MODULE-04

Class &Objects in PHP: What is Class & Object, Creating and accessing a Class &Object, Object properties, object methods, Overloading, inheritance, Constructor and Destructor

Form Handling: Creating HTML Form, Handling HTML Form data in PHP

Database Handling Using PHP with MySQL: Introduction to MySQL: Database terms, Data Types.

Class &Objects in PHP: What is Class & Object:

- The class keyword is used in PHP to define a class.
- In PHP, a class is a blueprint or template for creating objects.
- It defines the properties or attributes (variables) and methods (functions) that objects of that class will have.
- An object, on the other hand, is an instance of a class. It's a concrete realization of the blueprint laid out by the class.
- You can create multiple objects based on the same class, and each object will have its own set of properties and methods.

USES OF CLASS AND OBJECTS IN PHP

Classes and objects in PHP are fundamental to object-oriented programming (OOP) and offer various benefits and use cases:

1. **Code Organization**: Classes help organize code into logical units. For example, in a web application, you might have classes representing users, products, orders, etc. This makes your codebase more manageable and easier to maintain.
2. **Encapsulation**: Classes allow you to encapsulate data (properties) and behavior (methods) into objects. This means you can hide the internal state and implementation details of an object, exposing only what's necessary. Encapsulation helps in creating more robust and secure code.
3. **Reusability**: Once you define a class, you can create multiple objects based on that class. This promotes code reuse, as you can use the same class definition to create objects with similar properties and behaviors throughout your application.
4. **Inheritance**: PHP supports inheritance, allowing classes to inherit properties and methods from parent classes. This enables you to create specialized classes (child classes)

that inherit common functionality from a base class (parent class). Inheritance promotes code reuse and facilitates the creation of hierarchical relationships between classes.

5. **Polymorphism**: Polymorphism allows objects of different classes to be treated as objects of a common superclass. This means you can use a single interface to interact with objects of different types. Polymorphism helps in writing more flexible and maintainable code by allowing you to write code that works with objects at a higher level of abstraction.
6. **Modularity**: Classes and objects promote modularity by allowing you to break down complex systems into smaller, more manageable components. Each class can encapsulate a specific set of functionality, making it easier to understand, modify, and extend your codebase.
7. **Extensibility**: OOP principles like inheritance and polymorphism make it easier to extend the functionality of your application. You can create new classes that build upon existing classes, adding new features or modifying existing ones without affecting the original code.

CREATING AND ACCESSING A CLASS &OBJECT, OBJECT PROPERTIES:

CREATING AND ACCESSING A CLASS: Creating and accessing a class in PHP involves defining a blueprint for an object, including its properties (attributes) and methods (functions), and then creating instances of that class to work with. Let's break down the process:

1. **Defining a Class**: To define a class in PHP, you use the class keyword followed by the class name. Inside the class, you define properties and methods.
2. **Instantiating Objects**: Once you have defined a class, you can create instances of that class, also known as objects. To create an object in PHP, you use the `new` keyword followed by the class name and parentheses (if there are arguments to the constructor).
3. **Accessing Properties and Methods**: Once you have an object, you can access its properties and methods using the arrow operator `->`.
4. **Visibility Modifiers**: In PHP, you can specify the visibility of properties and methods using public, protected, or private keywords. public means the property or method can be accessed from outside the class, protected means they can only be accessed within the class itself and its subclasses, and private means they can only be accessed within the class itself.

Syntax:

```
<?php
class ClassName {
    // Properties (variables)
    public $property1;
    private $property2;
    protected $property3;

    // Constructor (optional)
    public function __construct($param1, $param2) {
        // Initialize properties or perform other setup tasks
        $this->property1 = $param1;
        $this->property2 = $param2;
    }

    // Methods (functions)
    public function method1() {
        // Method body
    }

    private function method2() {
        // Method body
    }

    protected function method3() {
        // Method body
    }
}
?>
```

S.No	Parameters	description
1	Class	To declare a class, we have to use the keyword class and then the name of the class that we want to declare.
2	Variable declaration	To declare a variable, we have to declare keyword var followed by $ convention and then the name of the declared variable. These are also known as member variables. These are also called properties
3	Function declaration	To declare a function, we have to declare the keyword function following the name of the declared function. These are also known as member functions, but these functions are only accessible to class only. These are also called methods
4	Braces	We have to enclose our class with curly braces { }

Example:

```
<?php
// Define the class
class Car {
  // Properties
  public $brand;
  public $model;

  // Constructor
  public function __construct($brand, $model) {
    $this->brand = $brand;
    $this->model = $model;
  }

  // Method to display car information
  public function displayInfo() {
    echo "This is a {$this->brand} {$this->model}.";
    }
}

// Create an object of the Car class
$car1 = new Car("Toyota", "Corolla");

// Accessing properties
echo "Brand: " . $car1->brand . "<br>";
echo "Model: " . $car1->model . "<br>";

// Call a method
$car1->displayInfo(); // Output: This is a Toyota Corolla.
?>
// Output: This is a Toyota Corolla.
```

Objects in PHP: In PHP, an object is an instance of a class.

- It's a concrete realization of the blueprint laid out by the class.
- Objects encapsulate data (properties) and behavior (methods) defined within the class.
- To create an object in PHP, you use the `new` keyword followed by the name of the class, optionally followed by parentheses containing any required constructor arguments.

Syntax:

$object_name = new ClassName();

- $object_name: This is the name you give to the object. You can choose any valid variable name.
- new: This keyword is used to create a new instance of the class.
- ClassName: This is the name of the class from which you want to create an object. Make sure it's spelled correctly and matches the class definition.
- (): These parentheses are used to call the constructor of the class. If the class has a constructor and requires arguments, you pass them inside these parentheses.

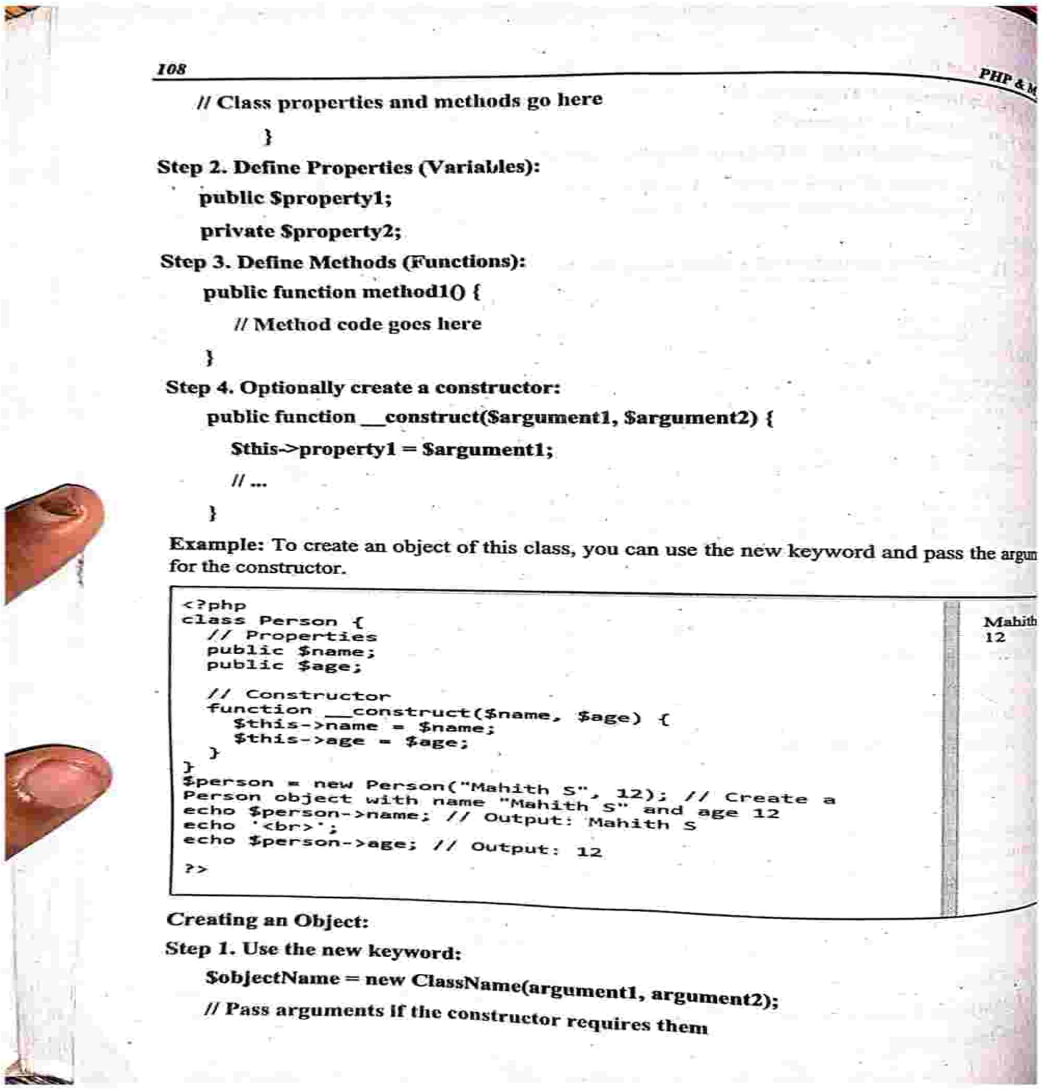
Example:

```
<?php
// Define a class
```

```
class Car {
    public $brand;
    public $model;

    public function __construct($brand, $model) {
        $this->brand = $brand;
        $this->model = $model;
    }

    public function displayInfo() {
        echo "This is a {$this->brand} {$this->model}.";
    }
}

// Create an object of the Car class
$car1 = new Car("Toyota", "Corolla");
?>
```

OUTPUT: **This is a Toyota Corolla.**

In this example, $car1 is an object of the Car class, created using the new keyword followed by the class name. We're also passing "Toyota" and "Corolla" as arguments to the constructor because the Car class has a constructor that expects two parameters.



108 PHP & M

```
    // Class properties and methods go here
        }
```

Step 2. Define Properties (Variables):

```
    public $property1;
    private $property2;
```

Step 3. Define Methods (Functions):

```
    public function method1() {
        // Method code goes here
    }
```

Step 4. Optionally create a constructor:

```
    public function __construct($argument1, $argument2) {
      $this->property1 = $argument1;
      // ...
    }
```

Example: To create an object of this class, you can use the new keyword and pass the argum for the constructor.

```
<?php
class Person {
  // Properties
  public $name;
  public $age;

  // Constructor
  function __construct($name, $age) {
    $this->name = $name;
    $this->age = $age;
  }
}
$person = new Person("Mahith S", 12); // Create a
Person object with name "Mahith S" and age 12
echo $person->name; // Output: Mahith S
echo '<br>';
echo $person->age; // Output: 12
?>
```

```
Mahith
12
```

Creating an Object:

Step 1. Use the new keyword:

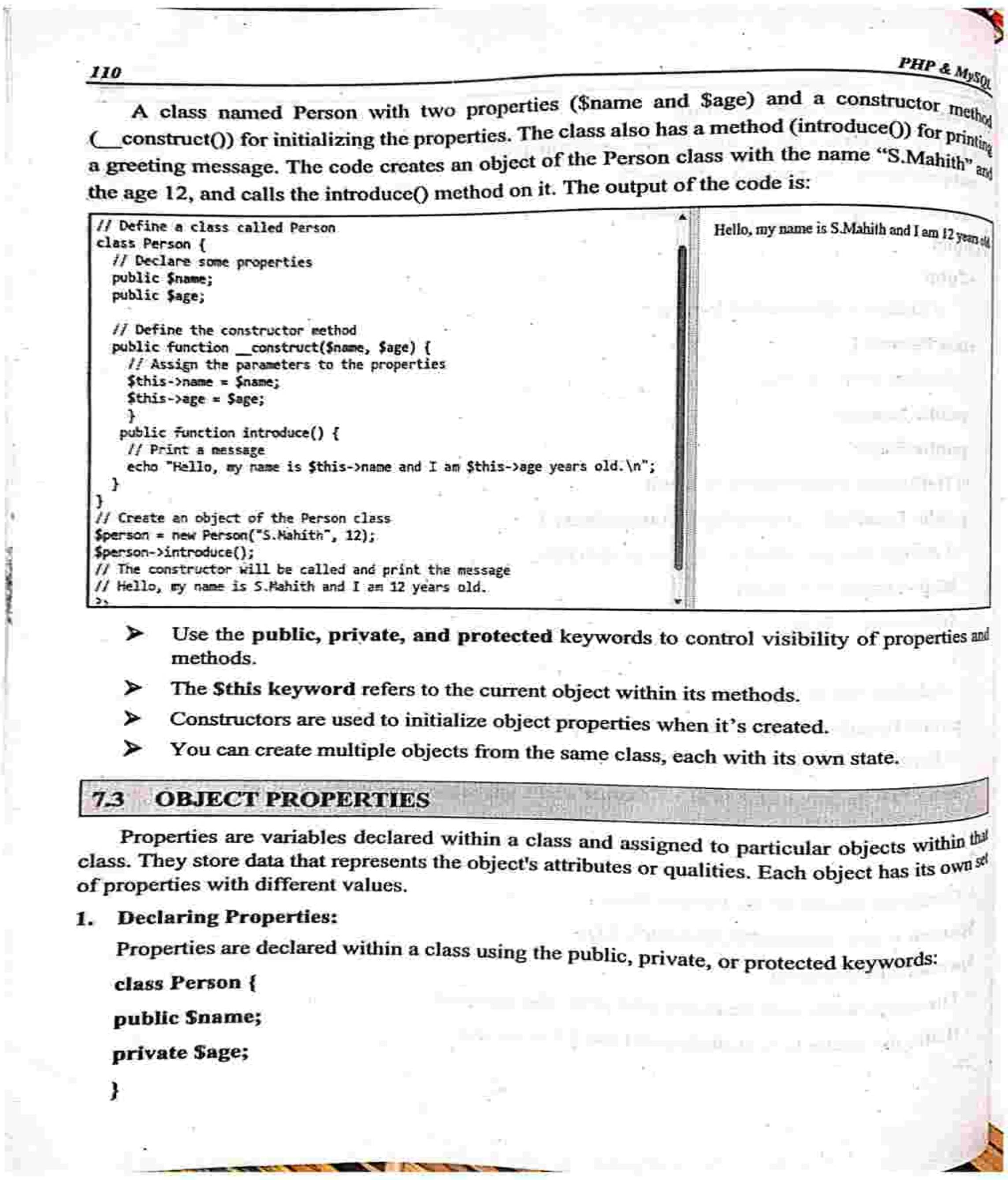
```
    $objectName = new ClassName(argument1, argument2);
    // Pass arguments if the constructor requires them
```

NOTE:

- This keyword: Used within methods to refer to the current object.Accessing properties and methods: Use the object's name followed by the property or method name and parentheses for methods (e.g., $object->property, $object->method()).
- The $this keyword refers to the current object, and is only available insidemethods.

OBJECT PROPERTIES:

- In PHP, object properties are variables that are associated with a particular instance of a class. They represent the state or characteristics of an object.
- Object properties define the data that an object holds. These properties can be accessed and modified within the class methods or from outside the class, depending on their visibility.



A class named Person with two properties ($name and $age) and a constructor method (__construct()) for initializing the properties. The class also has a method (introduce()) for printing a greeting message. The code creates an object of the Person class with the name "S.Mahith" and the age 12, and calls the introduce() method on it. The output of the code is:

```
// Define a class called Person
class Person {
  // Declare some properties
  public $name;
  public $age;

  // Define the constructor method
  public function __construct($name, $age) {
    // Assign the parameters to the properties
    $this->name = $name;
    $this->age = $age;
    }
   public function introduce() {
    // Print a message
    echo "Hello, my name is $this->name and I am $this->age years old.\n";
  }
}
// Create an object of the Person class
$person = new Person("S.Mahith", 12);
$person->introduce();
// The constructor will be called and print the message
// Hello, my name is S.Mahith and I am 12 years old.
?>
```

```
Hello, my name is S.Mahith and I am 12 years old
```

- Use the **public, private, and protected** keywords to control visibility of properties and methods.
- The **$this keyword** refers to the current object within its methods.
- Constructors are used to initialize object properties when it's created.
- You can create multiple objects from the same class, each with its own state.

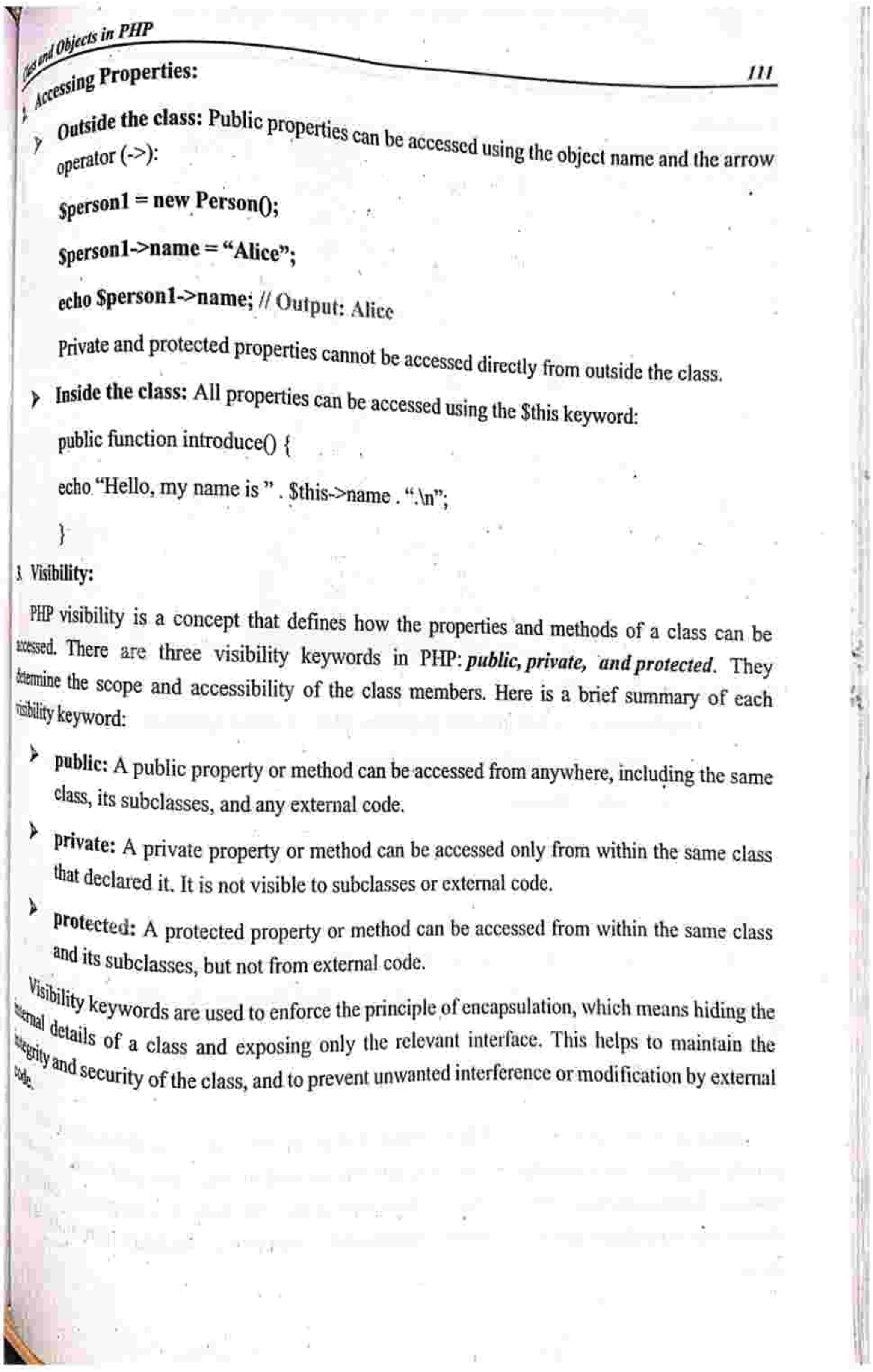
7.3 OBJECT PROPERTIES

Properties are variables declared within a class and assigned to particular objects within that class. They store data that represents the object's attributes or qualities. Each object has its own set of properties with different values.

