

The Basics

class

Basic class definitions begin with the keyword *class*, followed by a class name, followed by a pair of curly braces which enclose the definitions of the properties and methods belonging to the class.

new

To create an instance of a class, the ***new*** keyword must be used. An object will always be created unless the object has a constructor defined that throws an exception on error.

Properties and methods

Class properties and methods live in separate "namespaces", so it is possible to have a property and a method with the same name. Referring to both a property and a method has the same notation, and whether a property will be accessed or a method will be called, solely depends on the context, i.e. whether the usage is a variable access or a function call.

extends

A class can inherit the methods and properties of another class by using the keyword ***extends*** in the class declaration. It is not possible to extend multiple class is; a class can only inherit from one base class.

The inherited methods and properties can be overridden by re-declaring them with the same name defined in the parent class. However, if the parent class has defined a method as final, that method may not be overridden. It is possible to access the overridden methods or static properties by referencing them with parent:

: class

Since PHP 5.5, the ***class*** keyword is also used for class name resolution. You can get a string containing the fully qualified name of the ***Classmate*** class by using ***Class Name::class***.

Properties

Class member variables are called "properties". You may also see them referred to using other terms such as "attributes" or "fields", but for the purposes of this reference we will use "properties". They are defined by using one of the keywords *public*, *protected*, or *private*, followed by a normal variable declaration. This declaration may include an initialization, but this initialization must be a constant value--that is, it must be able to be evaluated at compile time and must not depend on run-time information in order to be evaluated.

```
<?php
class SimpleClass{
// valid as of PHP 5.6.0:
public $var1='hello '. 'geetanjali';
// valid as of PHP 5.3.0:
public $var2=<<<college
hello Geetanjali
college;
// valid as of PHP 5.6.0:
public $var3=1+2;
public $var7=array(true,false);
// valid as of PHP 5.3.0:
public $var8=<<<'geet'
hello Geetanjali College
geet;
}
$smpClass= new SimpleClass();
echo $smpClass->var1."<BR>";
echo $smpClass->var2."<BR>";
echo $smpClass->var7[0]."<BR>";
echo $smpClass->var8."<BR>";
?>
```

OP

hello geetanjali
hello Geetanjali
1
hello Geetanjali College

Class Constants

It is possible to define constant values on a per-class basis remaining the same and unchangeable. Constants differ from normal variables in that you don't use the \$ symbol to declare or use them. The default visibility of class constants is **public**. It's also possible for interfaces to have **constants**.

```
<?php
class MyClass{
    const geet='constant value';
    function showConstant(){
        echo self::geet;
    }
}
echo MyClass::geet;
$classname="MyClass";
echo $classname::geet;// As of PHP 5.3.0

$class=new MyClass();
$class->showConstant();
echo $class::geet;// As of PHP 5.3.0
?>
```

OP:constant valueconstantvalueconstantvalueconstant value

Autoloading Classes

Many developers writing object-oriented applications create one PHP source file per class definition. One of the biggest annoyances is having to write a long list of needed includes at the beginning of each script (one for each class).

```
<?php
spl_autoload_register( function ($class _name) {
    include $class _name . '.php';
});$obj = new MyClass1();
$obj2 = new MyClass2();
?>
```

In PHP 5, this is no longer necessary. The spl_autoload_register () function registers any number of autoloaders, enabling for classes and interfaces to be automatically loaded if they are currently not defined. By registering autoloaders, PHP is given a last chance to load the class or interface before it fails with an error.

Constructors and Destructors

Constructor

`void __construct ([mixed $args = "" [, $...]])`

PHP 5 allows developers to declare constructor methods for classes. Classes which have a constructor method call this method on each newly-created object, so it is suitable for any initialization that the object may need before it is used.

For backwards compatibility with PHP 3 and 4, if PHP cannot find a `__construct()` function for a given class, it will search for the old-style constructor function, by the name of the class. Effectively, it means that the only case that would have compatibility issues is if the class had a method named `construct()` which was used for different semantics.

Unlike with other methods, PHP will not generate an `E_STRICT` level error message when `__construct()` is overridden with different parameters than the `parent::__construct()` method has.

<pre><?php class BaseClass { function __construct() { print "In BaseClass constructor\n"; } } class SubClass extends BaseClass { function __construct() { parent::__construct(); print "In SubClass constructor\n"; } } class OtherSubClass extends BaseClass { // inherits BaseClass's constructor } // In BaseClass constructor \$obj = new BaseClass (); // In BaseClass constructor // In SubClass constructor \$obj = new SubClass (); // In BaseClass constructor \$obj = new OtherSubClass (); ?></pre>	<p>OP:</p> <p>In BaseClass constructor</p> <p>In BaseClass constructor</p> <p>In SubClass constructor</p> <p>In BaseClass constructor</p>
--	---

Destructor

`void __destruct (void)`

PHP 5 introduces a destructor concept similar to that of other object-oriented languages, such as C++. The destructor method will be called as soon as there are no other references to a particular object, or in any order during the shutdown sequence.

Like constructors, parent destructors will not be called implicitly by the engine. In order to run a parent destructor, one would have to explicitly call `parent::__destruct()` in the destructor body. Also like constructors, a child class may inherit the parent's destructor if it does not implement one itself.

The destructor will be called even if script execution is stopped using `exit()`. Calling `exit()` in a destructor will prevent the remaining shutdown routines from executing.

```
<?php
class new Account {
function happy(){
echo "<br>hello<br>";
}
function __construct(){
    echo "Creating new Savings Account<BR>";
    print "Adding MIN Balance : 5000<BR>";
    print " ----- <BR>";
    print " ----- <BR>";
    $this->name="Mr. ABC";
}
function __destruct(){
    print "Auto Generating Complete. <BR>Removing Traces
for ".$this->name;
    }
}
$obj= new new Account();
$obj->happy();
?>
```

Output

Creating new Savings Account
Adding MIN Balance : 5000

hello
Auto Generating Complete.
Removing Traces for Mr. ABC

Visibility

The visibility of a property, a method or (as of PHP 7.1.0) a constant can be defined by prefixing the declaration with the keywords `public`, `protected` or `private`. Class members declared `public` can be accessed everywhere. Members declared `protected` can be accessed only within the class itself and by inheriting and parent classes. Members declared as `private` may only be accessed by the class that defines the member.

Property Visibility

Class properties must be defined as `public`, `private`, or `protected`. If declared using `var`, the property will be defined as `public`.

```
<?php
class MyClass{
    public $a='Public';
    protected $b='Protected';
    private $c='Private';

    function printHello() {
        echo $this->a;
        echo $this->b;
        echo $this->c;
    }
}

$obj=new MyClass();
echo $obj->a; // Works
echo $obj->b; // Fatal Error
echo $obj->c; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private

class MyClass2 extends MyClass{
    // We can redeclare the public and protected properties, but not private
    public $a='Public 2';
    protected $b='Protected 2';

    function printHello() {
        echo $this->a;
        echo $this->b;
        echo $this->c;
    }
}

$obj2=new MyClass2();
echo $obj2->a; // Works
echo $obj2->b; // Fatal Error
echo $obj2->c; // Undefined
$obj2->printHello(); // Shows Public 2, Protected 2, Undefined
?>
```



Method Visibility

Class methods may be defined as public,private, or protected. Methods declared without any explicit visibility keyword are defined as public.

```
<?php
```

```
class MyClass{
// Declare a public constructor
public function __construct() {}

// Declare a public method
public function MyPublic() {}

// Declare a protected method
protected function MyProtected() {}

// Declare a private method
private function MyPrivate() {}

// This is public
function Foo() {
$this->MyPublic();
$this->MyProtected();
$this->MyPrivate();
}
}

$myclass= new MyClass;
$myclass->MyPublic(); // Works
$myclass->MyProtected(); // Fatal Error
$myclass->MyPrivate(); // Fatal Error
$myclass->Foo(); // Public,Protected and Private work
class MyClass2 extends MyClass{
// This is public
function Foo2()
{
$this->MyPublic();
$this->MyProtected();
$this->MyPrivate(); // Fatal Error
}
}

$myclass2=new MyClass2;
$myclass2->MyPublic(); // Works
$myclass2->Foo2(); // Public and Protected work, not Private

class Bar {
```



```

public function test() {
    $this->testPrivate();
    $this->testPublic();
}

public function testPublic() {
    echo "Bar::testPublic\n";
}

private function testPrivate() {
    echo "Bar::testPrivate\n";
}
}

class Foo extends Bar {
    public function testPublic() {
        echo "Foo::testPublic\n";
    }

    private function testPrivate() {
        echo "Foo::testPrivate\n";
    }
}

$myFoo = new Foo();
$myFoo->test(); // Bar::testPrivate
// Foo::testPublic
?>

```



Constant Visibility

As of PHP 7.1.0, class constants may be defined as public, private, or protected. Constants declared without any explicit visibility keyword are defined as public.

```

<?php
class MyClass
{
    // Declare a public constant
    public const MY_PUBLIC = 'public';

    // Declare a protected constant
    protected const MY_PROTECTED = 'protected';

    // Declare a private constant
    private const MY_PRIVATE = 'private';

    public function foo()
    {

```



```

echo self::MY_PUBLIC ;
echo self::MY_PROTECTED ;
echo self::MY_PRIVATE ;
}
}

$myclass= new MyClass();
MyClass::MY_PUBLIC ; // Works
MyClass::MY_PROTECTED ; // Fatal Error
MyClass::MY_PRIVATE ; // Fatal Error
$myclass->foo(); // Public, Protected and Private work
class MyClass2 extends MyClass
{
    // This is public
    function foo2()
    {
        echo self::MY_PUBLIC ;
        echo self::MY_PROTECTED ;
        echo self::MY_PRIVATE ; // Fatal Error
    }
}

$myclass2=new MyClass2;
echo MyClass2::MY_PUBLIC ; // Works
$myclass2->foo2(); // Public and Protected work, not Private
?>

```

Visibility from other objects

Objects of the same type will have access to each other's private and protected members even though they are not the same instances. This is because the implementation specific details are already known when inside those objects.

```

<?php
class Test{
    private $foo;
    public function __construct($foo){
        $this->foo=$foo;
    }

    private function bar(){
        echo 'Accessed the private method.';
    }

    public function baz(Test $other){
        // We can change the private property:
        $other->foo='hello';
        var_dump($other->foo);
    }
}

```

// We can also call the private method:

```
$other->bar();
}
}
$test= new Test('test');
$test->baz( new Test('other'));
?>
```

Output: string(5) "hello"Accessed the private method.

Object Inheritance

Inheritance is a well-established programming principle, and PHP makes use of this principle in its object model. This principle will affect the way many classes and objects relate to one another.

For example, when you extend a class, the subclass inherits all of the public and protected methods from the parent class. Unless a class overrides those methods, they will retain their original functionality.

This is useful for defining and abstracting functionality, and permits the implementation of additional functionality in similar objects without the need to re-implement all of the shared functionality.

```
<?php
class Foo{
public function printItem($string){
echo 'Foo: '.$string. PHP_EOL;
}

public function printPHP(){
echo 'PHP is great.'. PHP_EOL;
}
}
```

```
class Bar extends Foo{
public function printItem($string)
{
echo 'Bar: '.$string. PHP_EOL;
}
}
```

```
$foo= new Foo();
$bar= new Bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP();      // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP();      // Output: 'PHP is great'
?>
```



Scope Resolution Operator (::)

The Scope Resolution Operator (also called PaamayimNekudotayim) or in simpler terms, the double colon, is a token that allows access to static, constant, and overridden properties or methods of a class.

When referencing these items from outside the class definition, use the name of the class.

PaamayimNekudotayim would, at first, seem like a strange choice for naming a double-colon. However, while writing the Zend Engine 0.5 (which powers PHP 3), that's what the Zend team decided to call it. It actually does mean double-colon - in Hebrew!

```
<?php
class MyClass {
    const CONST_VALUE='A constant value';
}

$class name='MyClass';
echo $class name::CONST_VALUE; // As of PHP 5.3.0

echo MyClass::CONST_VALUE;
?>
```

OutPut: A constant valueA constant value

Static Keyword

Declaring class properties or methods as static makes them accessible without needing an instantiation of the class. A property declared as static cannot be accessed with an instantiated class object (though a static method can).

For compatibility with PHP 4, if no visibility declaration is used, then the property or method will be treated as if it was declared as public.

Static methods

Because static methods are callable without an instance of the object created, the pseudo-variable \$this is not available inside the method declared as static.

```
<?php
class Foo {
    public static function aStaticMethod() {
        // ...
    }
}

Foo::aStaticMethod();
$class name='Foo';
$class name::aStaticMethod(); // As of PHP 5.3.0
?>
```

Static properties

Static properties cannot be accessed through the object using the arrow operator ->.

Like any other PHP static variable, static properties may only be initialized using a literal or constant before PHP 5.6; expressions are not allowed. In PHP 5.6 and later, the same rules apply as **const** expressions: some limited expressions are possible, provided they can be evaluated at compile time.

As of PHP 5.3.0, it's possible to reference the class using a variable. The variable's value cannot be a keyword (e.g. self, parent and static).

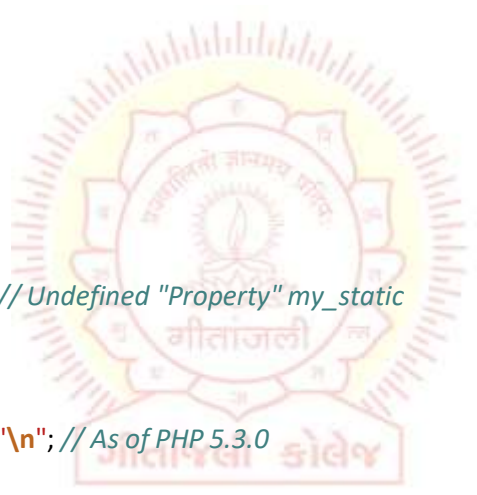
```
<?php
class Foo{
    public static $my_static='foo';

    public function staticValue() {
        return self::$my_static;
    }
}
class Bar extends Foo{
    public function fooStatic() {
        return parent::$my_static;
    }
}
print Foo::$my_static."\n";

$foo= new Foo();
print $foo->staticValue() . "\n";
print $foo->my_static. "\n"; // Undefined "Property" my_static

print $foo::$my_static. "\n";
$class_name='Foo';
print $class_name::$my_static. "\n"; // As of PHP 5.3.0

print Bar::$my_static. "\n";
$bar= new Bar();
print $bar->fooStatic() . "\n";
?>
```

A circular watermark logo for Geetanjali College is centered in the background. It features a central lamp (diya) with a flame, surrounded by text in Hindi. The outer ring of the logo contains the words 'Geetanjali College' in English and Hindi. Below the circle, the name 'Geetanjali' is written in a stylized font.

Class Abstraction

PHP 5 introduces abstract classes and methods. Classes defined as abstract may not be instantiated, and any class that contains at least one abstract method must also be abstract. Methods defined as abstract simply declare the method's signature - they cannot define the implementation.

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 (or a less restricted) visibility. For example, if the abstract method is defined as protected, the function implementation must be defined as either protected or public, but not private. Furthermore the signatures of the methods must match, i.e. the type hints and the number of required arguments must be the same. For example, if the child class defines an optional argument, where the abstract method's signature does not, there is no conflict in the signature. This also applies to constructors as of PHP 5.4. Before 5.4 constructor signatures could differ.

```
<?php
abstract class AbstractClass{

    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut() {
        print $this->getValue() . "<br>";
    }

    class ConcreteClass 1 extends AbstractClass {
        protected function getValue() {
            return "ConcreteClass 1";
        }
        public function prefixValue($prefix) {
            return "{$prefix}ConcreteClass 1";
        }
    }

    class ConcreteClass 2 extends AbstractClass {
        public function getValue() {
            return "ConcreteClass 2";
        }
        public function prefixValue($prefix) {
            return "{$prefix}ConcreteClass 2";
        }
    }

    $class 1= new ConcreteClass 1;
    $class 1->printOut();
    echo $class 1->prefixValue("FOO_") . "<br>";

    $class 2= new ConcreteClass 2;
    $class 2->printOut();
    echo $class 2->prefixValue("FOO_") . "<br>";
?>
```

The above example will output:

```
ConcreteClass 1
FOO_ConcreteClass 1

ConcreteClass 2
FOO_ConcreteClass 2
```

Object Interfaces

Object interfaces allow you to create code which specifies which methods a class must implement, without having to define how these methods are implemented.

Interfaces are defined in the same way as a class, but with the interface keyword replacing the class keyword and without any of the methods having their contents defined. All methods declared in an interface must be public; this is the nature of an interface.

Note that it is possible to declare a constructor in an interface, what can be useful in some contexts, e.g. for use by factories.

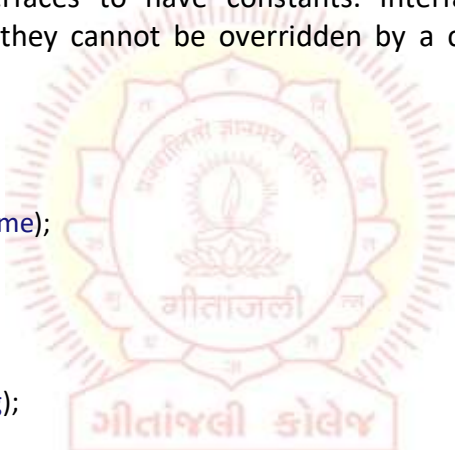
implements

To implement an interface, the implements operator is used. All methods in the interface must be implemented within a class; failure to do so will result in a fatal error. Classes may implement more than one interface if desired by separating each interface with a comma.

Constants

It's possible for interfaces to have constants. Interface constants work exactly like class constants except they cannot be overridden by a class /interface that inherits them.

```
<?php
interface car{
    function setModel($name);
    function getModel();
    const abc='CONST';
}
interface petrol{
    function setPetrol($avg);
    function getPetrol();
}
class minicar implements car, petrol{
    public $model;
    public $average;
    function setModel($name){
        $this->model=$name;
    }
    function getModel(){
        return $this->model;
    }
    function setPetrol($avg){
        $this->average=$avg;
    }
    function getPetrol(){
        return $this->average;
    }
}
$test=new minicar();
$test->setModel('Tata Nano');
```



```
$test->setPetrol('15');
echo car::abc;
echo $test->getModel();
echo $test->getPetrol();
?>
```

OUTPUT : CONSTTata Nano15

Anonymous classes

Support for anonymous classes was added in PHP 7. Anonymous classes are useful when simple, one-off objects need to be created.

```
<?php
```

```
$Anonymous=newclass{
    private $readOnly = 'Anonymous Class';

    public function printOnly(){
        return $this->readOnly;
    }
};
echo $Anonymous->printOnly();

?>
```

OUTPUT : Anonymous Class

Overloading

Overloading in PHP provides means to dynamically "create" properties and methods. These dynamic entities are processed via magic methods one can establish in a class for various action types.

The overloading methods are invoked when interacting with properties or methods that have not been declared or are not visible in the current scope. The rest of this section will use the terms "inaccessible properties" and "inaccessible methods" to refer to this combination of declaration and visibility.

All overloading methods must be defined as public.

Property overloading

```
public void __set ( string $name, mixed $value )
public mixed __get ( string $name )
public bool __isset ( string $name )
public void __unset ( string $name )
```

__set() is run when writing data to inaccessible properties.

__get() is utilized for reading data from inaccessible properties.

__isset() is triggered by calling isset() or empty() on inaccessible properties.

__unset() is invoked when unset() is used on inaccessible properties.

The \$name argument is the name of the property being interacted with.

The __set() method's \$value argument specifies value the \$name'ed property should set to.

Property overloading only works in object context. These magic methods will not be triggered in static context. Therefore these methods should not be declared static. As of PHP 5.3.0, a warning is issued if one of the magic overloading methods is declared static.

<pre> <?php class PropertyTest { private \$data=array(); public function __set(\$name, \$value) { echo "Setting '\$name' to '\$value'\n"; \$this->data[\$name]=\$value; } public function __get(\$name) { echo "Getting '\$name'\n"; return \$this->data[\$name]; } public function __isset(\$name) { echo "Is '\$name' set?\n"; return isset(\$this->data[\$name]); } public function __unset(\$name) { echo "Unsetting '\$name'\n"; unset(\$this->data[\$name]); } } echo "<pre>\n"; \$obj=new PropertyTest; \$obj->a=1; echo \$obj->a."\n\n"; var_dump(isset(\$obj->a)); unset(\$obj->a); var_dump(isset(\$obj->a)); ?> </pre>	<p>OutPut</p> <p>Setting 'a' to '1'</p> <p>Getting 'a'</p> <p>1</p> <p>Is 'a' set?</p> <p>bool(true)</p> <p>Unsetting 'a'</p> <p>Is 'a' set?</p> <p>bool(false)</p>
--	---

Method overloading

```
public mixed __call ( string $name, array $arguments )
```

```
public static mixed __callStatic ( string $name, array $arguments )
```

__call() is triggered when invoking inaccessible methods in an object context.

__callStatic() is triggered when invoking inaccessible methods in a static context.

The \$name argument is the name of the method being called. The \$arguments argument is an enumerated array containing the parameters passed to the \$name'd method.

<pre><?php class Overloading{ function __call(\$name,\$args) { echo"\$name does not exists."; echo"<pre>"; print_r (\$args); echo"</pre>"; } static function __callStatic(\$name,\$args){ echo"\$name does not exists."; echo"<pre>"; print_r (\$args); echo"</pre>"; } } \$test=new Overloading; \$test->anything('123',123); \$test::anything('123',123); ?></pre>	<p>OutPut</p> <p>anything does not exists.</p> <p>Array</p> <pre>([0] => 123 [1] => 123)</pre> <p>anything does not exists.</p> <p>Array</p> <pre>([0] => 123 [1] => 123)</pre>
---	---

Object Iteration

PHP 5 provides a way for objects to be defined so it is possible to iterate through a list of items, with, for example a foreach statement. By default, all visible properties will be used for the iteration.

