level;
    public function say_hello(){
        echo "Hello Student";}
}
```

Inheritance and final keyword

- PHP uses the keyword final. When you use this keyword in front of a function declaration, that function cannot be overridden in any subclasses.
- Final in front of the class, this means that cannot be extended.

```php
class Machine{
    public $brand;
    function sayHello(){
        echo "hello I am a machine";
    }
}
class Transportation extends Machine{
    public $type;
    final function canMove(){
        echo "I can move";
    }
}
```

```php
final class Car extends Transportation {
    public $model;
    function sayHello()
    {
        parent::sayHello();
  // TODO: Change the autogenerated stub
        echo "Called from the Car class";
    }
}
```

Abstract classes

- Any class that contains abstract methods must itself be abstract.
- The main use of abstract methods and classes is in a complex class hierarchy where you want to make sure each subclass contains and overrides some particular method; this can also be done with an interface.

Abstract classes

```php
abstract class Base {
    function __construct() {
        echo " <br> <b> This is abstract class constructor ";
    }
    // This is abstract function #abstract function cannot contain body
    abstract function printdata();
}
class Derived extends base {
    function __construct() {
        echo "<br> <b> Derived class constructor";
        Base::__construct();
    }
    function printdata() {
        echo "<br> <b> Derived class printdata function";
    }
}
$b1 = new Derived;
$b1->printdata();
```

Interface

- An Interface allows the users to create programs, and specifying the public methods that the class must implement without involving the complexity of function implementations.

- It is generally referred to as the next level of abstracti on.

- It resembles the abstract methods, resembling the abstract classes.

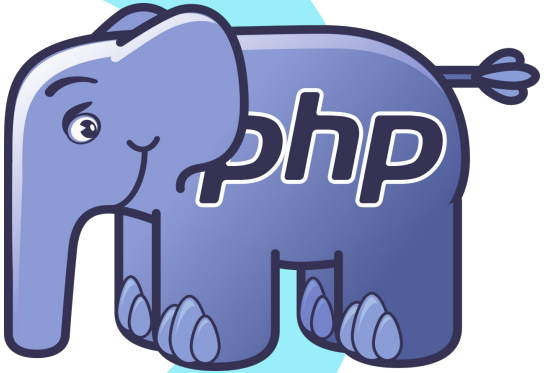
Interface

- The idea of an interface is that it specifies a set of functions that must be implemented in classes that implement that interface.
- Class can implement multiple interfaces.
- Solution for the single inheritance problem in PHP.

```php
Interface Transportation {
    public function setModel($model);
    public function setYear($car);
}
```

```php
class Car implements Transportation{
    public $model;
    public $year;
    public function setModel($model)
    {
        $this->model=$model;
    }
    public function setYear($year)
    {
        $this->year=$year;
    }
}
```

Anonymous classes



Anonymous classes

- Anonymous classes were introduced into PHP 7 to enable for quick one-off objects to be easily created. They can take constructor arguments, extend other classes, implement interfaces, and use traits just like normal classes can.

```php
interface DisplayMsg {
    public function printMsg(string $msg);
}
```

```php
class Application {
    private $displayer;
    public function getDisplayer(): DisplayMsg {
        return $this->displayer;
    }
    public function setPrinter(DisplayMsg $dismsg) {
        $this->displayer = $dismsg;
    }
}
```

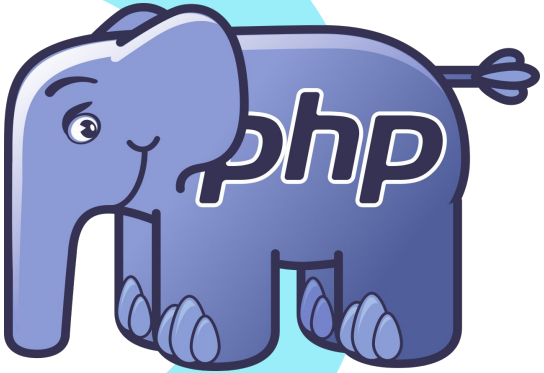
Anonymous classes

- To call the function set_printer you need to have an object with type DisplayMsg, this could be done smoothly with the anonymous classes.