1. Declaring Properties:

Properties are declared within a class using the public, private, or protected keywords:

```
class Person {
public $name;
private $age;
}
```



2. **Accessing Properties:**

- **Outside the class:** Public properties can be accessed using the object name and the arrow operator (->):

```
$person1 = new Person();
$person1->name = "Alice";
echo $person1->name; // Output: Alice
```

Private and protected properties cannot be accessed directly from outside the class.

- **Inside the class:** All properties can be accessed using the $this keyword:

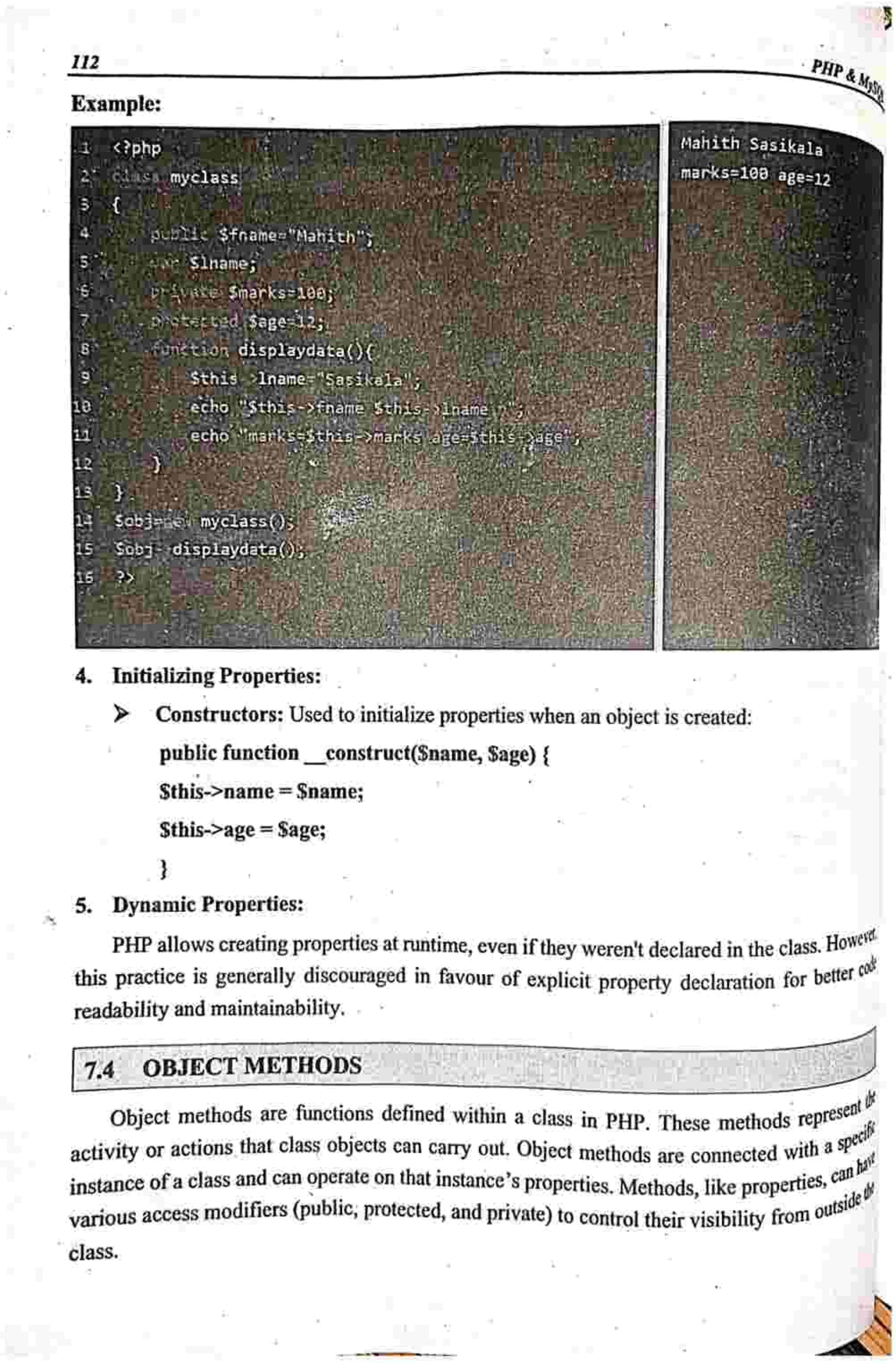
```
public function introduce() {
echo "Hello, my name is " . $this->name . ".\n";
}
```

3. **Visibility:**

PHP visibility is a concept that defines how the properties and methods of a class can be accessed. There are three visibility keywords in PHP: ***public, private, and protected***. They determine the scope and accessibility of the class members. Here is a brief summary of each visibility keyword:

- **public:** A public property or method can be accessed from anywhere, including the same class, its subclasses, and any external code.
- **private:** A private property or method can be accessed only from within the same class that declared it. It is not visible to subclasses or external code.
- **protected:** A protected property or method can be accessed from within the same class and its subclasses, but not from external code.

Visibility keywords are used to enforce the principle of encapsulation, which means hiding the internal details of a class and exposing only the relevant interface. This helps to maintain the integrity and security of the class, and to prevent unwanted interference or modification by external code.



Example:

```
<?php
class myclass
{
    public $fname="Mahith";
    var $lname;
    private $marks=100;
    protected $age=12;
    function displaydata(){
        $this->lname="Sasikala";
        echo "$this->fname $this->lname \n";
        echo "marks=$this->marks age=$this->age";
    }
}
$obj=new myclass();
$obj->displaydata();
?>
```

```
Mahith Sasikala
marks=100 age=12
```

4. **Initializing Properties:**

   ➤ **Constructors:** Used to initialize properties when an object is created:

   public function __construct($name, $age) {

   $this->name = $name;

   $this->age = $age;

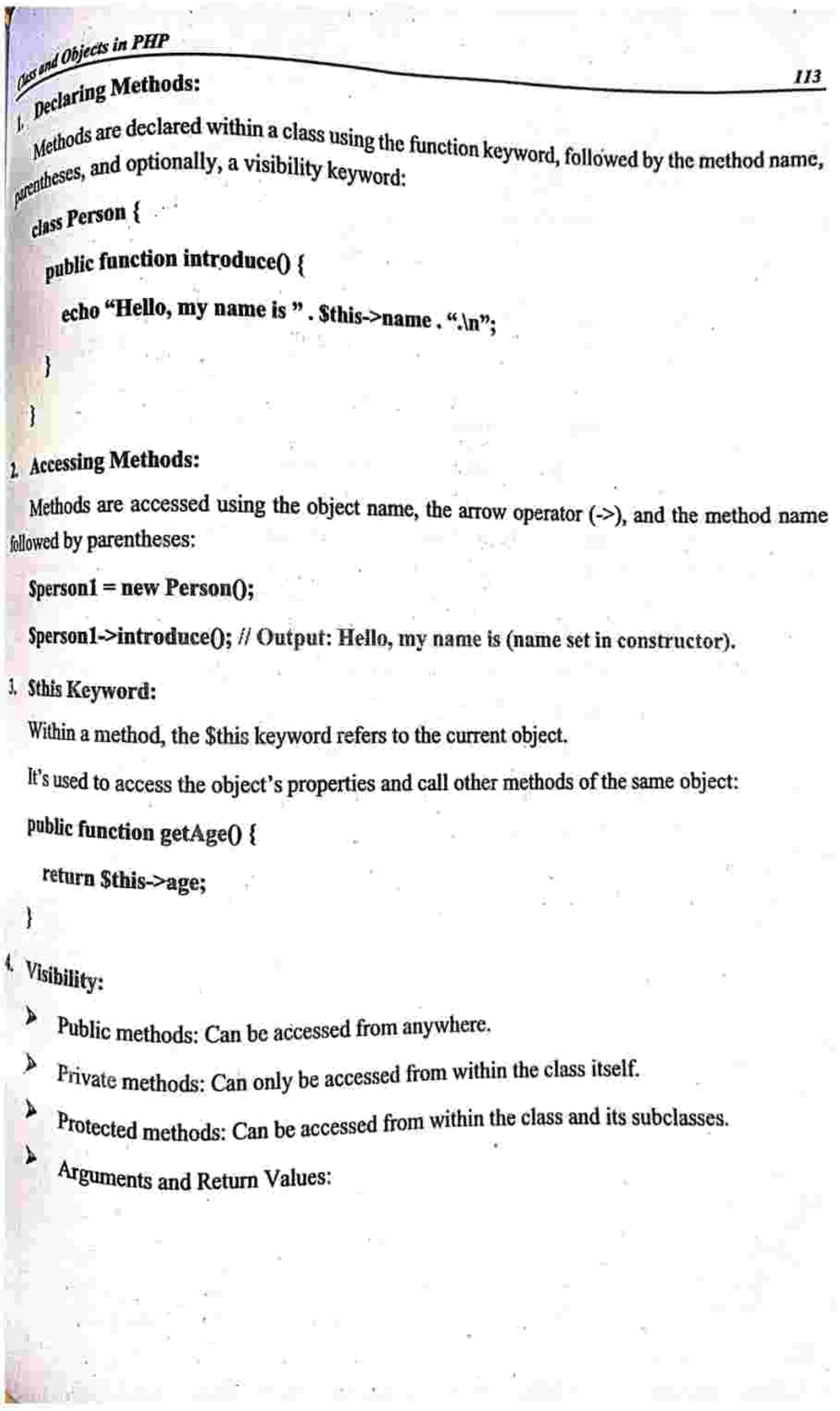
   }

5. **Dynamic Properties:**

   PHP allows creating properties at runtime, even if they weren't declared in the class. However, this practice is generally discouraged in favour of explicit property declaration for better code readability and maintainability.

7.4 OBJECT METHODS

Object methods are functions defined within a class in PHP. These methods represent the activity or actions that class objects can carry out. Object methods are connected with a specific instance of a class and can operate on that instance's properties. Methods, like properties, can have various access modifiers (public, protected, and private) to control their visibility from outside the class.



1. **Declaring Methods:**

Methods are declared within a class using the function keyword, followed by the method name, parentheses, and optionally, a visibility keyword:

```
class Person {
  public function introduce() {
    echo "Hello, my name is " . $this->name . ".\n";
  }
}
```

2. **Accessing Methods:**

Methods are accessed using the object name, the arrow operator (->), and the method name followed by parentheses:

```
$person1 = new Person();
$person1->introduce(); // Output: Hello, my name is (name set in constructor).
```

3. **$this Keyword:**

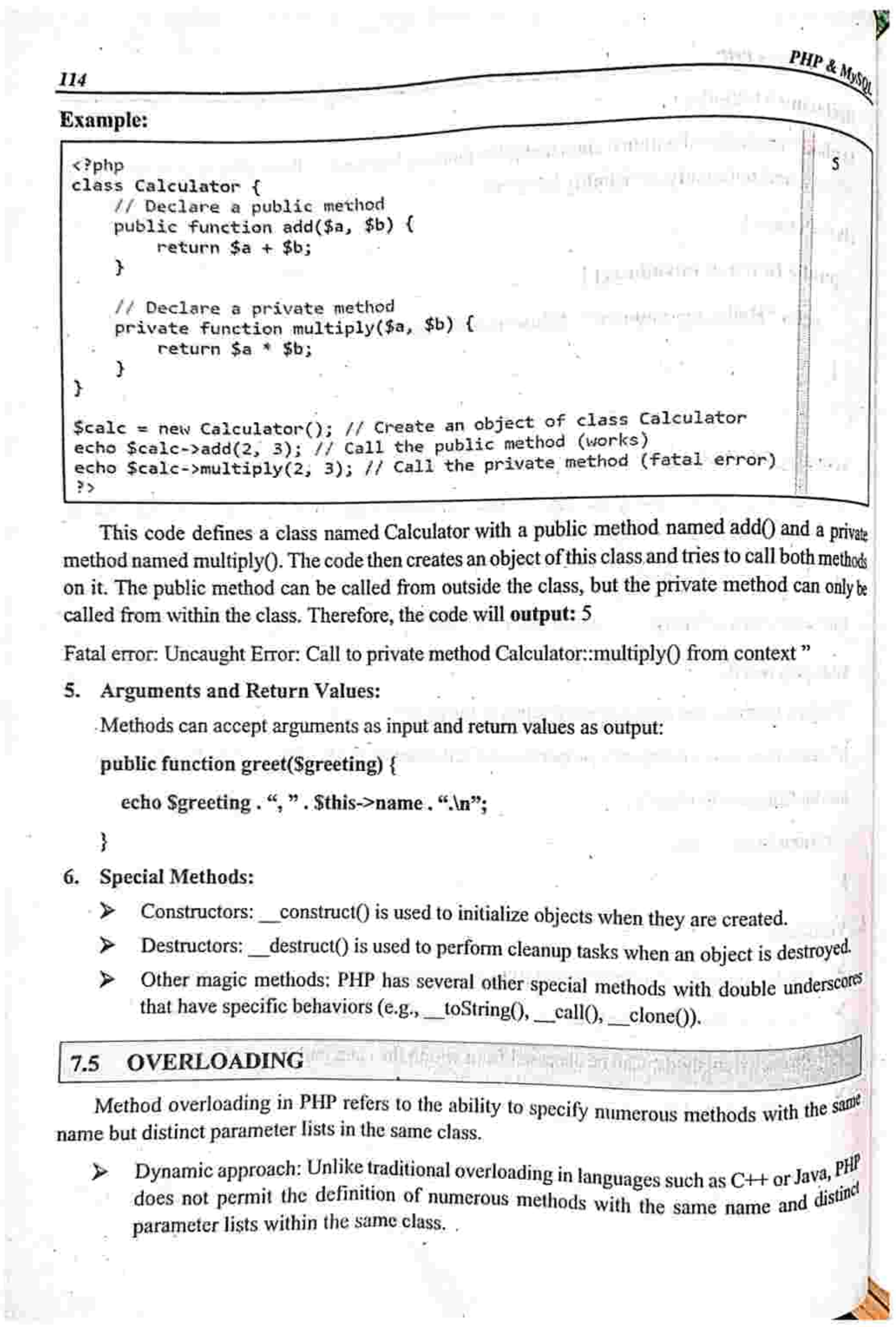
Within a method, the $this keyword refers to the current object.

It's used to access the object's properties and call other methods of the same object:

```
public function getAge() {
  return $this->age;
}
```

4. **Visibility:**

- Public methods: Can be accessed from anywhere.
- Private methods: Can only be accessed from within the class itself.
- Protected methods: Can be accessed from within the class and its subclasses.
- Arguments and Return Values:



Example:

```
<?php
class Calculator {
    // Declare a public method
    public function add($a, $b) {
        return $a + $b;
    }

    // Declare a private method
    private function multiply($a, $b) {
        return $a * $b;
    }
}

$calc = new Calculator(); // Create an object of class Calculator
echo $calc->add(2, 3); // Call the public method (works)
echo $calc->multiply(2, 3); // Call the private method (fatal error)
?>
```

This code defines a class named Calculator with a public method named add() and a private method named multiply(). The code then creates an object of this class and tries to call both methods on it. The public method can be called from outside the class, but the private method can only be called from within the class. Therefore, the code will **output:** 5

Fatal error: Uncaught Error: Call to private method Calculator::multiply() from context "

5. **Arguments and Return Values:**

   Methods can accept arguments as input and return values as output:

   public function greet($greeting) {

   echo $greeting . ", " . $this->name . ".\n";

   }

6. **Special Methods:**

   ➢ Constructors: __construct() is used to initialize objects when they are created.

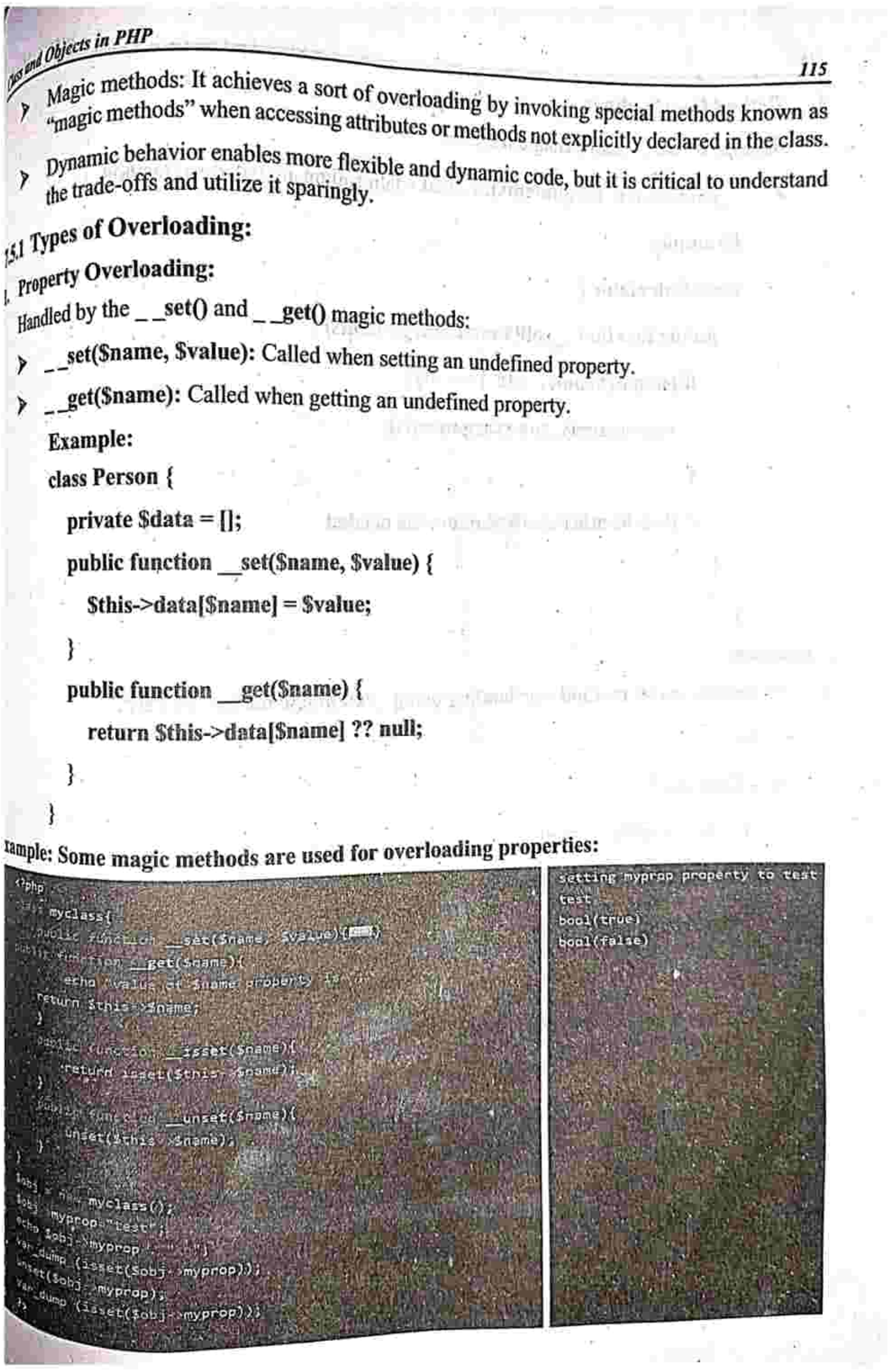
   ➢ Destructors: __destruct() is used to perform cleanup tasks when an object is destroyed.