<pre><?php class iterat{ public \$abc="Hello"; public \$def="Hello"; public \$hij="Hello"; private \$klm="Hello"; protected \$nop="hello"; function iterate(){ foreach(\$this as \$key=>\$value) print"\$key =>\$value
"; } } \$test=new iterat;</pre>	<p>Output</p> <pre>abc => Hello def => Hello hij => Hello klm => Hello nop => hello</pre>
---	--

```

echo"<pre>";
$test->iterate();
/*foreach($test as $key => $value){
    echo "$key => $value <br>";
}*/
//var_export(get_object_vars($test));
?>

```

Example Interfaceliterator

```

<?php
class titerator implements Iterator {
    private $myArray;

    public function __construct( $givenArray ) {
        $this->myArray=$givenArray;
    }
    function rewind() {
        var_dump(__METHOD__);

        return reset($this->myArray);
    }
    function current() {
        var_dump(__METHOD__);
        return current($this->myArray);
    }
    function key() {
        var_dump(__METHOD__);
        return key($this->myArray);
    }
    function next() {
        var_dump(__METHOD__);
        return next($this->myArray);
    }
    function valid() {
        var_dump(__METHOD__);
        return key($this->myArray) !== null;
    }
}
$it=new titerator(['Hello','I','AM','STUDENT']);
echo"<pre>";
foreach($itas $key=>$value) {
    echo "$key => $value";
    echo "<br>";
}
?>

```

OUTPUT

```

string(17) "titerator::rewind"
string(16) "titerator::valid"
string(18) "titerator::current"
string(14) "titerator::key"
0 => Hello
string(15) "titerator::next"
string(16) "titerator::valid"
string(18) "titerator::current"
string(14) "titerator::key"
1 => I
string(15) "titerator::next"
string(16) "titerator::valid"
string(18) "titerator::current"
string(14) "titerator::key"
2 => AM
string(15) "titerator::next"
string(16) "titerator::valid"
string(18) "titerator::current"
string(14) "titerator::key"
3 => STUDENT
string(15) "titerator::next"
string(16) "titerator::valid"

```

Example IteratorAggregate

<?php

```
class Collection implements IteratorAggregate{
    protected $items;
    public function __construct(array $items){
        $this->items=$items;
    }

    function getIterator() {
        return new ArrayIterator($this->items);
    }
}
```

```
class User{
    public $FirstName;
    public $LastName;
    public $email;
}
```

```
$user1=new User;
$user2=new User;
```

```
$user1->email='abc@gmail.com';
$user1->FirstName="ABC";
$user1->LastName="XYZ";
```

```
$user2->email='def@gmail.com';
$user2->FirstName="DEF";
$user2->LastName="XYZ";
```

```
$users=new Collection([$user1,$user2]);
```

```
foreach($users as $user)
{
    echo "$user->email";
    echo "$user->FirstName";
    echo "$user->LastName<br>";
}
```

?>

OUTPUT

abc@gmail.comABCXYZ

def@gmail.comDEFXYZ



Magic Methods

The function names

`__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` and `__debugInfo()`

are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them.

`__toString()`

```
public string __toString ( void )
```

The `__toString()` method allows a class to decide how it will react when it is treated like a string. For example, what `echo $obj;` will print. This method must return a string, as otherwise a fatal `E_RECOVERABLE_ERROR` level error is emitted.

It is worth noting that before PHP 5.2.0 the `__toString()` method was only called when it was directly combined with `echo` or `print`. Since PHP 5.2.0, it is called in any string context (e.g. in `printf()` with `%s` modifier) but not in other types contexts (e.g. with `%d` modifier). Since PHP 5.2.0, converting objects without `__toString()` method to string would cause `E_RECOVERABLE_ERROR`.

`__invoke()`

```
mixed __invoke ( [ $... ] )
```

The `__invoke()` method is called when a script tries to call an object as a function .

`__set_state()`

```
static object __set_state ( array $properties )
```

This static method is called for classes exported by `var_export()` since PHP 5.1.0. The only parameter of this method is an array containing exported properties in the form `array('property' => value, ...)`.

`__debugInfo()`

```
array __debugInfo ( void )
```

This method is called by `var_dump()` when dumping an object to get the properties that should be shown. If the method isn't defined on an object, then all public, protected and private properties will be shown.

This feature was added in PHP 5.6.0.

```
<?php
```

```
class MagicMethods{
    public $value1;
    public $value2;
```

```
/* __construct() __destruct() __get() __set() __isset() __unset() __call() __callStatic()
__invoke() clone() __toString() __sleep() __wakeup() __set_state() __debugInfo() */
```

```

function __invoke(){
    echo "<Br>This is completely out of Bounds.";
}
function __clone(){
    echo "<br>I M John\'s Copy";
}
function __toString(){
    return "<br>Here goes notihng";
}
function __debugInfo(){
    echo "<pre>";
    return ["Hello", 'sdsdsd', "I am new "];
    echo "</pre>";
}
function __set_state($args){
    $test3=new MagicMethods;
    $test3->value1=$args['value1'];
    $test3->value2=$args['value2'];
    return $test3;
}
}
$test=new MagicMethods;
$test->value1=15;
$test->value2='sdsdsd';
$str=var_export($test,true);
eval("$str.");
var_dump($str);
var_dump(new MagicMethods);
//echo serialize($test);
$test();
$test1=clone $test;
echo $test1;

```



?>

OUTPUT

```

string(79) "MagicMethods::__set_state(array( 'value1' => 15, 'value2' => 'sdsdsd', ))"
object(MagicMethods)#2 (3) {
    [0]=>
    string(5) "Hello"
    [1]=>
    string(6) "sdsdsd"
    [2]=>
    string(9) "I am new "
}

```

This is completely out of Bounds.

I M John's Copy

Here goes notihng

__sleep() and __wakeup()

```
public array __sleep ( void )
void __wakeup ( void )
```

serialize() checks if your class has a function with the magic name **__sleep()**. If so, that function is executed prior to any serialization. It can clean up the object and is supposed to return an array with the names of all variables of that object that should be serialized. If the method doesn't return anything then NULL is serialized and **E_NOTICE** is issued.

The intended use of **__sleep()** is to commit pending data or perform similar cleanup tasks. Also, the function is useful if you have very large objects which do not need to be saved completely.

Conversely, **unserialize()** checks for the presence of a function with the magic name **__wakeup()**. If present, this function can reconstruct any resources that the object may have.

The intended use of **__wakeup()** is to reestablish any database connections that may have been lost during serialization and perform other reinitialization tasks.

```
<?php
classdemoSleepWakeup {

public$arrayM=array(1, 2, 3, 4);

publicfunction__sleep() {
returnarray('arrayM');
}

publicfunction__wakeup() {
echo"array restarted ";
}
}

$obj=newdemoSleepWakeup();
$serializedStr=serialize($obj);
echo"<pre>";
var_dump($serializedStr);
var_dump(unserialize($serializedStr));
echo"</pre>";
?>
```

OutPut

```
string(78)
"O:15:"demoSleepWakeup":1:{s:6:"arrayM";a:4:{i:0
;i:1;i:1;i:2;i:2;i:3;i:3;i:4;}}"
array restarted object(demoSleepWakeup)#2 (1) {
["arrayM"]=>
array(4) {
[0]=>
int(1)
[1]=>
int(2)
[2]=>
int(3)
[3]=>
int(4)
}
}
```

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.

Example # Final methods example

```
<?php
class BaseClass {
    public function test() {
        echo "BaseClass::test() called\n";
    }

    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
    public function moreTesting() {
        echo "ChildClass::moreTesting() called\n";
    }
}
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()
?>
```



Example #2 Final class example

```
<?php
final class BaseClass {
    public function test() {
        echo "BaseClass::test() called\n";
    }

    // Here it doesn't matter if you specify the function as final or not
    final public function moreTesting() {
        echo "BaseClass::moreTesting() called\n";
    }
}

class ChildClass extends BaseClass {
}
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)
?>
```

Object Cloning

Creating a copy of an object with fully replicated properties is not always the wanted behavior. A good example of the need for copy constructors, is if you have an object which represents a GTK window and the object holds the resource of this GTK window, when you create a duplicate you might want to create a new window with the same properties and have the new object hold the resource of the new window. Another example is if your object holds a reference to another object which it uses and when you replicate the parent object you want to create a new instance of this other object so that the replica has its own separate copy.

An object copy is created by using the clone keyword (which calls the object's `__clone()` method if possible). An object's `__clone()` method cannot be called directly.

`$copy_of_object = clone $object;`

When an object is cloned, PHP 5 will perform a shallow copy of all of the object's properties. Any properties that are references to other variables will remain references.

`void __clone (void)`

Once the cloning is complete, if a `__clone()` method is defined, then the newly created object's `__clone()` method will be called, to allow any necessary properties that need to be changed.



```
<?php
class BOX {
    public $name="Hello";
}

$box=new BOX();
// $box->name= "Hello";

$box_ref=$box;
//DUPLICATE
// $box_ref->name="Sorry";

$box_clone=clone $box;
// $box_clone->name="NO";

$box_change=clone $box;
// $box_change->name="Hello";

$another_box=new BOX();
// $another_box->name="New";

//Comparison
echo $box==$box_ref?'true<br>':'false<br>';           //true
echo $box==$box_clone?'true<br>':'false<br>';         //true
echo $box==$box_change?'true<br>':'false<br>';        //true
```



```

echo $box==$another_box?'true<br>':'false<br>';           //true
//Identity Comparison
echo $box=== $box_ref?'true<br>':'false<br>';           //true
echo $box=== $box_clone?'true<br>':'false<br>';          //false
echo $box=== $box_change?'true<br>':'false<br>';         //false
echo $box=== $another_box?'true<br>':'false<br>'; //false

```

?>

Comparing Objects

When using the comparison operator (==), object variables are compared in a simple manner, namely: Two object instances are equal if they have the same attributes and values (values are compared with ==), and are instances of the same class.

When using the identity operator (===), object variables are identical if and only if they refer to the same instance of the same class.

Type Hinting

Type declarations

Note:

Type declarations were also known as type hints in PHP 5.

Type declarations allow functions to require that parameters are of a certain type at call time. If the given value is of the incorrect type, then an error is generated: in PHP 5, this will be a recoverable fatal error, while PHP 7 will throw a `TypeError` exception.

To specify a type declaration, the type name should be added before the parameter name. The declaration can be made to accept NULL values if the default value of the parameter is set to NULL.

Valid types

Type	Description	Min. PHP version
Class /interface name	The parameter must be an instance of the given class or interface name.	PHP 5.0.0
self	The parameter must be an instance of the same class as the one the method is defined on. This can only be used on class and instance methods.	PHP 5.0.0
array	The parameter must be an array.	PHP 5.1.0
callable	The parameter must be a valid callable.	PHP 5.4.0
bool	The parameter must be a boolean value.	PHP 7.0.0
float	The parameter must be a floating point number.	PHP 7.0.0
int	The parameter must be an integer.	PHP 7.0.0
string	The parameter must be a string.	PHP 7.0.0
Iterable	The parameter must be either an array or an instance of Traversable.	PHP 7.1.0

Strict typing

By default, PHP will coerce values of the wrong type into the expected scalar type if possible. For example, a function that is given an integer for a parameter that expects a string will get a variable of type string.

It is possible to enable strict mode on a per-file basis. In strict mode, only a variable of exact type of the type declaration will be accepted, or a `TypeError` will be thrown. The only exception to this rule is that an integer may be given to a function expecting a float. Function calls from within internal functions will not be affected by the `strict_types` declaration.

Type declarations


```
<?php
class Song{
    public $title;
    public $lyrics;
}
class Data{
    function sing(Song $song){
        $this->d_song=$song;
        echo "Best song of the year :".$this->d_song->title."<br>";
        echo "<p>".$this->d_song->lyrics." </p>";
    }
}
$data=new Data();
$hit=new Song();
$hit->title="VandeMatartam";
$hit->lyrics="Sujalamsuphalammalayajasheetalam";
$data->sing($hit);

//sing($hit);
?>
```



OUTPUT

Best song of the year :VandeMatartam
Sujalamsuphalammalayajasheetalam



Late Static Bindings

Late static bindings tries to solve that limitation by introducing a keyword that references the class that was initially called at runtime. It was decided not to introduce a new keyword but rather use `static` that was already reserved.

Example # static:: simple usage

```
<?php
class Department{
    protected static $x=10;
    public function myFunction(){
        echo static::$x;
    }
}
class HR extends Department{
    protected static $x=20;
}
```

```

class Marketing extends Department{
    protected static $x=30;
}
$test=new Marketing;
$test->myFunction();
?>

```

The above example will output: 30

Objects and References

One of the key-points of PHP 5 OOP that is often mentioned is that "objects are passed by references by default". This is not completely true. This section rectifies that general thought using some examples.

A PHP reference is an alias, which allows two different variables to write to the same value. As of PHP 5, an object variable doesn't contain the object itself as value anymore. It only contains an object identifier which allows object assessors to find the actual object. When an object is sent by argument, returned or assigned to another variable, the different variables are not aliases: they hold a copy of the identifier, which points to the same object.

```

<?php
class objRef{
    public $hi=5;
}
$a=new objRef();

$b=$a; // Copy.
// $b=&$a; // Reference.

echo "a=".$a->hi."<br>";
echo "b=".$b->hi."<br>";

echo "a=".$a->hi=3."<br>";
echo "b=".$b->hi=10."<br>";

$a=null;

echo "a=".$a->hi."<br>";
echo "b=".$b->hi."<br>";

?>

```

Output:

```

a=5
b=5
a=3
b=10

```

Notice: Trying to get property of non-object

```

a=
b=10

```

BOOTSTRAP

What is Bootstrap?

Bootstrap is an Open Source product from **Mark Otto** and **Jacob Thornton** who, when initially released were both employees at **Twitter**. There was a need to standardize the front end toolsets of engineers across the company. In the launch blog post, Mark Otto introduces the project like this:

In the earlier days of Twitter, engineers used almost any library they were familiar with to meet front-end requirements. Inconsistencies among the individual applications made it difficult to scale and maintain them. Bootstrap began as an answer to these challenges and quickly accelerated during Twitter's first Hack week. By the end of Hackweek, we had reached a stable version that engineers could use across the company.

— Mark Otto
<https://dev.twitter.com/blog/bootstrap-twitter>

Since Bootstrap launched in August, 2011 it has taken off in popularity. It has evolved away from being an entirely CSS driven project to include a host of JavaScript plugins, and icons that go hand in hand with forms and buttons. At its base, it allows for responsive web design, and features a robust 12 column, 940px wide grid. One of the highlights is the build tool on <http://getbootstrap.com> website where you can customize the build to suit your needs, choosing what CSS and JavaScript features that you want to include on your site. All of this, allows front-end web development to be catapulted forward, building on a stable foundation of forward-looking design, and development. Getting started with Bootstrap is as simple as dropping some CSS and JavaScript into the root of your site.

Starting a project new, Bootstrap comes with a handful of useful elements to get you started. Normally, when I start a project, I start with tools like Eric Meyer's CSS reset, and get going on my web project. With Bootstrap, you just need to include the `bootstrap.css` CSS file, and optionally the `bootstrap.js` JavaScript file into your website and you are ready to go.

File Structure

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   └── bootstrap.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
├── img/
│   ├── glyphs-halflings.png
│   └── glyphs-halflings-white.png
└── README.md
```

The Bootstrap download includes three folders: `css`, `js` and `img`. For simplicity, add these to the root of your project. Included are also minified versions of the CSS and JavaScript. Both the uncompressed and the minified versions do not need to be included. For the sake of brevity, I use the uncompressed during development, and then switch to the compressed version in production.

Basic HTML Template

Normally, a web project looks something like this:

Basic HTML Layout.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap 101 Template</title>
  </head>
  <body>
    <h1>Hello, world!</h1>
  </body>
</html>
```

With Bootstrap, we simply include the link to the CSS stylesheet, and the Javascript.

Basic Bootstrap Template.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Bootstrap 101 Template</title>
    <link href="css/bootstrap.min.css" rel="stylesheet">
  </head>
  <body>
    <h1>Hello, world!</h1>
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

Global Styles

With Bootstrap, a number of items come prebuilt. Instead of using the old reset block that was part of the Bootstrap 1.0 tree, Bootstrap 2.0 uses Normalize.css, a project from Nicolas Gallagher that is part of the HTML5 Boilerplate. This is included in the Bootstrap.css file.

In particular, these default styles give special treatment to typography and links.

- margin has been removed from the body, and content will snug up to the edges of the browser window.
- background-color: white; is applied to the body
- Bootstrap is using the @baseFontFamily, @baseFontSize, and @baseLine Height attributes as our typographic base. This allows the height of headings, and other content around the site to maintain a similar line height.
- Bootstrap sets the global link color via @linkColor and applies link underlines only on :hover

Default Grid System

The default Bootstrap grid system utilizes 12 columns, making for a 940px wide container without responsive features enabled. With the responsive CSS file added, the grid adapts to be 724px and 1170px wide depending on your viewport. Below 767px view-ports, for example, on tablets and smaller devices the columns become fluid and stack vertically. At the default width, each column is 60 pixels wide, offset 20 pixels to the left.



Basic Grid HTML

To create a simple layout, create a container with a div that has a class of .row, and add the appropriate amount of .span* columns. Since we have 12-column grid, we just need to have the amount of .span* columns add up to twelve. We could use a 3-6-3 layout, 4-8, 3-5-4, 2-8-2, we could go on and on, but I think you get the gist.

Basic Grid Layout.

```
<div class="row">
  <div class="span8">...</div>
  <div class="span4">...</div>
</div>
```

In the above example, we have .span8, and a .span4 adding up to 12

Offsetting Columns

You can move columns to the right using the .offset* class. Each class moves the span over that width. So an .offset2 would move a .span7 over two columns.

Offsetting Columns.

```
<div class="row">
  <div class="span2">...</div>
  <div class="span7 offset2">...</div>
</div>
```



Nesting Columns

To nest your content with the default grid, inside of a `.span*`, simply add a new `.row` with enough `.span*` that add up the number of spans of the parent container. So, let's say that you have a two columns layout, with a `span8`, and a `span4`, and you want to embed a two column layout inside of the layout, what spans would you use? For a four column layout?

Create a table that looks like this:

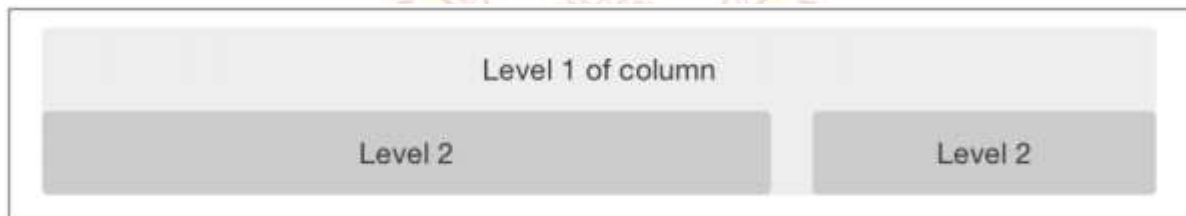
First	Last
Sanders	Kleinfeld
Karen	Tripp
Adam	Zaremba

Your markup should look something like this:

```
[options="header"]
|=====
|First  | Last
|Sanders| Kleinfeld
|Karen  | Tripp
|Adam   | Zaremba
|=====
```

Nesting Columns.

```
<div class="row">
  <div class="span9">
    Level 1 column
    <div class="row">
      <div class="span6">Level 2</div>
      <div class="span3">Level 2</div>
    </div>
  </div>
</div>
```



Fluid Grid System

Fluid Grid System The fluid grid system uses percent instead of pixels for column widths. It has the same responsive capabilities as our fixed grid system, ensuring proper proportions for key screen resolutions and devices. You can make any row “fluid” by changing `.row` to `.rowfluid`. The column classes stay the exact same, making it easy to flip between fixed and fluid grids. To offset, you operate in the same way as the fixed grid system works by adding `.offset*` to any column to shift by your desired number of columns.

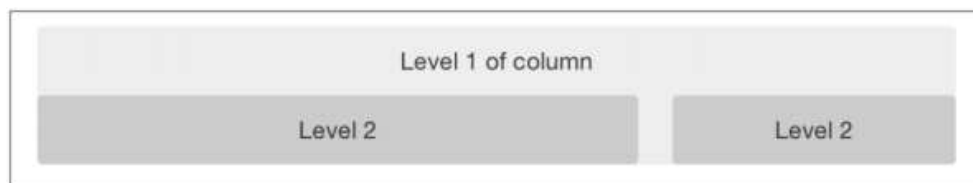
Fluid Column Layout.

```
<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span8">...</div>
</div>
<div class="row-fluid">
  <div class="span4">...</div>
  <div class="span4 offset2">...</div>
</div>
```


Nesting a fluid grid is a little different. Since we are using percentages, each `.row` resets the column count to 12. For example, If you were inside a `.span8`, instead of two `.span4` elements to divide the content in half, you would use two `.span6` divs. This is the case for responsive content, as we want the content to fill 100% of the container.

Nesting Fluid Column Layout.

```
<div class="row-fluid">
  <div class="span8">
    <div class="row">
      <div class="span6">...</div>
      <div class="span6">...</div>
    </div>
  </div>
</div>
```



Container Layouts

To add a fixed width, centered layout to your page, simply wrap the content in `<div class="container">...</div>`. If you would like to use a fluid layout, but want to wrap everything in a container, use the following: `<div class="container-fluid">...</div>`. Using a fluid layout is great when you are building applications, administration screens and other related projects.

Responsive Design

Responsive Design to turn on the responsive features of Bootstrap, you need to add a meta tag to the of your webpage. If you haven't downloaded the compiled source, you will also need to add the responsive CSS file. An example of required files looks like this:

Responsive Meta Tag and CSS.

```
<!DOCTYPE html>
<html>
<head>
  <title>My amazing Bootstrap site!</title>
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <link href="/css/bootstrap.css" rel="stylesheet">
  <link href="/css/bootstrap-responsive.css" rel="stylesheet">
</head>
```

What Is Responsive Design?

Responsive design is a method for taking all of the existing content that is on the page, and optimizing it for the device that is viewing it. For example, the desktop not only gets the normal version of the website, but might get also get a widescreen layout, optimized for the larger displays that many people have attached to their computers. Tab- lets get an optimized layout, taking advantage of the portrait or landscape layouts of those de-

vices. And then with phones, you can target the much narrower width of phones. To target these different widths, Bootstrap uses CSS media queries to measure the width of the browser viewport, and then using conditionals, change which parts of the stylesheets are loaded. Using the width of the browser viewport, Bootstrap can then optimize the content using a combination of ratios, widths, but mostly falls on minwidth and max-width properties.

At the core, Bootstrap supports five different layouts, each relying on CSS media queries. The largest layout has columns that are 70 pixels wide, contrasting the 60 pixels of the normal layout. The tablet layout brings the columns to 42 pixels wide, and when narrower than that, each column goes fluid, meaning the columns are stacked vertically and each column is the full width of the device.

Label	Layout width	Column width	Gutter width
Large display	1200px and up	70px	30px
Default	980px and up	60px	20px
Portrait Tablets	768px and above	42px	20px
Phones to Tablets	767px and below	Fluid columns, no fixed widths	
Phones	480px and below	Fluid columns, no fixed widths	

To add custom CSS based on the media query, you can either include all rules in one CSS file, via the media queries below, or use entirely different CSS files. CSS media queries in the CSS stylesheet.

```
/* Large desktop */
@media (min-width: 1200px) { ... }
/* Portrait tablet to landscape and desktop */
@media (min-width: 768px) and (max-width: 979px) { ... }
/* Landscape phone to portrait tablet */
@media (max-width: 767px) { ... }
/* Landscape phones and down */
@media (max-width: 480px) { ... }
```

For a larger site, you might want to separate them into separate files. In the HTML file, you can call them with the link tag in the head of your document. This is useful for keeping file sizes smaller, but does potentially increase the HTTP requests if being responsive.