```php
$app = new Application;
$app->setPrinter(new class implements DisplayMsg {
        public function printMsg(string $msg) {
            echo($msg)."<br>";
        }
    }
);
$app->getDisplayer()->printMsg("My first Log Message using anonymous class ");
```

My first Log Message using anonymous class

Reflection Classes



Reflection classes

- “Reflection” in software development means that a program knows its own structure at runtime and can also modify it. This capability is also referred to as “introspection”.

- Reflection is generally defined as a program's ability to inspect itself and modify its logic at execution time.

- In less technical terms, reflection is asking an object to tell you about its properties and methods, and altering those members

Reflection classes

```php
class OpenSource {
    private $instructor;
    protected $sub_tracks;
    public $list_of_courses;
    const PI = 3.1415;
    public function __construct() {
        $this->instructor = "Noha";
        $this->sub_tracks = "Application";
        $this->list_of_courses = ["Python","PHP","Scala","Laravel","Admin"];
    }
    public function getInstructor() {
        return $this->instructor;
    }
    public function setInstructor($instructor) {
        $this->instructor = $instructor;
    }
    private function getsub_tracks() {
        return $this->sub_tracks;
    }
}
```

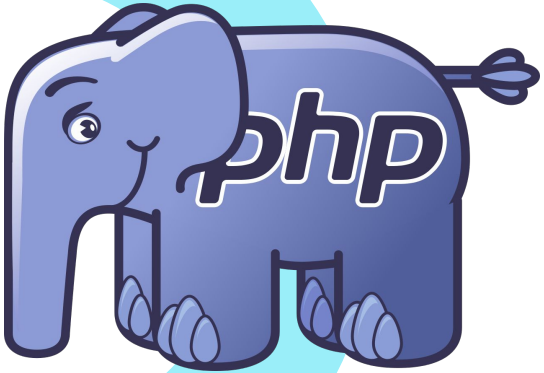
Reflection classes

```php
$reflection_class = new ReflectionClass("Opensource");
foreach ($reflection_class->getMethods() as $method) {
    echo $method->getName() . "<br>";
}
```

```
__construct
getInstructor
setInstructor
getsub_tracks
```

- When to use it:
  - Perform unit testing to your application
  - Getting information about unknown classes

Polymerphism



Polymorphism

- Overloading.
- Overriding.

Method overriding (Dynamic binding)

- Dynamic binding, also referred as method overriding is an example of **run time polymorphism** that occurs when multiple classes contain different implementations of the same method, but the object that the method will be called on is unknown until run time.

Dynamic binding

- Check this...

```php
Interface Animals{
    public function makeNoise();
}
class Cat implements Animals{
    public function makeNoise(){
        $this->meow();
    }
    function meow(){
        echo '<br> mewo';
    }
}
class Dog implements Animals{
    public function makeNoise(){
        $this->bark();
    }
    function bark(){
        echo '<br> bark';
    }
}
```

Dynamic binding

- Check this...

```
class Person {
    Const CAT='cat';
    Const DOG='dog';
    private $petpreference;
    private $pet;
    public function isCatLover() :bool{
        return $this->petpreference =Person::CAT;
    }
    public function isDogLover(): bool {
        return $this->petpreference =self::DOG;
    }
    public function setPet(Animals $pet) {
        $this->pet = $pet;
    }
    public function getPet(): Animals {
        return $this->pet;
    }
}
```

Dynamic binding

- Check this...