   ➢ Other magic methods: PHP has several other special methods with double underscores that have specific behaviors (e.g., __toString(), __call(), __clone()).

7.5 OVERLOADING

Method overloading in PHP refers to the ability to specify numerous methods with the same name but distinct parameter lists in the same class.

➢ Dynamic approach: Unlike traditional overloading in languages such as C++ or Java, PHP does not permit the definition of numerous methods with the same name and distinct parameter lists within the same class.



- Magic methods: It achieves a sort of overloading by invoking special methods known as "magic methods" when accessing attributes or methods not explicitly declared in the class.
- Dynamic behavior enables more flexible and dynamic code, but it is critical to understand the trade-offs and utilize it sparingly.

5.1 Types of Overloading:

1. Property Overloading:

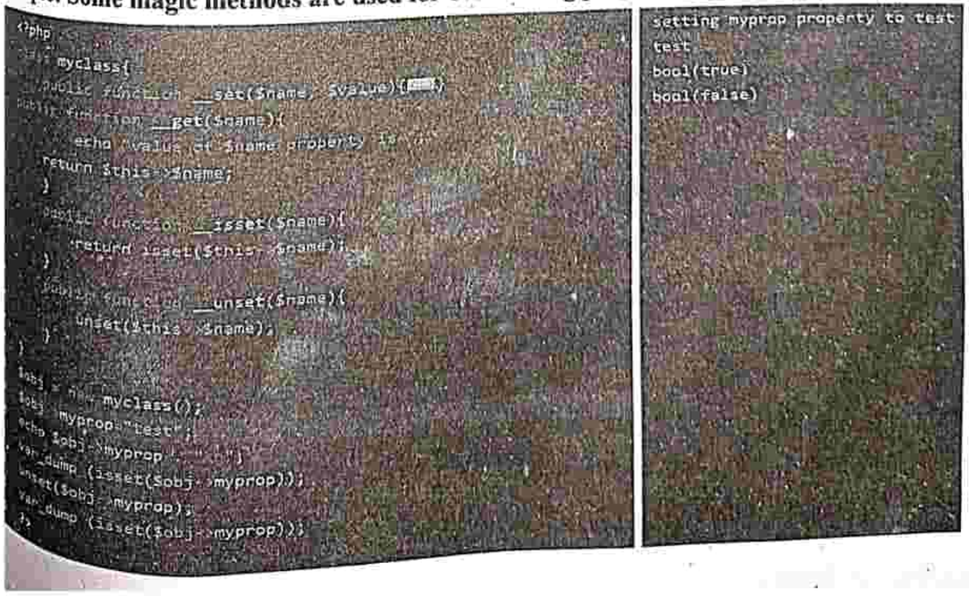
Handled by the __set() and __get() magic methods:

- **__set($name, $value):** Called when setting an undefined property.
- **__get($name):** Called when getting an undefined property.

Example:

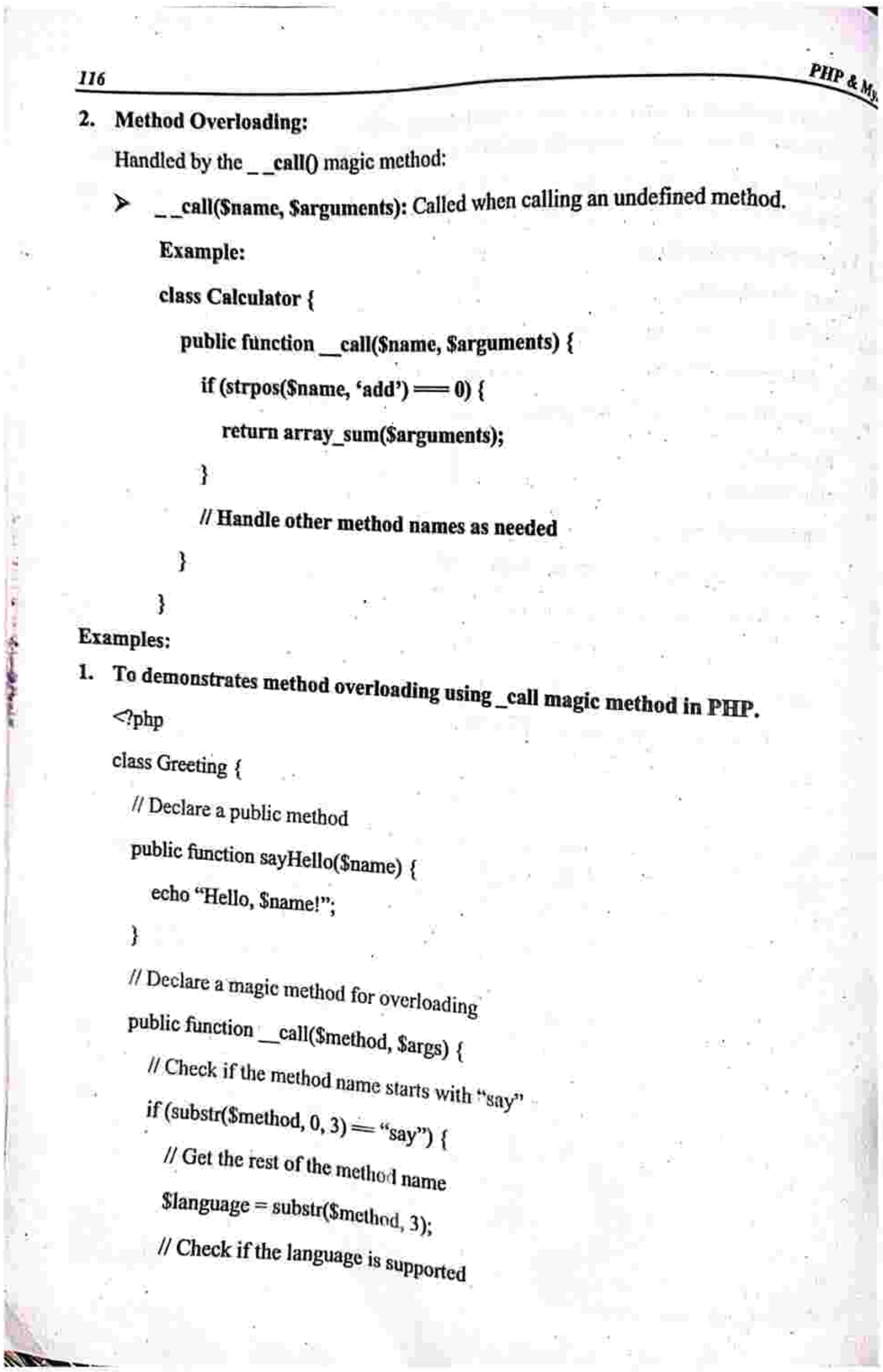
```
class Person {
  private $data = [];
  public function __set($name, $value) {
    $this->data[$name] = $value;
  }
  public function __get($name) {
    return $this->data[$name] ?? null;
  }
}
```

xample: Some magic methods are used for overloading properties:



```
<?php
class myclass{
  public function __set($name, $value){[illegible]}
public function __get($name){
    echo "value of $name property is ";
return $this->$name;
}
public function __isset($name){
   return isset($this->$name);
}
public function __unset($name){
    unset($this->$name);
}
}
$obj = new myclass();
$obj->myprop="test";
echo $obj->myprop . "\n";
var_dump (isset($obj->myprop));
unset($obj->myprop);
var_dump (isset($obj->myprop));
?>
```

```
setting myprop property to test
test
bool(true)
bool(false)
```



2. **Method Overloading:**

Handled by the _ _call() magic method:

➤ **_ _call($name, $arguments):** Called when calling an undefined method.

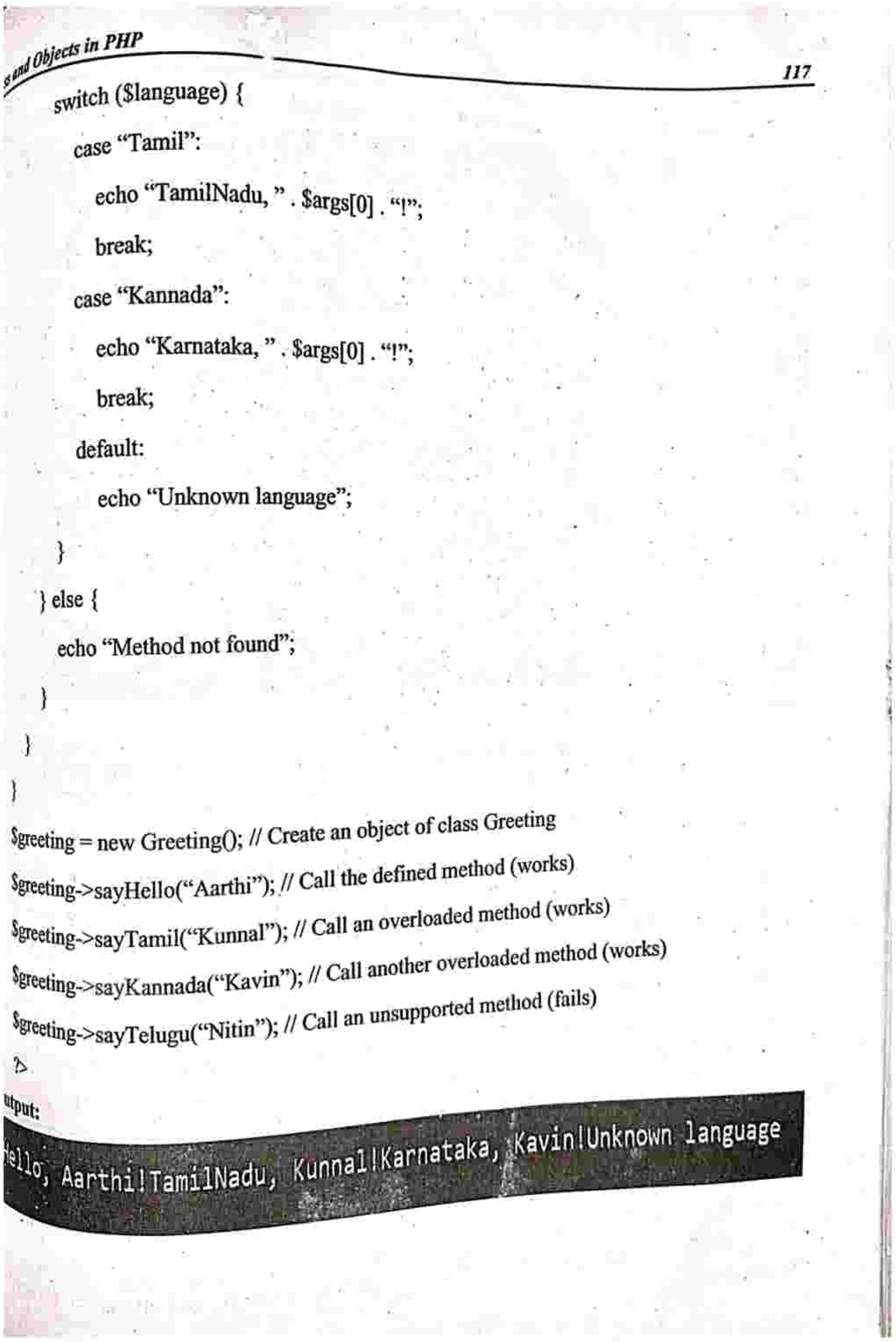
Example:

```
class Calculator {
  public function __call($name, $arguments) {
    if (strpos($name, 'add') === 0) {
      return array_sum($arguments);
    }
    // Handle other method names as needed
  }
}
```

Examples:

1. **To demonstrates method overloading using _call magic method in PHP.**

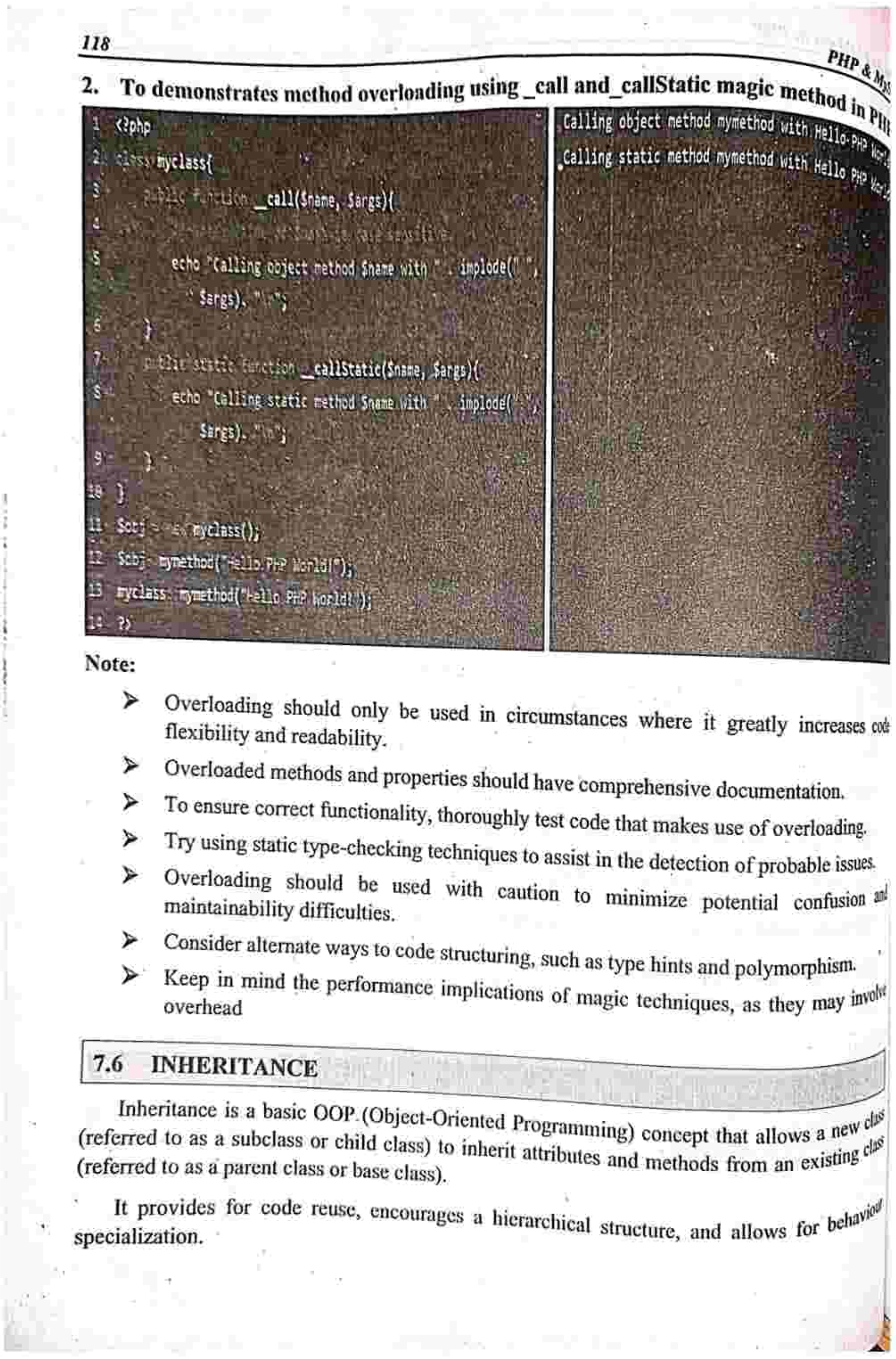
```
<?php
class Greeting {
  // Declare a public method
  public function sayHello($name) {
    echo "Hello, $name!";
  }
  // Declare a magic method for overloading
  public function __call($method, $args) {
    // Check if the method name starts with "say"
    if (substr($method, 0, 3) == "say") {
      // Get the rest of the method name
      $language = substr($method, 3);
      // Check if the language is supported
```



```
    switch ($language) {
      case “Tamil”:
        echo “TamilNadu, ” . $args[0] . “!”;
        break;
      case “Kannada”:
        echo “Karnataka, ” . $args[0] . “!”;
        break;
      default:
        echo “Unknown language”;
    }
  } else {
    echo “Method not found”;
  }
 }
}
$greeting = new Greeting(); // Create an object of class Greeting
$greeting->sayHello(“Aarthi”); // Call the defined method (works)
$greeting->sayTamil(“Kunnal”); // Call an overloaded method (works)
$greeting->sayKannada(“Kavin”); // Call another overloaded method (works)
$greeting->sayTelugu(“Nitin”); // Call an unsupported method (fails)
?>
```

utput:

```
ello, Aarthi!TamilNadu, Kunnal!Karnataka, Kavin!Unknown language
```



2. **To demonstrates method overloading using _call and_callStatic magic method in PH**

```php
<?php
class myclass{
    public function __call($name, $args){
        // [illegible] is case sensitive
        echo "Calling object method $name with " . implode(" ",
          $args). "\n";
    }
    public static function __callStatic($name, $args){
        echo "Calling static method $name with " . implode(" ",
          $args). "\n";
    }
}
$obj = new myclass();
$obj->mymethod("Hello PHP World!");
myclass::mymethod("Hello PHP World!");
?>
```

```
Calling object method mymethod with Hello PHP Worl
Calling static method mymethod with Hello PHP Worl
```

Note:

- ➢ Overloading should only be used in circumstances where it greatly increases code flexibility and readability.
- ➢ Overloaded methods and properties should have comprehensive documentation.
- ➢ To ensure correct functionality, thoroughly test code that makes use of overloading.
- ➢ Try using static type-checking techniques to assist in the detection of probable issues.
- ➢ Overloading should be used with caution to minimize potential confusion and maintainability difficulties.
- ➢ Consider alternate ways to code structuring, such as type hints and polymorphism.
- ➢ Keep in mind the performance implications of magic techniques, as they may involve overhead

7.6 INHERITANCE

Inheritance is a basic OOP (Object-Oriented Programming) concept that allows a new class (referred to as a subclass or child class) to inherit attributes and methods from an existing class (referred to as a parent class or base class).

