Manipulating Files

PHP offers a number of functions to use when creating, reading, uploading, and editing files.
The **fopen()** function creates or opens a file. If you use **fopen()** with a file that does not exist, the file will be created, given that the file has been opened for writing (w) or appending (a).

Use one of the following **modes** to open the file.
r: Opens file for read only.
w: Opens file for write only. Erases the contents of the file or creates a new file if it doesn't exist.
a: Opens file for write only.
x: Creates new file for write only.
r+: Opens file for read/write.
w+: Opens file for read/write. Erases the contents of the file or creates a new file if it doesn't exist.
a+: Opens file for read/write. Creates a new file if the file doesn't exist
x+: Creates new file for read/write.

The example below creates a new file, "file.txt", which will be created in the same directory that houses the PHP code.

```php
$myfile = fopen("file.txt", "w");
```

PHP offers a number of functions to use when creating, reading, uploading, and editing files.

Drag and drop from the options below to open a file called 'test.txt' for writing.

```
     (     ,     );
```

`fopen` `'w'` `'test.txt'`

Write to File

When writing to a file, use the **fwrite()** function. The first parameter of **fwrite()** is the file to write to; the second parameter is the string to be written.

The example below writes a couple of names into a new file called "names.txt".

```php
<?php
$myfile = fopen("names.txt", "w");

$txt = "John\n";
fwrite($myfile, $txt);
$txt = "David\n";
fwrite($myfile, $txt);

fclose($myfile);

/* File contains:
John
David
*/
?>
```

Notice that we wrote to the file "names.txt" twice, and then we used the **fclose()** function to close the file.

The **\n** symbol is used when writing **new lines**.

What is the symbol for a new line in a text file?

Type

fclose()

The **fclose()** function closes an open file and returns TRUE on success or FALSE on failure.

It's a good practice to close all files after you have finished working with them.

Rearrange the code to write 12 to the file 'num.txt'.

```
fwrite($handle, '1');
```

```
$handle = fopen('num.txt', 'w');
```

```
fwrite($handle, '2');
```

```
fclose($handle);
```

Appending to a File

If you want to append content to a file, you need to open the file in **append mode**.

For example:

```php
$myFile = "test.txt";
$fh = fopen($myFile, 'a');
fwrite($fh, "Some text");
fclose($fh);
```

When appending to a file using the **'a' mode**, the file pointer is placed at the end of the file, ensuring that all new data is added at the end of the file.

Which of the following is not a supported file access mode for the fopen function?

- `w`
- `r`
- `d`
- `a`

Appending to a File

Let's create an example of a form that adds filled-in data to a file.

```php
<?php
if(isset($_POST['text'])) {
  $name = $_POST['text'];
  $handle = fopen('names.txt', 'a');
  fwrite($handle, $name."\n");
  fclose($handle);
}
?>
<form method="post">
  Name: <input type="text" name="text" /
>
  <input type="submit" name="submit" />
</form>
```

Now, each time a name is entered and submitted, it's added to the "names.txt" file, along with a new line.

The **isset()** function determined whether the form had been submitted, as well as whether the text contained a value.

We did not specify an **action** attribute for the form, so it will submit to itself.

Fill in the blanks to open the file using append mode, write to it, and close it.

```
$h = [Type] ('my.txt', '[Type]');

fwrite($h, 'test');

[Type] ($h);
```

Reading a File

The **file()** function reads the entire file into an array. Each element within the array corresponds to a line in the file:

CODE PLAYGROUND PHP

```php
$read = file('names.txt');
foreach ($read as $line) {
  echo $line .", ";
}

Tap to edit |
```

This prints all of the lines in the file, and separates them with commas.

We used the **foreach** loop, because the **$read** variable is an **array**.

Which function is used to read the content of a file?

`file()`

`read()`

`read_file()`

Reading a File

At the end of the output in the previous example, we would have a comma, as we print it after each element of the array.
The following code lets us avoid printing that final comma.

CODE PLAYGROUND **PHP**

```php
$read = file('names.txt');
$count = count($read);
$i = 1;
foreach ($read as $line) {
  echo $line;
  if($i < $count) {
    echo ', ';
  }
  $i++;
}
```

The $count variable uses the **count** function to obtain the number of elements in the $read array. Then, in the foreach loop, after each line prints, we determine whether the current line is less than the total number of lines, and print a comma if it is.

This avoids printing that final comma, as for the last line, $i is equal to $count.

Which function was used to get the number of elements in the array?

