

WEBD 236

Web Information Systems Programming

Week 7

Copyright © 2013-2017
Todd Whittaker and Scott Sharkey
(sharkesc@franklin.edu)

Agenda

- This week's expected outcomes
- This week's topics
- This week's homework
- Upcoming deadlines
- Questions and answers

Week 7 Outcomes

- Employ advanced features of functions (pass by reference, closures, variable argument lists) to solve problems.
- Distinguish between objects and scalar data types.
- Describe the five properties of object-orientation.
- Employ encapsulation, inheritance, and polymorphism to build web applications.

Functions

- Basic review
 - Use function keyword to create a function
 - Use the name of the function with () to call it.

```
function uninteresting() {  
    return 42;  
}
```

```
$x = uninteresting();
```

Functions

- Basic review
 - Use function keyword to create a function
 - Use the name of the function with () to call it.

```
function uninteresting() {  
    return 42;  
}
```

```
$x = uninteresting();
```

Function names become global symbols. Defining two functions with the same name (or including the defining file multiple times) is an error.

Functions

- Basic review
 - Can specify arguments to be passed into parameters.

```
function double($num) {  
    return 2 * num;  
}
```

```
$x = double(21);
```

Functions

- Basic review
 - All arguments are passed by *value*, including arrays.

```
function doubleAll($arr) {  
    foreach ($arr as $key => $value) {  
        $arr[$key] = $arr[$key] * 2;  
    }  
}
```

```
$numbers = array(1, 2, 3, 4, 5);  
doubleAll($numbers);  
print_r($numbers);
```

Functions

- Basic review

- All arguments are passed by *value*, including arrays.

```
function doubleAll($arr) {  
    foreach ($arr as $key => $value) {  
        $arr[$key] = $arr[$key] * 2;  
    }  
}
```

```
$numbers = array(1, 2, 3, 4, 5);  
doubleAll($numbers);  
print_r($numbers);
```

```
Array  
(  
    [0] => 1  
    [1] => 2  
    [2] => 3  
    [3] => 4  
    [4] => 5  
)
```


Functions

- Basic review
 - Arguments can be passed by *reference* so that the modifications made are seen outside the function.

```
function doubleAll(&$arr) {  
    foreach ($arr as $key => $value) {  
        $arr[$key] = $arr[$key] * 2;  
    }  
}
```

```
$numbers = array(1, 2, 3, 4, 5);  
doubleAll($numbers);  
print_r($numbers);
```

Functions

- Basic review

- Arguments can be passed by *reference* so that the modifications made *inside* the function.

```
function doubleAll(&$arr) {  
    foreach ($arr as $key => $value) {  
        $arr[$key] = $arr[$key] * 2;  
    }  
}
```

```
$numbers = array(1, 2, 3, 4, 5);  
doubleAll($numbers);  
print_r($numbers);
```

```
Array  
(  
    [0] => 2  
    [1] => 4  
    [2] => 6  
    [3] => 8  
    [4] => 10  
)
```

Functions

- Basic review

- Arguments can be passed by *reference* so that the modifications made *inside* the function.

```
function doubleAll(&$arr) {  
    foreach ($arr as $key => $value) {  
        $arr[$key] = $arr[$key] * 2;  
    }  
}
```

```
$numbers = array(1, 2, 3, 4, 5);  
doubleAll($numbers);  
print_r($numbers);
```

```
Array  
(  
    [0] => 2  
    [1] => 4  
    [2] => 6  
    [3] => 8  
    [4] => 10
```

Functions that affect the program this way are said to have side effects, and these must be documented in comments.

Functions

- Basic review
 - You can specify default values for arguments.

```
function multiplyAll(&$arr, $by = 2) {  
    foreach ($arr as $key => $value) {  
        $arr[$key] = $arr[$key] * $by;  
    }  
}
```

```
$numbers = array(1, 2, 3, 4, 5);  
multiplyAll($numbers);  
print_r($numbers);
```

If you don't specify a second parameter, then it will use the default value of 2 for \$by.

Functions

- Advanced features
 - To prevent name clashes, use a namespace to enclose your libraries of functions

```
namespace webd236 {  
    function largest() {  
        // ... omitted ...  
    }  
    // more functions defined here  
}  
$result = webd236\largest(4, 6, 3, 1, 8, 2);
```

Functions

- Advanced features
 - To prevent name clashes, use a namespace to enclose your libraries of functions

```
namespace webd236 {  
    function largest() {  
        // ... omitted  
    }  
    // more fun  
}  
$result = webd
```

Namespace declaration must be the first statement in a script.