It provides for code reuse, encourages a hierarchical structure, and allows for behaviour specialization.

INHERITANCE: Inheritance in PHP is a fundamental object-oriented programming concept that allows a class to inherit properties and methods from another class, referred to as the parent or base class.

- The class that inherits properties and methods is called the child or derived class.

Inheritance is one of the four pillars of Object-Oriented Programming (OOPs).

- Inheritance is the phenomenon by which a child class can inherit all the properties and characteristics of the parent class.

Uses of iheritance:

- **Code Reusability**: One of the primary advantages of inheritance is code reuse. By defining common properties and methods in a parent class, subclasses can inherit and reuse this code, reducing redundancy and promoting modular design.
- **Abstraction and Encapsulation**: Inheritance allows for abstraction by defining a generalized parent class that provides a common interface for its subclasses. This abstraction hides the implementation details of the parent class and allows subclasses to focus on specific behaviors or attributes. Encapsulation is also promoted as related properties and methods are grouped together in a single class hierarchy.
- **Promoting Polymorphism**: Inheritance facilitates polymorphism, which allows objects of different classes to be treated uniformly through a common interface. This enables flexibility in designing systems where different objects can respond to the same message or method call in different ways.
- **Enhancing Maintainability and Extensibility**: Inheritance promotes maintainability by centralizing common code in a parent class, making it easier to modify or update functionality without affecting multiple subclasses. It also enhances extensibility by allowing new subclasses to be created without modifying existing code, thus minimizing the risk of introducing bugs.
- **Simplifying Class Hierarchies**: Inheritance helps in organizing class hierarchies by creating a logical relationship between classes. This hierarchical structure makes it easier to understand and manage complex systems by grouping related classes together and defining their relationships.

- **Specialization and Generalization**: Inheritance allows for specialization, where subclasses can extend or override behavior inherited from a parent class to meet specific requirements.

Advantages and Disadvanages of inheritance in php

Advantages:

1. **Code Reusability**: Inheritance allows you to reuse code from existing classes in new classes, reducing redundancy and promoting a more modular design.
2. **Promotes Polymorphism**: Inheritance enables polymorphism, where objects of different classes can be treated interchangeably if they share a common parent class. This enhances flexibility and promotes cleaner, more flexible code.
3. **Encourages Abstraction**: Inheritance allows you to define a common interface or behavior in a parent class, which can be inherited by subclasses. This abstraction hides implementation details and promotes a clearer separation of concerns.
4. **Simplifies Maintenance**: By centralizing common functionality in a parent class, inheritance can make code maintenance easier. Changes made to the parent class are automatically propagated to its subclasses, reducing the risk of introducing errors.
5. **Enhances Extensibility**: Inheritance facilitates the creation of new subclasses without modifying existing code. This makes it easier to extend the functionality of a system by adding new features or behaviors.

Disadvantages:

1. **Tight Coupling**: Inheritance can lead to tight coupling between classes, where changes in the parent class can have unintended effects on subclasses. This can make the codebase more fragile and difficult to maintain.
2. **Inheritance Hierarchy Complexity**: As the inheritance hierarchy grows, it can become more complex and difficult to understand. This complexity can make it harder to reason about the behavior of subclasses and their relationships.
3. **Overuse**: Overuse of inheritance can lead to an overly complex class hierarchy, known as the "diamond problem" or "inheritance hell." This can make the codebase harder to maintain and extend.
4. **Inflexibility**: Inheritance ties subclasses to the implementation details of the parent class, making it harder to change or refactor the inheritance hierarchy without affecting existing code.

Type of inheritance in php:

1) Single Inheritance

2) Multilevel Inheritance.

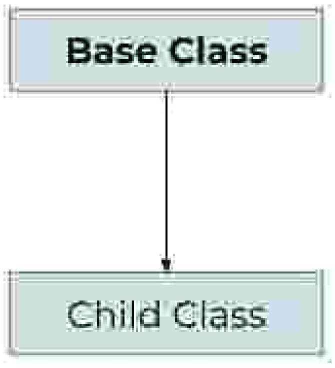
3)Multiple iheritance

4) Hierarchical Inheritance.

5)Hybrid inheritance

1) Single Inheritance

Single inheritance is the most basic and simple inheritance type. As the name suggests, in a single inheritance, there is only one base class and one sub or derived class. It directly inherits the subclass from the base class.

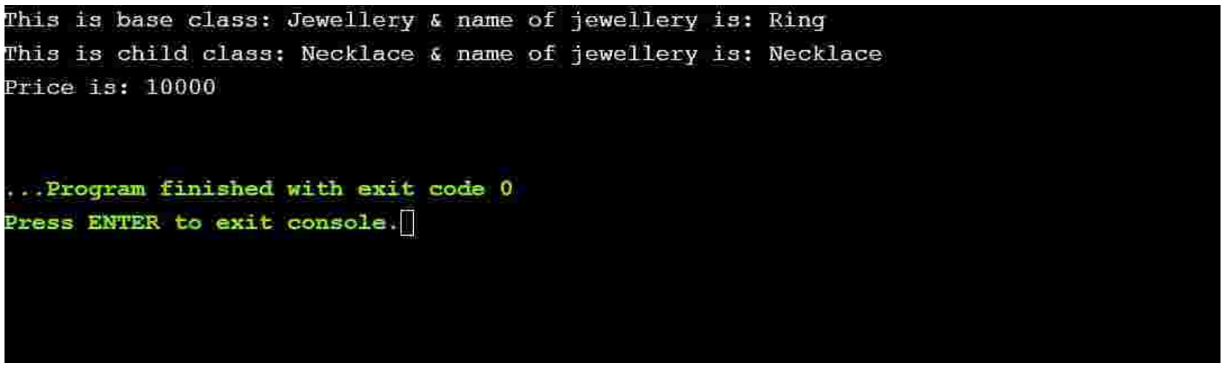


- In single inheritance, a class can inherit properties and methods from only one parent class.
- PHP supports single inheritance directly. A class can extend only one parent class at a time.
- This is the most common type of inheritance in PHP.

Syntax:
```
class ParentClass {
   // Properties and methods
}
class ChildClass extends ParentClass {
   // Inherits properties and methods from ParentClass
}
```

Example:
```
<?php
// base class named "Jewellery"
class Jewellery {
  var $cost = 10000;
```

```
    public function printName($name) {
        echo 'This is base class: Jewellery & name of jewellery is: ' . $name . PHP_EOL;
    }
}
// derived class named "Necklace"
class Necklace extends Jewellery {
    public function print Name($name) {
        echo 'This is child class: Necklace & name of jewellery is: ' . $name . PHP_EOL;
       // this class can access
        // data member of its parent class.
        echo 'Price is: ' . $this->cost . PHP_EOL;
    }
}
$f = new Jewellery();
$s = new Necklace();
 $f->printName('Ring');
$s->printName('Necklace');
 ?>
```



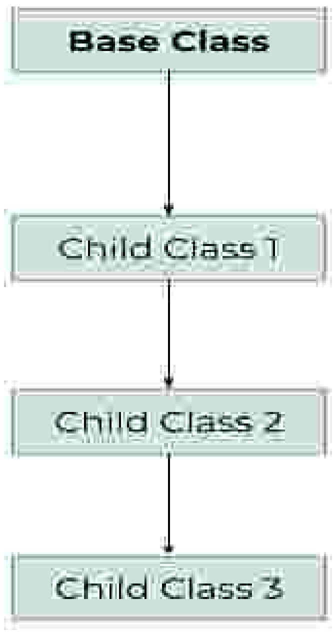
```
This is base class: Jewellery & name of jewellery is: Ring
This is child class: Necklace & name of jewellery is: Necklace
Price is: 10000


...Program finished with exit code 0
Press ENTER to exit console.
```

2) Multilevel Inheritance. Multilevel inheritance is a concept in object-oriented programming where a class extends another class, which in turn extends yet another class, forming a chain or hierarchy of inheritance. This creates a parent-child relationship between the classes involved, with each child class inheriting properties and methods from its parent class, as well as from all ancestors up the inheritance chain.

Here's a step-by-step explanation of multilevel inheritance:

1. **Parent Class**: At the top of the hierarchy is the parent class. This class typically contains common properties and methods that are shared among multiple child classes.

2. **First-level Child Class**: This class extends the parent class, inheriting all of its properties and methods. The first-level child class can also add its own unique properties and methods.
3. **Second-level Child Class (Optional)**: If desired, another class can extend the first-level child class, creating a second level of inheritance. This class inherits properties and methods from both the parent class and the first-level child class.
4. **Subsequent Levels (Optional)**: The process can continue with additional levels of inheritance, with each subsequent child class inheriting properties and methods from its immediate parent class and all ancestors up the inheritance chain.



Syntax: **class ParentClass {**

```
class ParentClass {
   // Parent class properties and methods
}

class ChildClass extends ParentClass {
   // Child class properties and methods
}

class GrandChildClass extends ChildClass {
        // Grandchild class properties and methods

    }
```

Example:

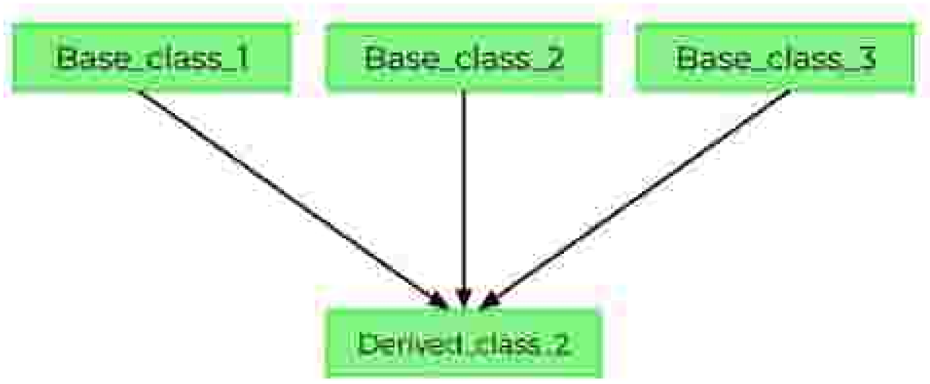
```
<?php
class grandparent
{
        function dis1()
        {
                echo "Grand-Parent";
        }
}
class parents extends grandparent
{
        function dis2()
        {
                echo "Parents";
        }
}
class child extends parents
{
        function dis3()
        {
                echo "Child";
        }
}
$obj=new child();
$obj->dis1();
$obj->dis2();
$obj->dis3();
?>
OUTPUT:Grand-ParentParentsChild
```

```
 class Animal {

   public function eat() {

       echo "Animal is eating.";

   }

}

class Mammal extends Animal {

   public function breathe() {

       echo "Mammal is breathing.";

   }
}

class Dog extends Mammal {
    public function bark() {
        echo "Dog is barking.";
    }
}
// Creating an instance of the Dog class
$dog = new Dog();
// Accessing methods from all levels of inheritance
$dog->eat();    // Output: Animal is eating.
$dog->breathe(); // Output: Mammal is breathing.
$dog->bark();   // Output: Dog is barking.
```

- Animal is the parent class.
- Mammal extends Animal, forming the first level of inheritance.
- Dog extends Mammal, forming the second level of inheritance.
- The Dog class inherits properties and methods from both Animal and Mammal.

3)Multiple iheritance:Multiple Inheritance (Emulated using Interfaces and Traits)

- Multiple inheritance refers to a class inheriting properties and methods from more than one parent class.
- PHP does not support multiple inheritance directly to avoid the diamond problem and ambiguity.
- Instead, multiple inheritance can be emulated using interfaces and traits.
- PHP doesn't support multiple inheritance but by using Interfaces in PHP or using Traits in PHP instead of classes, we can implement it. Traits (Using Class along with Traits): The trait is a type of class which enables multiple inheritance.



In PHP, while direct multiple inheritances is not supported; you can achieve similar behavior using interfaces and traits. Let me explain both approaches with examples:

Multiple Inheritance Using Interfaces:

Interfaces define a contract that classes must adhere to. By implementing multiple interfaces, a class can inherit behavior from multiple sources.

Using multiple inhritance

1.Interface: An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.

Syntax:

```
interface Interface1 {
   // Method declarations
}
interface Interface2 {
   // Method declarations
}
class ChildClass implements Interface1, Interface2 {
   // Implement methods from Interface1 and Interface2
}
// Define multiple interfaces
interface Interface1 {
   public function method1();
}
```

```
interface Interface2 {
    public function method2();
}
```

Example:

```
<?php
// Implement the interfaces
class MyClass implements Interface1, Interface2 {
    public function method1() {
        echo "Method 1 implementation.\n";
    }

    public function method2() {
        echo "Method 2 implementation.\n";
    }
}

// Usage
$obj = new MyClass();
$obj->method1(); // Output: Method 1 implementation.
$obj->method2(); // Output: Method 2 implementation.
?>
```

Example2:

```
<?php
// Define Interface 1
interface Animal {
    public function makeSound();
}
```

```
// Define Interface 2
interface Vehicle {
    public function drive();
}
// Define Trait 1
trait Dog {
    public function makeSound() {
        echo "Woof! ";
    }
}
// Define Trait 2
trait Car {
    public function drive() {
        echo "Vroom! ";
    }
}

// Implementing multiple inheritance using a class
class DogCar implements Animal, Vehicle {
    use Dog, Car;
}
// Create an object and call methods
$obj = new DogCar();
```

```
$obj->makeSound(); // Outputs: Woof!

$obj->drive();     // Outputs: Vroom!
```

2. Traits: In PHP, multiple inheritance can be emulated using traits. Traits allow you to reuse methods within multiple classes without the need for a class hierarchy. Here's an example demonstrating how to achieve multiple inheritance using traits.

Syntax:

```
traitTrait1 {
   // Methods
}
trait Trait2 {
   // Methods
}
class ChildClass {
   use Trait1, Trait2;
}
```

Example:

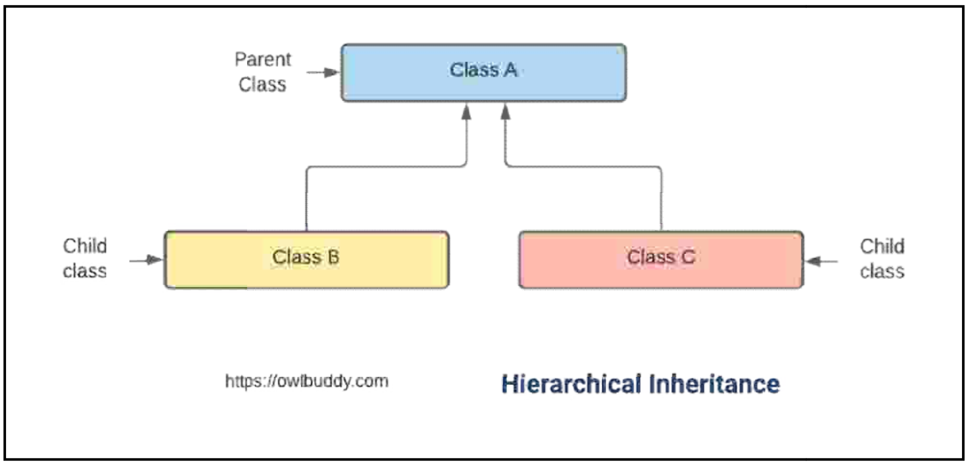
```
<?php
// Define a trait for flying ability
trait Fly {
   public function fly() {
      echo "Flying...\n";
   }
}
// Define a trait for swimming ability
trait Swim {
   public function swim() {
      echo "Swimming...\n";
   }

}

// Define a class for animals

class Animal {

   public function eat() {

      echo "Eating...\n";

   }
}

// Define a class for a bird that can fly

class Bird extends Animal {

   use Fly;

}

// Define a class for a fish that can swim

class Fish extends Animal {

   use Swim;

}

// Define a class for a duck that can both fly and swim

class Duck extends Animal {

   use Fly, Swim;

}

// Create instances and demonstrate

$bird = new Bird();

$bird->fly();

$fish = new Fish();

$fish->swim();

$duck = new Duck();

$duck->fly();

$duck->swim();
?>
```

- We define two traits: Fly and Swim, each containing a method.
- We have a base class Animal with a method eat.
- We have classes Bird and Fish that extend Animal and use the Fly and Swim traits, respectively.
- We have a class Duck that extends Animal and uses both Fly and Swim traits.

Using traits, we can mix and match behaviors across different classes, effectively achieving multiple inheritance.

4) HIERARCHICAL INHERITANCE:

In PHP, hierarchical inheritance is achieved by extending classes. Subclasses inherit properties and methods from their parent classes. This promotes code reuse and allows for a structured organization of code.