CSS media queries via the link tag in the HTML <head>.

```
<link rel="stylesheet" href="base.css" />
<link rel="stylesheet" media="(min-width:1200px)" href="large.css" />
<link rel="stylesheet" media="(min-width:768px) and (max-width: 979px)" href="tablet.css" />
<link rel="stylesheet" media="(max-width: 767px)" href="tablet.css" />
<link rel="stylesheet" media="(max-width: 480px)" href="phone.css" />
```

Helper Classes

Bootstrap also includes a handful of helper classes for doing responsive development. It would be best practice to use these sparingly. A couple of use cases that I have seen involve loading custom elements based on certain layouts. Perhaps you have a really nice header on the main layout, but on mobile you want to pare it down, leaving only a few of the elements. In this scenario, you could use the .hidden-phone class to hide either parts, or entire dom elements from the header.

Class	Phones	Tablets	Desktops
.visible-phone	Visible	Hidden	Hidden
.visible-tablet	Hidden	Visible	Hidden
.visible-desktop	Hidden	Hidden	Visible
.hidden-phone	Hidden	Visible	Visible
.hidden-tablet	Visible	Hidden	Visible
.hidden-desktop	Visible	Visible	Hidden

Typography

Starting with Typography, with the default font stack, Bootstrap uses Helvetica Neue, Helvetica, Arial, and sans-serif. These are all standard fonts, and included as defaults on all major computers, falling back to sans-serif, the catch all to tell the browser to use the default font that the user has decided. All body copy has the font-size set at 14 pixels, with the line-height set at 20 pixels. The tag has a margin-bottom of 10 pixels, or half the line-height.

Headings

All six standard heading levels have been styled in Bootstrap, with the 36 pixels high, and the down to 12 pixels in height (for reference, body text is 14 pixels in height by default). In addition, to add an inline sub-heading to any of the headings, simply add around any of the elements, and you will get smaller text, in a lighter color. In the case of the, the small text is 24 pixels tall, normal font weight (i.e., not bold), and gray instead of black.

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

```
h1 small {
  font-size:24px;
  font-weight:normal;
  line-height:1;
  color:#999;
}
```



Lead body copy

To add some emphasis to a paragraph, add class="lead". This will give you larger font size, lighter weight, and a taller line height. This is usually used for the first few paragraphs in a section, but can really be used anywhere.

<p class="lead">Bacon ipsum dolor sit amet tri-tip pork loin ball tip ... rump. </p>

Lead Example

Bacon ipsum dolor sit amet tri-tip pork loin ball tip frankfurter swine boudin meatloaf shoulder short ribs cow drumstick beef jowl. Meatball chicken sausage tail, kielbasa strip steak turducken venison prosciutto. Chuck filet mignon tri-tip ribeye, flank brisket leberkas. Swine turducken turkey shank, hamburger beef ribs bresaola pastrami venison rump.

Emphasis

In addition to using the <small> tag within headings, as discussed above, you can also use it with body copy. When <small> is applied to body text, the font shrinks to 85% of its original size.

Bold

To add emphasis to text, simply wrap it in a `` tag. This will add font-weightbold to the selected text.

Italics

For italics, wrap your content in the `` tag. “em” derives from the word “emphasis”, and is meant to add stress to your text.

Emphasis classes

Along with `` and ``, Bootstrap offers a few other classes that can be used to provide emphasis. These could be applied to paragraphs, or spans.

Emphasis Classes

```
<p class="muted">This content is muted</p>
<p class="text-warning">This content carries a warn-
ing class</p>
<p class="text-error">This content carries an error
class</p>
<p class="text-info">This content carries an info
class</p>
<p class="text-success">This content carries a suc-
cess class</p>
<p>This content has <em>emphasis</em>, and can
be <strong>bold</strong></p>
```

Bootstrap Emphasis Classes

This content is muted

This content carries a warning class

This content carries an error class

This content carries an info class

This content carries a success class

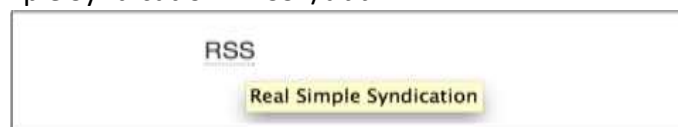
This content has *emphasis*, and can be **bold**

Abbreviations

The HTML `<abbr>` element provides markup for abbreviations or acronyms, like WWW or HTTP. By marking up abbreviations, you can give useful information to browsers, spell checkers, translation systems or search engines. Bootstrap styles `<abbr>` elements with a light dotted border along the bottom, and reveals the full text on hover (as long as you add that text to the `<abbr>` title attribute).

Abbreviation Example.

```
<abbr title="Real Simple Syndication">RSS</abbr>
```



Add `.initialism` to an `<abbr>` for a slightly smaller font size.

Abbreviation Example.

```
<abbr title="Real Simple Syndication">RSS</abbr>
```



Addresses

Adding <address> elements to your page can help screen readers and search engines locate any physical addresses and phone numbers in the text. It can also be used to markup email addresses. Since the <address> defaults to display: block; you'll need to use
 tags to add line breaks to the enclosed address text (for example, to split the street address and city onto separate lines).

Address Markup.

```
<address>
```

```
  <strong>Geetanjali College</strong><br>
```

```
  Red Cross Building <br>
```

```
  SUchak Road<br>
```

```
  <abbr title="Phone">P:</abbr><a  
href="tel:02812464447">0281-2464447</a>
```

```
</address>
```

```
<address>
```

```
  <strong>Hardik Chavda</strong><br />
```

```
  <a href="mailto:#">hardikkchavda@gmail.com</a>
```

```
</address>
```

Geetanjali College,
Red Cross Building,
Suchak Road,
Rajkot.
P: 0281-2464447

Hardik Chavda
hardikkchavda@gmail.com

Blockquotes

To add blocks of quoted text to your document—or for any quotation that you want to set apart from the main text flow—add the <blockquote> tag around the text. For best results, and for line breaks, wrap each subsection in a <p> tag. Bootstrap's default styling indents the text, and adds a thick grey border along the left side. To identify the source of the quote, add the <small> tag, and inside that, add the source's name wrapped in a <cite> tag. When you put it all together, you get something that looks like this:

That this is needed, desperately needed, is indicated by the incredible uptake of Bootstrap. I use it in all the server software I'm working on. And it shows through in the templating language I'm developing, so everyone who uses it will find it's "just there" and works, any time you want to do a Bootstrap technique. Nothing to do, no libraries to include. It's as if it were part of the hardware. Same approach that Apple took with the Mac OS in 1984.

— Developer of RSS, Dave Winer

Blockquote Markup

```
<blockquote>
```

```
<p>
```

That this is needed, desperately needed, is indicated by the incredible uptake of Bootstrap. I use it in all the server software I'm working on. And it shows through in the templating language I'm developing, so everyone who uses it will find it's "just there" and works, any time you want to do a Bootstrap technique. Nothing to do, no libraries to include. It's as if it were part of the hardware. Same approach that Apple took with the Mac OS in 1984.

```
</p>
```

```
<small>Developer of RSS, <cite title="Source Title">Dave Winer</cite></small>
```

```
</blockquote>
```

Lists

Bootstrap offers support and styling for the three main list types that HTML offers: ordered, unordered, and definition lists. An unordered list is a list that doesn't have any particular order, and is traditionally styled with bullets.

Unordered List

Unordered List Markup.

```
<h3>Favorite Outdoor Activities</h3>
<ul>
  <li>Backpacking in Yosemite</li>
  <li>Hiking in Arches</li>
  <li>Delicate Arch</li>
  <li>Park Avenue</li>
</ul>
<li>Biking the Flintstones Trail</li>
</ul>
```

Favorite Outdoor Activities

- Backpacking in Yosemite
- Hiking in Arches
 - Delicate Arch
 - Park Avenue
- Biking the Flintstones Trail

If you have an ordered list that you would like to remove the bullets from, add `class="unstyled"` to the opening `` tag.

Ordered List

An ordered list is a list that falls in some sort of sequential order, and is prefaced by numbers rather than bullets. This is handy when you want to build a list of numbered items, like a task list, guide items, or even a list of comments on a blog post.

Ordered List Markup.

```
<h3>Self-Referential Task List</h3>
<ol>
  <li>Turn off the internet.
  <li>Write the book</li>
  <li>... Profit?</li>
</ol>
```

Self-Referential Task List

1. Turn off the internet.
2. Right the book
3. ... Profit?

Definition List

The third type of list you get with Bootstrap is the definition list. The definition list differs from the ordered and unordered list in that instead of just having a block level `` element, each list item can consist of both the `<dt>` and the `<dd>` elements. `<dt>` stands for "definition term," and like a dictionary, this is the term (or phrase) that is being defined. Subsequently, the `<dd>` is the definition of the `<dt>`.

A lot of times in markup, you will see people using headings inside an unordered list. This works, but maybe isn't the most semantic way to markup the text. A better idea would be to create a `<dl>` and then style the `<dt>` and `<dd>` as you would the heading and the text.

That being said, out of the box, Bootstrap offers some clean default styles, and an option for a side-by-side layout of each definition.

Definition List Markup.

```
<h3>Common Electronics Parts</h3>
```

```
<dl>
```

```
  <dt>LED</dt>
```

```
  <dd>A light-emitting diode (LED) is a semiconductor light source.</dd>
```

```
  <dt>Servo</dt>
```

```
  <dd>Servos are small, cheap, mass-produced actuators used for radio control and
    small robotics.</dd>
```

```
</dl>
```

Common Electronics Parts

LED

A light-emitting diode (LED) is a semiconductor light source.

Servo

Servos are small, cheap, mass-produced actuators used for radio control and small robotics.

To change the <dl> to a horizontal layout, with the <dt> on the left side, and the <dd> on the right, simply add class="dl-horizontal" to the opening tag.

Common Electronics Parts

LED A light-emitting diode (LED) is a semiconductor light source.

Servo Servos are small, cheap, mass-produced actuators used for radio control and small robotics.

Code

There are two different key ways to display code with Bootstrap. The first is the <code>tag, and the second is with the <pre> tag. Generally, if you are going to be displaying code inline, then you should use the <code> tag, but if it needs to be displayed as a standalone block element, or if it has multiple lines, then you should use the <pre> tag. Instead of always using divs, in HTML5, you can use new elements like <code><section></code>,</p>
</div>
<pre>
 <article>
 <h1>Article Heading</h1>
 </article>
</pre>

```
<pre>
```

```
  &lt;article&gt;
```

```
  &lt;h1&gt;Article Heading&lt;/h1&gt;
```

```
  &lt;/article&gt;
```

```
</pre>
```

Tables

One of my favorite parts of Bootstrap is the nice way that tables are handled. I do a lot of work looking at and building tables, and the clean layout is great feature that's included in Bootstrap right off the bat. Bootstrap supports the following elements:

Tag	Description
<code><table></code>	Wrapping element for displaying data in a tabular format
<code><thead></code>	Container element for table header rows (<code><tr></code>) to label table columns
<code><tbody></code>	Container element for table rows (<code><tr></code>) in the body of the table
<code><tr></code>	Container element for a set of table cells (<code><td></code> or <code><th></code>) that appears on a single row
<code><td></code>	Default table cell
<code><th></code>	Special table cell for column (or row, depending on scope and placement) labels. Must be used within a <code><thead></code>
<code><caption></code>	Description or summary of what the table holds, especially useful for screen readers

If you want a nice basic table style with just some light padding and horizontal dividers only, add the base class of `.table` to any table. The basic layout has a top border on all of the `<td>` elements.

Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Optional Table Classes

With the base table markup, and adding the `.table` class, there are few additional classes that you can add to style the markup. There are three classes, `.table-striped`, `.table-bordered`, `.table-hover`, and `.table-condensed`.

Striped Table

By adding the `.table-striped` class, you will get stripes on rows within the `<tbody>`. This is done via the CSS: `nth-child` selector which is not available on Internet Explorer 7-8.

Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Bordered Table

If you add the `.table-bordered` class, you will get borders surrounding every element, and rounded corners around the entire table.

Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Hover Table

If you add the `.table-hover` class, when you hover over a row, a light grey background will be added to rows while the user hovers over them.

Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Condensed Table

If you add the `.table-condensed` class, padding is cut in half on rows to condense the table. Useful if you want denser information.

Name	Phone Number	Rank
Kyle West	707-827-7001	Eagle
Davey Preston	707-827-7003	Eagle
Taylor Lemmon	707-827-7005	Eagle

Table Row Classes

If you want to style the table rows, you could add the following classes to change the background color.

Class	Description	Background Color
<code>.success</code>	Indicates a successful or positive action.	Green
<code>.error</code>	Indicates a dangerous or potentially negative action.	Red
<code>.warning</code>	Indicates a warning that might need attention.	Yellow
<code>.info</code>	Used as an alternative to the default styles.	Blue

#	Product	Payment Taken	Status
1	TB - Monthly	01/04/2012	Approved
2	TB - Monthly	02/04/2012	Declined
3	TB - Monthly	03/04/2012	Pending
4	TB - Monthly	04/04/2012	Call in to confirm

Forms

Another one of the highlights of using Bootstrap is the attention that is paid to forms. As a web developer, one of my least favorite things to do is style forms. Bootstrap makes it easy to do with the simple HTML markup and extended classes for different styles of forms.

The basic form structure comes styled in Bootstrap, without needing to add any extra helper classes. If you use the placeholder, it is only supported in newer browsers. In older ones, no text will be displayed.



Basic Form Structure.

```
<form>
  <fieldset>
    <legend>Legend</legend>
    <label for="name">Label name</label>
    <input type="text" id="name" placeholder="Type something...">
      <span class="help-block">Example block-level help text here.</span>
    <label class="checkbox" for="checkbox">
      <input type="checkbox" id="checkbox">Check me out
    </label>
    <button type="submit" class="btn">Submit</button>
  </fieldset>
</form>
```

Optional Form Layouts

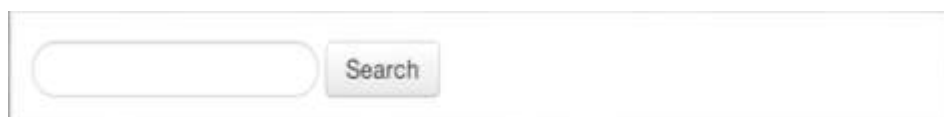
With a few helper classes, you can dynamically update the layout of your form. Bootstrap comes with a few preset styles you can use.

Search Form

Add `.form-search` to the form tag, and then `.search-query` to the `<input>` for an input box with rounded corners, and an inline submit button.

Basic Form Structure.

```
<form class="form-search">
  <input type="text" class="input-medium search-query">
  <button type="submit" class="btn">Search</button>
</form>
```



Inline Form

To create a form where all of the elements are inline, and labels are along side, add the class .form-inline to the form tag. To have the label and the input on the same line, use the horizontal form below.

Inline Form Code.

```
<form class="form-inline">
  <input type="text" class="input-sm" placeholder="Email">
  <input type="password" class="input-sm" placeholder="Password">
  <label class="checkbox">
    <input type="checkbox"> Remember me
  </label>
  <button type="submit" class="btn">Sign in</button>
</form>
```

Horizontal Form

Bootstrap also comes with a pre-baked horizontal form; this one stands apart from the others not only in the amount of markup, but also in the presentation of the form. Traditionally you'd use a table to get a form layout like this, but Bootstrap manages to do it without. Even better, if you're using the responsive CSS, the horizontal form will automatically adapt to smaller layouts by stacking the controls vertically.

To create a form that uses the horizontal layout, do the following:

- Add a class of form-horizontal to the parent <form> element
- Wrap labels and controls in a div with class control-group
- Add a class of control-label to the labels
- Wrap any associated controls in a div with class controls for proper alignment

Horizontal Form Code.

```
<form class="form-horizontal">
<div class="control-group">
  <label class="control-label" for="inputEmail">Email</label>
  <div class="controls">
    <input type="text" id="inputEmail" placeholder="Email">
  </div>
</div>
```

```

</div>
<div class="control-group">
  <label class="control-label" for="inputPassword">Password</label>
  <div class="controls">
    <input type="password" id="inputPassword" placeholder="Password">
  </div>
</div>
<div class="control-group">
<div class="controls">
  <label class="checkbox">
    <input type="checkbox"> Remember me
  </label>
  <button type="submit" class="btn">Sign in</button>
</div>
</div>
</form>

```

Supported Form Controls

Bootstrap natively supports the most common form controls. Chief among them, input, text area, checkbox and radio, and select.

Inputs

The most common form text field is the input—this is where users will enter most of the essential form data. Bootstrap offers support for all native HTML5 input types: text, password, date time, date time-local, date, month, time, week, number, email, url, search, tel and color.



Input Code.

```
<input type="text" placeholder="Text input">
```



Textarea

The textarea is used when you need multiple lines of input. You'll find you mainly modify the rows attribute, changing it to the number of rows that you need to support (fewer rows = smaller box, more rows = bigger box).



Textarea Example.

```
<textarea rows="3"></textarea>
```

Checkboxes and radios

Checkboxes and radio buttons are great for when you want users to be able to choose from a list of preset options. When building a form, use checkbox if you want the user to select any number of options from a list, and radio if you want to limit them to just one selection.

Radio and Checkbox Code Example.

```
<label class="checkbox">
<input type="checkbox" value="">
  Option one is this and that—be sure to include why it's great
</label>
<label class="radio">
<input type="radio" name="optionsRadios" id="optionsRadios1" value="option1" checked>
  Option one is this and that—be sure to include why it's great
</label>
<label class="radio">
<input type="radio" name="optionsRadios" id="optionsRadios2" value="option2">
  Option two can be something else and selecting it will deselect option one
</label>
```

If you want multiple checkboxes to appear on the same line together, simply add the `.inline` class to a series of checkboxes or radios.

```
<label for="option1" class="checkbox inline">
<input id="option1" type="checkbox" id="inlineCheckbox1" value="option1"> 1
</label>
<label for="option2" class="checkbox inline">
<input id="option2" type="checkbox" id="inlineCheckbox2" value="option2"> 2
</label>
<label for="option3" class="checkbox inline">
<input id="option3" type="checkbox" id="inlineCheckbox3" value="option3"> 3
</label>
```

Selects

A select is used when you want to allow the user to pick from multiple options, but by default it only allows one. It's best to use `<select>` for list options of which the user is familiar such as states or numbers. Use `multiple="multiple"` to allow the user to select more than one option. If you only want the user to choose one option, use `type="radio"`.

Select Code Example.

```
<select>
  <option>1</option>
  <option>2</option>
  <option>3</option>
  <option>4</option>
  <option>5</option>
</select>
<select multiple="multiple">
  <option>1</option>
  <option>2</option>
  <option>3</option>
  <option>4</option>
  <option>5</option>
</select>
```

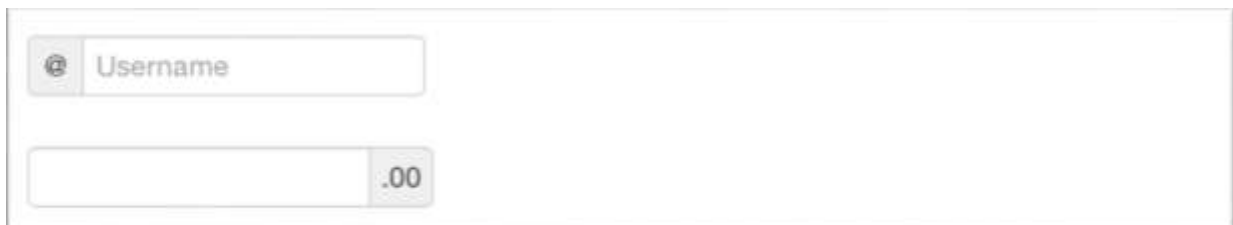


Extending Form Controls

In addition to the basic form controls listed in the previous section, Bootstrap offers few other form components to complement the standard HTML form elements; for example, it lets you easily prepend and append content to inputs.

Prepended and Appended Inputs

By adding prepended and appended content to an input field, you can add common elements to the text users input, like the dollar symbol, the @ for a Twitter username or anything else that might be common for your application interface. To use, wrap the input in a `div` with class `input-prepend` (to add the extra content before the user input) or `input-append` (to add it after). Then, within that same `<div>`, place your extra content inside a `` with an add-on class, and place the `` either before or after the `<input>` element.



Prepend and Append Code Example.

```
<div class="input-prepend">
  <span class="add-on">@</span>
  <input class="span2" id="prependedInput" type="text" placeholder="Username">
</div>
```

```
<div class="input-append">
<input class="span2" id="appendedInput" type="text">
<span class="add-on">.00</span>
</div>
```

If you combine both of them, you simply need to add both the .input-prepend and .input-append classes to the parent <div>.



Append and Prepend Code Example.

```
<div class="input-prepend input-append">
<span class="add-on">$</span>
<input class="span2" id="appendedPrependedInput" type="text">
<span class="add-on">.00</span>
</div>
```

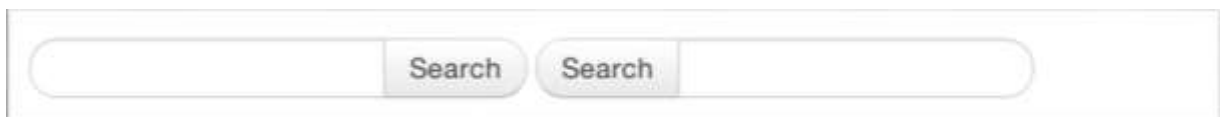
Rather than using a , you can instead use <button> with a class of btn to attach (surprise!) a button or two to the input.



Attach Multiple Buttons Code Example.

```
<div class="input-append">
<input class="span2" id="appendedInputButtons" type="text">
<button class="btn" type="button">Search</button>
<button class="btn" type="button">Options</button>
</div>
```

If you are appending a button to a search form, you will get the same nice rounded corners that you would expect.



```
<form class="form-search">
<div class="input-append">
<input type="text" class="span2 search-query">
<button type="submit" class="btn">Search</button>
</div>
<div class="input-prepend">
<button type="submit" class="btn">Search</button>
<input type="text" class="span2 search-query">
</div>
</form>
```

Form Control Sizing

With the default grid system that is inherent in Bootstrap, you can use the .span* system for sizing form controls. In addition to the span column-sizing method, you can also use a handful of classes that take a relative approach to sizing. If you want the input to act as a block level element, you can add .input-block-level and it will be the full width of the container element.

```
<input class="input-block-level" type="text" placeholder="input-block-level">
```

Relative Input Controls

```
<input class="input-mini" type="text" placeholder="input-mini">
<input class="input-small" type="text" placeholder="input-small">
<input class="input-medium" type="text" placeholder="input-medium">
<input class="input-large" type="text" placeholder="input-large">
<input class="input-xlarge" type="text" placeholder="input-xlarge">
<input class="input-xxlarge" type="text" placeholder="input-xxlarge">
```

Grid Sizing

You can use any .span from .span1 to .span12 for form control sizing.

```

<input class="span1" type="text" placeholder=".span1">
<input class="span2" type="text" placeholder=".span2">
<input class="span3" type="text" placeholder=".span3">
<select class="span1">
...
</select>
<select class="span2">
...
</select>
<select class="span3">
...
</select>

```

If you want to use multiple inputs on a line, simply use the `.controls-row` modifier class to apply the proper spacing. It floats the inputs to collapse the white space, and set the correct margins, and like the `.row` class, it also clears the float.

```

<div class="controls">
  <input class="span5" type="text" placeholder=".span5">
</div>
<div class="controls controls-row">
  <input class="span4" type="text" placeholder=".span4">
  <input class="span1" type="text" placeholder=".span1">
</div>

```

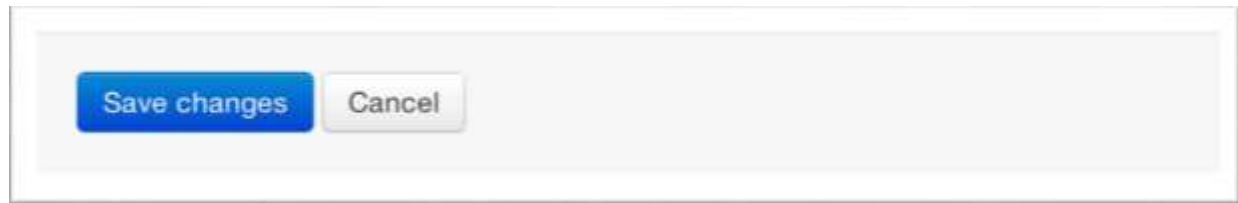
Uneditable Text

If you want to present a form control, but not have it editable, simply add the class `.uneditable-input`.

```
<span class="input-xlarge uneditable-input">Some value here</span>
```

Form Actions

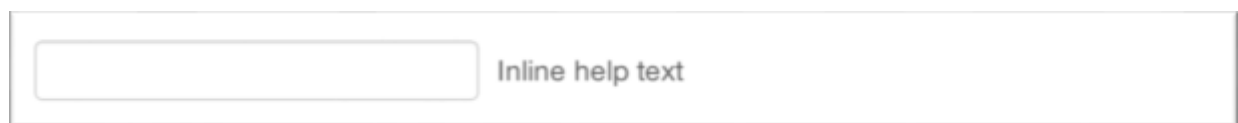
At the bottom of a horizontal-form you can place the form actions. Then inputs will correctly line up with the floated form controls.