Functions

- Advanced features
 - You can pass a function as a parameter to another function.

```
function map($arr, $func) {  
    $result = array();  
    foreach ($arr as $key => $value) {  
        $result[$key] = $func($value);  
    }  
    return $result;  
}  
  
function double($num) {  
    return 2 * $num;  
}  
  
$numbers = array(1, 2, 3, 4, 5);  
$doubled = map($numbers, 'double');
```

Functions

- Advanced features
 - Functions can be assigned to variables and both called and passed around.

```
function map($arr, $func) {  
    $result = array();  
    foreach ($arr as $key => $value) {  
        $result[$key] = $func($value);  
    }  
    return $result;  
}  
  
$f = function($num) {  
    return $num * 2;  
};  
  
$numbers = array(1, 2, 3, 4, 5);  
$doubled = map($numbers, $f);
```


Functions

- Advanced features
 - For a “throw away” function, you can use an anonymous function.

```
function map($arr, $func) {  
    $result = array();  
    foreach ($arr as $key => $value) {  
        $result[$key] = $func($value);  
    }  
    return $result;  
}  
$numbers = array(1, 2, 3, 4, 5);  
$doubled = map($numbers, function($num) {return $num * 2;});
```

Functions

- Advanced features
 - When you treat functions as variables, it creates a *closure*.

```
function counter($start = 1) {  
    $counter = $start;  
    return function() use (&$counter) {  
        return $counter++;  
    };  
}  
$func = counter(5);  
$arr = array();  
for ($i = 0; $i < 5; ++$i) {  
    $arr[] = $func();  
}
```

```
Array  
(  
    [0] => 5  
    [1] => 6  
    [2] => 7  
    [3] => 8  
    [4] => 9  
)
```

Functions

- Advanced features
 - When you treat functions as variables, it creates a *closure*.

```
function makeCounter($start = 1) {  
    $counter = $start;  
    return function() use (&$counter) {  
        return $counter++;  
    };  
}  
$counter = makeCounter(5);  
$arr = array();  
for ($i = 0; $i < 5; ++$i) {  
    $arr[] = $counter();  
}
```

```
Array  
(  
    [0] => 5  
    [1] => 6
```

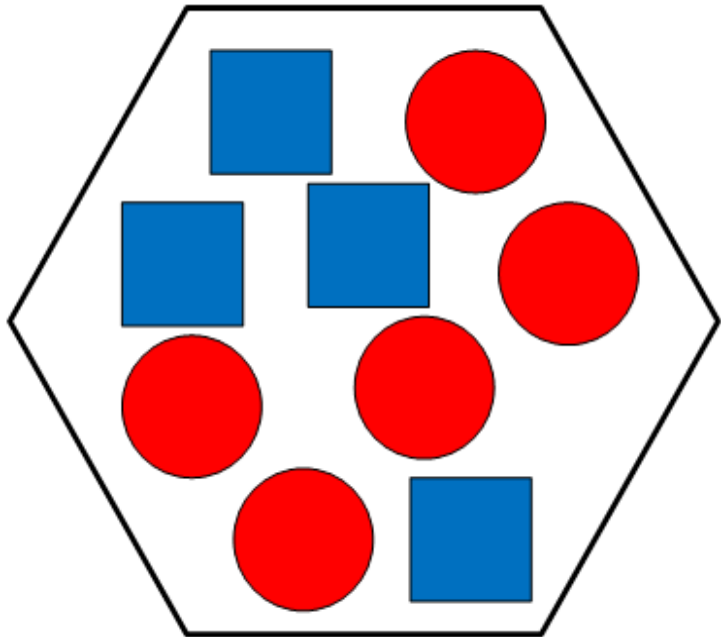
Data for the operation of the function is kept hidden inside the function, and the function retains this state from call to call. Much like an *object*.

Object Oriented Concepts

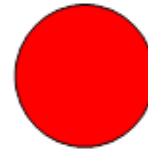
- What is an object?
 - › All objects have 3 characteristics
 - › **State** – data associated with the object
 - › **Behavior** – code associated with the object
 - › **Identity** – a location where the object exists in memory

Object Oriented Concepts

- What is an object



Behavior (methods)



State (properties)



Identity (container)

Object Oriented Concepts

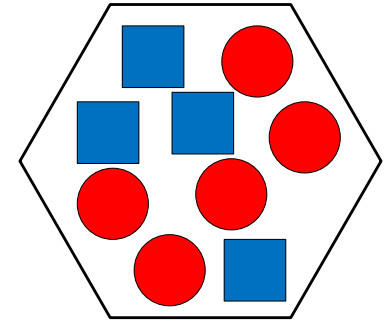
- What is an object?