Syntax:

```
class ParentClass {
 // Parent class properties and methods
}
class ChildClass extends ParentClass {
// Child class properties and methods
}
```

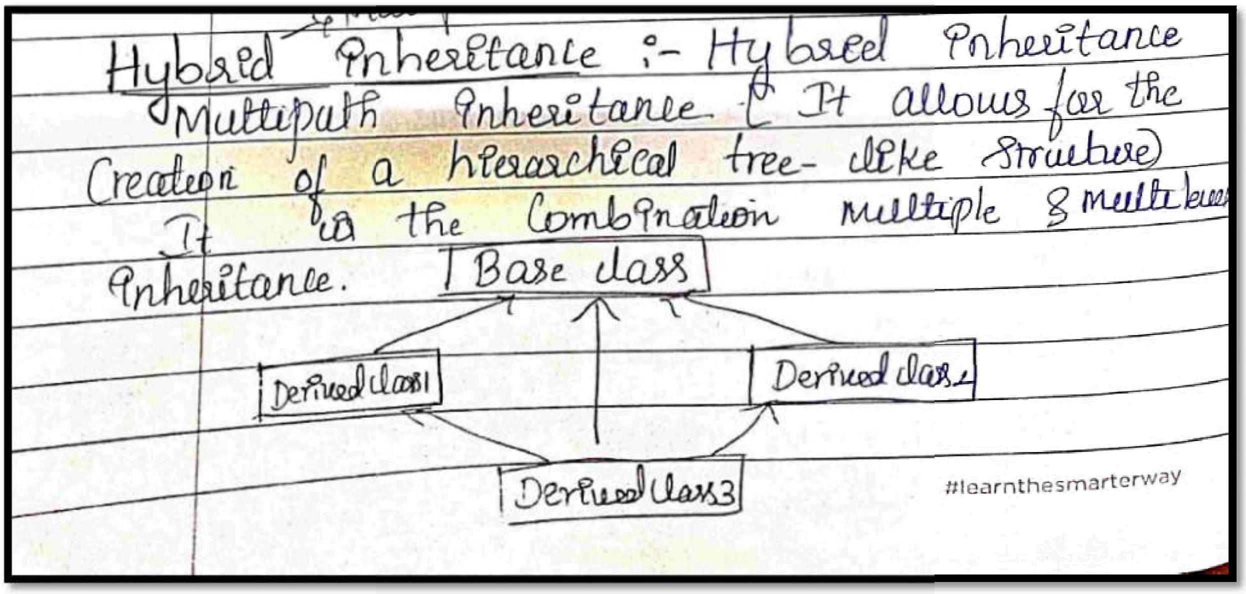
Example:

```
<?php
class Animal {
protected $name;
public function __construct($name) {
$this->name = $name;
}
public function getName() {
 return $this->name;
}
public function speak() {
echo "The animal makes a sound\n"
 }
}
class Dog extends Animal {
   public function speak() {
      echo "Woof!\n";
   }
}
class Cat extends Animal {
    public function speak() {
        echo "Meow!\n";
    }
}
// Create instances of subclasses
$dog = new Dog("Buddy");
$cat = new Cat("Whiskers");
echo $dog->getName() . " says: ";
$dog->speak();
echo $cat->getName() . " says: ";
$cat->speak();
?>
```

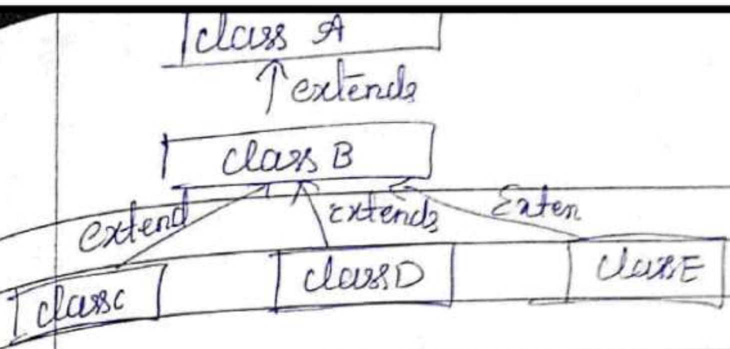
In this example, Animal is the parent class, and Dog and Cat are its subclasses. Both Dog and Cat inherit the getName() method from Animal, and they override the speak() method with their own implementations. When you run this code, you'll see that each subclass has its own behavior for the speak() method.

5)Hybrid inheritance:

Hybrid (Multipath) Inheritance (Not supported in PHP): Hybrid or multipath inheritance is a combination of multiple inheritance and multilevel inheritance. It involves inheriting from multiple classes and forming a multilevel inheritance structure simultaneously.

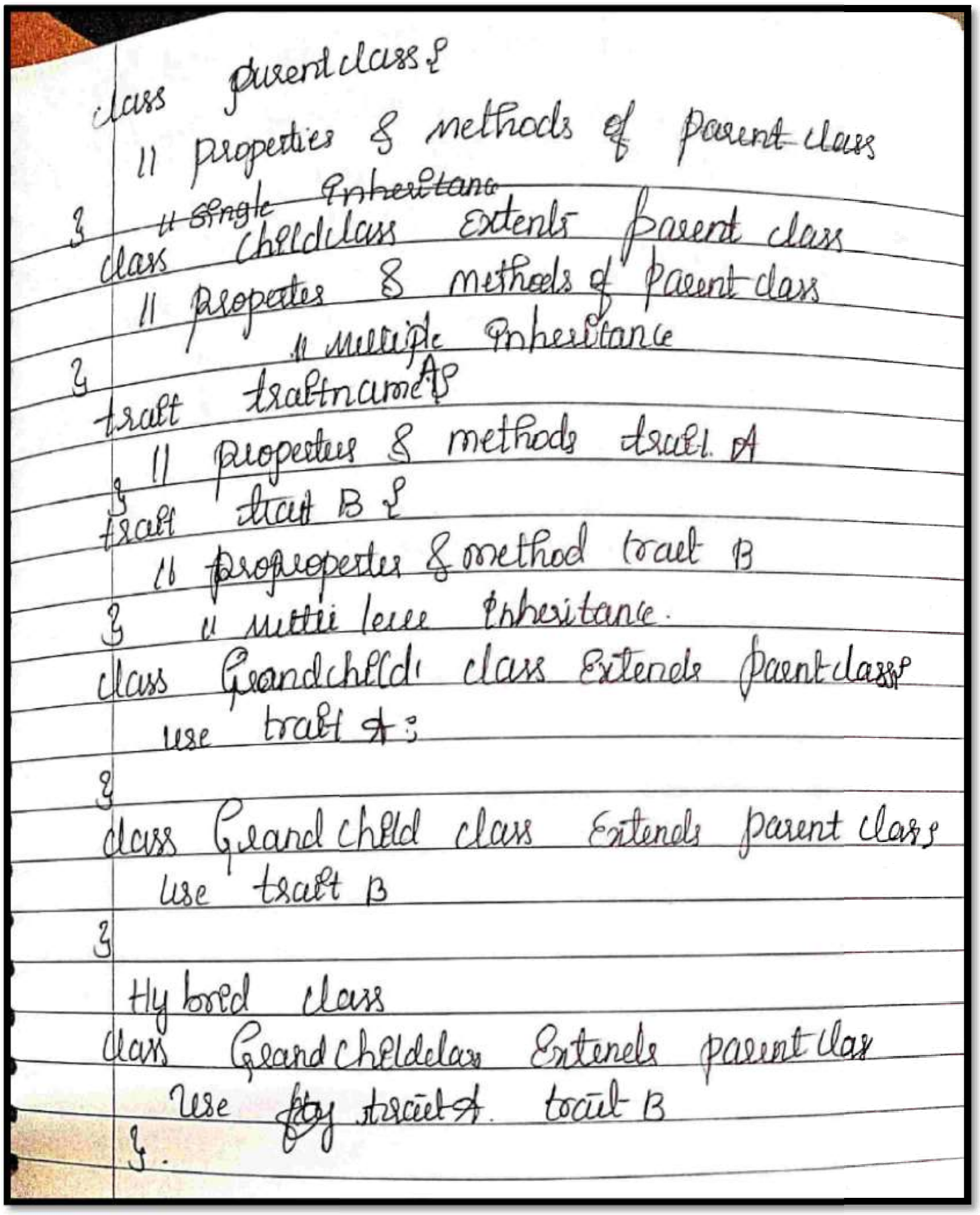


Hybrid Inheritance :- Hybreed Inheritance (Multipath Inheritance (It allows for the Creation of a hierarchical tree-like structure) It is the Combination multiple & multilevel Inheritance.



```
<?php
class Animal {
 public function Sound(){
 echo " Animal Make aSound ";
 }
} // Single Inheritance
class Dog extends Animal {
   public function bark(){
   echo " Woof! ";
 }
}
 traits fly{
  public function fly(){
      echo " Animal flies. ";
   }
  }
```

```
trait Swim{
 Public function Swim(){            Date: __/__/__
  echo " Animal Swim ";
 }
} // Multilevel Inheritance
class Bird Extend Animal{
 use fly;
 }
 class fish Extends Animal{
 use Swim;
} // Hybrid Inheritance
class Duck Extends Animal{
 use fly ,Swim;
}
$dog= new Dog();
$dog→ Sound(); o/p Animal make sound
$dog→ bark(); o/p woof!

$bird = new Bird();
$bird → Sound(); o/p Animal make Sound
$bird → fly(); o/p Animal flies

$fish = new fish();
$fish→ Sound(); Animal make a Sound
$fish→ Swim(); Animal Swim.
$duck → new Duck();
 $duck → Sound(); Animal make sound
 $duck → fly(); o/p Animal fly
 $ ... Swim(); o/p Animal Swims
```



```
class parentclass{
    // properties & methods of parent class
}   // ~~single~~ ~~Inheritance~~
class Childclass Extends parent class
    // properties & methods of parent class
        // multiple Inheritance
}
trait traitnameA{
    // properties & methods trait A
}
trait trait B {
    // ~~propropertes~~ & method trait B
}   // multi level Inheritance.
class Grandchild class Extends parentclass{
    use trait A;
}
class Grand child class Extends parent class
    use trait B
}
    Hybrid class
class Grandchildclass Extends parent clas
    Use ~~[illegible]~~ traitA. trait B
}.
```

CONSTRUCTOR AND DESTRUCTOR

Constructor: A constructor is a method that gets called automatically when an object is created from a class. It's typically used for initializing object properties or performing any setup tasks needed for the object to function properly. In PHP, the constructor method is named.

Uses of constructor in php

- In PHP, a constructor is a special method within a class that is automatically called when an object of that class is instantiated (i.e., when a new instance of the class is created using the `new` keyword).
- Constructors are useful for initializing object properties or performing any setup tasks that are required when an object is created.
- **Initialization**: Constructors are used to initialize object properties with default values or values passed as arguments during object creation. This ensures that objects are in a valid state when they are created.
- **Dependency Injection**: Constructors can be used for dependency injection, where objects are passed as arguments to the constructor. This allows for greater flexibility and decoupling of classes, making the code more maintainable and testable.
- **Resource Allocation**: Constructors are often used to allocate resources such as opening files, establishing database connections, or initializing other objects needed by the class. This ensures that resources are properly initialized and available for use when needed.
- **Configuration**: Constructors can be used to configure objects with initial settings or parameters. This can include setting up default values, loading configuration from external sources, or performing any necessary setup tasks before the object is used.

Syntax:

```
class ClassName {
  // Properties, methods, and other class members
    // Constructor method
 public function __construct(/* constructor parameters */) {
        // Constructor logic
    }
}
```

- ClassName: Replace this with the name of your class.
- __construct(): This is the constructor method name. It must be named exactly like this for it to be recognized as a constructor.
- /* constructor parameters */: You can optionally specify parameters for the constructor. These parameters are passed when creating an object of the class and used for initialization purposes.

- // Constructor logic: This is where you define the logic that should be executed when the object is created.

Example:

```
<?php

class Person {

   public $name;

   public $age;

   // Constructor method

   public function __construct($name, $age) {

       $this->name = $name;

       $this->age = $age;

       echo "Constructor called. Initializing $name with age $age.\n";

   }

   // Method to display person's information

   public function displayInfo() {

       echo "Name: $this->name, Age: $this->age\n";

   }

}

// Creating objects of the Person class

$person1 = new Person("madesh", 30);

$person2 = new Person("Alice", 25);

// Calling the displayInfo method to display information about the persons

$person1->displayInfo();
```

```
$person2->displayInfo();

?>
```

OUTPUT:

```
Constructor called. Initializing John with age 30.
Constructor called. Initializing Alice with age 25.
Name: madesh, Age: 30
Name: Alice, Age: 25
```

- We have a Person class with properties $name and $age.
- We define a constructor method __construct() within the class. This constructor takes two parameters ($name and $age) and initializes the object properties $name and $age with the provided values.
- When we create objects of the Person class ($person1 and $person2), the constructor method __construct() is automatically called, and the object properties are initialized.
- We also have a method displayInfo() within the Person class to display information about the person.

DESTRUCTOR IN PHP: In PHP, a destructor is a special method within a class that is automatically called when an object is destroyed or goes out of scope. The destructor method has a predefined name `__destruct()` and is useful for performing cleanup tasks, releasing resources, or executing finalization logic before an object is destroyed.

The syntax of a destructor method in PHP

Syntax:

```
class ClassName {

   // Properties, methods, and other class members

   // Destructor method

   public function __destruct() {

       // Destructor logic

   }
}
```

- ClassName: Replace this with the name of your class.

- __destruct(): This is the destructor method name. It must be named exactly like this for it to be recognized as a destructor.
- // Destructor logic: This is where you define the logic that should be executed when the object is destroyed. You can perform cleanup tasks, release resources, or execute any other finalization logic required before the object is destroyed.

Example:

```
class MyClass {

   public function __construct() {

      echo "Object created.\n";

   }

   public function __destruct() {

      echo "Object destroyed.\n";

   }

}

// Creating an object

$obj = new MyClass();

// Object goes out of scope

unset($obj);
?>
```

Output:

```
Object created.

Object destroyed.
```

In this example, MyClass is the class name, and __destruct() is the destructor method. When an object of the MyClass class is created, the constructor method __construct() is called, and when the object goes out of scope or is explicitly destroyed using unset($obj), the destructor method __destruct() is automatically called.

Form Handling: Handling HTML Form data in PHP

Form Handling: Form handling in web development is a fundamental aspect that enables interaction between users and web applications.

Here are some common uses of form handling.

1. **Data Collection**: Forms are used to collect various types of data from users, such as user registration information, contact details, feedback, surveys, and more.
2. **User Authentication**: Forms can be used for user authentication, where users provide their credentials (username and password) to log in to secure areas of a website.
3. **Content Submission**: Websites often allow users to submit content, such as articles, comments, reviews, or product ratings, through forms.
4. **E-commerce Transactions**: Forms facilitate e-commerce transactions by allowing users to provide shipping addresses, payment details, and order preferences during the checkout process.
5. **Search Functionality**: Search forms enable users to input search queries and retrieve relevant results from a website's database or search engine.

Creating HTML Form:

An HTML form is a section of a web page that contains interactive elements, such as text fields, checkboxes, radio buttons, buttons, and more, allowing users to input and submit data to a server. Forms are a crucial component of web development as they facilitate user interaction and data submission.

HTML5 introduced several new input types to make form validation easier and improve user experience.

HTML elements

1. **Search (<input type="search">):** This input type is designed for search fields. It typically provides a search box with a clear button for clearing the input.
2. **Tel (<input type="tel">):** Used for telephone numbers. It's designed to provide a keypad interface for entering phone numbers on devices with touchscreens.
3. **Email (<input type="email">):** Specifically for email addresses. It validates that the input text is in a proper email format.
4. **Number (<input type="number">):** This type is for numeric input. It typically provides a numeric keypad on mobile devices and restricts input to valid numbers.

5. **Range (<input type="range">):** Creates a slider control for selecting a value from a range. It's often used for settings or configurations where users need to select a value within a specific range.

HTML TAGS:HTML (Hypertext Markup Language) tags are the building blocks of HTML documents. They are used to define the structure and content of web pages. HTML tags consist of angled brackets < > enclosing a tag name, with attributes (if any) specified inside the opening tag. Here's a breakdown of HTML tags:

1. **Opening Tag:** Contains the name of the element enclosed in angled brackets, e.g., <tagname>.
2. **Closing Tag:** Similar to the opening tag but with a forward slash before the tag name, e.g., </tagname>.
3. **Content:** The text, images, or other elements enclosed between the opening and closing tags.
4. **Attributes:** Provide additional information about the element and are included within the opening tag. Attributes are written as name-value pairs, e.g., attribute="value".
   some common HTML tags along with their definitions and purposes:

1. **<html>:** Defines the root element of an HTML document. All other elements must be descendants of this element.
2. **<head>:** Contains meta-information about the HTML document, such as the title, links to stylesheets or scripts, and other metadata.
3. **<title>:** Sets the title of the HTML document, which appears in the browser's title bar or tab.
4. **<body>:** Contains the content of the HTML document that is visible to the user, including text, images, links, and other elements.
5. **<h1> to <h6>:** Defines headings of decreasing importance. <h1> is the highest level heading, while <h6> is the lowest.
6. **<p>:** Defines a paragraph of text.
7. **<a>:** Defines a hyperlink, which allows users to navigate to another page or resource when clicked.
8. **<img>:** Embeds an image into the HTML document.
9. **<ul>:** Defines an unordered list, typically rendered with bullet points.

10. **<ol>:** Defines an ordered list, typically rendered with numbers or letters.
11. **<li>:** Defines a list item within an ordered or unordered list.
12. **<div>:** Defines a division or section of an HTML document, often used for grouping and styling purposes.
13. **<span>:** Defines a small section of text within a larger block of text, often used for applying styles or scripting.
14. **<table>:** Defines a table for displaying tabular data.
15. **<tr>:** Defines a row within a table.
16. **<td>:** Defines a cell within a table row.
17. **<form>:** Defines a form for collecting user input.
18. **<input>:** Defines an input field within a form, such as a text field, checkbox, radio button, etc.
19. **<textarea>:** Defines a multi-line text input field within a form.
20. **<button>:** Defines a clickable button, often used to submit a form or trigger a JavaScript function

To create an HTML form in PHP, you'll typically use the <form> tag along with various input elements like <input>, <textarea>, and <select>.

Here's a basic example:

```
<!DOCTYPE html>

<html>

<head>

  <title>Form Handling</title>

</head>

<body>

  <form method="post" action="process_form.php">

    <label for="name">Name:</label>

    <input type="text" id="name" name="name"><br>

    <label for="email">Email:</label>
```

```
        <input type="email" id="email" name="email"><br>

        <label for="message">Message:</label><br>

        <textarea id="message" name="message" rows="4" cols="50"></textarea><br>

        <input type="submit" value="Submit">

    </form>

</body>

</html>
```

In this example:

- The <form> tag defines the form and specifies the method (POST in this case) and the action attribute, which indicates where to send the form data for processing. In this case, it's pointing to a PHP file called process_form.php.
- Various input fields like text, email, and textarea are used to collect user input.
- The <input type="submit"> creates a submit button for the form.

Handling HTML Form data in PHP:

To handle HTML form data in PHP, you need to follow these steps:

1. Create an HTML form with appropriate input fields.
2. Set the form action attribute to the PHP script where you want to handle the form data.
3. Use PHP to retrieve form data using the $_POST or $_GET superglobals, depending on the form submission method (POST or GET).
4. Process the form data as required, such as validation, sanitization, and database operations.

