Magic methods

magic methods in PHP are special methods that allow you to define how objects behave in certain situations. They are prefixed with double underscores (__) and provide hooks for various operations

__get and __set method

The __get magic method is invoked when an inaccessible or nonexistent property is accessed. This allows you to define custom behavior when trying to read a property that is

عند محاولة الوصول إلى خاصية غير موجودة أو خاصة (private/protected)

```php
<?php
class User {
    private $data = [];
    private $name ;

    public function __get($name) {
        echo "Getting value of '$name'\n";
        return isset($this->data[$name]) ? $this->data[$name] : null;
    }}
?>
```

Use Cases:

- Dynamic Data Access: Suitable for objects where properties may vary or be dynamically defined.

- Backward Compatibility: Useful when maintaining legacy code while adding new

The __Set method in PHP is a magic method that allows you to define custom behavior when setting the value of an inaccessible or non-existent property of an object

عند محاولة تعيين قيمة لخاصية غير موجودة أو غير قابلة للوصول في الكائن

```php
  <?php
class User {
    private $data = [];
    private $name ;

        public function __set($name, $value) {
            echo $name;
            $this->data[$name] = $value;
        }
}
?>
```

Use cases:

- Dynamic Property Handling: Useful when dealing with objects that may have properties added or changed dynamically.

- Custom Validation: Implement custom validation or transformation logic when setting properties.

- Logging or Debugging: Track property changes for debugging or logging purposes.

__call method

to call an inaccessible or nonexistent method on an object. This allows you to handle method calls dynamically.

لاستدعاءات إلى الدوال غير المعرفة في كائن ما

```php
   <?php
class Method {
    public function __call($name, $arguments) {
        echo "Method '$name' was called with arguments: " . implode(', ', $arguments) . "\n";
    }
}

$obj = new Method();
$obj->foo('set', 'this method');
echo $obj;
?>
```

Use cases:

- Dynamic Method Handling
- Implementing a Proxy:

  You can create a proxy class that forwards method calls to another object

- Method Overloading
- You can use __call() to simulate method overloading based on the number of arguments

__call static mthod

magic method is used to handle calls to static methods that are not defined in a class. This method works similarly to the __call() method but is specifically for static method calls

تُستخدم لمعالجة الاستدعاءات إلى الدوال الثابتة غير المعرفة

```php
<?php
class Method {
    public static function __callStatic($name, $arguments) {
        echo "Static method '$name' was called with arguments: " . implode(', ', $arguments) . "\n";
    }
}

echo Method::foo('ahmed', 'student');

?>
```

Use Cases:

Dynamic Static APIs: Create APIs that can handle a variety of static method calls dynamically.

- Static Logging Systems: Implement flexible logging mechanisms without needing to define each log level method statically.

- Proxies and Decorators: Forward static method calls to other classes dynamically.

__clone

method is invoked when an object is cloned using the clone keyword. This allows you to define custom behavior for object cloning, which is particularly useful when you want to control how properties of an object are copied, especially when dealing with objects that contain references to other objects

```php
<?php
class Address {
    public $street;

    public function __construct($street) {
        $this->street = $street;
    }
}

class User {
    public $name;
    public $address;

    public function __construct($name, Address $address) {
        $this->name = $name;
        $this->address = $address;
    }

    public function __clone() {

        $this->address = clone $this->address;
    }
}
```

```php
$originalAddress = new Address("cairo.13");
$user1 = new User("ali", $originalAddress);
var_dump( $originalAddress);
var_dump($user1);
?>
```

Use Cases:

-Deep Copying Objects

- Resetting State on Clone

- Managing Resource Handles

__toString

magic method allows you to define how an object should be represented as a string when it is treated as a string.

```php
<?php
class Product {
    public $id;
    public $name;
    public $price;

    public function __construct($id, $name, $price) {
        $this->id = $id;
        $this->name = $name;
        $this->price = $price;
    }

    public function __toString() {
        return "Product ID: {$this->id}, Name: {$this->name}, Price: \${$this->price}";
    }
}

$product = new Product(15, "bag", 999);
```

```php
error_log($product);
?>
```

Use Cases:

-imple String Representation

-Debugging and Logging

-Formatting Output

__invoke Method

Allow an object to be called as a function. This means that you can create instances of a class that can be invoked directly like a function

method can accept any number of parameters, and it should contain the logic you want to execute when the object is invoked

```php
<?php
class User {
    private $greeting;

    public function __construct($greeting) {
        $this->greeting = $greeting;
    }

    public function __invoke($name) {
        return "{$this->greeting}, {$name}!";
    }
}
```

```php
$greeter = new User("hello");
echo $greeter("ahmed");
?>
```

Use Cases:

Callable Objects: Create objects that can be called as functions, allowing for cleaner and more expressive code.

- Data Transformation: Implement complex data transformation logic in a way that is easy to use and understand.

- API Design: Design APIs where objects can be used interchangeably with functions, enhancing usability and readability.