```php
$person=new Person();

/// inside if will the set function
if($person->isDogLover()) {
    $person->setPet(new Dog());
}
echo $person->getPet()->makeNoise()."<br>";

if($person->isCatLover()) {
    $person->setPet(new Cat());
}
echo $person->getPet()->makeNoise()."<br>";

var_dump($person);
var_dump ($person->isDogLover());
var_dump( $person->getPet());
var_dump(Person::CAT);
```

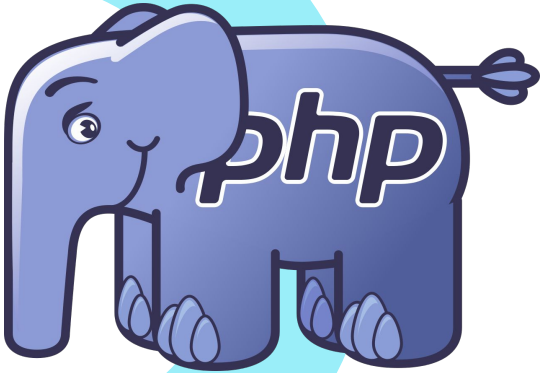
Dynamic overloading

- Used to create dynamic functions in the class.
- For creating these properties no separate line of code is needed.
- Allows calling for static and non static method
- Use __call and __callStatic with static method

Dynamic overloading

```php
class phpClass {
    public function __call($name, $arguments) {
        echo "Calling object method '$name' "
             . implode(', ', $arguments)."<br>";
    }
    public static function __callStatic($name, $arguments) {
        echo "Calling static method '$name' "
             . implode(', ', $arguments)."<br>";
    }
}
// Create new object
$obj = new phpClass;
#function declared on the run time
$obj->runTest('in object context')."<br>";
phpClass::runTest('in static context');
```

Cloning object



Cloning objects

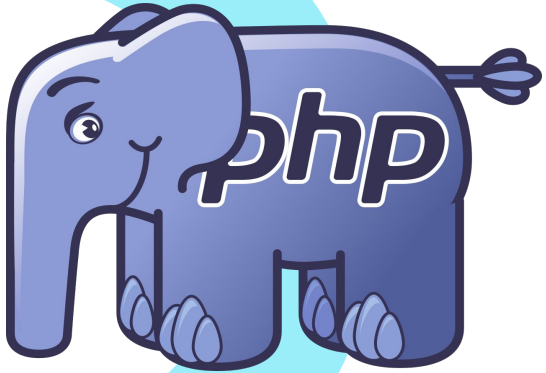
- Object cloning is creating a copy of an object.
- An object copy is created by using the clone keyword and the __clone() method
- In PHP, cloning an object is doing a shallow copy and not a deep copy.
- Meaning, the contained objects of the copied objects are not copied.
- If you wish for a deep copy, then you need to define the __clone() method.
- When there is a need that you do not want the outer enclosing object to modify the internal state of the object then the default PHP cloning can be used.

Cloning objects

```php
class Product{
        public $name;
        public $category;
    }

    $objProduct = new Product();
    //Assigning values
    $objProduct->name = "Apple";
// $objProduct->category = "Fruit";
    //Cloning the original object
    $objCloned = clone $objProduct;
    $objCloned->name = "Orange";
    $objCloned->category = "Fruit";
    print_r($objProduct);
    print_r($objCloned);
```

Traits



Traits

- PHP only allows single inheritance.
- Traits allow you to basically "copy and paste" a portion of a class into your main class.
- A Trait is similar to a class, but only intended to group functionality in a fine-grained and consistent way. It is not possible to instantiate a Trait on its own.
- 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.

Traits

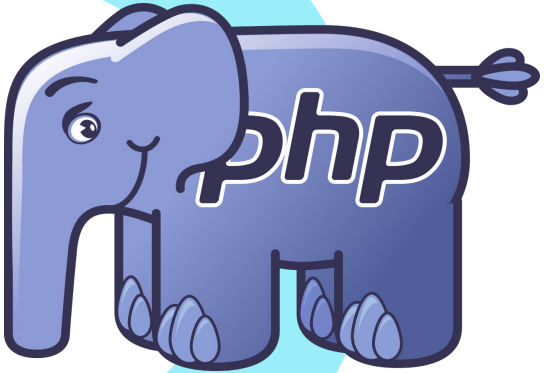
- Define the trait with the keyword trait and use it inside a class.