- › State (properties)

- › Data kept inside the object.

- › The internal representation of the object need not be the same as how it is seen from the outside.

- › Ex: PDO object in represents a connection to a database using a DSN and username/password credentials.



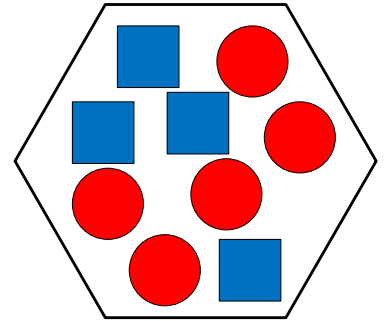
Object Oriented Concepts

- What is an object?

- › Behavior (method)

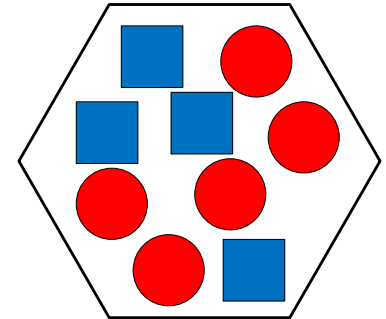
- › A function kept inside an object

- › Has access to all the properties of the object as well as any parameters and global variables.



Object Oriented Concepts

- What is an object?
 - › Identity (container)
 - › Memory location of the object.
 - › One variable that holds many other variables (methods and properties) within itself.
 - › Very similar to an associative array.



Object-oriented Programming

- Definition of *object*.
 - › An object is the combination of data and the functions (called *methods*) that act on that data in a single package.
 - › Contrast with structured approach
 - › Write many independent functions that accept parameters containing the data to act on.
 - › E.g. all array functions have an array as the first parameter: `array_slice($array, $offset, $length, $preserve_keys)`

Object-oriented Programming

- Contrasting the two approaches
 - A structured approach:

```
$array = array(1, 2, 3, 4, 5);  
$array = array_slice($array, 3);
```

- An object-oriented approach:

```
$array = array(1, 2, 3, 4, 5);  
$array -> slice(3);
```

The data on which slice operates is already part of the object. *Note: this isn't working PHP code; it just illustrates the OO approach.*

Object-oriented Programming

- What is a *class*?
 - › A *class* is the template for making an object.
 - › Defines what data the object holds
 - › Defines the methods that act on that object
 - › Defines a special method that builds the object called a *constructor*.

Think of it as a blueprint. You can build many houses based on the same blueprint, but they're not the *same* house.

Object-oriented Programming

- Ex: a counter class

```
class Counter {  
    private $count;  
  
    public function __construct($start = 1) {  
        $this -> count = $start;  
    }  
  
    public function next() {  
        $result = $this -> count;  
        ++$this -> count;  
        return $result;  
    }  
}
```

Object-oriented Programming

- Ex: a counter class

```
class Counter {  
    private $count;  
  
    public function __construct($start = 1) {  
        $this -> count = $start;  
    }  
  
    public function next() {  
        $result = $this -> count;  
        ++$this -> count;  
        return $result;  
    }  
}
```

```
$arr = array();  
$counter = new Counter(5);  
for ($i = 0; $i < 5; ++$i) {  
    $arr[] = $counter -> next();  
}
```

Object-oriented Programming

- Mechanics of OOP
 - › Inside a class, the current object is called `$this`.
 - › Use the member access operator (`->`) both external to and internal to the object to access methods and data items.
 - › Constructor is called `__construct`
 - › Use the new operator and the name of the class to construct an object.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Increased modularity:** the unit of modularity becomes the object and systems become a set of cooperating objects. Objects are typically smaller, and therefore there are more modules.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Simplified analysis:** The real world consists of objects. In the real world, objects have attributes and behaviors. When the method of programming and the real world align, then the process of analyzing the problem becomes simpler.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Easier testing:** With increased modularity (i.e. smaller, more tightly focused objects) comes easier testing of those objects. Tests can be written to validate the behavior of each object independently of the entire system.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Increased comprehension:** Since objects are kept small (on the order of perhaps a couple of hundred lines of code) programmers are better able to keep the entire state of the object in their working memory at once.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Looser coupling:** *Coupling* is a measure of the degree to which a class depends on other classes to work properly. It is rare that an object acts in isolation of other objects, the connections between objects are clearly defined by the methods.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Tighter cohesion:** *Cohesion* is a measure of the degree to which a class models a single concept. Objects are smaller modules of modeling than those found in non-object oriented systems, and hence tend to promote tighter cohesion.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Increased reuse:** Because objects are loosely coupled and highly cohesive, they are easier to reuse within the same or different systems.