   Here's a basic example:

   HTML form (form.html):

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Form Example</title>

</head>

<body>

    <form action="process_form.php" method="post">

        <label for="name">Name:</label><br>

        <input type="text" id="name" name="name"><br>

        <label for="email">Email:</label><br>

        <input type="email" id="email" name="email"><br>

        <input type="submit" value="Submit">

    </form>

</body>

</html>
```

PHP script to handle form data (process_form.php)

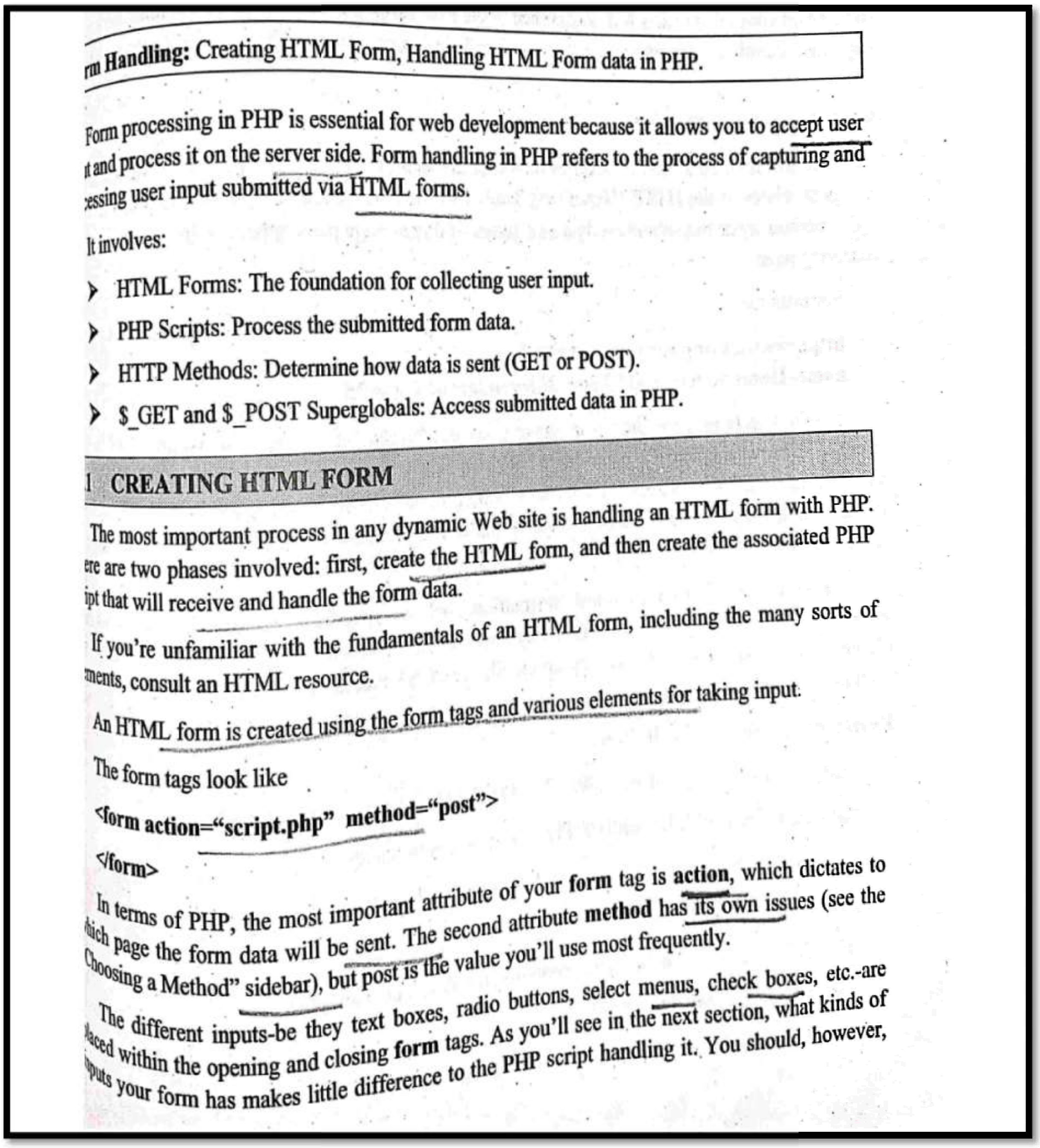
```
<?php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Retrieve form data
    $name = $_POST['name'];
    $email = $_POST['email'];
    // Validate form data
    if (empty($name) || empty($email)) {
        echo "Please fill all fields.";
    } else {
        // Sanitize form data (optional)
        $name = htmlspecialchars($name);
        $email = filter_var($email, FILTER_SANITIZE_EMAIL);
        // Process the form data (e.g., store in a database)
        // In this example, we just echo the submitted data
        echo "Name: $name<br>";
        echo "Email: $email";
    }
} else {
    echo "Form submission method not allowed.";
}
?>
```

In this example, when the form is submitted, the data is sent to the process_form.php script using the POST method. The PHP script retrieves the form data using the $_POST superglobal, validates it, and then processes it as needed.

Note: To handle the forn data in html form using get and post method.

Parameter	**GET**	**POST**
Visibility	Data is appended to the URL and visible in the browser's address bar.	Data is included in the request body and not visible in the URL.
Data Length	Limited by the URL length, which can vary by browser and server. Typically, around 2048 characters.	No inherent limit on data size, allowing for large amounts of data to be sent.
Security	Less secure due to visibility in the URL. Sensitive data can be easily exposed in browser history, server logs, etc.	More secure for transmitting sensitive data since it's not exposed in the URL.
Use Case	Ideal for simple data retrieval where the data can be bookmarked or shared.	Suited for transactions that result in a change on the server, such as updating data, submitting forms, and uploading files.
Idempotency	Idempotent, meaning multiple identical requests have the same effect as a single request.	Non-idempotent, meaning multiple identical requests, may have different outcomes.
Caching	Can be cached by the browser and proxies.	Not cached by default since it can change the server state.
Data Type	Only ASCII characters are allowed. Non-ASCII characters must be encoded.	Can handle binary data in addition to text, making it suitable for file uploads.
Back/Forward Buttons	Reloading a page requested by GET does not usually require browser confirmation.	Reloading a page can cause the browser to prompt the user for confirmation to resubmit the POST request.
Bookmarks and Sharing	Easily bookmarked and shared since the data is part of the URL.	Cannot be bookmarked or shared through the URL since the data is in the

		request body.
Impact on Server	Generally used for retrieving data without any side effects on the server.	Often causes a change in server state (e.g., database updates) or side effects.



rm Handling: Creating HTML Form, Handling HTML Form data in PHP.

Form processing in PHP is essential for web development because it allows you to accept user t and process it on the server side. Form handling in PHP refers to the process of capturing and essing user input submitted via HTML forms.

It involves:

- HTML Forms: The foundation for collecting user input.
- PHP Scripts: Process the submitted form data.
- HTTP Methods: Determine how data is sent (GET or POST).
- $_GET and $_POST Superglobals: Access submitted data in PHP.

1 CREATING HTML FORM

The most important process in any dynamic Web site is handling an HTML form with PHP. ere are two phases involved: first, create the HTML form, and then create the associated PHP ipt that will receive and handle the form data.

If you're unfamiliar with the fundamentals of an HTML form, including the many sorts of ments, consult an HTML resource.

An HTML form is created using the form tags and various elements for taking input.

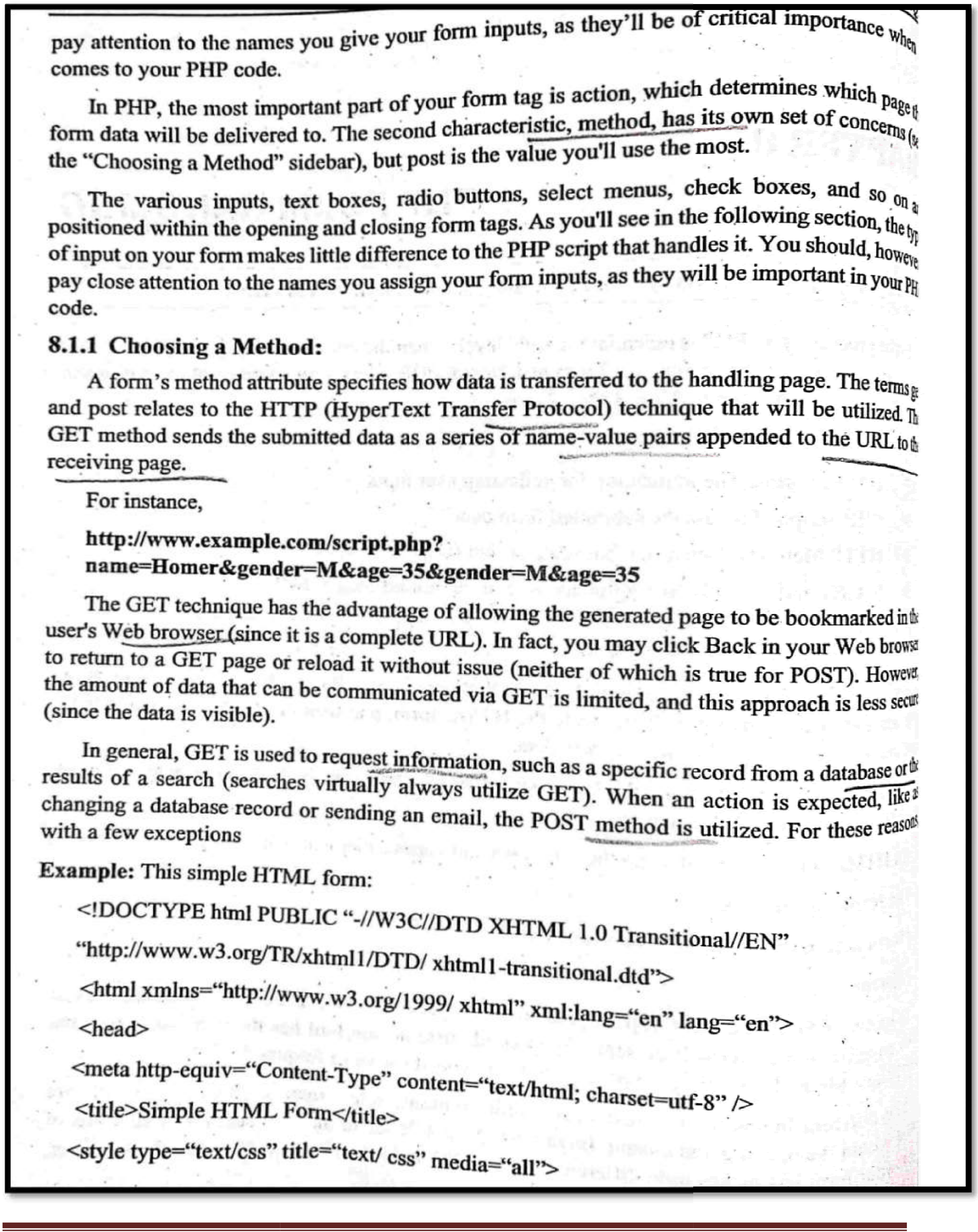
The form tags look like

<form action="script.php" method="post">

</form>

In terms of PHP, the most important attribute of your **form** tag is **action**, which dictates to hich page the form data will be sent. The second attribute **method** has its own issues (see the Choosing a Method" sidebar), but post is the value you'll use most frequently.

The different inputs-be they text boxes, radio buttons, select menus, check boxes, etc.-are laced within the opening and closing **form** tags. As you'll see in the next section, what kinds of nputs your form has makes little difference to the PHP script handling it. You should, however,



pay attention to the names you give your form inputs, as they'll be of critical importance when comes to your PHP code.

In PHP, the most important part of your form tag is action, which determines which page t form data will be delivered to. The second characteristic, method, has its own set of concerns (s the "Choosing a Method" sidebar), but post is the value you'll use the most.

The various inputs, text boxes, radio buttons, select menus, check boxes, and so on a positioned within the opening and closing form tags. As you'll see in the following section, the ty of input on your form makes little difference to the PHP script that handles it. You should, howeve pay close attention to the names you assign your form inputs, as they will be important in your PH code.

8.1.1 Choosing a Method:

A form's method attribute specifies how data is transferred to the handling page. The terms g and post relates to the HTTP (HyperText Transfer Protocol) technique that will be utilized. Th GET method sends the submitted data as a series of name-value pairs appended to the URL to th receiving page.

For instance,

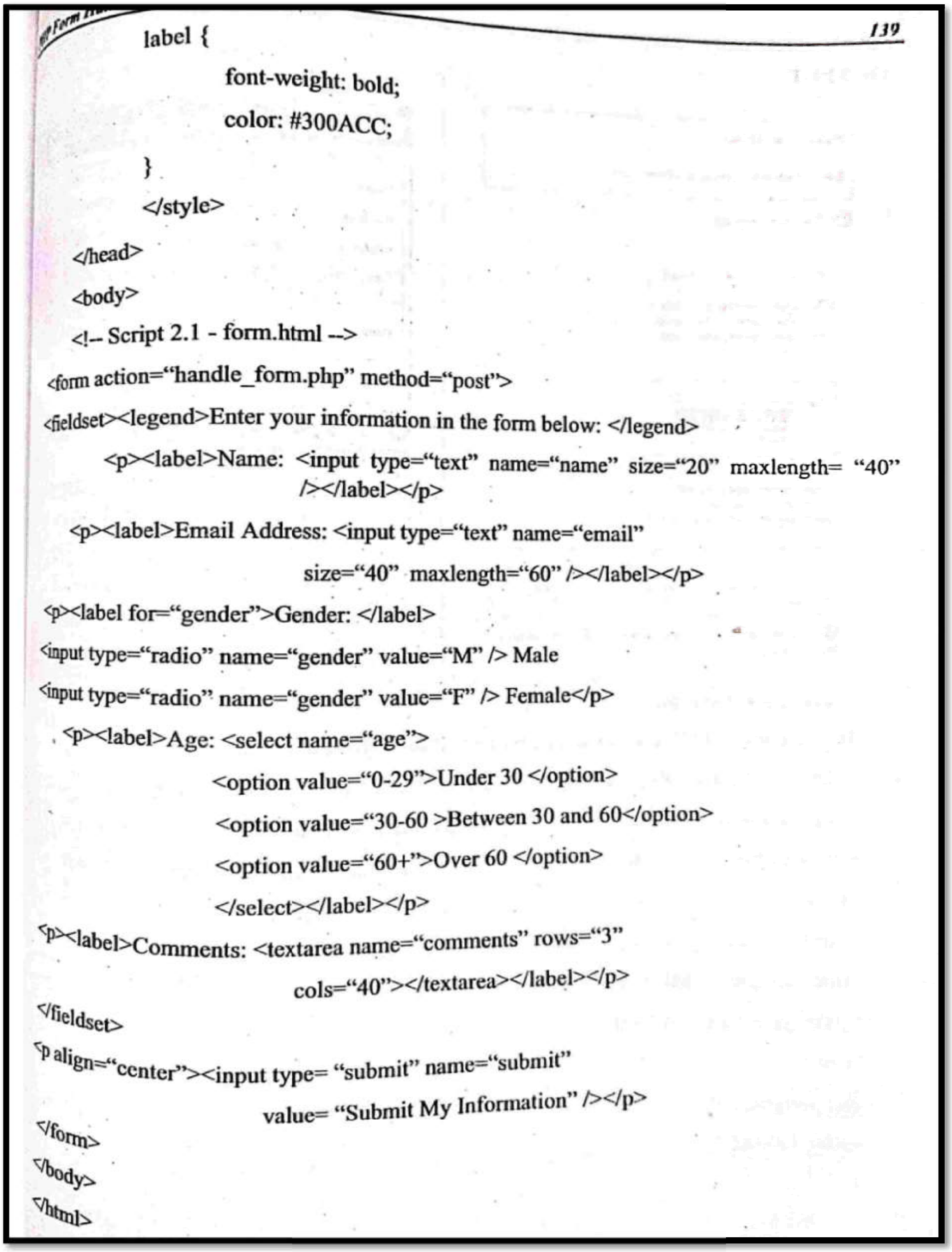
http://www.example.com/script.php?
name=Homer&gender=M&age=35&gender=M&age=35

The GET technique has the advantage of allowing the generated page to be bookmarked in th user's Web browser (since it is a complete URL). In fact, you may click Back in your Web brows to return to a GET page or reload it without issue (neither of which is true for POST). However the amount of data that can be communicated via GET is limited, and this approach is less secur (since the data is visible).

In general, GET is used to request information, such as a specific record from a database or th results of a search (searches virtually always utilize GET). When an action is expected, like a changing a database record or sending an email, the POST method is utilized. For these reasons with a few exceptions

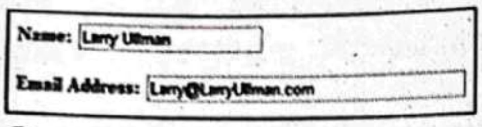
Example: This simple HTML form:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/ xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/ xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Simple HTML Form</title>
<style type="text/css" title="text/ css" media="all">
```



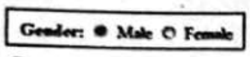
```
            label {
                        font-weight: bold;
                        color: #300ACC;
            }
            </style>
    </head>
    <body>
    <!-- Script 2.1 - form.html -->
<form action="handle_form.php" method="post">
<fieldset><legend>Enter your information in the form below: </legend>
        <p><label>Name: <input type="text" name="name" size="20" maxlength= "40"
                        /></label></p>
    <p><label>Email Address: <input type="text" name="email"
                              size="40" maxlength="60" /></label></p>
<p><label for="gender">Gender: </label>
<input type="radio" name="gender" value="M" /> Male
<input type="radio" name="gender" value="F" /> Female</p>
  <p><label>Age: <select name="age">
                    <option value="0-29">Under 30 </option>
                    <option value="30-60 >Between 30 and 60</option>
                    <option value="60+">Over 60 </option>
                    </select></label></p>
<p><label>Comments: <textarea name="comments" rows="3"
                              cols="40"></textarea></label></p>
</fieldset>
<p align="center"><input type= "submit" name="submit"
                        value= "Submit My Information" /></p>
</form>
</body>
</html>
```

OUTPUT:

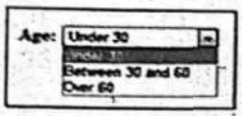

Name: Larry Ullman

Email Address: Larry@LarryUllman.com

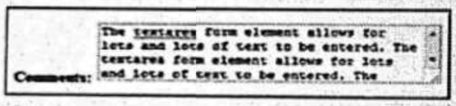
A Two text inputs.



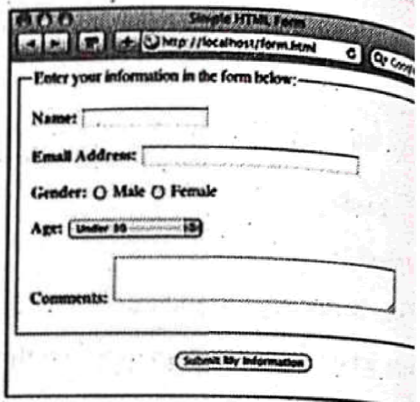
B If multiple radio buttons have the same name value, only one can be selected by the user.



C The pull-down menu offers three options, of which only one can be selected (in this example).


Comments: The textarea form element allows for lots and lots of text to be entered. The textarea form element allows for lots and lots of text to be entered. The

D The textarea form element type allows for lots and lots of text.