```php
trait Hello {
    public function sayHello() {
        echo ' <br> Hello ';
    }
}
trait World {
    public function sayWorld() {
        echo 'World';
    }
}
```

```php
class MyHelloWorld {
    use Hello, World;
    public function sayExclamationMark() {
        echo '!';
    }
}
$o = new MyHelloWorld();
$o->sayHello();
$o->sayWorld();
$o->sayExclamationMark();
```

Hello World!

Generators



Generators

- Generators provide an easy way to implement simple iterators without the overhead or complexity of implementing a class that implements the Iterator interface.
- A yield statement is similar to a return statement, except that instead of stopping execution of the function and returning, yield instead returns a Generator object and pauses execution of the generator function.
- Generators are useful when you need to generate a large collection to later iterate over.
- They're a simpler alternative to creating a class that implements an Iterator

Generators

```php
function randomNumbers($length)
{
    $array = [];
    for($i=0;$i<$length;$i++){
        $array[]=mt_rand(1,10);
    }
    return $array;
}
```

- This function generates an array that's filled with random numbers.
- What if we want to generate one million random numbers randomNumbers 1000000) will do that for us, but at a cost of memory. One million integers stored in an array uses approximately 33 megabytes of memory.

Generators

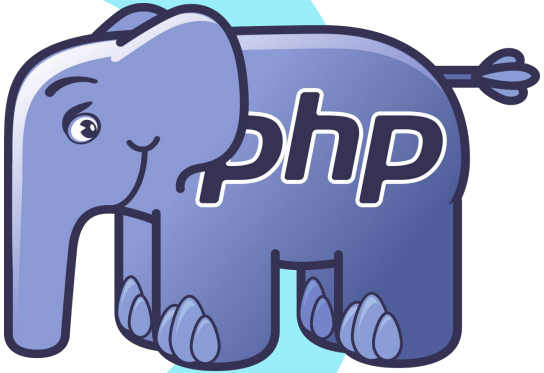
```php
function randomNumbers($length)
{
    for($i=0;$i<$length;$i++){
        yield mt_rand(1,100);
    }
} // function return with a generator object
```

- This function will return with a generator object that can be iterated to generate the required values, see this

```php
$genObj=randomNumbers(10);
foreach ($genObj as $num){
    echo $num."<br>";
}
```

81
71
59
38
89
92
88
70
65
70

Required and Include

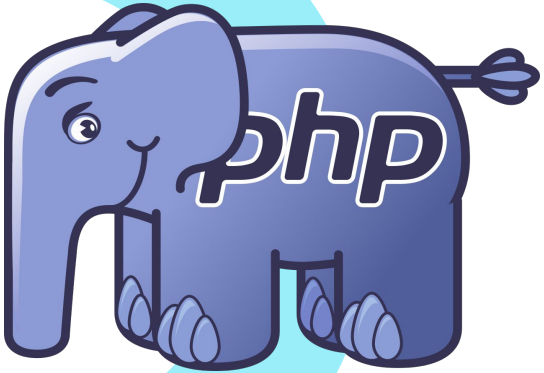


Required and include.

- Using a require() or include() statement, you can load a file into your PHP script.

- require_once () and include_once (). The purpose of these constructs is, as you might guess, to ensure that an included file can be included only once.

Namespaces



Namespaces

- A way of encapsulating items to avoid name conflicts
- Uses keyword namespace
- Namespace can be defined either with name or without.
- You can define multiple namespaces inside each other
- Namespaces should be in the top of your script
- Recommended to declare one name space for a file

Declare namespace

- namespace MyProject;
  - Declare the namespace MyProject

- namespace MyProject\Security\Cryptography;
  - Declare a nested namespace

- Namespace ITI{}

Declare namespace

- To create a new namespace

```php
namespace Project\Shapes;
class Rectangle {
}
```

- To call it from outside,

```php
require("namespace.php");
use Project\Shapes;
$rec= new Shapes\Rectangle();
```

Lab 05

Implement a class called Database that have methods
connect-> takes the connection credentials, insert accepts table names and columns names.
function select that accepts table name and retrieves the data from it,
Function update that accepts the table name and the id of the record needed to be updated and the fields values.
Function called delete, that accepts the table name and the id of the record needed to deleted.

Thanks ^^

Noha Shehab
nshehab@iti.gov.eg