Type

Which is the correct way to open the file "time.txt" as readable?

```
fopen("time.txt","r");
```

```
open("time.txt","read");
```

```
fopen("times.txt", "d");
```

```
open("time.txt");
```

Fill in the blanks to read and output the content of the file "nums.txt".

```
$nums =    Type    ("nums.txt");

foreach(  $    Type    as $num) {

   echo $num."<br />";

}
```

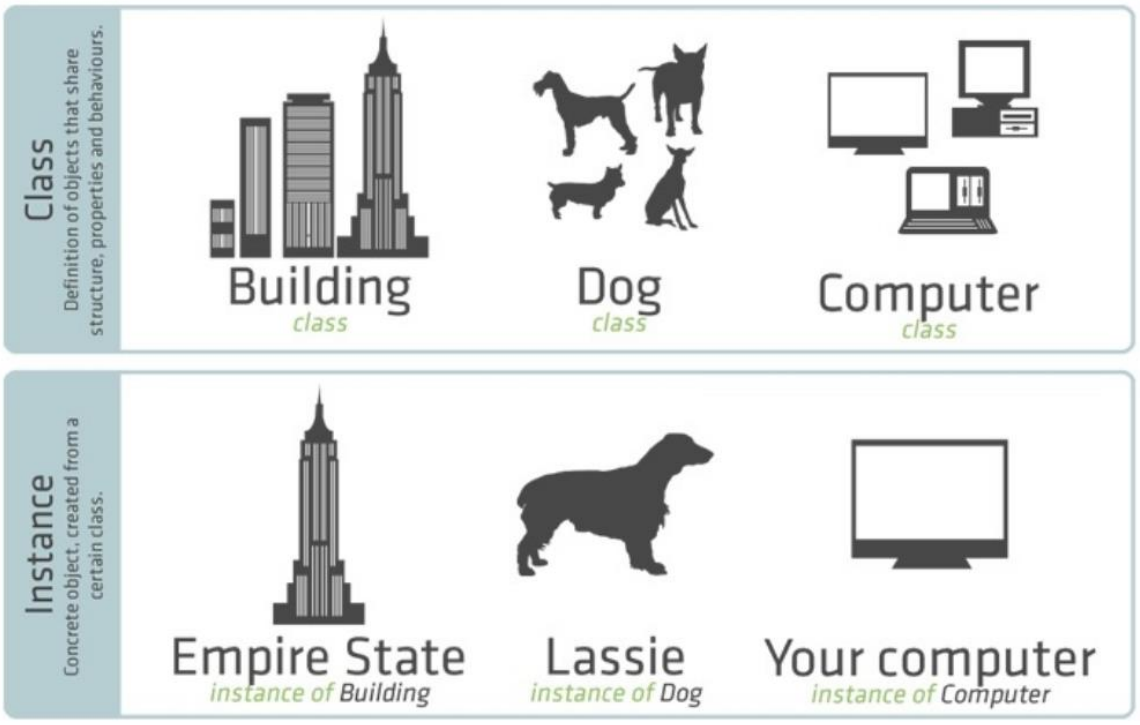
Fill in the blanks to write the numbers 1 to 10 to the file.

```
$h =    Type    ('file.txt', 'a');

for($i=1;$i<=10; $i++) {

      Type    ($h, $i);

}

Type    ($h);
```

Classes & Objects in PHP

Object **O**riented **P**rogramming (OOP) is a programming style that is intended to make thinking about programming closer to thinking about the real world.

Objects are created using **classes**, which are the focal point of OOP.
The class describes what the object will be, but is separate from the object itself. In other words, a class can be thought of as an object's **blueprint**, description, or definition.
Take a look at the following examples:



Here, **Building** is a class. It defines the features of a generic building and how it should work.
The **Empire State Building** is a specific object (or **instance**) of that class.

You can use the same class as a blueprint for creating multiple different objects.

An object is an instance of a class.

`False`

`True`

PHP Classes

In PHP, a class can include member variables called **properties** for defining the features of an object, and functions, called **methods**, for defining the behavior of an object. A class definition begins with the keyword **class**, followed by a class name. Curly braces enclose the definitions of the properties and methods belonging to the class.

For example:

```php
class Person {
  public $age; //property
  public function speak() { //method
    echo "Hi!"
  }
}
```

The code above defines a **Person** class that includes an **age** property and a **speak**() method.

A valid class name starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

Notice the keyword **public** in front of the speak method; it is a **visibility specifier**.
The **public** keyword specifies that the member can be accessed from anywhere in the code.
There are other visibility keywords and we will learn about them in later lessons.

Fill in the blanks to declare a class Student with a sayHi() method:

```
Type    Student {
  public $name;
  public $age;
  public    Type
 sayHi() { echo "Hi!"; }
}
```

PHP Objects

The process of creating an object of a class is called **instantiation**.
To instantiate an object of a class, use the keyword **new**, as in the example below:

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

In the code above, **$bob** is an object of the **Person** class.

To access the properties and methods of an object, use the arrow (->) construct, as in:

```php
echo $bob->age;
```

This statement outputs the value of the **age** property for **$bob**. If you want to assign a value to a property use the assignment operator = as you would with any variable.