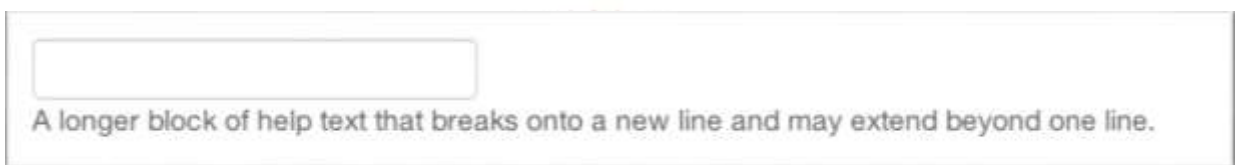
```
<div class="form-actions">
  <button type="submit" class="btn btn-primary">Save changes</button>
  <button type="button" class="btn">Cancel</button>
</div>
```

Help Text

Bootstrap form controls can have either block or inline text that flows with the inputs.



```
<input type="text"><span class="help-inline">Inline help text</span>
```



```
<input type="text">
<span class="help-block">A longer block of help text that breaks onto a new line.</span>
```

Form Control States

In addition to the: focus state, Bootstrap offers styling for disabled inputs, and classes for form validation.

Input Focus

When an input receives: focus, that is to say, a user clicks into the input, or tabs into it, the outline of the input is removed, and a box-shadow is applied. I remember the first time that I saw this on Twitter's site, it blew me away, and I had to dig into the code to see how they did it. In WebKit, this accomplished in the following manner:

```
input {
  -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075);
  -webkit-transition: box-shadow linear 0.2s;
}
input:focus {
  -webkit-box-shadow: inset 0 1px 1px rgba(0, 0, 0, 0.075), 0 0 8px rgba(82, 168, 236, 0.6);
}
```

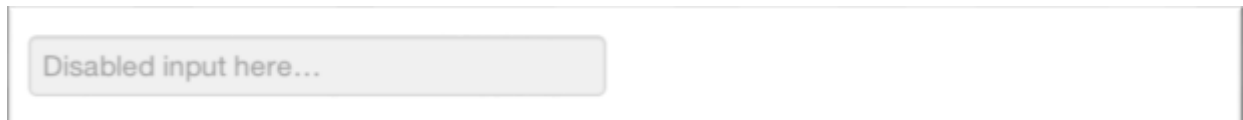
The `<input>` has a small inset box-shadow, this gives the appearance that the input sits lower than the page. When `:focus` is applied, an 8px light blue code is applied. The `-webkit-transition` tells the browser to apply the effect in a linear manner over 0.2seconds. Nice and subtle, a great effect.



```
<input class="input-xlarge" id="focusedInput" type="text" value="This is focused...">
```

Disabled Input

If you need to disable an input, simply add the disabled attribute to not only disable it, but change the styling, and the mouse cursor when it hover over the element.



```
<input class="input-xlarge" id="disabledInput" type="text" placeholder="Disabled input here..." disabled>
```

Validation States

Bootstrap includes validation styles for error, warning, info, and success messages. To use, simply add the appropriate class to the surrounding .control-group.



```
<div class="control-group warning">
<label class="control-label" for="inputWarning">Input with warning</label>
<div class="controls">
<input type="text" id="inputWarning">
<span class="help-inline">Something may have gone wrong</span>
</div>
</div>
<div class="control-group error">
<label class="control-label" for="inputError">Input with error</label>
<div class="controls">
<input type="text" id="inputError">
<span class="help-inline">Please correct the error</span>
</div>
</div>
<div class="control-group success">
<label class="control-label" for="inputSuccess">Input with success</label>
```

```
<div class="controls">
<input type="text" id="inputSuccess">
<span class="help-inline">Woohoo!</span>
</div>
</div>
```



Buttons

One of my favorite features of Bootstrap is the way that buttons are styled. Dave Winner, inventor of RSS, and big fan of Bootstrap has this to say about it:

That this is needed, desperately needed, is indicated by the incredible uptake of Bootstrap. I use it in all the server software I'm working on. And it shows through in the templating language I'm developing, so everyone who uses it will find it's "just there" and works, anytime you want to do a Bootstrap technique. Nothing to do, no libraries to include. It's as if it were part of the hardware. Same approach that Apple took with the Mac OS in 1984.

— Dave Winer
scripting.com

I like to think that Bootstrap is doing that, unifying the web, and allowing a unified experience of what an interface can look like across the web. With the advent of Bootstrap, you can spot the sites that have taken it up usually first by the buttons that they use. A grid layout, and many of the other features fade into the background, but buttons, forms and other unifying elements are a key part of Bootstrap. Maybe I'm the only person that does this, but when I come across a site that is using Bootstrap, I want to give a high five to whomever answers the webmaster email at that domain, as they probably just get it. It reminds me of a few years ago I would do the same thing when I would see content in the HTML of sites that I would visit. Now, buttons, and links can all look alike with Bootstrap, anything that is given that class of btn will inherit the default look of a grey button with rounded corners. Adding extra classes will add colors to the buttons.

Buttons	Class	Description
	btn	Standard gray button with gradient
	btn btn-primary	Provides extra visual weight and identifies the primary action in a set of buttons
	btn btn-info	Used as an alternative to the default styles
	btn btn-success	Indicates a successful or positive action
	btn btn-warning	Standard gray button with gradient
	btn btn-danger	Indicates a dangerous or potentially negative action
	btn btn-inverse	Alternate dark gray button, not tied to a semantic action or use
	btn btn-link	Deemphasize a button by making it look like a link while maintaining button behavior

Button Sizes

If you need larger or smaller buttons, simply add `.btn-large`, `.btn-small`, or `.btn-mini` to links or buttons.



```
<p>
<button class="btn btn-large btn-primary" type="button">Large button</button>
<button class="btn btn-large" type="button">Large button</button>
</p>
<p>
<button class="btn btn-primary" type="button">Default button</button>
<button class="btn" type="button">Default button</button>
</p>
<p>
<button class="btn btn-small btn-primary" type="button">Small button</button>
<button class="btn btn-small" type="button">Small button</button>
</p>
<p>
<button class="btn btn-mini btn-primary" type="button">Mini button</button>
<button class="btn btn-mini" type="button">Mini button</button>
</p>
```

If you want to create buttons that display like a block level element, simply add the `btn-block` class. These buttons will display at 100% width.



```
<button class="btn btn-large btn-block btn-primary" type="button">Block level
button</button>
<button class="btn btn-large btn-block" type="button">Block level button</button>
```

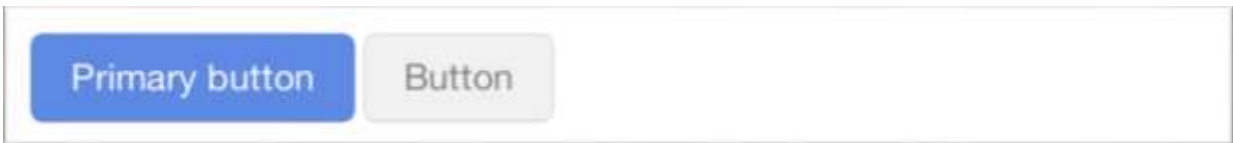
Disabled Button Styling

For anchor elements, simply add the class of .disabled to the tag, and the link will drop back in color, and will lose the gradient.



```
<a href="#" class="btn btn-large btn-primary disabled">Primary link</a>
<a href="#" class="btn btn-large disabled">Link</a>
```

For a button, simply add the disabled attribute to the button. This will actually disable the button, so javascript is not directly needed.



```
<button type="button" class="btn btn-large btn-primary disabled" disabled="disabled">Primary button</button>
<button type="button" class="btn btn-large" disabled>Button</button>
```

Images

Images have three classes to apply some simple styles. They are .img-rounded that adds border-radius:6px to give the image rounded corners, .img-circle that adds makes the entire image a circle by adding border-radius:500px making the image round, and lastly, img-polaroid, that adds a bit of padding and a grey border.



```



```

Icons

Bootstrap bundles 140 icons into one sprite that can be used with buttons, links, navigation, and form fields. The icons are provided by Glyphicons.

 icon-glass	 icon-music	 icon-search	 icon-envelope
 icon-heart	 icon-star	 icon-star-empty	 icon-user
 icon-film	 icon-th-large	 icon-th	 icon-th-list
 icon-ok	 icon-remove	 icon-zoom-in	 icon-zoom-out
 icon-off	 icon-signal	 icon-cog	 icon-trash
 icon-home	 icon-file	 icon-time	 icon-road
 icon-download-alt	 icon-download	 icon-upload	 icon-inbox
 icon-play-circle	 icon-repeat	 icon-refresh	 icon-list-alt
 icon-lock	 icon-flag	 icon-headphones	 icon-volume-off
 icon-volume-down	 icon-volume-up	 icon-qr-code	 icon-barcode
 icon-tag	 icon-tags	 icon-book	 icon-bookmark
 icon-print	 icon-camera	 icon-font	 icon-bold
 icon-italic	 icon-text-height	 icon-text-width	 icon-align-left
 icon-align-center	 icon-align-right	 icon-align-justify	 icon-list
 icon-indent-left	 icon-indent-right	 icon-facetime-video	 icon-picture
 icon-pencil	 icon-map-marker	 icon-adjust	 icon-tint
 icon-edit	 icon-share	 icon-check	 icon-move
 icon-step-backward	 icon-fast-backward	 icon-backward	 icon-play
 icon-pause	 icon-stop	 icon-forward	 icon-fast-forward
 icon-step-forward	 icon-eject	 icon-chevron-left	 icon-chevron-right
 icon-plus-sign	 icon-minus-sign	 icon-remove-sign	 icon-ok-sign
 icon-question-sign	 icon-info-sign	 icon-screenshot	 icon-remove-circle
 icon-ok-circle	 icon-ban-circle	 icon-arrow-left	 icon-arrow-right
 icon-arrow-up	 icon-arrow-down	 icon-share-alt	 icon-resize-full
 icon-resize-small	 icon-plus	 icon-minus	 icon-asterisk
 icon-exclamation-sign	 icon-gift	 icon-leaf	 icon-fire
 icon-eye-open	 icon-eye-close	 icon-warning-sign	 icon-plane
 icon-calendar	 icon-random	 icon-comment	 icon-magnet
 icon-chevron-up	 icon-chevron-down	 icon-retweet	 icon-shopping-cart
 icon-folder-close	 icon-folder-open	 icon-resize-vertical	 icon-resize-horizontal
 icon-hdd	 icon-bullhorn	 icon-bell	 icon-certificate
 icon-thumbs-up	 icon-thumbs-down	 icon-hand-right	 icon-hand-left
 icon-hand-up	 icon-hand-down	 icon-circle-arrow-right	 icon-circle-arrow-left
 icon-circle-arrow-up	 icon-circle-arrow-down	 icon-globe	 icon-wrench
 icon-tasks	 icon-filter	 icon-briefcase	 icon-fullscreen

Glyphicon Attribution

Users of Bootstrap are fortunate to use the Glyphicons free of use on Bootstrap projects. The developers have asked that you use a link back to Glyphicons when practical.

Usage

To use the icons, simply use an `<i>` tag with the name spaced `.icon-` class. For example, if you wanted to use the edit icon, you would simply add the `.icon-edit` class to the `<i>` tag.

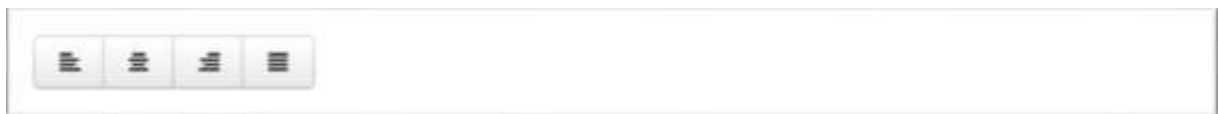
```
<i class="icon-edit"></i>
```

If you want to use the white icon, simply add the `.icon-white` class to the tag.

```
<i class="icon-edit icon-white"></i>
```

Button Groups

Using button groups, combined with icons, you can create nice interface elements with minimal markup.



```
<div class="btn-toolbar">
  <div class="btn-group">
    <a class="btn" href="#"><i class="icon-align-left"></i></a>
    <a class="btn" href="#"><i class="icon-align-center"></i></a>
    <a class="btn" href="#"><i class="icon-align-right"></i></a>
    <a class="btn" href="#"><i class="icon-align-justify"></i></a>
  </div>
</div>
```

Navigation

When you are using icons next to a string of text, make sure to add a space to provide the proper alignment of the image.



```
<ul class="nav nav-list">
  <li class="active"><a href="#"><i class="icon-home icon-white"></i> Home</a></li>
  <li><a href="#"><i class="icon-book"></i> Library</a></li>
  <li><a href="#"><i class="icon-pencil"></i> Applications</a></li>
  <li><a href="#"><i class="i"></i> Misc</a></li>
</ul>
```


Dropdown Menus

Dropdown Menus Dropdown menus are a toggle able, contextual menu for displaying links in a list format. The dropdowns can be used on a variety of different elements, navs, buttons, and more. You can have a single dropdown, or extend the dropdown into another sub-menu.



```
<ul class="dropdown-menu" role="menu" aria-labelledby="dropdownMenu">
  <li><a tabindex="-1" href="#">Action</a></li>
  <li><a tabindex="-1" href="#">Another action</a></li>
  <li><a tabindex="-1" href="#">Something else here</a></li>
  <li class="divider"></li>
```

45

```
<li><a tabindex="-1" href="#">Separated link</a></li>
</ul>
```

Options

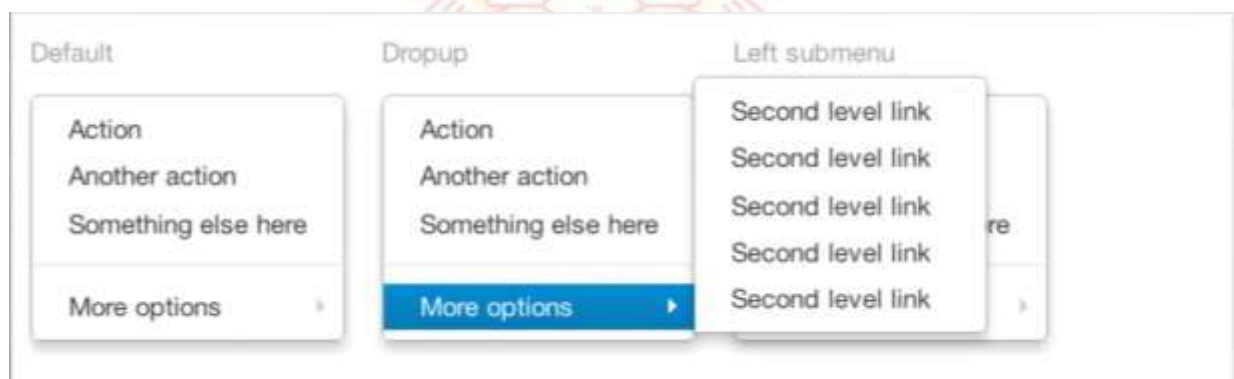
Right align the dropdown

Add .pull-right to a .dropdown-menu to right-align the dropdown menu to the parent object.

```
<ul class="dropdown-menu pull-right" role="menu" aria-labelledby="dLabel">
  ...
</ul>
```

If you would like to add a second layer of dropdowns, simply add .dropdownsubmenu to any in an existing dropdown menu for automatic styling.

Dropdown Submenu



```
<ul class="dropdown-menu" role="menu" aria-labelledby="dLabel">
  ...
  <li class="dropdown-submenu">
    <a tabindex="-1" href="#">More options</a>
    <ul class="dropdown-menu">
      ...
    </ul>
  </li>
</ul>
```


Button Groups

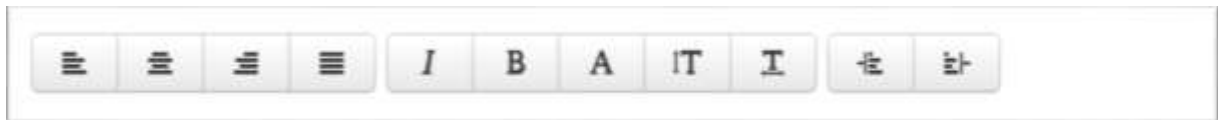
Button groups allow multiple buttons to be stacked together. This is useful like the example below when you want to place items like alignment buttons together. Simply, to create a button group, simply wrap a series of anchors or buttons with a class of .btn with a <div> that has .btn-group as a class.



Inline Button Group Code Example.

```
<div class="btn-group">
  <button class="btn">1</button>
  <button class="btn">2</button>
  <button class="btn">3</button>
</div>
```

If you have multiple button groups that you want to align on a single line, wrap multiple .btn-group with .btn-toolbar.



Button Toolbar Code Example.

```
<div class="btn-toolbar">
<div class="btn-group">
  <a class="btn" href="#"><i class="icon-align-left"></i></a>
  <a class="btn" href="#"><i class="icon-align-center"></i></a>
  <a class="btn" href="#"><i class="icon-align-right"></i></a>
  <a class="btn" href="#"><i class="icon-align-justify"></i></a>
</div>
<div class="btn-group">
  <a class="btn" href="#"><i class="icon-italic"></i></a>
  <a class="btn" href="#"><i class="icon-bold"></i></a>
  <a class="btn" href="#"><i class="icon-font"></i></a>
  <a class="btn" href="#"><i class="icon-text-height"></i></a>
  <a class="btn" href="#"><i class="icon-text-width"></i></a>
</div>
<div class="btn-group">
  <a class="btn" href="#"><i class="icon-indent-left"></i></a>
  <a class="btn" href="#"><i class="icon-indent-right"></i></a>
</div>
</div>
```

To make the buttons stack, simply add .btn-group-vertical to the .btn-group class.

Vertical Button Group Code.

```
<div class="btn-group btn-group-vertical">
...
</div>
```



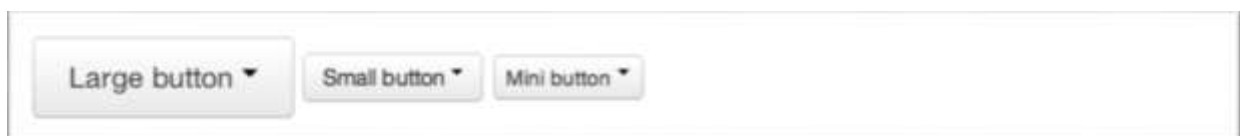
Buttons with Dropdowns

To add a dropdown to a button, simply wrap a button and a dropdown menu in a `.btn-group`. You can also use a `` to act as an indicator that the button is a dropdown.



```
<div class="btn-group">
<button class="btn btn-danger">Danger</button>
<button class="btn btn-danger dropdown-toggle" data-toggle="dropdown">
<span class="caret"></span>
</button>
<ul class="dropdown-menu">
<li><a href="#">Action</a></li>
<li><a href="#">Another action</a></li>
<li><a href="#">Something else here</a></li>
<li class="divider"></li>
<li><a href="#">Separated link</a></li>
</ul>
</div>
```

You can use the dropdowns with any button size, `.btn-large`, `.btn`, `.btn-small` and `.btn-mini`.



Split Button Dropdowns

Using the same general styles of the dropdown button, but adding a primary action along with the dropdown, split buttons have the primary action on the left, and the a toggle on the right for the dropdown.

Split Button Dropdown Code Example

```
<div class="btn-group">
<button class="btn">Action</button>
<button class="btn dropdown-toggle" data-toggle="dropdown">
<span class="caret"></span>
</button>
```

```
<ul class="dropdown-menu">
<!-- dropdown menu links -->
</ul>
</div>
```



Dropup Menus

Menus can also be built to dropup, rather than down. To make this change, simply add `.dropup` to the `.btn-group` container. To have the button pullup from the right hand side, add `.pull-right` to the `.dropdown-menu`. Take notice, the caret is now pointed up, as the menu will be going up instead of down.



Dropup Menu Code Example.

```
<div class="btn-group dropup">
  <button class="btn">Dropup</button>
  <button class="btn dropdown-toggle" data-toggle="dropdown">
    <span class="caret"></span>
  </button>
  <ul class="dropdown-menu">
    <!-- dropdown menu links -->
  </ul>
</div>
```

Navigation Elements

Bootstrap provides a few different opportunities for styling navigation elements. All of them share the same markup and base class `.nav`. Bootstrap also provides a helper class, `.active`. In principal, it generally adds distinction to the current element, and sets it apart from the rest of the navigation elements. You could add this class to the home page links, or add it to the links of the page that you are currently on.

Tabular Navigation

To create a tabbed navigation menu, start with a basic unordered list with the base class of `.nav` and add `.nav-tabs`.



Tabbed Navigation Code Example.

```
<ul class="nav nav-tabs">
<li class="active">
  <a href="#">Home</a>
</li>
```

```

</li>
<li><a href="#">Profile</a></li>
<li><a href="#">Messages</a></li>
</ul>

```

Basic Pills Navigation

To turn the tabs into pills, instead of using the .nav-tabs use .nav-pills.



Tabbed Navigation Code Example.

```

<ul class="nav nav-pills">
<li class="active">
<a href="#">Home</a>
</li>
<li><a href="#">Profile</a></li>
<li><a href="#">Messages</a></li>
</ul>

```

Disabled Class

For each of the .nav classes, if you add the .disabled class, it will create gray link that also disables the hover state. The link is still clickable unless the href is removed, with JavaScript or some other method.