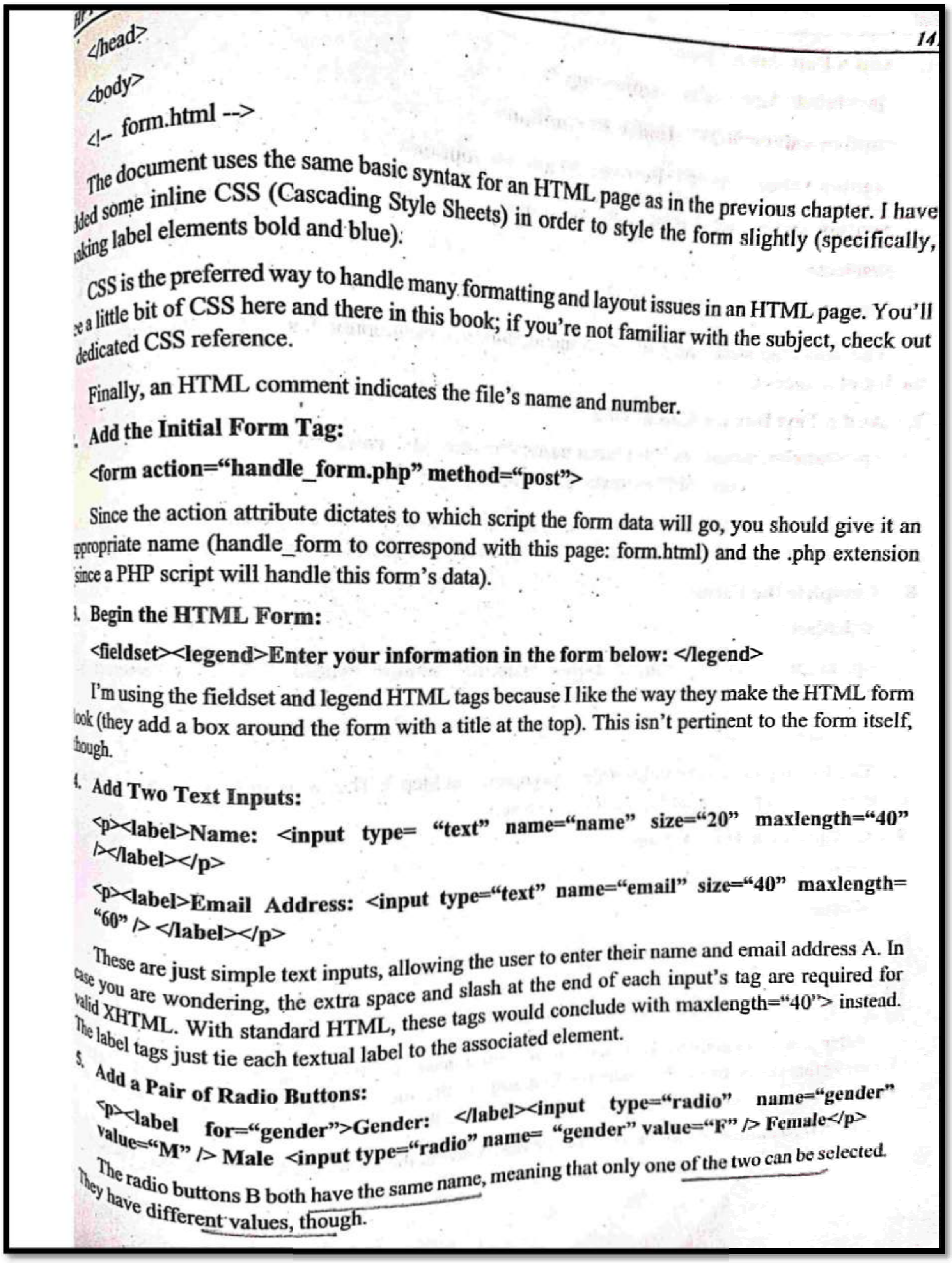


E The complete form, which requests some b information from the user.

To Create an HTML Form:

1. Begin a new HTML document in your text editor or IDE, to be named form.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/ xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"  lang="en">
<head>
<meta http-equiv="Content-Type"  content="text/html; charset=utf-8" />
<title>Simple HTML Form</title>
<style type="text/css"  title="text/css" media="all">
label {
font-weight: bold;
color: #300ACC;
}
</style>
```



```
</head>
<body>
<!-- form.html -->
```

The document uses the same basic syntax for an HTML page as in the previous chapter. I have dded some inline CSS (Cascading Style Sheets) in order to style the form slightly (specifically, aking label elements bold and blue).

CSS is the preferred way to handle many formatting and layout issues in an HTML page. You'll e a little bit of CSS here and there in this book; if you're not familiar with the subject, check out dedicated CSS reference.

Finally, an HTML comment indicates the file's name and number.

. **Add the Initial Form Tag:**

<form action="handle_form.php" method="post">

Since the action attribute dictates to which script the form data will go, you should give it an ppropriate name (handle_form to correspond with this page: form.html) and the .php extension since a PHP script will handle this form's data).

3. **Begin the HTML Form:**

<fieldset><legend>Enter your information in the form below: </legend>

I'm using the fieldset and legend HTML tags because I like the way they make the HTML form look (they add a box around the form with a title at the top). This isn't pertinent to the form itself, though.

4. **Add Two Text Inputs:**

<p><label>Name: <input type= "text" name="name" size="20" maxlength="40" /></label></p>

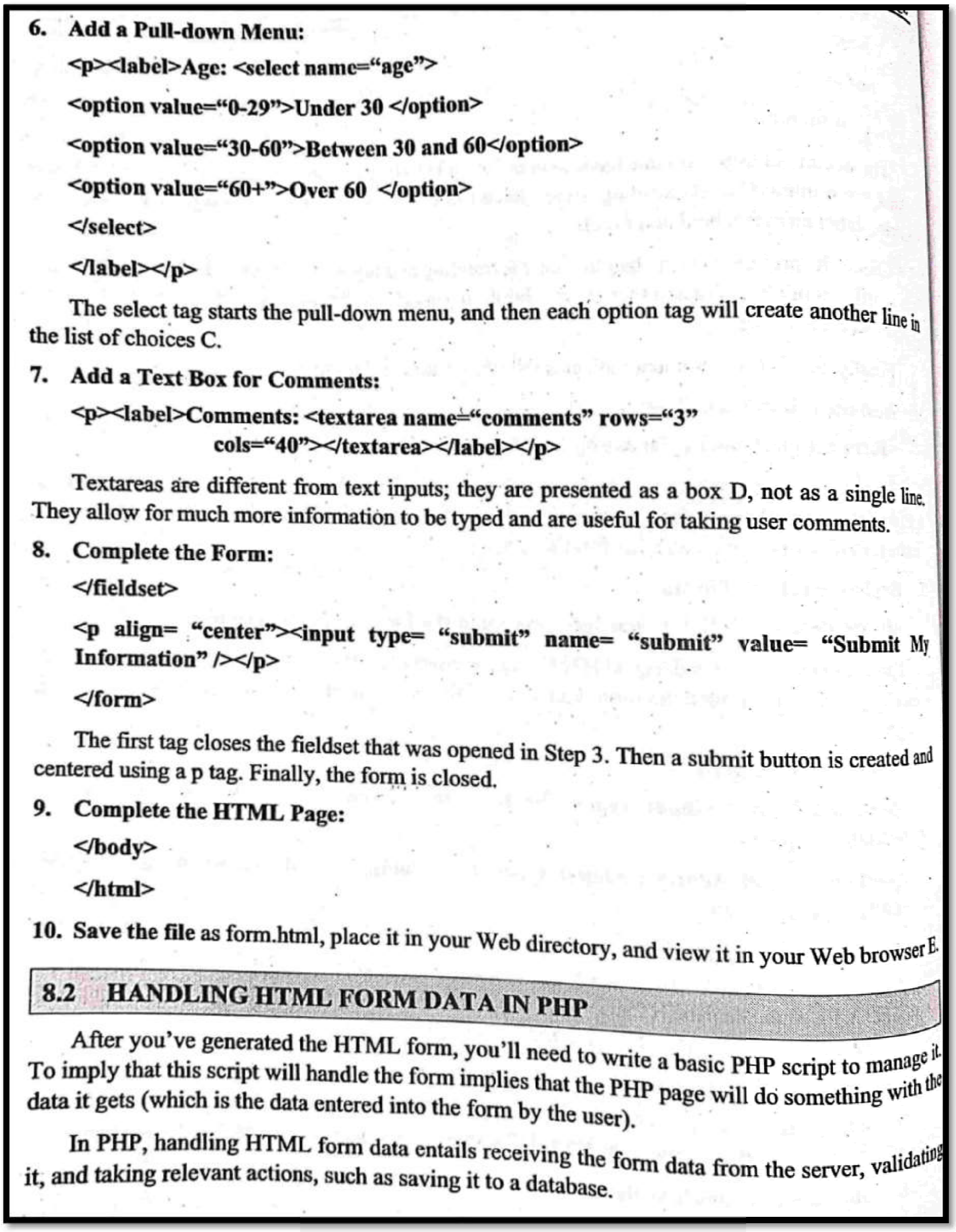
<p><label>Email Address: <input type="text" name="email" size="40" maxlength= "60" /> </label></p>

These are just simple text inputs, allowing the user to enter their name and email address A. In case you are wondering, the extra space and slash at the end of each input's tag are required for valid XHTML. With standard HTML, these tags would conclude with maxlength="40"> instead. The label tags just tie each textual label to the associated element.

5. **Add a Pair of Radio Buttons:**

<p><label for="gender">Gender: </label><input type="radio" name="gender" value="M" /> Male <input type="radio" name= "gender" value="F" /> Female</p>

The radio buttons B both have the same name, meaning that only one of the two can be selected. They have different values, though.



6. **Add a Pull-down Menu:**

```
<p><label>Age: <select name="age">
<option value="0-29">Under 30 </option>
<option value="30-60">Between 30 and 60</option>
<option value="60+">Over 60  </option>
</select>
</label></p>
```

The select tag starts the pull-down menu, and then each option tag will create another line in the list of choices C.

7. **Add a Text Box for Comments:**

```
<p><label>Comments: <textarea name="comments" rows="3"
                cols="40"></textarea></label></p>
```

Textareas are different from text inputs; they are presented as a box D, not as a single line. They allow for much more information to be typed and are useful for taking user comments.

8. **Complete the Form:**

```
</fieldset>
<p align= "center"><input type= "submit" name= "submit" value= "Submit My
Information" /></p>
</form>
```

The first tag closes the fieldset that was opened in Step 3. Then a submit button is created and centered using a p tag. Finally, the form is closed.

9. **Complete the HTML Page:**

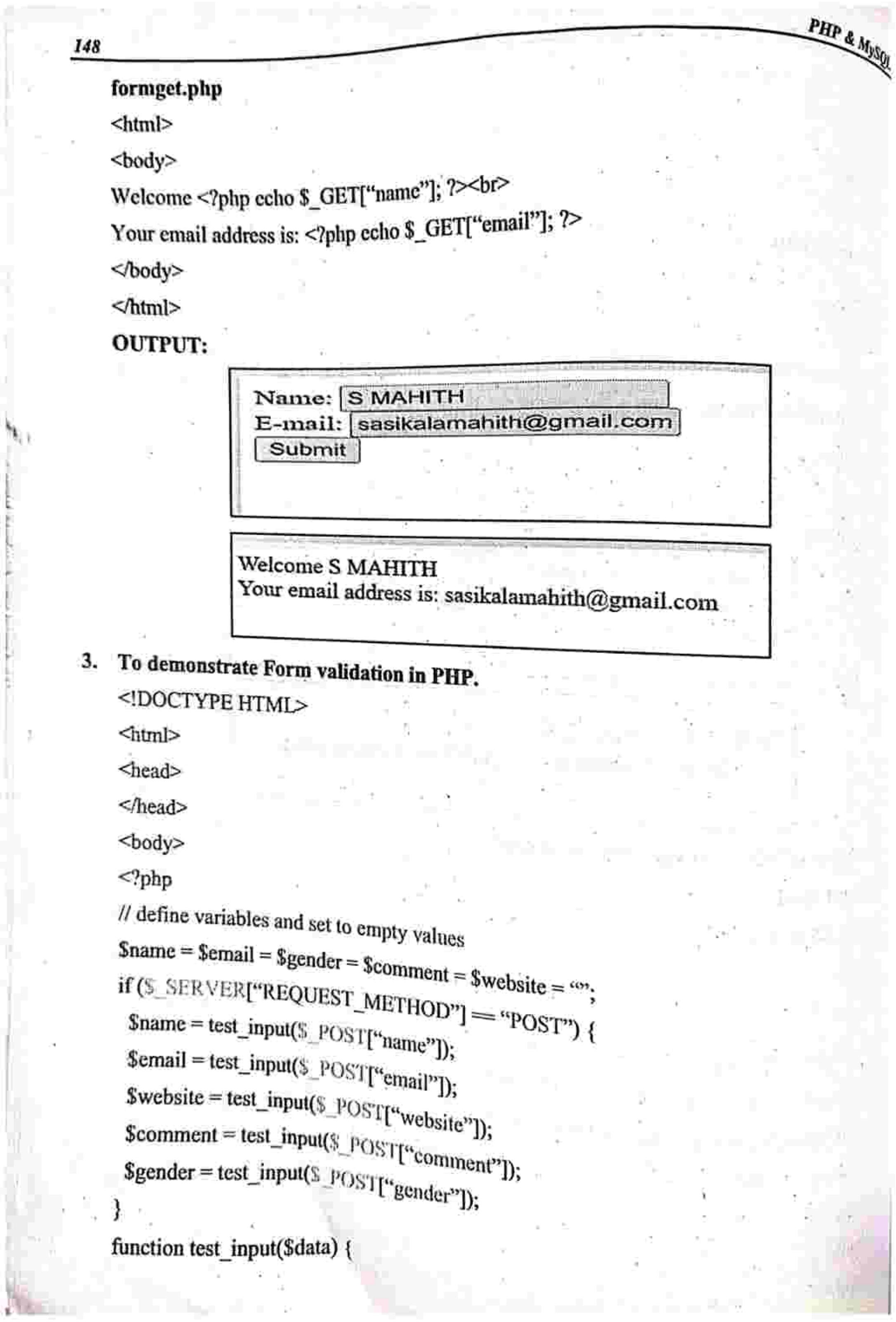
```
</body>
</html>
```

10. **Save the file** as form.html, place it in your Web directory, and view it in your Web browser E.

8.2 HANDLING HTML FORM DATA IN PHP

After you've generated the HTML form, you'll need to write a basic PHP script to manage it. To imply that this script will handle the form implies that the PHP page will do something with the data it gets (which is the data entered into the form by the user).

In PHP, handling HTML form data entails receiving the form data from the server, validating it, and taking relevant actions, such as saving it to a database.



formget.php

```
<html>
<body>
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
</body>
</html>
```

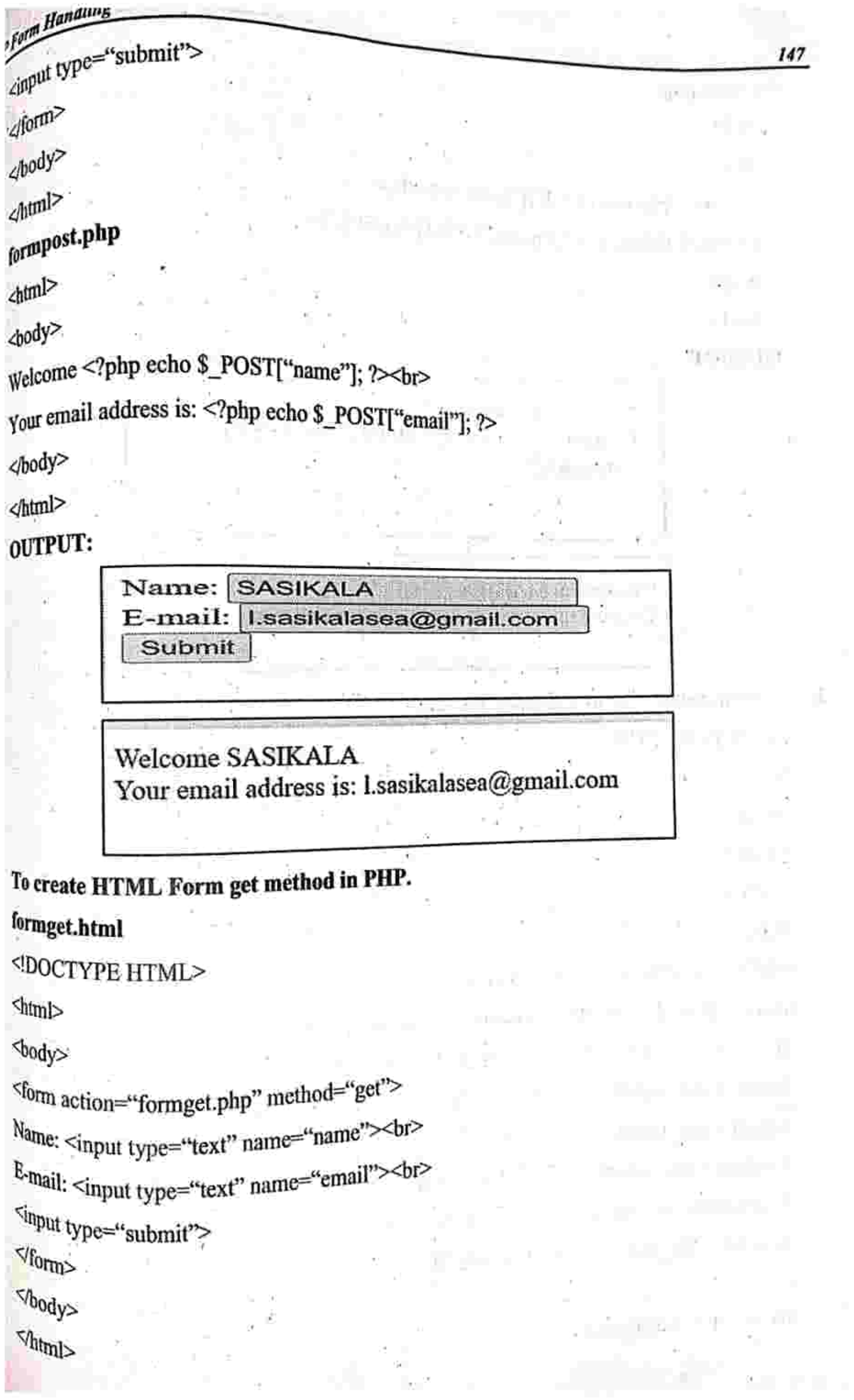
OUTPUT:

```
Name: S MAHITH
E-mail: sasikalamahith@gmail.com
Submit
```

```
Welcome S MAHITH
Your email address is: sasikalamahith@gmail.com
```

3. **To demonstrate Form validation in PHP.**