Let's define the **Person** class, instantiate an object, make an assignment, and call the **speak**() method:

CODE PLAYGROUND **PHP**

```php
class Person {
  public $age;
  function speak() {
    echo "Hi!";
  }
}
$p1 = new Person(); //instantiate an
object
$p1->age = 23; // assignment
echo $p1->age; // 23
$p1->speak(); // Hi!

Tap to edit |
```

Create a "dog" object which is an instance of the "Animal" class.

```
$ Type = Type Animal();
```

$this

$this is a pseudo-variable that is a reference to the calling object. When working within a method, use **$this** in the same way you would use an object name outside the class.

For example:

CODE PLAYGROUND PHP

```php
class Dog {
  public $legs=4;
  public function display() {
    echo $this->legs;
  }
}
$d1 = new Dog();
$d1->display(); //4
```

For example:

CODE PLAYGROUND PHP

```php
class Dog {
  public $legs=4;
  public function display() {
    echo $this->legs;
  }
}
$d1 = new Dog();
$d1->display(); //4

$d2 = new Dog();
$d2->legs = 2;
$d2->display(); //2

Tap to edit |
```

We created two objects of the Dog class and called their **display**() methods. Because the **display**() method uses **$this**, the **legs** value referred to the appropriate calling object's property value.

As you can see, each object can have its own values for the properties of the class.

Fill in the blanks to declare a class Student and a method that prints its name and age properties.

```
Type   Student {
  public $name;
  public $age;
  public function printData() {
      Type   $this->name;
    echo   Type   ->age;
  }
}
```

PHP Class Constructor

PHP provides the constructor magic method **__construct()**, which is called automatically whenever a new object is instantiated.

For example:

CODE PLAYGROUND **PHP**

```php
class Person {
  public function __construct() {
    echo "Object created";
  }
}
$p = new Person();

Tap to edit |
```

The **__construct()** method is often used for any initialization that the object may need before it is used. Parameters can be included in **__construct()** to accept values when the object is created.

For example:

CODE PLAYGROUND **PHP**

```php
class Person {
  public $name;
  public $age;
  public function __construct($name,
$age) {
    $this->name = $name;
    $this->age = $age;
  }
}
$p = new Person("David", 42);

Tap to edit
```

In the code above, the constructor uses arguments in the new statement to initialize corresponding class properties.

You can't write multiple **__construct()** methods with different numbers of parameters. Different constructor behavior must be handled with logic within a single **__construct()** method.

Fill in the blanks to implement a constructor method for the User class.

```
class User {

  public  Type  __  Type  (){

    echo "constructed";

  }

}

$n = new User();
```

PHP Class Destructor

Similar to the class constructor, there is a destructor magic method **__destruct()**, which is automatically called when an object is destroyed.

For example:

CODE PLAYGROUND PHP

```php
class Person {
  public function __destruct() {
    echo "Object destroyed";
  }
}
$p = new Person();

Tap to edit
```

What is the output of the following code?

```
class TestMe {
  public function __construct() { echo "2"; }
  public function __destruct() { echo "1"; }
}
$test = new TestMe();
unset($test);
```

21

2

12

1

PHP Class Inheritance

Classes can inherit the methods and properties of another class. The class that inherits the methods and properties is called a **subclass**. The class a subclass inherits from is called the **parent class**.

Inheritance is achieved using the **extends** keyword.

For example:

CODE PLAYGROUND PHP

```php
class Animal {
  public $name;
  public function hi() {
    echo "Hi from Animal";
  }
}
class Dog extends Animal {
}

$d = new Dog();
$d->hi();

Tap to edit
```

Here the **Dog** class inherits from the **Animal** class. As you can see, all the properties and methods of **Animal** are accessible to **Dog** objects.

Parent constructors are not called implicitly if the subclass defines a constructor. However, if the child does not define a constructor then it will be inherited from the parent class if it is not declared **private**.

Notice all our properties and methods have **public** visibility.
For added control over objects, declare methods and properties using a visibility keyword. This controls how and from where properties and methods can be accessed.
Check out the next lesson for more on **visibility**.

Fill in the blanks to define a class Singer that inherits from the Musician class.

```
class Musician    Type

  public $name;

  public function toPlay() {

    echo "Playing on piano";

  }

}

Type    Singer    Type    Musician {

}
```

PHP Visibility

Visibility controls how and from where **properties** and **methods** can be accessed.
So far, we have used the **public** keyword to specify that a property/method is accessible from anywhere.
There are two more keywords to declare visibility:
protected: Makes members accessible only within the class itself, by inheriting, and by parent classes.
private: Makes members accessible only by the class that defines them.

Class properties must always have a visibility type. Methods declared without any explicit visibility keyword are defined as **public**.

Protected members are used with inheritance.
Private members are used only internally in a class.