Disabled Navigation Code Example.

```

<ul class="nav nav-pills">
...
<li class="disabled"><a href="#">Home</a></li>
...
</ul>

```

Stackable Navs

Both tabs and pills are horizontal by default, to make them stackable, just add the navstacked class to make them appear vertically stacked.



Stacked Tabs Code Example.

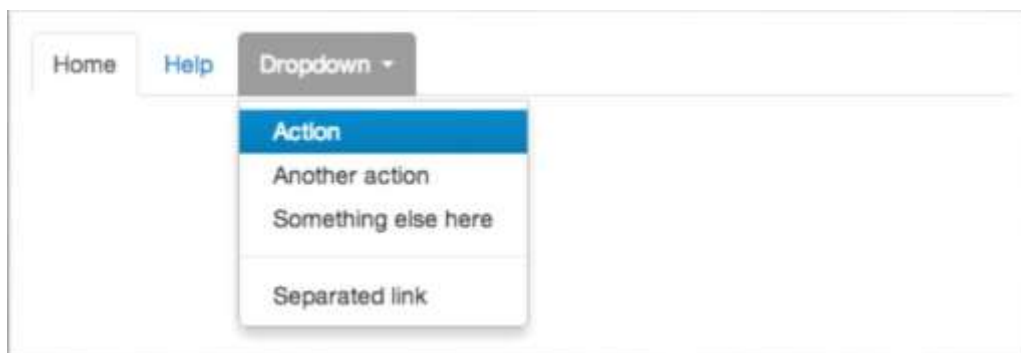
```
<ul class="nav nav-tabs nav-stacked">
...
</ul>
```

**Stacked Pills Code Example.**

```
<ul class="nav nav-pills nav-stacked">
...
</ul>
```

Dropdowns

Navigation menus share a similar syntax to dropdown menus. By default, you have a list item that has an anchor that works in conjunction with some data- attributes to trigger an unordered list with a .dropdown-menu class.

**Tabbed Navigation Dropdown Code Example.**

```
<ul class="nav nav-tabs">
<li class="dropdown">
<a class="dropdown-toggle"
data-toggle="dropdown"
href="#">
Dropdown
<b class="caret"></b>
</a>
<ul class="dropdown-menu">
<li><a href="#">Action</a></li>
<li><a href="#">Another action</a></li>
<li><a href="#">Something else here</a></li>
<li class="divider"></li>
<li><a href="#">Separated link</a></li>
</ul>
</li>
</ul>
```



Pill Navigation Dropdown Code Example.

```

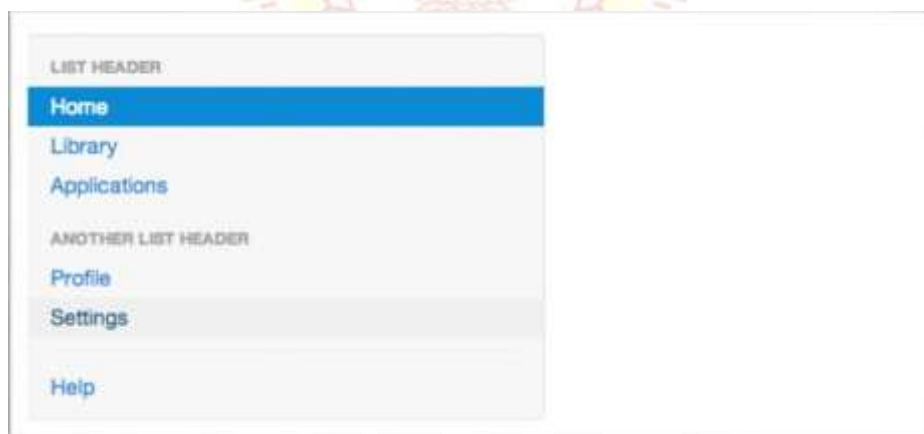
<ul class="nav nav-pills">
<li class="dropdown">
<a class="dropdown-toggle"
data-toggle="dropdown"
href="#">
Dropdown
<b class="caret"></b>
</a>
<ul class="dropdown-menu">
<!--links-->
</ul>
</li>
</ul>

```

**Navigation Lists**

Navigation lists are useful when you need to display a group of navigation links. This type of interface element is common when building admin interfaces in websites. In the MAKE admin interface, I have one of these on the sidebar of every page with quick links to common pages. A form of this is what that Bootstrap developers use for their documentation.

Like all of the lists that we have discussed thus far, this is simply an unordered list with the .nav class, and to give it its specific styling, we add the .nav-list class.

**Navigation List Code Example.**

```

<ul class="nav nav-list">
<li class="nav-header">List Header</li>
<li class="active"><a href="/">Home</a></li>
<li><a href="#">Blog</a></li>
<li class="divider"></li>
<li><a href="#">Contact</a></li>
</ul>

```

Horizontal Divider

To create a divider, much like an `<hr />`, use an empty `` with a class of `.divider`.

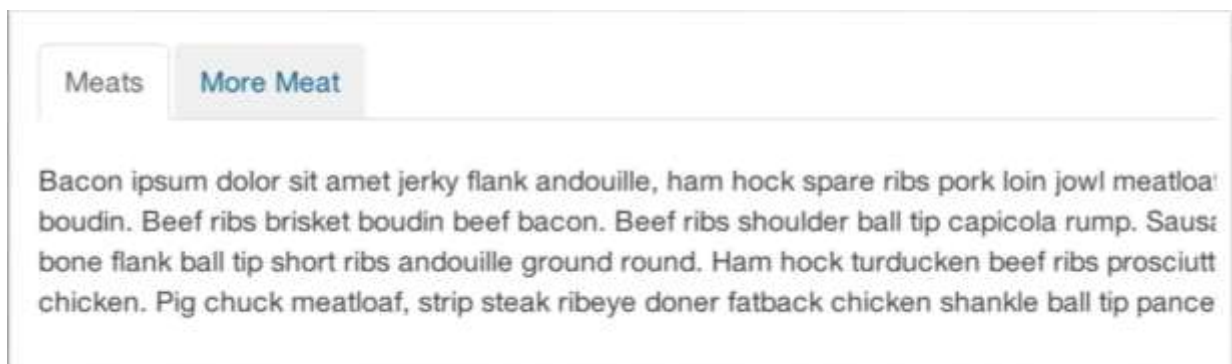
Horizontal Divider.

```
<ul class="nav-menu">
...
<li class="divider"></li>
....
</ul>
```

Tabbable Navigation

Not only can you create a tabbed navigation, but by using the JavaScript plugin, you can also add some interaction by making them tab able to open different windows of content.

To make navigation tabs tabbable, create a `.tab-pane` with a unique ID for every tab, and then wrap them in `.tab-content`.

**Table Navigation Code Example.**

```
<div class="tabbable">
<ul class="nav nav-tabs">
<li class="active"><a href="#tab1" data-toggle="tab">Meats</a></li>
<li><a href="#tab2" data-toggle="tab">More Meats</a></li>
</ul>
<div class="tab-content">
<div class="tab-pane active" id="tab1">
<p>Bacon ipsum dolor sit amet jerky flank...</p>
</div>
<div class="tab-pane" id="tab2">
<p>Beef ribs, turducken ham hock...</p>
</div>
</div>
```


If you want to make the tabs fade when switching, add `.fade` to each `.tab-pane`.

Tab Position

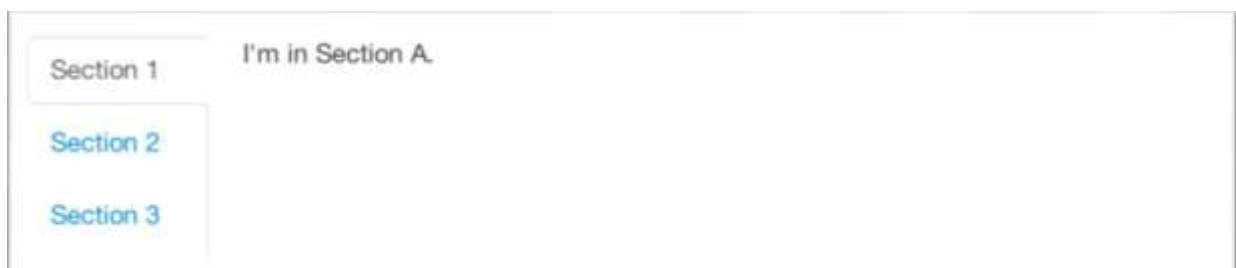
The tabs are fully position able, you can have them above, below, or on the sides of the content.



Bottom Tab Code Example.

```
<div class="tabbable tabs-below">
<div class="tab-content">
<div class="tab-pane active" id="tab1">
<p>I'm in section A.</p>
</div>
<div class="tab-pane" id="tab2">
<p>I'm in section B.</p>
</div>
<div class="tab-pane" id="tab3">
<p>I'm in section C.</p>
</div>
</div>
<ul class="nav nav-tabs">
<li class="active"><a href="#tab1" data-toggle="tab">Section 1</a></li>
<li><a href="#tab2" data-toggle="tab">Section 2</a></li>
<li><a href="#tab3" data-toggle="tab">Section 3</a></li>
</ul>
</div>
```

Tabs on the left get the `.tabs-left` class.



Left Tab Code Example.

```
<div class="tabbable tabs-left">
<div class="tab-content">
<div class="tab-pane active" id="tab1">
<p>I'm in section A.</p>
</div>
```

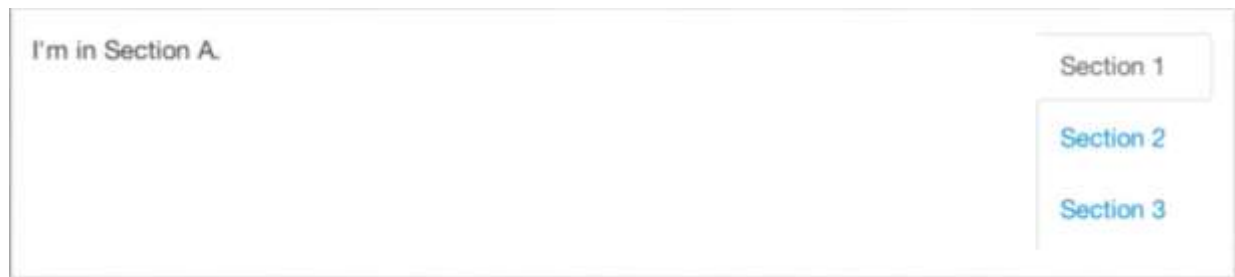


```

<div class="tab-pane" id="tab2">
<p>I'm in section B.</p>
</div>
<div class="tab-pane" id="tab3">
<p>I'm in section C.</p>
</div>
</div>
<ul class="nav nav-tabs">
<li class="active"><a href="#tab1" data-toggle="tab">Section 1</a></li>
<li><a href="#tab2" data-toggle="tab">Section 2</a></li>
<li><a href="#tab3" data-toggle="tab">Section 3</a></li>
</ul>
</div>

```

Tabs on the right get the .tabs-right class.



Right Tab Code Example.

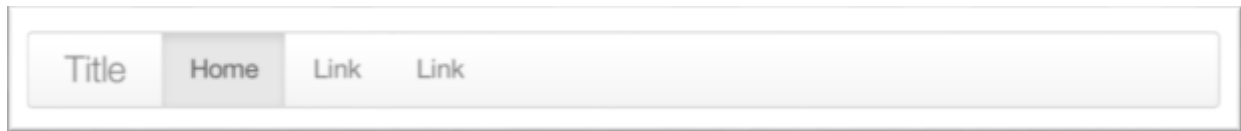
```

<div class="tabbable tabs-right">
<div class="tab-content">
<div class="tab-pane active" id="tab1">
<p>I'm in section A.</p>
</div>
<div class="tab-pane" id="tab2">
<p>I'm in section B.</p>
</div>
<div class="tab-pane" id="tab3">
<p>I'm in section C.</p>
</div>
</div>
<ul class="nav nav-tabs">
<li class="active"><a href="#tab1" data-toggle="tab">Section 1</a></li>
<li><a href="#tab2" data-toggle="tab">Section 2</a></li>
<li><a href="#tab3" data-toggle="tab">Section 3</a></li>
</ul>
</div>

```

Navbar

The Navbar is a nice feature, and one of the prominent features of Bootstrap sites. At the core, the navbar includes styling for site names, and basic navigation. It can later be extended by adding form specific controls, and specialized dropdowns. To be sure that the navbar is constrained to the width of the content of the page, either place it inside of a .span12 or the .container class.



Basic Navbar Code Example.

```
<div class="navbar">
<div class="navbar-inner">
<a class="brand" href="#">Title</a>
<ul class="nav">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">Link</a></li>
<li><a href="#">Link</a></li>
</ul>
</div>
</div>
```

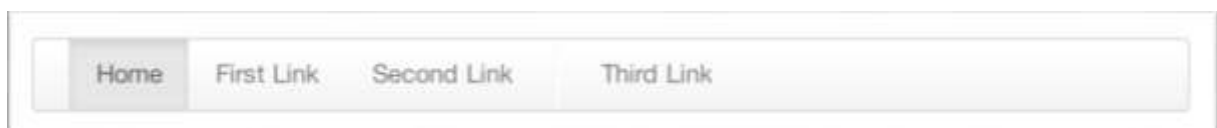
In the code above, note the .brand class, this will give the text a lighter font-weight and slightly larger size.

Brand Class Example.

```
<a class="brand" href="#">Project name</a>
```

Nav Links

To add links to the navbar, simply add an unordered list with the class of .nav. If you want to add a divider to your links, you can do that by adding an empty list-item with a class of .divider-vertical.



Navbar Links Code Example.

```
<ul class="nav">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">First Link</a></li>
<li><a href="#">Second Link</a></li>
<li class="divider-vertical"></li>
<li><a href="#">Third Link</a></li>
</ul>
```

Forms

Instead of using the default, class based forms above, forms that are in the navbar use the `.navbar-form` class. This ensures that the forms margins are properly set, and match the nav stylings. Of note, `.pull-left`, and `.pull-right` helper classes may help move the form in the proper position.

Default Navbar Form Styling.

```
<form class="navbar-form pull-left">
<input type="text" class="span2" id="fname">
<button type="submit" class="btn">
</form>
```

To add rounded corners, taking style cues from the search inputs of iOS devices, instead of using `.navbar-form`, use the `.navbar-search` class.

Navbar Search Input Code Example.

```
<form class="navbar-search" accept-charset="utf-8">
<input type="text" class="search-query" placeholder="Search">
</form>
```

Navbar Menu Variations

The Bootstrap navbar can be dynamic in its positioning. By default, it is a block level element that takes its positioning based on its placement in the HTML. With a few helper classes, you can place it either to the top or bottom of the page, or have it scroll statically with the page.

Fixed to the top

If you want the navbar fixed to the top, simply add `.navbar-fixed-top` to the `.nav bar` class. To prevent the navbar from sitting on top of other content in the body of the page, add at least 40 pixels of padding to the `<body>` tag.

Fixed Top Navbar.

```
<div class="navbar navbar-fixed-top">
<div class="navbar-inner">
<a class="brand" href="#">Title</a>
<ul class="nav">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">Link</a></li>
<li><a href="#">Link</a></li>
</ul>
</div></div>
```

Fixed Bottom Navbar

To affix the navbar to the bottom of the page, simply add `.fixed-navbar-bottom` class to the navbar. Once again, to prevent overlap, add at least 40 pixels of padding to the `<body>` tag.

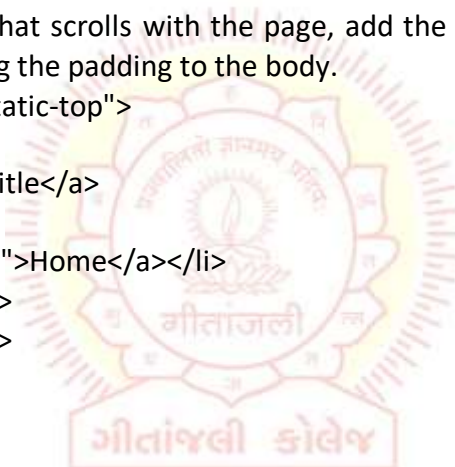
Fixed Bottom Navbar.

```
<div class="navbar navbar-fixed-bottom">
<div class="navbar-inner">
<a class="brand" href="#">Title</a>
<ul class="nav">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">Link</a></li>
<li><a href="#">Link</a></li>
</ul>
</div>
</div>
```

Static Top Navbar

To create a navbar that scrolls with the page, add the `.navbar-static-top` class. This class does not require adding the padding to the body.

```
<div class="navbar navbar-static-top">
<div class="navbar-inner">
<a class="brand" href="#">Title</a>
<ul class="nav">
<li class="active"><a href="#">Home</a></li>
<li><a href="#">Link</a></li>
<li><a href="#">Link</a></li>
</ul>
</div>
</div>
```

**Responsive Navbar**

Like the rest of Bootstrap, the navbar can be totally responsive. To add the responsive features, the content that you want to be collapsed needs to be wrapped in a `<div>` with `.nav-collapse.collapse` as a class. The collapsing nature is tripped by a button that has a the class of `.btn-navbar` and then features two data- elements. The first, `data-toggle` is used to tell the JavaScript what to do with the button, and the second, `data-target` tells which element to toggle. In the example below, three `` with a class of `.icon-bar` create what I like to call the hamburger button. This will toggle the elements that in the `.nav-collapse <div>`. For this to work, the `bootstrap-responsive.css`, and either the `collapse.js` or the full `bootstrap.js` files must be included for this feature to work.



Responsive Navbar Code Example.

```

<div class="header">
<div class="navbar-inner">
<div class="container">
<a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
<span class="icon-bar"></span>
<span class="icon-bar"></span>
<span class="icon-bar"></span>
</a>
<!-- Leave the brand out if you want it to be shown when other elements are <a href="#"
class="brand">Project Name</a>
<!-- Everything that you want collapsed, add it to the collapse div. -->
<div class="nav-collapse collapse">
<!-- .nav, .navbar-search etc... -->
</div>
</div>
</div>
</div>

```

Inverted Navbar

To create an inverted navbar, where the background is black, with white text, simply add `.navbar-inverse` to the `.navbar` class.

**Inverted Navbar Code Example.**

```

<div class="navbar navbar-inverse">
...
</div>

```

Breadcrumbs

Breadcrumbs are a great way to show hierarchy based information for a site. In the case of blogs, it could show the dates of publishing, categories or tags, or for a full CMS, any type of information. A breadcrumb in Bootstrap is simply an unordered list with a class of `.breadcrumb`. There is also a helper class of `.divider` that mutes the colors and makes the text a little smaller too. You could use forward slashes, arrows, or any divider that you choose. Note that the divider here in the breadcrumbs has slightly different markup than the navbar example.



Breadcrumb Code Example.

```

<ul class="breadcrumb">
<li><a href="#">Home</a><span class="divider"></span></li>
<li><a href="#">2012</a><span class="divider"></span></li>
<li><a href="#">December</a><span class="divider"></span></li>
<li><a href="#">5</a></li>
</ul>
<ul class="breadcrumb">
<li><a href="#">Home</a><span class="divider">&rarr;</span></li>
<li><a href="#">Dinner Menu</a><span class="divider">&rarr;</span></li>
<li><a href="#">Specials</a><span class="divider">&rarr;</span></li>
<li><a href="#">Steaks</a></li>
</ul>
<ul class="breadcrumb">
<li><a href="#">Home</a><span class="divider">&raquo;</span></li>
<li><a href="#">Electronics</a><span class="divider">&raquo;</span></li>
<li><a href="#">Raspberry Pi</a></li>
</ul>

```

Pagination

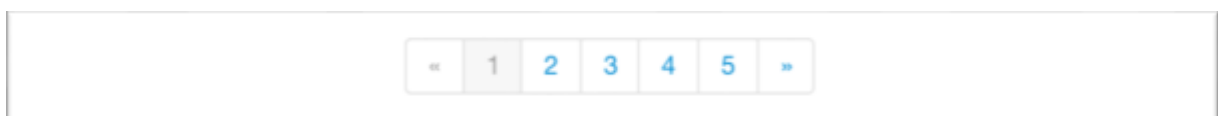
Bootstrap handles pagination like a lot of interface elements, an unordered list, with wrapper <div> that has a specific class that identifies the element. In the basic form, adding .pagination do the parent <div> creates a row of bordered links. Each of the list items can be additionally styled by using the .disabled or .active class.

**Basic Pagination Code Example.**

```

<div class="pagination">
<ul>
<li><a href="#">Prev</a></li>
<li><a href="#">1</a></li>
<li><a href="#">2</a></li>
<li><a href="#">3</a></li>
<li><a href="#">4</a></li>
<li><a href="#">Next</a></li>
</ul>
</div>

```

**Pagination with helper classes code examples.**

```

<div class="pagination pagination-centered">

```

```

<ul>
<li class="disabled"><a href="#"><</a></li>
<li class="active"><a href="#">1</a></li>
<li><a href="#">2</a></li>
<li><a href="#">3</a></li>
<li><a href="#">4</a></li>
<li><a href="#">5</a></li>
<li><a href="#">></a></li>
</ul>
</div>

```

In addition to the .active and .disabled classes for list items, you can also add .pagination-centered to the parent <div>. This will center the contents of the div. If you want the items right aligned in the <div> add .pagination-right. For sizing, in addition to the normal size, there are three other sizes, applied by adding a class to the wrapper <div>. They are: .pagination-large, pagination-small and paginationmini.



Pagination Code Example

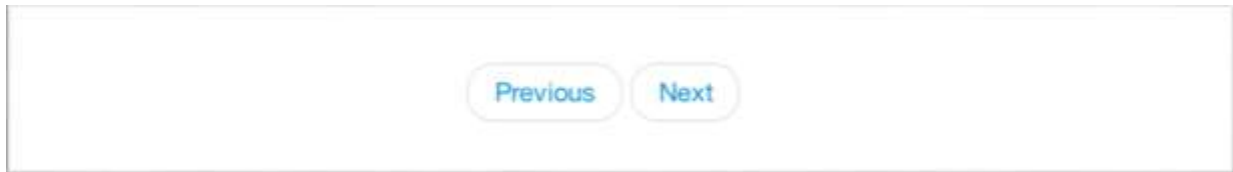
```

<div class="pagination pagination-large">
<ul>
...
</ul>
</div>
<div class="pagination">
<ul>
...
</ul>
</div>
<div class="pagination pagination-small">
<ul>
...
</ul>
</div>
<div class="pagination pagination-mini">
<ul>
...
</ul>
</div>

```

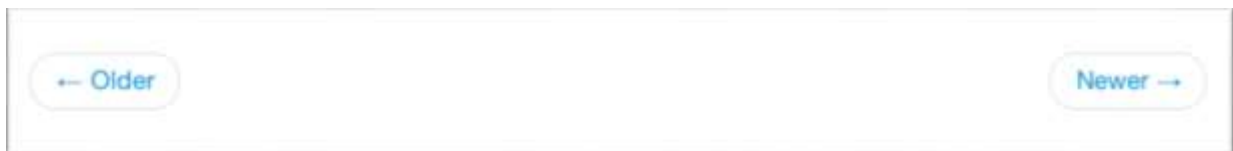

Pager

If you need to create simple pagination links that go beyond text, the pager can work quite well. Like the pagination links, the markup is an unordered list that sheds the wrapper `<div>`. By default, the links are centered.