Object-Oriented Benefits

- Many benefits to grouping data and methods together:
 - **Better maintainability:** All of the aforementioned benefits lead to systems that are much more flexible to change and much easier to fix when bugs are encountered.

5 Properties of OOP

- “C A P I E”

- **Composition:** one object becomes data within another object. Defined by the phrase “has-a”
- **Abstraction:** identifying those real-world attributes that we should model in software.
- **Polymorphism:** Behavior varies based on the type of the object.
- **Inheritance:** A relationship between classes defined by the phrase “is-a”
- **Encapsulation:** information hiding

5 Properties of OOP

- Composition

A user “has-a” date object as an attribute.

```
class User {  
  private $email;  
  private $dob;  
  public function __construct($email, $dob) {  
    $this -> email = $email;  
    $this -> dob = $dob;  
  }  
  public function isOver21() {  
    $now = new DateTime();  
    $now -> modify("-21 years");  
    return $now->getTimestamp() >  
      $this -> dob -> getTimeStamp();  
  }  
}  
$usr = new User("user@foo.com", new DateTime('1989-03-12'));
```


5 Properties of OOP

- Abstraction
 - › What attributes of a “user” should we model?
 - › What attributes of a “user” are irrelevant?
 - › What behaviors of a “user” should we model?
 - › What behaviors of a “user” are irrelevant?
 - › How do we model them?
 - › Nouns become classes/objects
 - › Verbs become methods

5 Properties of OOP

- Encapsulation (information hiding):
 - public: the item (member variable, method, etc.) is accessible from outside the class and also from inside the class
 - private: the item is not accessible from outside the class, but is accessible from inside the class.
 - protected: the item is not accessible from outside the class, but is accessible from inside the class and from within *descendant* classes.

5 Properties of OOP

- Inheritance
 - › Determined by the “is-a” (or “is a kind of”) relationship.
 - › Examples:
 - › All dogs are animals.
 - › All administrators are users.
 - › All users are “models.”
 - › When administrator inherits from user, administrator gets all the properties and methods of user for “free.”

5 Properties of OOP

- Inheritance

```
class Animal {  
    public function eat() {  
        print "Animal is eating."  
    }  
}  
  
class Dog extends Animal {  
    public function speak() {  
        print "Woof!";  
    }  
}  
  
$dog = new Dog();  
$dog -> eat();  
$dog -> speak();
```

Outputs:
Animal is eating.
Woof!

5 Properties of OOP

- Inheritance

```
class Animal {  
    public function eat() {  
        print "Animal is eating."  
    }  
}  
  
class Dog extends Animal {  
    public function speak() {  
        print "Woof!"  
    }  
}  
  
$dog = new Dog();  
$dog -> eat();  
$dog -> speak();
```

Animal is the base (or parent) class and Dog is the derived (or child) class.

5 Properties of OOP

- Polymorphism
 - › Often the base class provides almost, but not quite, the right behavior (method).
 - › If you need to change the behavior, you *override* the method in the derived class by defining a new method with the same name.

5 Properties of OOP

- Polymorphism

```
class Animal {  
    public function eat() {  
        print "Animal is eating."  
    }  
}
```

```
class Dog extends Animal {  
    public function eat() {  
        print "Chomp!"  
    }  
}
```

```
$dog = new Dog();  
$dog -> eat();
```

5 Properties of OOP

- Polymorphism
 - › If there is no reasonable implementation of the parent class function, make both the function and the class *abstract*.
 - › This forces the derived classes to write their own implementation, but the base class can still write methods that use it.
 - › To prevent overriding in a derived class, use the key word *final*.