Which of the following defines a visibility control allowing access only from subclasses?

- `private`
- `final`
- `protected`
- `invisible`

PHP Interfaces

An **interface** specifies a list of methods that a class **must** implement. However, the interface itself does not contain any method implementations. This is an important aspect of interfaces because it allows a method to be handled differently in each class that uses the interface.

The **interface** keyword defines an interface. The **implements** keyword is used in a class to implement an interface.

For example, **AnimalInterface** is defined with a declaration for the **makeSound**() function, but it isn't implemented until it is used in a class:

CODE PLAYGROUND PHP

```php
<?php
interface AnimalInterface {
  public function makeSound();
}

class Dog implements AnimalInterface {
  public function makeSound() {
    echo "Woof! <br />";
  }
}
class Cat implements AnimalInterface {
  public function makeSound() {
    echo "Meow! <br />";
  }
}

$myObj1 = new Dog();
$myObj1->makeSound();

$myObj2 = new Cat();
$myObj2->makeSound();
?>
```

A class can implement multiple interfaces. More than one interfaces can be specified by separating them with commas. For example:

```php
class Demo implements AInterface,
BInterface, CInterface {
  // Functions declared in interfaces
must be defined here
}
```

An interface can be inherit another interface by using the **extends** keyword.

All the methods specified in an interface require **public** visibility.

Fill in the blanks to define an interface IMusician with a declaration for the play() method. Then, define a Guitarist class that implements IMusician and its play method.

```
Type    IMusician {
  public function play();
}

Type    Guitarist    Type    IMusician {
  public function    Type    () {
    echo "playin a guitar";
  }
```

PHP Abstract Classes

Abstract classes can be inherited but they cannot be instantiated.
They offer the advantage of being able to contain both methods with definitions and abstract methods that aren't defined until they are inherited.

A class inheriting from an abstract class must implement all the abstract methods.

The **abstract** keyword is used to create an abstract class or an abstract method.

For example:

CODE PLAYGROUND PHP

```php
<?php
abstract class Fruit {
  private $color;

  abstract public function eat();

  public function setColor($c) {
    $this->color = $c;
  }
}

class Apple extends Fruit {
  public function eat() {
    echo "Omnomnom";
  }
}

$obj = new Apple();
$obj->eat();
?>
```

Tap to edit |

Abstract functions can only appear in an abstract class.

What is the output of the following code?

```
abstract class Calc {
  abstract public function calculate($param);
  protected function getConst() { return 4; }
}
class FixedCalc extends Calc {
  public function calculate($param) {
    return $this->getConst() + $param;
  }
}
$obj = new FixedCalc();
echo $obj->calculate(38);
```

Type

The static Keyword

The PHP **static** keyword defines static properties and static methods.
A static property/method of a class can be accessed without creating an object of that class.

A static property or method is accessed by using the **scope resolution operator ::** between the class name and the property/method name.

For example:

CODE PLAYGROUND PHP

```php
<?php
class myClass {
  static $myStaticProperty = 42;
}

echo myClass::$myStaticProperty;
?>

Tap to edit
```

The **self** keyword is needed to access a static property from a static method in a class definition.

For example:

CODE PLAYGROUND PHP

```php
<?php
class myClass {
  static $myProperty = 42;
  static function myMethod() {
    echo self::$myProperty;
  }
}

myClass::myMethod();
?>
```

Tap to edit |

Objects of a class cannot access static properties in the class but they can access static methods.

Fill in the blanks to declare a static property 'name' and access it from a static method in a class definition.

```
class Singer {

    Type    $name = "Jone";

  static function toSing() {

    echo    Type    ::$name;

  }

}

Singer    Type    toSing();
```

The final Keyword

The PHP **final** keyword defines methods that cannot be overridden in child classes. Classes that are defined final cannot be inherited.

This example demonstrates that a final method cannot be overridden in a child class:

CODE PLAYGROUND **PHP**

```php
<?php
class myClass {
  final function myFunction() {
    echo "Parent";
  }
}
// ERROR because a final method cannot be overridden in child classes.
class myClass2 extends myClass {
  function myFunction() {
    echo "Child";
  }
}
?>
```

Tap to edit

The following code demonstrates that a final class cannot be inherited:

CODE PLAYGROUND **PHP**

```php
<?php
final class myFinalClass {
}

// ERROR because a final class cannot be inherited.
class myClass extends myFinalClass {
}
?>

Tap to edit
```

Unlike classes and methods, properties cannot be marked **final**.

What does OOP stand for?

- Object Oriented Process
- Object Oriented Programming
- Out of Print
- Object Oriented Principles

The process of creating an object of a class is called:

declaration

creation

instantiation

execution

Which keyword is used to refer to properties or methods within the class itself?

- `protected`
- `public`
- `this`
- `final`