**Basic Pager Code Example.**

```
<ul class="pager">
<li><a href="#">Previous</a></li>
<li><a href="#">Next</a></li>
</ul>
```

To left/right align the different links, you just need to add the `.previous` and `.next` class to the list-items. Also, like `.pagination` above, you can add the `disabled` class for a muted look.

**Aligned Page Links Code Example.**

```
<ul class="pager">
<li class="previous">
<li>
<li class="next">
<a href="#">Newer &rarr;</a>
</li>
</ul>
```

**Labels**

Labels and Badges are great for offering counts, tips, or other markup for pages. Another one of my favorite little Bootstrap touches.

**Label Markup.**

```
<span class="label">Default</span>
<span class="label label-success">Success</span>
<span class="label label-warning">Warning</span>
<span class="label label-important">Important</span>
<span class="label label-info">Info</span>
<span class="label label-inverse">Inverse</span>
```

Badges

Badges are similar to labels, the primary difference is that they have more rounded corners. The colors of badges reflect the same classes as labels.

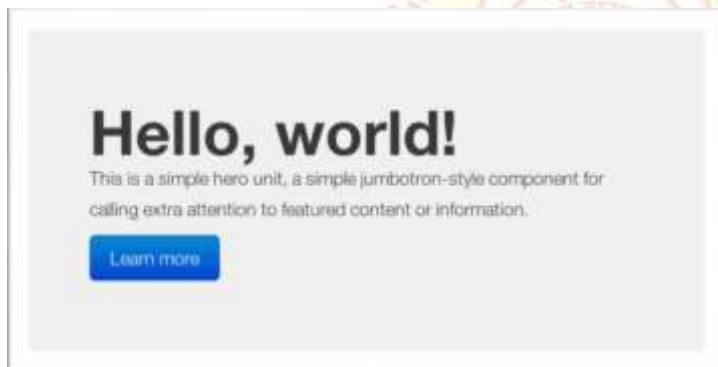


Badges Code Example.

```
<span class="badge">1</span>
<span class="badge badge-success">2</span>
<span class="badge badge-warning">4</span>
<span class="badge badge-important">6</span>
<span class="badge badge-info">8</span>
<span class="badge badge-inverse">10</span>
```

Typographic Elements

In addition to buttons, labels, forms, tables and tabs, Bootstrap has a few more elements for basic page layout. The hero unit is a large, content area that increased the size of headings, and adds a lot of margin for landing page content. To use, simply create a container `<div>` with the class of `.hero-unit`. In addition to a larger `<h1>`, all the fontweight is reduced to 200.



Hero Unit Code Example.

```
<div class="hero-unit">
<h1>Heading</h1>
<p>Tagline</p>
<p>
<a class="btn btn-primary btn-large">Learn more</a>
</p>
</div>
```

Page Header

The page header is nice little feature to add appropriate spacing around the headings on a page. This is particularly helpful on a blog archive page where you may have several post titles, and need a way to add distinction to each of them. To use, wrap your heading in a `<div>` with a class of `.page-header`.

Example page header

Subtext for header

Page Header Code Example.

```
<div class="page-header">
<h1>Example page header <small>Subtext for header</small></h1>
</div>
```

Thumbnails

A lot of sites need a way to layout images in a grid, and Bootstrap has an easy way to do this. At the simplest, you add an `<a>` tag with the class of `.thumbnail` around an image. This adds four pixels of padding, and a grey border. On hover, an animated glow is added around the image.

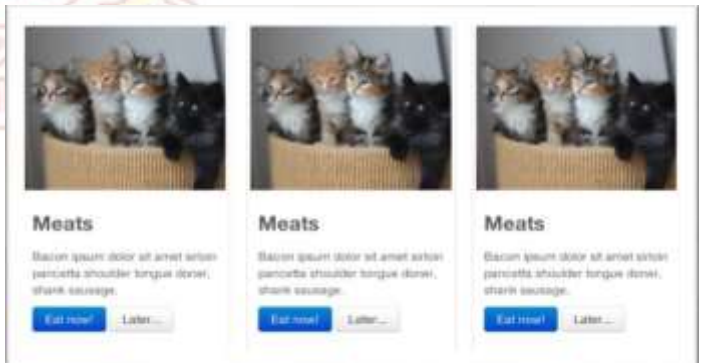


Thumbnail Code Example

```
<a href="#" class="thumbnail">

</a>
```

To add more content to the markup, as an example, you could add headings, buttons and more, swap the `<a>` tag that has a class of `.thumbnail` to be a `<div>`. Inside of that `<div>`, you can add anything you need. Since this is a `<div>` we can use the default span based naming convention for sizing. If you want to group multiple images, place them in an unordered list, and each list item will be floated to left.



Customizable Code Example.

```
<ul class="thumbnails">
<li class="span4">
<div class="thumbnail">

<div class="caption">
<h3>Meats</h3>
<p>Bacon ipsum dolor sit amet sirloin pancetta shoulder tongue doner, shank sausage.</p>
```

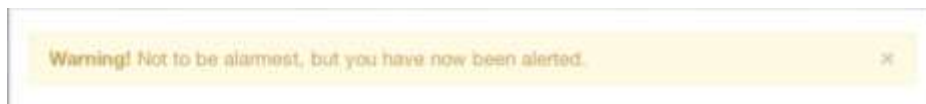
```

<p><a href="#" class="btn btn-primary">Eat now!</a><a href="#"
class="btn">Later...</a></div>
</div>
</li>
<li class="span4">
...
</li>
</ul>

```

Alerts

Alerts provide a way to style messages to the user. The default alert is created by creating a wrapper `<div>` and adding a class of `.alert`.



Basic Alert Code Example.

```

<div class="alert">
<a href="#" class="close" data-dismiss="alert">&times;</a>
<strong>Warning!</strong> Not to be alarmist, but you have now been alerted.
</div>

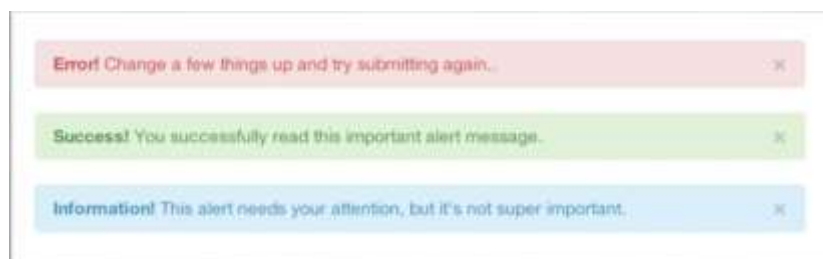
```

The alert uses the alerts jquery plugin that is covered in chapter 4. To close the alert, you can use a button that contains the `data-dismiss="alert"` attribute. Mobile Safari, and Mobile Opera browsers require an `href="#"` to close.

If you have a longer message in your alert, you can use the `.alert-block` class. This provides a little more padding above and below the content contained in the alert, particularly useful for multi-page lines of content.



There are also three other color options, to help provide a more semantic method for the alert. They are added by adding either `.alert-error`, `.alert-success`, or `alertinfo`.



Progress bars

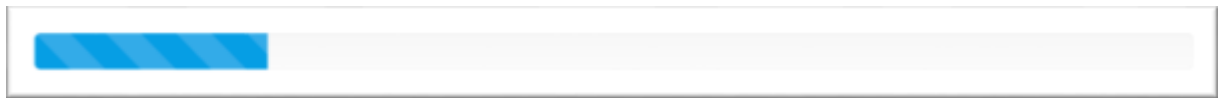
The purpose of progress bars is to show that assets are loading, in progress, or that there is action taking place regarding elements on the page. Personally, I think that these elements are more an exercise in markup, and have little purpose beyond that in the Bootstrap framework. That being said, with thousands of people using Bootstrap, there are likely

a few outliers that have a good reason. By nature, these are static elements, and need some sort of JavaScript interaction to provide any interaction. The default progress bar has a light background and a blue progress bar. To create, add a `<div>` with a class of `.progress`. And then inside, add an empty `<div>` with a class of `.bar`. Add a style attribute with the width in percentage. In the case below, I added `style="60%";` to indicate that the progress bar was at 60%.

**Progress Bar Example.**

```
<div class="progress">
<div class="bar" style="width: 60%;"></div>
</div>
```

To create a striped progress bar, just add `.progress-striped` to the container `<div>`. Of note, striped progress bars are not available in Internet Explorer 7 and 8.

**Striped Progress Bar Code Example.**

```
<div class="progress progress-striped">
<div class="bar" style="width: 20%;"></div>
</div>
```

Like the striped version of the progress bar, you can animate the stripes, making them look like the blue light special barbershop pole.

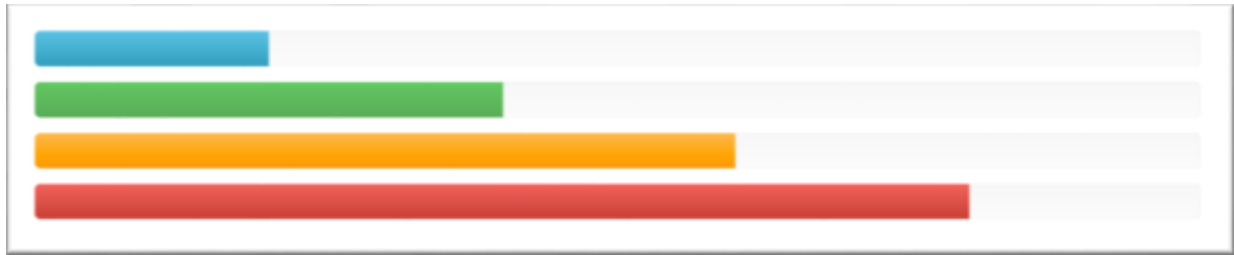
**Animated Progress Bar Code Example**

```
<div class="progress progress-striped active">
<div class="bar" style="width: 40%;"></div>
</div>
```

In addition to the blue progress bar, there are options for green, yellow, and red by using the `.bar-success`, `bar-warning`, and `bar-danger` classes. Progress bars can be stacked, indicating a graph of sorts by adding multiple elements together like so:

**Stacked Progress Bar Example.**

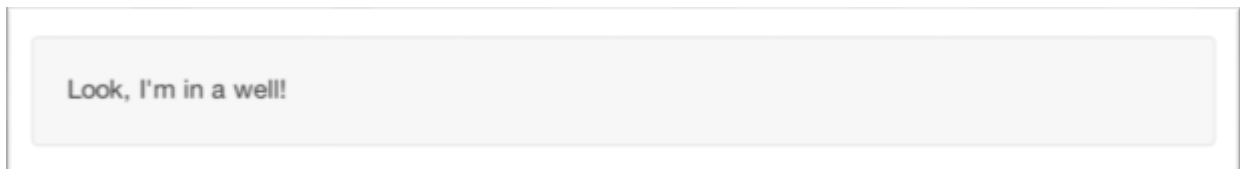
```
<div class="progress">
<div class="bar bar-success" style="width: 35%;"></div>
<div class="bar bar-warning" style="width: 20%;"></div>
<div class="bar bar-danger" style="width: 10%;"></div>
</div>
```



Well

There are a few more components that Bootstrap offers. There are a few that are layout based, and a few that are production based helper classes. The first among these are the wells.

A well is a container `<div>` that add some styles to appear sunken on the page. I have used them before for blog post Meta information, like author, date, categories. To create, simply wrap the content that you would like to appear in the well with a `<div>` containing the class of `.well`.



Well Example

```
<div class="well">  
...  
</div>
```

There are two additional classes that can be used in conjunction with `.well`, `.well-large` and `.well-small`. These affect the padding, making the well larger or smaller depending on the class.



```
<div class="well well-large">  
Look, I'm in a .well-large!  
</div>  
<div class="well well-small">  
Look, I'm in a .well-small!  
</div>
```

What is Laravel?

The need for frameworks

Of all the server-side programming languages, PHP undoubtedly has the lowest entry barriers. It is almost always installed by default on even the cheapest web hosts, and it is also extremely easy to set up on any personal computer. For newcomers who have some experience with authoring web pages in HTML and CSS, the concepts of variables, inline conditions, and include statements are easy to grasp. PHP also provides many commonly used functions that one might need when developing a dynamic website. All of this contributes to what some refer to as the immediacy of PHP. However, this instant gratification comes at a cost. It gives a false sense of productivity to beginners, who almost inevitably end up with convoluted spaghetti code as they add more features and functionality to their site. This is mainly because PHP, out of the box, does not do much to encourage the separation of concerns.

The limitations of homemade tools

If you already have a few PHP projects under your belt, but have not used a web application framework before, then you will probably have amassed a personal collection of commonly used functions and classes that you can use on new projects. These homegrown utilities might help you with common tasks, such as sanitizing data, authenticating users, and including pages dynamically. You might also have a predefined directory structure where these classes and the rest of your application code reside. However, all of this will exist in complete isolation; you will be solely responsible for the maintenance, inclusion of new features, and documentation. For a lone developer or an agency with ever-changing staff, this can be a tedious and time-consuming task, not to mention that if you were to collaborate with other developers on the project, they would first have to get acquainted with the way in which you build applications.

Laravel to the rescue

This is exactly where a web application framework such as Laravel comes to the rescue. Laravel reuses and assembles existing components to provide you with a cohesive layer upon which you can build your web applications in a more structured and pragmatic way. Drawing inspiration from popular frameworks written not just in PHP but other programming languages too, Laravel offers a robust set of tools and an application architecture that incorporates many of the best features of frameworks like CodeIgniter, Yii, ASP.NET MVC, Ruby on Rails, Sinatra, and others. Most of these frameworks use the **Model-View-Controller (MVC)** paradigm or design pattern. If you have used one of the aforementioned tools or the MVC pattern, then you will find it quite easy to get started with Laravel 5.

Features

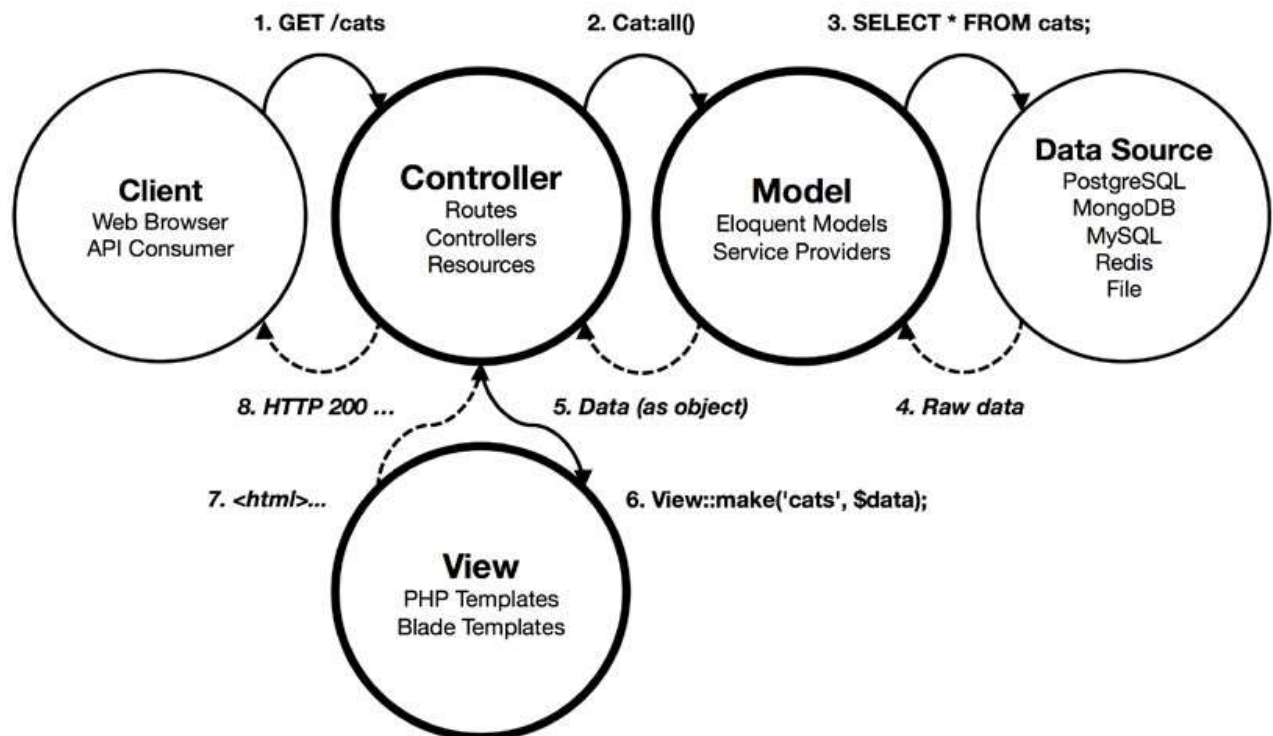
So, what do you get out of the box with Laravel 5? Let's take a look and see how the following features can help boost your productivity:

- **Modularity:** Laravel was built on top of over 20 different libraries and is itself split into individual modules. Tightly integrated with **Composer** dependency manager, these components can be updated with ease.
- **Testability:** Built from the ground up to ease testing, Laravel ships with several helpers that let you visit routes from your tests, crawl the resulting HTML, ensure that methods are called on certain classes, and even impersonate authenticated users in order to make sure the right code is run at the right time.
- **Routing:** Laravel gives you a lot of flexibility when you define the routes of your application. For example, you could manually bind a simple anonymous function to a route with an HTTP and HTTPS verb, such as GET, POST, PUT, or DELETE. This feature is inspired by micro-frameworks, such as **Sinatra** (Ruby) and **Silex** (PHP).
- **Configuration management:** More often than not, your application will be running in different environments, which means that the database or e-mail server credential's settings or the displaying of error messages will be different when your app is running on a local development server to when it is running on a production server. Laravel has a consistent approach to handle configuration settings, and different settings can be applied in different environments via the use of an `.env` file, containing settings unique for that environment.
- **Query builder and ORM:** Laravel ships with a fluent query builder, which lets you issue database queries with a PHP syntax, where you simply chain methods instead of writing SQL. In addition to this, it provides you with an **Object Relational Mapper (ORM)** and **ActiveRecord** implementation, called **Eloquent**, which is similar to what you will find in Ruby on Rails, to help you define interconnected models. Both the query builder and the ORM are compatible with different databases, such as PostgreSQL, SQLite, MySQL, and SQL Server.
- **Schema builder, migrations, and seeding:** Also inspired by Rails, these features allow you to define your database schema in PHP code and keep track of any changes with the help of database migrations. A migration is a simple way of describing a schema change and how to revert to it. Seeding allows you to populate the selected tables of your database, for example, after running a migration.
- **Template engine:** Partly inspired by the **Razor** template language in ASP.NET MVC, Laravel ships with **Blade**, a lightweight template language with which you can create hierarchical layouts with predefined blocks in which dynamic content is injected.
- **E-mailing:** With its Mail class, which wraps the popular **SwiftMailer** library, Laravel makes it very easy to send an e-mail, even with rich content and attachments from your application. Laravel also comes with drivers for popular e-mail sending services such as **SendGrid**, **Mailgun**, and **Mandrill**.

- **Authentication:** Since user authentication is such a common feature in webapplications, out of the box Laravel comes with a default implementation to register, authenticate, and even send password reminders to users.
- **Redis:** This is an in-memory key-value store that has a reputation for being extremely fast. If you give Laravel a Redis instance that it can connect to, it can use it as a session and general purpose cache, and also give you the possibility to interact with it directly.
- **Queues:** Laravel integrates with several queue services, such as AmazonSQS, Beanstalkd, and IronMQ, to allow you to delay resource-intensive tasks, such as the e-mailing of a large number of users, and run them in the background, rather than keep the user waiting for the task to complete.
- **Event and command bus:** Although not new in version 5, Laravel has brought a command bus to the forefront in which it's easy to dispatch events (a class that represents something that's happened in your application), handle commands (another class that represents something that should happen in your application), and act upon these at different points in your application's lifecycle.



MVC architecture



- **Models:** Models represent *resources* in your application. More often than not, they correspond to records in a data store, most commonly a database table. In this respect, you can think of models as representing *entities* in your application, be that a user, a news article, or an event, among others. In Laravel, models are classes that usually extend Eloquent's base Model class and are named in **CamelCase** (that is, NewsArticle). This will correspond to a database table with the same name, but in **snake_case** and plural (that is, news_articles). By default, Eloquent also expects a primary key named `id`, and will also look for—and automatically update—the `created_at` and `updated_at` columns. Models can also describe the relationships they have with other models. For example, a NewsArticle model might be associated with a User model, as a User model might be able to author a NewsArticle model.

- **Controllers or routes:** Controllers, at their simplest, take a request, do something, and then send an appropriate response. Controllers are where the actual processing of data goes, whether that is retrieving data from a database, or handling a form submission, and saving data back to a database. Although you are not forced to adhere to any rules when it comes to creating controller classes in Laravel, it does offer you two sane approaches: RESTful controllers and resource controllers. A RESTful controller allows you to define your own actions and what HTTP methods they should respond to. Resource controllers are based around an entity and allow you to perform common operations on that entity, based on the HTTP method used.

- **Views or Templates:** Views are responsible for displaying the response returned from a controller in a suitable format, usually as an HTML webpage. They can be conveniently built by using the Blade template language or by simply using standard PHP. The file extension of the view, either `.blade.php` or simply `.php`, determines whether or not Laravel treats your view as a Blade template or not.

Structure of a Laravel application

Here is what the directory of a new Laravel 5 application looks like:

./app/	# Your Laravel application
./app/Commands/	# Commands classes
./app/Console/	
./app/Console/Commands/	# Command-line scripts
./app/Events/	# Events that your application can raise
./app/Exceptions/	
./app/Handlers/	# Exception handlers
./app/Handlers/Commands	# Handlers for command classes
./app/Handlers/Events	# Handlers for event classes
./app/Http/	
./app/Http/Controllers/	# Your application's controllers
./app/Http/Middleware/	# Filters applied to requests
./app/Http/Requests/	# Classes that can modify requests
./app/Http/routes.php	# URLs and their corresponding handlers
./app/Providers	# Service provider classes
./app/Services	# Services used in your application
./bootstrap/	# Application bootstrapping scripts
./config/	# Configuration files
./database/	
./database/migrations/	# Database migration classes
./database/seeds/	# Database seeder classes
./public/	# Your application's document root
./public/.htaccess	# Sends incoming requests to index.php
./public/index.php	# Starts Laravel application
./resources/	
./resources/assets/	# Hold raw assets like LESS & Sass files
./resources/lang/	# Localization and language files
./resources/views/	# Templates that are rendered as HTML
./storage/	
./storage/app/	# App storage, like file uploads etc
./storage/framework/	# Framework storage (cache)
./storage/logs/	# Contains application-generated logs
./tests/	# Test cases
./vendor/	# Third-party code installed by Composer
./env.example	# Example environment variable file
./artisan	# Artisan command-line utility
./composer.json	# Project dependencies manifest
./phpunit.xml	# Configures PHPUnit for running tests
./server.php	# A lightweight local development server

Basic requirements for Laravel

However, if you are not using Homestead, you will need to make sure your server meets the following requirements:

- PHP version between 5.5.9 - 7.1.*
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension

Using Laravel Installer

Laravel utilizes Composer to manage its dependencies. So, before using Laravel, make sure you have Composer installed on your machine.