5 Properties of OOP

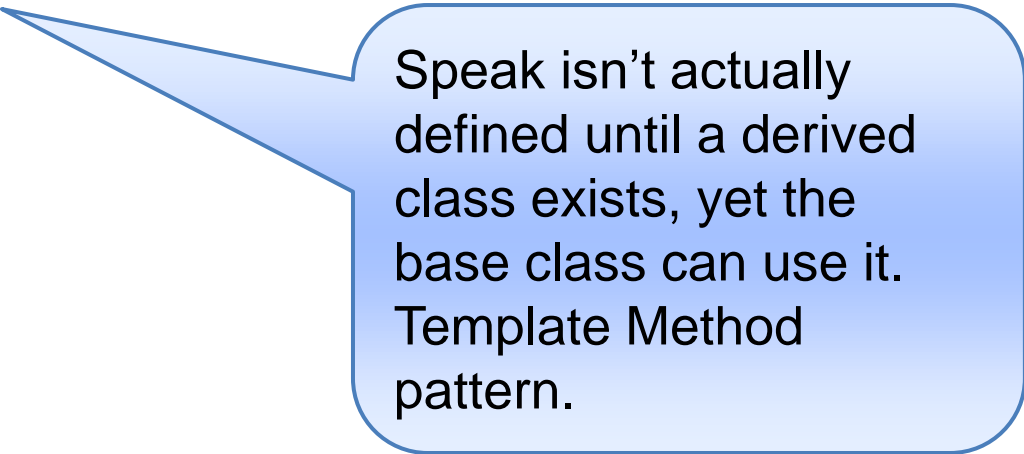
- Polymorphism

```
abstract class Animal {  
    public abstract function speak();  
    public final function greet() {  
        $this -> speak();  
        $this -> speak();  
        $this -> speak();  
    }  
}  
  
class Dog extends Animal {  
    public function speak() {  
        print "Woof!";  
    }  
}  
  
$dog = new Dog();  
$dog -> greet();
```

5 Properties of OOP

- Polymorphism

```
abstract class Animal {  
    public abstract function speak();  
    public final function greet() {  
        $this -> speak();  
        $this -> speak();  
        $this -> speak();  
    }  
}  
  
class Dog extends Animal {  
    public function speak() {  
        print "Woof!";  
    }  
}  
  
$dog = new Dog();  
$dog -> greet();
```



Speak isn't actually defined until a derived class exists, yet the base class can use it. Template Method pattern.

5 Properties of OOP

- Polymorphism
 - A class that is 100% abstract is called an *interface*.
 - It only determines what behaviors should be provided.
 - Class names often end in “-able”
 - Use the key word *implements* to create a concrete class that corresponds to the interface.

5 Properties of OOP

- Polymorphism

```
interface Saveable {  
    public function save();  
}  
abstract class Model implements Saveable {  
    private $id;  
    public function save() {  
        // save the id  
    }  
}  
class User extends Model {  
    private $email;  
    public function save() {  
        parent::save();  
        // save the email  
    }  
}
```

OOP Miscellany

- Sometimes an attribute or method belongs to the entire class rather than to each object.
 - Examples
 - Constants that are used to affect the behaviors of methods (i.e. FETCH_ASSOC)
 - Variables that all instances share (i.e. a database object shared among all models)
 - Methods that don't act on an object (i.e. a findById function)
 - Called *static* members of the class.

OOP Miscellany

```
abstract class Model implements Saveable {
    private static $db = null;
    public static function getDb() {
        if (self::$db == null) {
            try {
                self::$db = new PDO('sqlite:scratch.db3');
            } catch (PDOException $e) {
                die("Could not open database.");
            }
        }
        return self::$db;
    }
    // ...
}
```

OOP Miscellany

```
abstract class Model implements Saveable {  
    private static $db = null;  
    public static function getDb() {  
        if (self::$db == null) {  
            try {  
                self::$db = new PDO('sqlite:scratch.db3');  
            } catch (PDOException $e) {  
                die("Could not open database.");  
            }  
        }  
        return self::$db;  
    }  
    // ...  
}
```

```
class User extends Model {  
    private $email;  
    public function save() {  
        $db = parent::getDb();  
        // use the database here  
    }  
}
```

OOP Miscellany

- PHP is *reflective*
 - › You can ask questions like
 - › What's the name of the class for this object?
 - › Does a method named "foo" exist in this object?
 - › Does the current object inherit from the class "bar"?
 - › Very useful for building frameworks for active records (database access abstraction).
 - › List of functions at <http://www.php.net/manual/en/ref.classobj.php>

Example: Better Models

- Base models on convention
 - › Primary key is always a field called 'id'
 - › Get rid of global variable 'db'
 - › Support common operations like
 - › Create, retrieve, update, delete (CRUD)
- Screen share (code available from <http://cs.franklin.edu/~sharkesc/webd236>)

Upcoming Deadlines

- Readings for next week
 - › Chapter 15 in *PHP and MySQL*
- Assignments
 - › Homework 6 due end of week 7
 - › Midterm exam due end of week 8
 - › Lab 3 due end of week 10
- Next week:
 - › Regular expressions

General Q & A

- Questions?
- Comments?
- Concerns?