[composer global require "laravel/installer"]

Make sure to place the ~/.composer/vendor/bin directory (or the equivalent directory for your OS) in your PATH so the laravel executable can be located by your system.

Once installed, the laravel new command will create a fresh Laravel installation in the directory you specify. For instance, laravel new blog will create a directory named blog containing a fresh Laravel installation with all of Laravel's dependencies already installed. This method of installation is much faster than installing via Composer:

[laravel new blog]

Using Composer

Strongly inspired by popular dependency managers in other languages, such as Ruby's Bundler or Node.js's **Node Package Manager (npm)**, Composer brings these features to PHP and has quickly become the de-facto dependency manager in PHP.

How does Composer work?

A few years ago, you may have used **PHP Extension and Application Repository (PEAR)** to download libraries. PEAR differs from Composer, in that PEAR would install packages on a system-level basis, whereas a dependency manager, such as Composer, installs them on a project-level basis. With PEAR, you could only have one version of a package installed on a system. Composer allows you to use different versions of the same package in different applications, even if they reside on the same system.

Installation

Linux

Locally

To install Composer locally, run the installer in your project directory. The installer will check a few PHP settings and then download composer.phar to your working directory. This file is the Composer binary. It is a PHAR (PHP archive), which is an archive format for PHP which can be run on the command line, amongst other things.

Now run `php composer.phar` in order to run Composer.

You can install Composer to a specific directory by using the `--install-dir` option and additionally (re)name it as well using the `--filename` option. When running the installer when following [the Download page instructions](#) add the following parameters:

```
php composer-setup.php --install-dir=bin --filename=composer
```

Now run `php bin/composer` in order to run Composer.

Globally

You can place the Composer PHAR anywhere you wish. If you put it in a directory that is part of your PATH, you can access it globally. On UNIX systems you can even make it executable and invoke it without directly using the php interpreter. After running the installer following [the Download page instructions](#) you can run this to move composer.phar to a directory that is in your path:

```
mv composer.phar /usr/local/bin/composer
```

If you like to install it only for your user and avoid requiring root permissions, you can use `~/.local/bin` instead which is available by default on some Linux distributions.

Windows

Using the Installer

This is the easiest way to get Composer set up on your machine. Download and run [Composer-Setup.exe](#). It will install the latest Composer version and set up your PATH so that you can call composer from any directory in your command line.

Finding and installing new packages

Via Composer Create-Project

Alternatively, you may also install Laravel by issuing the Composer create-project command in your terminal:

```
[composer create-project --prefer-dist laravel/laravel blog "6.0*"]
```


Introduction

All of the configuration files for the Laravel framework are stored in the config directory. Each option is documented, so feel free to look through the files and get familiar with the options available to you.

Environment configuration

It is often helpful to have different configuration values based on the environment the application is running in. For example, you may wish to use a different cache driver locally than you do on your production server. It's easy using environment based configuration.

To make this a cinch, Laravel utilizes the DotEnv PHP library by Vance Lucas. In a fresh Laravel installation, the root directory of your application will contain a .env.example file. If you install Laravel via Composer, this file will automatically be renamed to .env. Otherwise, you should rename the file manually.

All of the variables listed in this file will be loaded into the \$_ENV PHP super-global when your application receives a request. However, you may use the env helper to retrieve values from these variables in your configuration files. In fact, if you review the Laravel configuration files, you will notice several of the options already using this helper:

```
'debug' => env('APP_DEBUG', false),
```

The second value passed to the env function is the "default value". This value will be used if no environment variable exists for the given key.

Your .env file should not be committed to your application's source control, since each developer / server using your application could require a different environment configuration.

If you are developing with a team, you may wish to continue including a .env.example file with your application. By putting place-holder values in the example configuration file, other developers on your team can clearly see which environment variables are needed to run your application.

Protecting sensitive configuration

For "real" applications, it is advisable to keep all of your sensitive configuration out of your configuration files. Things such as database passwords, Stripe API keys, and encryption keys should be kept out of your configuration files whenever possible. So, where should we place them? Thankfully, Laravel provides a very simple solution to protecting these types of configuration items using "dot" files.

First, configure your application to recognize your machine as being in the local environment. Next, create a .env.local.php file within the root of your project, which is usually the same directory that contains your composer.json file. The .env.local.php should return an array of key-value pairs, much like a typical Laravel configuration file:

```
<?php
return array(
    'TEST_STRIPE_KEY' => 'super-secret-sauce',
);
```


All of the key-value pairs returned by this file will automatically be available via the `$_ENV` and `$_SERVER` PHP "superglobals". You may now reference these globals from within your configuration files:

```
'key' => $_ENV['TEST_STRIPE_KEY']
```

Be sure to add the `.env.local.php` file to your `.gitignore` file. This will allow other developers on your team to create their own local environment configuration, as well as hide your sensitive configuration items from source control.

Now, on your production server, create a `.env.php` file in your project root that contains the corresponding values for your production environment. Like the `.env.local.php` file, the production `.env.php` file should never be included in source control.

Maintenance mode

When your application is in maintenance mode, a custom view will be displayed for all routes into your application. This makes it easy to "disable" your application while it is updating or when you are performing maintenance. A call to the `App::down` method is already present in your `app/start/global.php` file. The response from this method will be sent to users when your application is in maintenance mode.

To enable maintenance mode, simply execute the down Artisan command:

```
php artisan down
```

To disable maintenance mode, use the up command:

```
php artisan up
```

To show a custom view when your application is in maintenance mode, you may add something like the following to your application's `app/start/global.php` file:

```
App::down(function()  
{  
    return Response::view('maintenance', array(), 503);  
});
```

If the Closure passed to the `down` method returns `NULL`, maintenance mode will be ignored for that request.

Database configuration

Configuration

Laravel makes connecting with databases and running queries extremely simple. The database configuration for your application is located at `config/database.php`. In this file you may define all of your database connections, as well as specify which connection should be used by default. Examples for all of the supported database systems are provided in this file.

By default, Laravel's sample [environment configuration](#) is ready to use with [Laravel Homestead](#), which is a convenient virtual machine for doing Laravel development on your local machine. Of course, you are free to modify this configuration as needed for your local database.

SQL Server Configuration

Laravel supports SQL Server out of the box; however, you will need to add the connection configuration for the database:

```
'sqlsrv' => [
    'driver' => 'sqlsrv',
    'host' => env('DB_HOST', 'localhost'),
    'database' => env('DB_DATABASE', 'forge'),
    'username' => env('DB_USERNAME', 'forge'),
    'password' => env('DB_PASSWORD', ''),
    'charset' => 'utf8',
    'prefix' => '',
],
```

Read / Write Connections

Sometimes you may wish to use one database connection for SELECT statements, and another for INSERT, UPDATE, and DELETE statements. Laravel makes this a breeze, and the proper connections will always be used whether you are using raw queries, the query builder, or the Eloquent ORM.

To see how read / write connections should be configured, let's look at this example:

```
'mysql' => [
    'read' => [
        'host' => '192.168.1.1',
    ],
    'write' => [
        'host' => '196.168.1.2'
    ],
    'driver' => 'mysql',
    'database' => 'database',
    'username' => 'root',
    'password' => '',
    'charset' => 'utf8',
    'collation' => 'utf8_unicode_ci',
    'prefix' => '',
],
```



Note that two keys have been added to the configuration array: read and write. Both of these keys have array values containing a single key: host. The rest of the database options for the read and write connections will be merged from the main mysql array.

So, we only need to place items in the read and write arrays if we wish to override the values in the main array. So, in this case, 192.168.1.1 will be used as the "read" connection, while 192.168.1.2 will be used as the "write" connection. The database credentials, prefix, character set, and all other options in the main mysql array will be shared across both connections.

Artisan Command Line Tool

Introduction

Artisan is the name of the command-line interface included with Laravel. It provides a number of helpful commands for your use while developing your application. It is driven by the powerful Symfony Console component.

Usage

Listing All Available Commands

To view a list of all available Artisan commands, you may use the list command:

```
php artisan list
```

Viewing The Help Screen For A Command

Every command also includes a "help" screen which displays and describes the command's available arguments and options. To view a help screen, simply precede the name of the command with help:

```
php artisan help migrate
```

Specifying The Configuration Environment

You may specify the configuration environment that should be used while running a command using the `--env` switch:

```
php artisan migrate --env=local
```

Displaying Your Current Laravel Version

You may also view the current version of your Laravel installation using the `--version` option:

```
php artisan --version
```

Database Creation

By default there is no method given by laravel that can create database because developers have flexibility to use database of their own choice and create via their default interface and can connect via **config/database.php**.

Artisan Migration

Migrations are like version control for your database, allowing a team to easily modify and share the application's database schema. Migrations are typically paired with Laravel's schema builder to easily build your application's database schema.

The Laravel Schema facade provides database agnostic support for creating and manipulating tables. It shares the same expressive, fluent API across all of Laravel's supported database systems.

Generating Migrations

To create a migration, use the make:migration Artisan command:

```
php artisan make:migration create_users_table
```

The new migration will be placed in your database/migrations directory. Each migration file name contains a timestamp which allows Laravel to determine the order of the migrations.

The --table and --create options may also be used to indicate the name of the table and whether the migration will be creating a new table. These options simply pre-fill the generated migration stub file with the specified table:

```
php artisan make:migration add_votes_to_users_table --table=users
php artisan make:migration create_users_table --create=users
```

If you would like to specify a custom output path for the generated migration, you may use the --path option when executing the make:migration command. The provided path should be relative to your application's base path.

Migration Structure

A migration class contains two methods: up and down. The up method is used to add new tables, columns, or indexes to your database, while the down method should simply reverse the operations performed by the up method.

Within both of these methods you may use the Laravel schema builder to expressively create and modify tables. To learn about all of the methods available on the Schema builder, check out its documentation. For example, let's look at a sample migration that creates a flights table:

```
<?php
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
class CreateFlightsTable extends Migration{
    public function up() {
        Schema::create('flights', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
            $table->string('airline');
            $table->timestamps();
        });
    }
    public function down() {
        Schema::drop('flights');
    }
}
```

Creation Migration

To run all outstanding migrations for your application, use migrate Artisan command. If you are using the Homestead virtual machine, you should run this command from within your VM:

```
php artisan migrate
```

If you receive a "class not found" error when running migrations, try running the composer dump-autoload command and re-issuing the migrate command.

Rolling Back Migrations

To rollback the latest migration "operation", you may use the rollback command. Note that this rolls back the last "batch" of migrations that ran, which may include multiple migration files:

```
php artisan migrate:rollback
```

Writing Migrations

Creating Tables

To create a new database table, use the create method on the Schema facade. The create method accepts two arguments. The first is the name of the table, while the second is a Closure which receives a Blueprint object used to define the new table:

```
Schema::create('users', function (Blueprint $table) {  
    $table->increments('id');  
});
```

Of course, when creating the table, you may use any of the schema builder's column methods to define the table's columns.

Checking For Table / Column Existence

You may easily check for the existence of a table or column using the hasTable and hasColumn methods:

```
if (Schema::hasTable('users')) {  
    //  
}  
if (Schema::hasColumn('users', 'email')) {  
    //  
}
```

Renaming / Dropping Tables

To rename an existing database table, use the rename method:

```
Schema::rename($from, $to);
```

To drop an existing table, you may use the drop or dropIfExists methods:

```
Schema::drop('users');  
Schema::dropIfExists('users');
```

Renaming Tables with Foreign Keys

Before renaming a table, you should verify that any foreign key constraints on the table have an explicit name in your migration files instead of letting Laravel assign a convention based name. Otherwise, the foreign key constraint name will refer to the old table name.

Creating Columns

To update an existing table, we will use the table method on the Schema facade. Like the create method, the table method accepts two arguments: the name of the table and a Closure that receives a Blueprint instance we can use to add columns to the table:

```
Schema::table('users', function ($table) {
    $table->string('email');
});
```

Available Column Types

Of course, the schema builder contains a variety of column types that you may use when building your tables:

Command	Description
<code>\$table->bigIncrements('id');</code>	Incrementing ID (primary key) using a "UNSIGNED BIG INTEGER" equivalent.
<code>\$table->bigInteger('votes');</code>	BIGINT equivalent for the database.
<code>\$table->binary('data');</code>	BLOB equivalent for the database.
<code>\$table->boolean('confirmed');</code>	BOOLEAN equivalent for the database.
<code>\$table->char('name', 4);</code>	CHAR equivalent with a length.
<code>\$table->date('created_at');</code>	DATE equivalent for the database.
<code>\$table->dateTime('created_at');</code>	DATETIME equivalent for the database.
<code>\$table->decimal('amount', 5, 2);</code>	DECIMAL equivalent with a precision and scale.
<code>\$table->double('column', 15, 8);</code>	DOUBLE equivalent with precision, 15 digits in total and 8 after the decimal point.
<code>\$table->enum('choices', ['foo', 'bar']);</code>	ENUM equivalent for the database.
<code>\$table->float('amount');</code>	FLOAT equivalent for the database.
<code>\$table->increments('id');</code>	Incrementing ID (primary key) using a "UNSIGNED INTEGER" equivalent.
<code>\$table->integer('votes');</code>	INTEGER equivalent for the database.
<code>\$table->ipAddress('visitor');</code>	IP address equivalent for the database.
<code>\$table->json('options');</code>	JSON equivalent for the database.
<code>\$table->longText('description');</code>	LONGTEXT equivalent for the database.
<code>\$table->macAddress('device');</code>	MAC address equivalent for the database.
<code>\$table->mediumInteger('numbers');</code>	MEDIUMINT equivalent for the database.
<code>\$table->mediumText('description');</code>	MEDIUMTEXT equivalent for the database.
<code>\$table->rememberToken();</code>	Adds remember_token as VARCHAR(100) NULL.
<code>\$table->smallInteger('votes');</code>	SMALLINT equivalent for the database.
<code>\$table->string('email');</code>	VARCHAR equivalent column.

Command	Description
\$table->string('name', 100);	VARCHAR equivalent with a length.
\$table->text('description');	TEXT equivalent for the database.
\$table->time('sunrise');	TIME equivalent for the database.
\$table->tinyInteger('numbers');	TINYINT equivalent for the database.
\$table->timestamp('added_on');	TIMESTAMP equivalent for the database.
\$table->timestamps();	Adds created_at and updated_at columns.

Database seeding

Introduction

Laravel includes a simple method of seeding your database with test data using seed classes. All seed classes are stored in database/seeds. Seed classes may have any name you wish, but probably should follow some sensible convention, such as UsersTableSeeder, etc. By default, a DatabaseSeeder class is defined for you. From this class, you may use the call method to run other seed classes, allowing you to control the seeding order.

Writing Seeders

To generate a seeder, you may issue the make:seeder Artisan command. All seeders generated by the framework will be placed in the database/seeds directory:

```
php artisan make:seeder UsersTableSeeder
```

A seeder class only contains one method by default: run. This method is called when the db:seed Artisan command is executed. Within the run method, you may insert data into your database however you wish. You may use the query builder to manually insert data or you may use Eloquent model factories.

As an example, let's modify the DatabaseSeeder class which is included with a default installation of Laravel. Let's add a database insert statement to the run method:

```
<?php
use Illuminate\Database\Seeder;
use Illuminate\Database\Eloquent\Model;
class DatabaseSeeder extends Seeder
{
    public function run()
    {
        DB::table('users')->insert([
            'name' => str_random(10),
            'email' => str_random(10).'@gmail.com',
            'password' => bcrypt('secret'),
        ]);
    }
}
```


Running Seeders

Once you have written your seeder classes, you may use the `db:seed` Artisan command to seed your database. By default, the `db:seed` command runs the `DatabaseSeeder` class, which may be used to call other seed classes. However, you may use the `--class` option to specify a specific seeder class to run individually:

```
php artisan db:seed
php artisan db:seed --class=UsersTableSeeder
```

You may also seed your database using the `migrate:refresh` command, which will also rollback and re-run all of your migrations. This command is useful for completely rebuilding your database:

```
php artisan migrate:refresh --seed
```

Basic Routing

All Laravel routes are defined in the `app/Http/routes.php` file, which is automatically loaded by the framework. The most basic Laravel routes simply accept a URI and a Closure, providing a very simple and expressive method of defining routes:

```
Route::get('foo', function () {
    return 'Hello World';
});
```

The Default Routes File

The default `routes.php` file is loaded by the `RouteServiceProvider` and is automatically included in the web middleware group, which provides access to session state and CSRF protection. Most of the routes for your application will be defined within this file.

Available Router Methods

The router allows you to register routes that respond to any HTTP verb:

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
```

Route Parameters

Required Parameters

Of course, sometimes you will need to capture segments of the URI within your route. For example, you may need to capture a user's ID from the URL. You may do so by defining route parameters:

```
Route::get('user/{id}', function ($id) {
    return 'User ' . $id;
});
```

You may define as many route parameters as required by your route:

```
Route::get('posts/{post}/comments/{comment}', function ($postId, $commentId) {  
    //  
});
```

Route parameters are always encased within "curly" braces. The parameters will be passed into your route's Closure when the route is executed.

Optional Parameters

Occasionally you may need to specify a route parameter, but make the presence of that route parameter optional. You may do so by placing a ? mark after the parameter name. Make sure to give the route's corresponding variable a default value:

```
Route::get('user/{name?}', function ($name = null) {  
    return $name;  
});  
Route::get('user/{name?}', function ($name = 'John') {  
    return $name;  
});
```

Regular Expression Constraints

You may constrain the format of your route parameters using the where method on a route instance. The where method accepts the name of the parameter and a regular expression defining how the parameter should be constrained:

```
Route::get('user/{name}', function ($name) {  
    //  
})  
->where('name', '[A-Za-z]+');
```

```
Route::get('user/{id}', function ($id) {  
    //  
})  
->where('id', '[0-9]+');
```

```
Route::get('user/{id}/{name}', function ($id, $name) {  
    //  
})  
->where(['id' => '[0-9]+', 'name' => '[a-z]+']);
```

Route Filters (also known as Middleware)

Note: Below is prepared from Laravel 4.2

Route filters provide a convenient way of limiting access to a given route, which is useful for creating areas of your site which require authentication. There are several filters included in the Laravel framework, including an auth filter, an auth.basic filter, a guest filter, and a csrf filter. These are located in the app/filters.php file.

```
Route::filter('old', function(){
    if (Input::get('age') < 200) {
        return Redirect::to('home');
    }
});
```

If the filter returns a response, that response is considered the response to the request and the route will not execute. Any after filters on the route are also cancelled.

Attaching A Filter To A Route

```
Route::get('user', array('before' => 'old', function(){
    return 'You are over 200 years old!';
}));
```

Attaching A Filter To A Controller Action

```
Route::get('user', array('before' => 'old', 'uses' => 'UserController@showProfile'));
```

Attaching Multiple Filters To A Route

```
Route::get('user', array('before' => 'auth|old', function()
{
    return 'You are authenticated and over 200 years old!';
}));
```

Attaching Multiple Filters Via Array

```
Route::get('user', array('before' => array('auth', 'old'), function()
{
    return 'You are authenticated and over 200 years old!';
}));
```

Specifying Filter Parameters

```
Route::filter('age', function($route, $request, $value)
{
    //
});

Route::get('user', array('before' => 'age:200', function()
{
    return 'Hello World';
}));
```

After filters receive a \$response as the third argument passed to the filter:

```
Route::filter('log', function($route, $request, $response)
```

```
{
    //
});
```

Pattern Based Filters

You may also specify that a filter applies to an entire set of routes based on their URI.

```
Route::filter('admin', function()
{
    //
});
```

```
Route::when('admin/*', 'admin');
```

In the example above, the admin filter would be applied to all routes beginning with admin/. The asterisk is used as a wildcard, and will match any combination of characters.

You may also constrain pattern filters by HTTP verbs:

```
Route::when('admin/*', 'admin', array('post'));
```

Filter Classes

For advanced filtering, you may wish to use a class instead of a Closure. Since filter classes are resolved out of the application IoC Container, you will be able to utilize dependency injection in these filters for greater testability.

Registering A Class Based Filter

```
Route::filter('foo', 'FooFilter');
```

By default, the filter method on the FooFilter class will be called:

```
class FooFilter {
    public function filter() {
        // Filter logic...
    }
}
```

If you do not wish to use the filter method, just specify another method:

```
Route::filter('foo', 'FooFilter@foo');
```

Named Routes

Named routes make referring to routes when generating redirects or URLs more convenient. You may specify a name for a route like so:

```
Route::get('user/profile', array('as' => 'profile', function()
{
    //
}));
```

You may also specify route names for controller actions:

```
Route::get('user/profile', array('as' => 'profile', 'uses' => 'UserController@showProfile'));
```

Now, you may use the route's name when generating URLs or redirects:

```
$url = URL::route('profile');
$redirect = Redirect::route('profile');
```

You may access the name of a route that is running via the `currentRouteName` method:

```
$name = Route::currentRouteName();
```

Route Groups

Sometimes you may need to apply filters to a group of routes. Instead of specifying the filter on each route, you may use a route group:

```
Route::group(array('before' => 'auth'), function(){
    Route::get('/', function() {
        // Has Auth Filter
    });

    Route::get('user/profile', function() {
        // Has Auth Filter
    });
});
```

You may also use the namespace parameter within your group array to specify all controllers within that group as being in a given namespace:

```
Route::group(array('namespace' => 'Admin'), function()
{
    //
});
```

Sub-Domain Routing

Laravel routes are also able to handle wildcard sub-domains, and will pass your wildcard parameters from the domain:

Registering Sub-Domain Routes

```
Route::group(array('domain' => '{account}.myapp.com'), function(){
    Route::get('user/{id}', function($account, $id) {
        //
    });
});
```

Route Prefixing

A group of routes may be prefixed by using the prefix option in the attributes array of a group:

```
Route::group(array('prefix' => 'admin'), function(){
    Route::get('user', function() {
        //
    });
});
```

Route Model Binding

Model binding provides a convenient way to inject model instances into your routes. For example, instead of injecting a user's ID, you can inject the entire User model instance that matches the given ID. First, use the `Route::model` method to specify the model that should be used for a given parameter:

Binding A Parameter To A Model

```
Route::model('user', 'User');
```

Next, define a route that contains a `{user}` parameter:

```
Route::get('profile/{user}', function(User $user){  
    //  
});
```

Since we have bound the `{user}` parameter to the User model, a User instance will be injected into the route. So, for example, a request to `profile/1` will inject the User instance which has an ID of 1.

If you wish to specify your own "not found" behavior, you may pass a Closure as the third argument to the model method:

```
Route::model('user', 'User', function(){  
    throw new NotFoundHttpException;  
});
```

Throwing 404 Errors

There are two ways to manually trigger a 404 error from a route. First, you may use the `App::abort` method:

```
App::abort(404);
```

Second, you may throw an instance of `Symfony\Component\HttpFoundation\Exception\NotFoundHttpException`.

More information on handling 404 exceptions and using custom responses for these errors may be found in the errors section of the documentation.

Routing To Controllers

Laravel allows you to not only route to Closures, but also to controller classes, and even allows the creation of resource controllers.

Template Inheritance

Introduction

Blade is the simple, yet powerful templating engine provided with Laravel. Unlike other popular PHP templating engines, Blade does not restrict you from using plain PHP code in your views. All Blade views are compiled into plain PHP code and cached until they are modified, meaning Blade adds essentially zero overhead to your application. Blade view files use the `.blade.php` file extension and are typically stored in the `resources/views` directory.

Template Inheritance

Master Layout

Two of the primary benefits of using Blade are *template inheritance* and *sections*. To get started, let's take a look at a simple example. First, we will examine a "master" page layout. Since most web applications maintain the same general layout across various pages, it's convenient to define this layout as a single Blade view:

```
<!-- Stored in resources/views/layouts/master.blade.php -->
<html>
  <head>
    <title>App Name - @yield('title')</title>
  </head>
  <body>
    @section('sidebar')
      This is the master sidebar.
    @show

    <div class="container">
      @yield('content')
    </div>
  </body>
</html>
```

As you can see, this file contains typical HTML mark-up. However, take note of the `@section` and `@yield` directives. The `@section` directive, as the name implies, defines a section of content, while the `@yield` directive is used to display the contents of a given section.

Now that we have defined a layout for our application, let's define a child page that inherits the layout.

Extending the master layout

When defining a child page, you may use the Blade `@extends` directive to specify which layout the child page should "inherit". Views which `@extends` a Blade layout may inject content into the layout's sections using `@section` directives. Remember, as seen in the example above, the contents of these sections will be displayed in the layout using `@yield`:

```
<!-- Stored in resources/views/child.blade.php -->

@extends('layouts.master')

@section('title', 'Page Title')

@section('sidebar')
    @parent
    <p>This is appended to the master sidebar.</p>
@endsection

@section('content')
    <p>This is my body content.</p>
@endsection
```

In this example, the sidebar section is utilizing the `@parent` directive to append (rather than overwriting) content to the layout's sidebar. The `@parent` directive will be replaced by the content of the layout when the view is rendered.

Of course, just like plain PHP views, Blade views may be returned from routes using the global view helper function:

```
Route::get('blade', function () {
    return view('child');
});
```

Display Variables

You may display data passed to your Blade views by wrapping the variable in "curly" braces. For example, given the following route:

```
Route::get('greeting', function () {
    return view('welcome', ['name' => 'Samantha']);
});
```

You may display the contents of the name variable like so:

```
Hello, {{ $name }}.
```

Of course, you are not limited to displaying the contents of the variables passed to the view. You may also echo the results of any PHP function. In fact, you can put any PHP code you wish inside of a Blade echo statement:

The current UNIX timestamp is `{{ time() }}`.

Note: Blade `{ }` statements are automatically sent through PHP's **htmlspecialchars** function to prevent XSS attacks.

Blade & JavaScript Frameworks

Since many JavaScript frameworks also use "curly" braces to indicate a given expression should be displayed in the browser, you may use the `@` symbol to inform the Blade rendering engine an expression should remain untouched.

For example:

```
<h1>Laravel</h1>
Hello, @{{ name }}.
```

In this example, the `@` symbol will be removed by Blade; however, `{{ name }}` expression will remain untouched by the Blade engine, allowing it to instead be rendered by your JavaScript framework.

Echoing Data If It Exists

Sometimes you may wish to echo a variable, but you aren't sure if the variable has been set. We can express this in verbose PHP code like so:

```
{{ isset($name) ? $name : 'Default' }}
```

However, instead of writing a ternary statement, Blade provides you with the following convenient short-cut:

```
{{ $name or 'Default' }}
```

In this example, if the `$name` variable exists, its value will be displayed. However, if it does not exist, the word `Default` will be displayed.

Displaying Unescaped Data

By default, Blade `{{ }}` statements are automatically sent through PHP's **htmlspecialchars** function to prevent XSS attacks. If you do not want your data to be escaped, you may use the following syntax:

```
Hello, {!! $name !!}.
```

Blade conditional statements

In addition to template inheritance and displaying data, Blade also provides convenient short-cuts for common PHP control structures, such as conditional statements and loops. These short-cuts provide a very clean, terse way of working with PHP control structures, while also remaining familiar to their PHP counterparts.

If Statements

You may construct if statements using the `@if`, `@elseif`, `@else`, and `@endif` directives. These directives function identically to their PHP counterparts:

```
@if (count($records) === 1)
    I have one record!
}elseif (count($records) > 1)
    I have multiple records!
@else
    I don't have any records!
@endif
```

For convenience, Blade also provides an `@unless` directive:

```
@unless (Auth::check())
    You are not signed in.
@endunless
```

You may also determine if a given layout section has any content using the `@hasSection` directive:

```
<title>
    @hasSection ('title')
        @yield('title') - App Name
    @else
        App Name
    @endif
</title>
```

Blade Loops

In addition to conditional statements, Blade provides simple directives for working with PHP's supported loop structures. Again, each of these directives functions identically to their PHP counterparts:

```
@for ($i = 0; $i < 10; $i++)
    The current value is {{ $i }}
@endfor

@foreach ($users as $user)
    <p>This is user {{ $user->id }}</p>
@endforeach

@forelse ($users as $user)
    <li>{{ $user->name }}</li>
@empty
    <p>No users</p>
@endforelse

@while (true)
    <p>I'm looping forever.</p>
@endwhile
```

When using loops you might need to end the loop or skip the current iteration:

```
@foreach ($users as $user)
    @if ($user->type == 1)
        @continue
    @endif

    <li>{{ $user->name }}</li>

    @if ($user->number == 5)
        @break
    @endif
@endforeach
```

You may also include the condition with the directive declaration in one line:

```
@foreach ($users as $user)
    @continue($user->type == 1)

    <li>{{ $user->name }}</li>

    @break($user->number == 5)
@endforeach
```

Comments

Blade also allows you to define comments in your views. However, unlike HTML comments, Blade comments are not included in the HTML returned by your application:

```
{{-- This comment will not be present in the rendered HTML --}}
```

Executing PHP functions in blade

Laravel recipes suggest a simple but effective way to do it without including the php tags

```
{{--*/ $var = 'test' /*--}}
```

{{ ----}} works as a blade comment /and/ reverts the effect of comment resulting on

```
<?php $var = 'test' ?>
```

The problem is that is longer than including php tags.

Running Raw SQL Queries

Once you have configured your database connection, you may run queries using the DB facade. The DB facade provides methods for each type of query: select, update, insert, delete, and statement.

Running A Select Query

To run a basic query, we can use the select method on the DB facade:

```
<?php
namespace App\Http\Controllers;
use DB;
use App\Http\Controllers\Controller;

class UserController extends Controller{
    public function index() {
        $users = DB::select('select * from users where active = ?', [1]);
        return view('user.index', ['users' => $users]);
    }
}
```

The first argument passed to the select method is the raw SQL query, while the second argument is any parameter bindings that need to be bound to the query. Typically, these are the values of the where clause constraints. Parameter binding provides protection against SQL injection.

The select method will always return an array of results. Each result within the array will be a PHP stdClass object, allowing you to access the values of the results:

```
foreach ($users as $user) {
    echo $user->name;
}
```

Using Named Bindings

Instead of using ? to represent your parameter bindings, you may execute a query using named bindings:

```
$results = DB::select('select * from users where id = :id', ['id' => 1]);
```

Running An Insert Statement

To execute an insert statement, you may use the insert method on the DB facade. Like select, this method takes the raw SQL query as its first argument, and bindings as the second argument:

```
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);
```

Running An Update Statement

The update method should be used to update existing records in the database. The number of rows affected by the statement will be returned by the method:

```
$affected = DB::update('update users set votes = 100 where name = ?', ['John']);
```

Running A Delete Statement

The delete method should be used to delete records from the database. Like update, the number of rows deleted will be returned:

```
$deleted = DB::delete('delete from users');
```

Running A General Statement

Some database statements should not return any value. For these types of operations, you may use the statement method on the DB facade:

```
DB::statement('drop table users');
```

Database Transactions

To run a set of operations within a database transaction, you may use the transaction method on the DB facade. If an exception is thrown within the transaction Closure, the transaction will automatically be rolled back. If the Closure executes successfully, the transaction will automatically be committed. You don't need to worry about manually rolling back or committing while using the transaction method:

```
DB::transaction(function () {  
    DB::table('users')->update(['votes' => 1]);  
  
    DB::table('posts')->delete();  
});
```

Manually Using Transactions

If you would like to begin a transaction manually and have complete control over rollbacks and commits, you may use the beginTransaction method on the DB facade:

```
DB::beginTransaction();
```

You can rollback the transaction via the rollBack method:

```
DB::rollBack();
```

Lastly, you can commit a transaction via the commit method:

```
DB::commit();
```

Eloquent

Introduction

The Eloquent ORM included with Laravel provides a beautiful, simple ActiveRecord implementation for working with your database. Each database table has a corresponding "Model" which is used to interact with that table. Models allow you to query for data in your tables, as well as insert new records into the table.

Naming conventions

To get started, let's create an Eloquent model. Models typically live in the app directory, but you are free to place them anywhere that can be auto-loaded according to your composer.json file. All Eloquent models extend Illuminate\Database\Eloquent\Model class.

The easiest way to create a model instance is using the make:model Artisan command:

```
php artisan make:model User
```

If you would like to generate a database migration when you generate the model, you may use the --migration or -m option:

```
php artisan make:model User --migration  
php artisan make:model User -m
```

Model Conventions

Now, let's look at an example Flight model class, which we will use to retrieve and store information from our flights database table:

```
<?php  
namespace App;  
use Illuminate\Database\Eloquent\Model;  
class Flight extends Model{  
    //  
}
```

Table Names

Note that we did not tell Eloquent which table to use for our Flight model. The "snake case", plural name of the class will be used as the table name unless another name is explicitly specified. So, in this case, Eloquent will assume the Flight model stores records in the flights table. You may specify a custom table by defining a table property on your model:

```
<?php  
namespace App;  
use Illuminate\Database\Eloquent\Model;  
class Flight extends Model{  
    protected $table = 'my_flights';  
}
```


Primary Keys

Eloquent will also assume that each table has a primary key column named `id`. You may define a `$primaryKey` property to override this convention.

In addition, Eloquent assumes that the primary key is an incrementing integer value, which means that by default the primary key will be cast to an `int` automatically. If you wish to use a non-incrementing or a non-numeric primary key you must set the public `$incrementing` property on your model to `false`.

Timestamps

By default, Eloquent expects `created_at` and `updated_at` columns to exist on your tables. If you do not wish to have these columns automatically managed by Eloquent, set the `$timestamps` property on your model to `false`:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Flight extends Model{
    public $timestamps = false;
}
```

If you need to customize the format of your timestamps, set the `$dateFormat` property on your model. This property determines how date attributes are stored in the database, as well as their format when the model is serialized to an array or JSON:

```
<?php
namespace App;
use Illuminate\Database\Eloquent\Model;
class Flight extends Model{
    protected $dateFormat = 'U';
}
```

Create

To create a new record in the database, simply create a new model instance, set attributes on the model, and then call the save method:

```
<?php
namespace App\Http\Controllers;
use App\Flight;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
class FlightController extends Controller{
    public function store(Request $request) {
        // Validate the request...
        $flight = new Flight;
        $flight->name = $request->name;
        $flight->save();
    }
}
```

In this example, we simply assign the name parameter from the incoming HTTP request to the name attribute of the App\Flight model instance. When we call the save method, a record will be inserted into the database. The created_at and updated_at timestamps will automatically be set when the save method is called, so there is no need to set them manually.

Read

Of course, in addition to retrieving all of the records for a given table, you may also retrieve single records using find and first. Instead of returning a collection of models, these methods return a single model instance:

```
// Retrieve a model by its primary key...
$flight = App\Flight::find(1);
// Retrieve the first model matching the query constraints...
$flight = App\Flight::where('active', 1)->first();
```

You may also call the find method with an array of primary keys, which will return a collection of the matching records:

```
$flights = App\Flight::find([1, 2, 3]);
```

Not Found Exceptions

Sometimes you may wish to throw an exception if a model is not found. This is particularly useful in routes or controllers. The findOrFail and firstOrFail methods will retrieve the first result of the query. However, if no result is found, a Illuminate\Database\Eloquent\ModelNotFoundException will be thrown:

```
$model = App\Flight::findOrFail(1);
$model = App\Flight::where('legs', '>', 100)->firstOrFail();
```

If the exception is not caught, a 404 HTTP response is automatically sent back to the user, so it is not necessary to write explicit checks to return 404 responses when using these methods:

```
Route::get('/api/flights/{id}', function ($id) {  
    return App\Flight::findOrFail($id);  
});
```

Update

The save method may also be used to update models that already exist in the database. To update a model, you should retrieve it, set any attributes you wish to update, and then call the save method. Again, the updated_at timestamp will automatically be updated, so there is no need to manually set its value:

```
$flight = App\Flight::find(1);  
$flight->name = 'New Flight Name';  
$flight->save();
```

Updates can also be performed against any number of models that match a given query. In this example, all flights that are active and have a destination of San Diego will be marked as delayed:

```
App\Flight::where('active', 1)  
->where('destination', 'San Diego')  
->update(['delayed' => 1]);
```

The update method expects an array of column and value pairs representing the columns that should be updated.

Delete

To delete a model, call the delete method on a model instance:

```
$flight = App\Flight::find(1);  
$flight->delete();
```

Deleting an Existing Model by Key

In the example above, we are retrieving the model from the database before calling the delete method. However, if you know the primary key of the model, you may delete the model without retrieving it. To do so, call the destroy method:

```
App\Flight::destroy(1);  
App\Flight::destroy([1, 2, 3]);  
App\Flight::destroy(1, 2, 3);  
Deleting Models By Query
```

Of course, you may also run a delete query on a set of models. In this example, we will delete all flights that are marked as inactive:

```
$deletedRows = App\Flight::where('active', 0)->delete();
```

Introduction

Laravel provides several different approaches to validate your application's incoming data. By default, Laravel's base controller class uses a `ValidatesRequests` trait which provides a convenient method to validate incoming HTTP request with a variety of powerful validation rules.

Defining the Routes

Defining the Routes

First, let's assume we have the following routes defined in our `app/Http/routes.php` file:

```
// Display a form to create a blog post...
Route::get('post/create', 'PostController@create');

// Store a new blog post...
Route::post('post', 'PostController@store');
```

Of course, the GET route will display a form for the user to create a new blog post, while the POST route will store the new blog post in the database.

Creating the Controller

Next, let's take a look at a simple controller that handles these routes. We'll leave the store method empty for now:

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller{
    public function create() {
        return view('post.create');
    }
    public function store(Request $request) {
        // validate and store the blog post...
    }
}
```

Writing the Validation Logic

Now we are ready to fill in our store method with the logic to validate the new blog post. If you examine your application's base controller (`App\Http\Controllers\Controller`)

class, you will see that the class uses a `ValidatesRequests` trait. This trait provides a convenient `validate` method in all of your controllers.

The `validate` method accepts an incoming HTTP request and a set of validation rules. If the validation rules pass, your code will keep executing normally; however, if validation fails, an exception will be thrown and the proper error response will automatically be sent back to the user. In the case of a traditional HTTP request, a redirect response will be generated, while a JSON response will be sent for AJAX requests.

To get a better understanding of the `validate` method, let's jump back into the `store` method:

```
public function store(Request $request)
{
    $this->validate($request, [
        'title' => 'required|unique:posts|max:255',
        'body' => 'required',
    ]);

    // The blog post is valid, store in database...
}
```

Displaying the Validation Errors

So, what if the incoming request parameters do not pass the given validation rules? As mentioned previously, Laravel will automatically redirect the user back to their previous location. In addition, all of the validation errors will automatically be flashed to the session.

Again, notice that we did not have to explicitly bind the error messages to the view in our GET route. This is because Laravel will check for errors in the session data, and automatically bind them to the view if they are available. The `$errors` variable will be an instance of `Illuminate\Support\MessageBag`.

So, in our example, the user will be redirected to our controller's `create` method when validation fails, allowing us to display the error messages in the view:

```
<!-- /resources/views/post/create.blade.php -->
<h1>Create Post</h1>
@if (count($errors) > 0)
    <div class="alert alert-danger">
        <ul>
            @foreach ($errors->all() as $error)
                <li>{{ $error }}</li>
            @endforeach
        </ul>
    </div>
@endif
```

```

</div>
@endif
<!-- Create Post Form -->

```

Array validations

Validating array form input fields doesn't have to be a pain. For example, to validate that each e-mail in a given array input field is unique, you may do the following:

```

$validator = Validator::make($request->all(), [
    'person.*.email' => 'email|unique:users',
    'person.*.first_name' => 'required_with:person.*.last_name',
]);

```

Likewise, you may use the * character when specifying your validation messages in your language files, making it a breeze to use a single validation message for array based fields:

```

'custom' => [
    'person.*.email' => [
        'unique' => 'Each person must have a unique e-mail address',
    ]
],

```

Creating new validators

If you do not want to use the ValidatesRequests trait's validate method, you may create a validator instance manually using the Validator facade. The make method on the facade generates a new validator instance:

```

<?php
namespace App\Http\Controllers;
use Validator;
use Illuminate\Http\Request;
use App\Http\Controllers\Controller;

class PostController extends Controller{
    public function store(Request $request) {
        $validator = Validator::make($request->all(), [
            'title' => 'required|unique:posts|max:255',
            'body' => 'required',
        ]);

        if ($validator->fails()) {
            return redirect('post/create')
                ->withErrors($validator)
                ->withInput();
        }
        // Store the blog post...
    }
}

```

```
}
}
```

The first argument passed to the make method is the data under validation. The second argument is the validation rules that should be applied to the data.

After checking if the request failed to pass validation, you may use the withErrors method to flash the error messages to the session. When using this method, the \$errors variable will automatically be shared with your views after redirection, allowing you to easily display them back to the user. The withErrors method accepts a validator, a MessageBag, or a PHP array.

Error messages

After calling the errors method on a Validator instance, you will receive an Illuminate\Support\MessageBag instance, which has a variety of convenient methods for working with error messages.

Retrieving the First Error Message for a Field

To retrieve the first error message for a given field, use the first method:

```
$messages = $validator->errors();
echo $messages->first('email');
```

Retrieving All Error Messages for a Field

If you wish to simply retrieve an array of all of the messages for a given field, use the get method:

```
foreach ($messages->get('email') as $message) {
    //
}
```

Retrieving All Error Messages for All Fields

To retrieve an array of all messages for all fields, use the all method:

```
foreach ($messages->all() as $message) {
    //
}
```

Determining If Messages Exist For a Field

```
if ($messages->has('email')) {
    //
}
```

Retrieving an Error Message with a Format

```
echo $messages->first('email', '<p>:message</p>');
Retrieving All Error Messages With A Format
foreach ($messages->all('<li>:message</li>') as $message) {
    //
}
```


Custom validation rules.

Custom Error Messages

If needed, you may use custom error messages for validation instead of the defaults. There are several ways to specify custom messages. First, you may pass the custom messages as the third argument to the Validator::make method:

```
$messages = [
    'required' => 'The :attribute field is required.',
];
$validator = Validator::make($input, $rules, $messages);
```

In this example, the :attribute place-holder will be replaced by the actual name of the field under validation. You may also utilize other place-holders in validation messages. For example:

```
$messages = [
    'same' => 'The :attribute and :other must match.',
    'size' => 'The :attribute must be exactly :size.',
    'between' => 'The :attribute must be between :min - :max.',
    'in' => 'The :attribute must be one of the following types: :values',
];
```

Specifying a Custom Message for a Given Attribute

Sometimes you may wish to specify a custom error messages only for a specific field. You may do so using "dot" notation. Specify the attribute's name first, followed by the rule:

```
$messages = [
    'email.required' => 'We need to know your e-mail address!',
];
```

Specifying Custom Messages in Language Files

In many cases, you may wish to specify your attribute specific custom messages in a language file instead of passing them directly to the Validator. To do so, add your messages to custom array in the resources/lang/xx/validation.php language file.

```
'custom' => [
    'email' => [
        'required' => 'We need to know your e-mail address!',
    ],
],
```

Available Validators:

Accepted

The field under validation must be yes, on, 1, or true. This is useful for validating "Terms of Service" acceptance.

After (Date)

after:date

The field under validation must be a value after a given date. The dates will be passed into the strtotime PHP function:

```
'start_date' => 'required|date|after:tomorrow'
```

Instead of passing a date string to be evaluated by strtotime, you may specify another field to compare against the date:

```
'finish_date' => 'required|date|after:start_date'
```

Alpha

The field under validation must be entirely alphabetic characters.

Alpha Dash

alpha_dash

The field under validation may have alpha-numeric characters, as well as dashes and underscores.

Alpha Numeric

alpha_num

The field under validation must be entirely alpha-numeric characters.

Array

The field under validation must be a PHP array.

Before (Date)

before:date

The field under validation must be a value preceding the given date. The dates will be passed into the PHP strtotime function.

Between

between:min,max

The field under validation must have a size between the given min and max. Strings, numerics, and files are evaluated in the same fashion as the size rule.

Boolean

The field under validation must be able to be cast as a boolean. Accepted input are true, false, 1, 0, "1", and "0".

Date

The field under validation must be a valid date according to the strtotime PHP function.

Date Format

date_format:format

The field under validation must match the given format. The format will be evaluated using the PHP date_parse_from_format function. You should use either date or date_format when validating a field, not both.

Different

different:field

The field under validation must have a different value than field.

Digits

digits:value

The field under validation must be numeric and must have an exact length of value.

Digits Between

digits_between:min,max

The field under validation must have a length between the given min and max.

E-Mail

The field under validation must be formatted as an e-mail address.

Exists (Database)

exists:table,column

The field under validation must exist on a given database table.

'state' => 'exists:states'

Image (File)

The field under validation must be a successfully uploaded file.

image

The file under validation must be an image (jpeg, png, bmp, gif, or svg)

In

in:foo,bar,...

The field under validation must be included in the given list of values.

in_array:anotherfield

The field under validation must exist in anotherfield's values.

Integer

The field under validation must be an integer.

Max

The field under validation must be less than or equal to a maximum value. Strings, numerics, and files are evaluated in the same fashion as the size rule.

Min

min:value

The field under validation must have a minimum value. Strings, numerics, and files are evaluated in the same fashion as the size rule.

Not In

not_in:foo,bar,...

The field under validation must not be included in the given list of values.

Numeric

The field under validation must be numeric.

Regular Expression

regex:pattern

The field under validation must match the given regular expression.

Required

The field under validation must be present in the input data and not empty. A field is considered "empty" if one of the following conditions are true:

- The value is null.
- The value is an empty string.
- The value is an empty array or empty Countable object.
- The value is an uploaded file with no path.

String

The field under validation must be a string.