```
<!DOCTYPE HTML>
<html>
<head>
</head>
<body>
<?php
// define variables and set to empty values
$name = $email = $gender = $comment = $website = "";
if ($_SERVER["REQUEST_METHOD"] == "POST") {
  $name = test_input($_POST["name"]);
  $email = test_input($_POST["email"]);
  $website = test_input($_POST["website"]);
  $comment = test_input($_POST["comment"]);
  $gender = test_input($_POST["gender"]);
}
function test_input($data) {
```



```
<input type="submit">
</form>
</body>
</html>
```

formpost.php

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

OUTPUT:

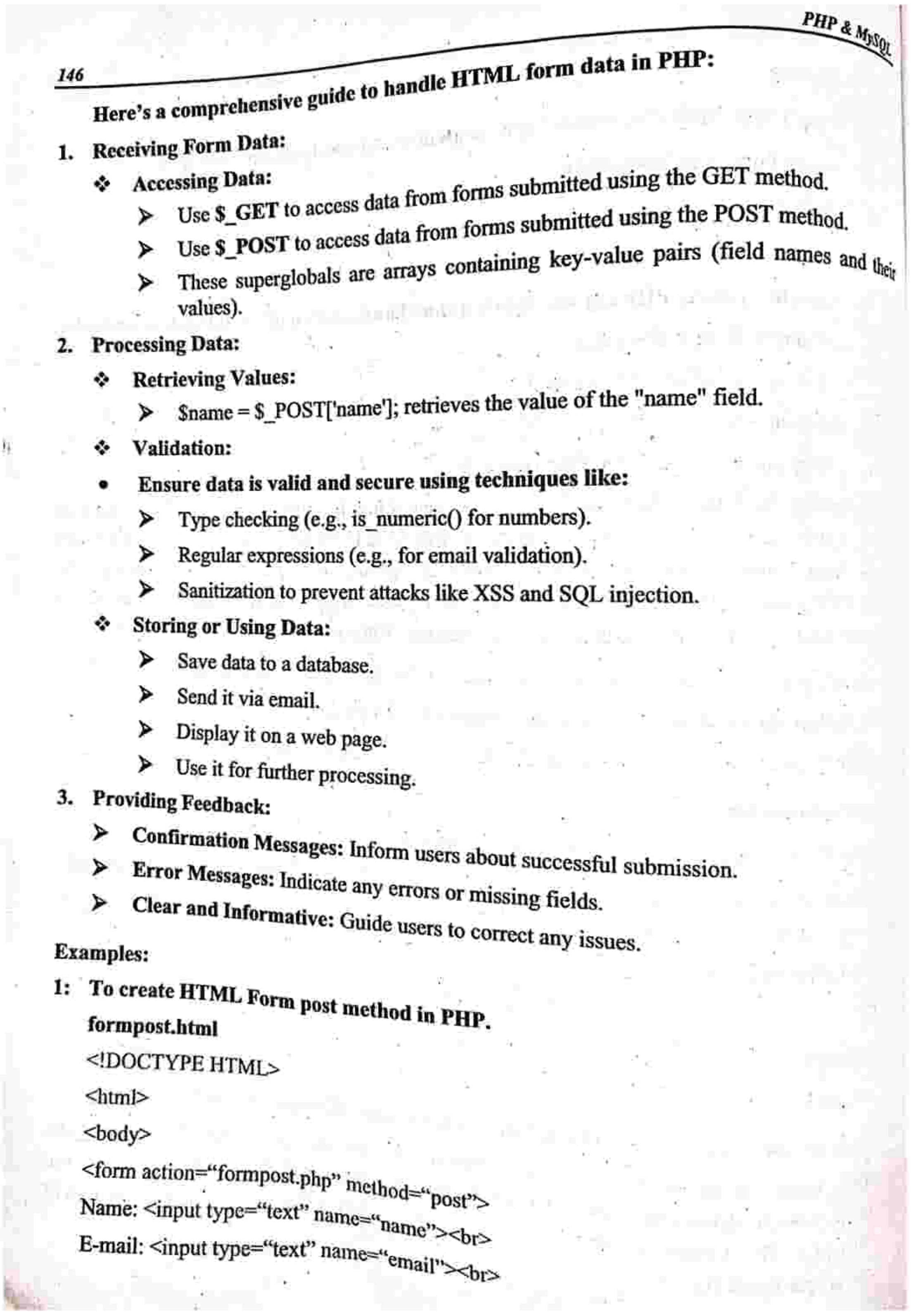
```
Name: SASIKALA
E-mail: l.sasikalasea@gmail.com
Submit
```

```
Welcome SASIKALA
Your email address is: l.sasikalasea@gmail.com
```

To create HTML Form get method in PHP.

formget.html

```
<!DOCTYPE HTML>
<html>
<body>
<form action="formget.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```



Here's a comprehensive guide to handle HTML form data in PHP:

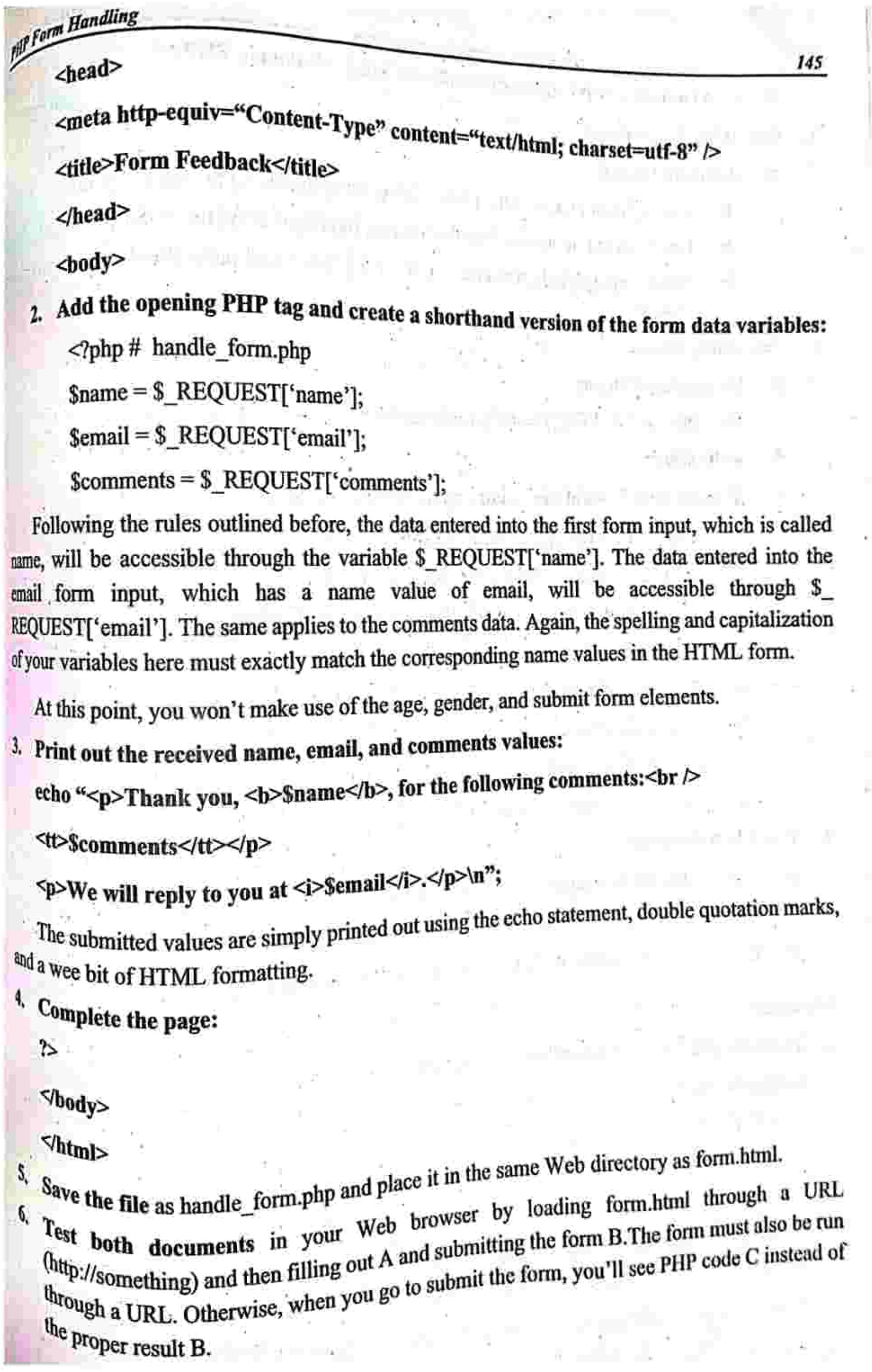
1. **Receiving Form Data:**
   - ❖ **Accessing Data:**
     - ➢ Use **$_GET** to access data from forms submitted using the GET method.
     - ➢ Use **$_POST** to access data from forms submitted using the POST method.
     - ➢ These superglobals are arrays containing key-value pairs (field names and their values).
2. **Processing Data:**
   - ❖ **Retrieving Values:**
     - ➢ $name = $_POST['name']; retrieves the value of the "name" field.
   - ❖ **Validation:**
   - • **Ensure data is valid and secure using techniques like:**
     - ➢ Type checking (e.g., is_numeric() for numbers).
     - ➢ Regular expressions (e.g., for email validation).
     - ➢ Sanitization to prevent attacks like XSS and SQL injection.
   - ❖ **Storing or Using Data:**
     - ➢ Save data to a database.
     - ➢ Send it via email.
     - ➢ Display it on a web page.
     - ➢ Use it for further processing.
3. **Providing Feedback:**
   - ➢ **Confirmation Messages:** Inform users about successful submission.
   - ➢ **Error Messages:** Indicate any errors or missing fields.
   - ➢ **Clear and Informative:** Guide users to correct any issues.

Examples:

1: To create HTML Form post method in PHP.

formpost.html

```
<!DOCTYPE HTML>
<html>
<body>
<form action="formpost.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
```



```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Form Feedback</title>
</head>
<body>
```

2. **Add the opening PHP tag and create a shorthand version of the form data variables:**

```
<?php #  handle_form.php
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$comments = $_REQUEST['comments'];
```

Following the rules outlined before, the data entered into the first form input, which is called name, will be accessible through the variable $_REQUEST['name']. The data entered into the email form input, which has a name value of email, will be accessible through $_REQUEST['email']. The same applies to the comments data. Again, the spelling and capitalization of your variables here must exactly match the corresponding name values in the HTML form.

At this point, you won't make use of the age, gender, and submit form elements.

3. **Print out the received name, email, and comments values:**

```
echo "<p>Thank you, <b>$name</b>, for the following comments:<br />
<tt>$comments</tt></p>
<p>We will reply to you at <i>$email</i>.</p>\n";
```

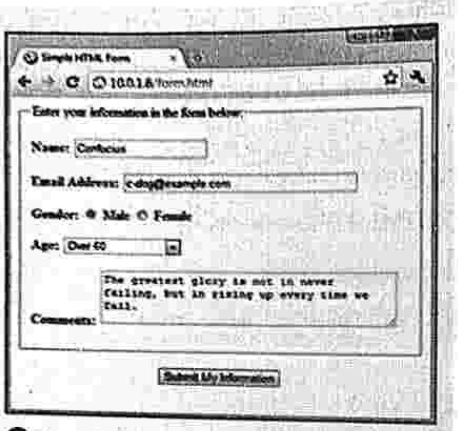
The submitted values are simply printed out using the echo statement, double quotation marks, and a wee bit of HTML formatting.

4. **Complete the page:**

```
?>
</body>
</html>
```

5. **Save the file** as handle_form.php and place it in the same Web directory as form.html.
6. **Test both documents** in your Web browser by loading form.html through a URL (http://something) and then filling out A and submitting the form B.The form must also be run through a URL. Otherwise, when you go to submit the form, you'll see PHP code C instead of the proper result B.

```
?>
</body>
</html>
```



Enter your information in the form below:

Name: Confucius

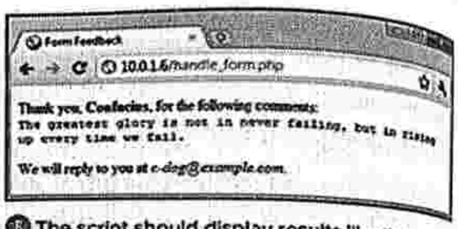
Email Address: c-dog@example.com

Gender: [✓ Male] Female

Age: Over 60

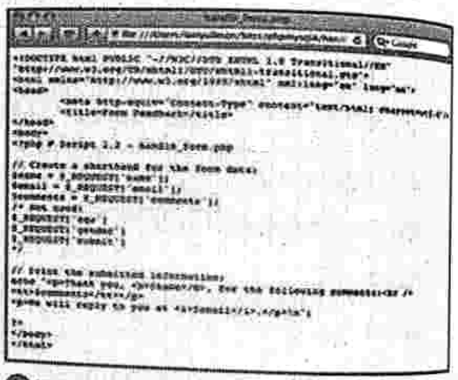
Comments:
```
The greatest glory is not in never
failing, but in rising up every time we
fall.
```

Ⓐ To test `handle_form.php`, you must load the form through a URL, then fill it out and submit it.



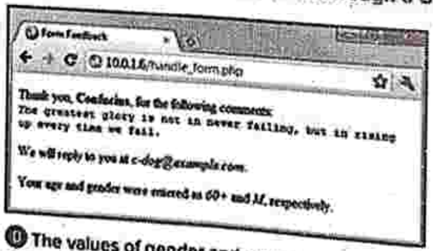
Thank you, Confucius, for the following comments:
```
The greatest glory is not in never failing, but in rising
up every time we fall.
```
We will reply to you at c-dog@example.com.

Ⓑ The script should display results like this.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>Form Feedback</title>
</head>
<body>
<?php # Script 2.2 - handle_form.php

// Create a shorthand for the form data:
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$comments = $_REQUEST['comments'];
/* Not used:
$_REQUEST['age']
$_REQUEST['gender']
$_REQUEST['submit']
*/

// Print the submitted information:
echo "<p>Thank you, <b>$name</b>, for the following comments:<br />
<tt>$comments</tt></p>
<p>We will reply to you at <i>$email</i>.</p>\n";

?>
</body>
</html>
```

Ⓒ If you see the PHP code after submitting the form, the problem is likely that you did not access the form through a URL.



Thank you, Confucius, for the following comments:
```
The greatest glory is not in never failing, but in rising
up every time we fall.
```
We will reply to you at c-dog@example.com.

Your age and gender were entered as *60+* and *M*, respectively.

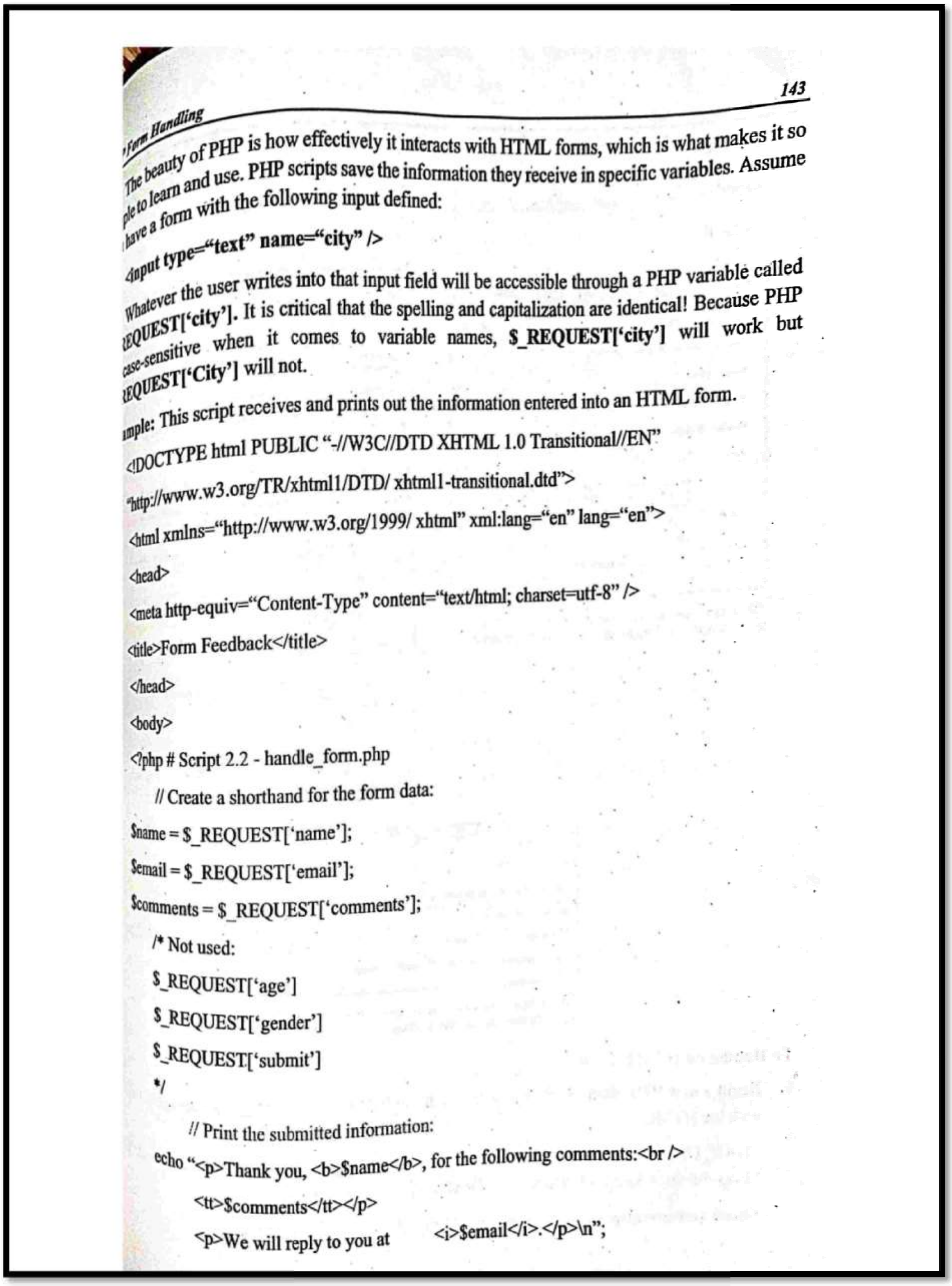
Ⓓ The values of gender and age correspond to those defined in the form's HTML.

To Handle an HTML Form:

1. **Begin a new PHP document** in your text editor or IDE, to be named handle_form.php starting with the HTML.

<!DOCTYPE html PUBLIC "-//W3C// DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/ xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/ 1999/xhtml" xml:lang="en" lang="en">



The beauty of PHP is how effectively it interacts with HTML forms, which is what makes it so ple to learn and use. PHP scripts save the information they receive in specific variables. Assume have a form with the following input defined:

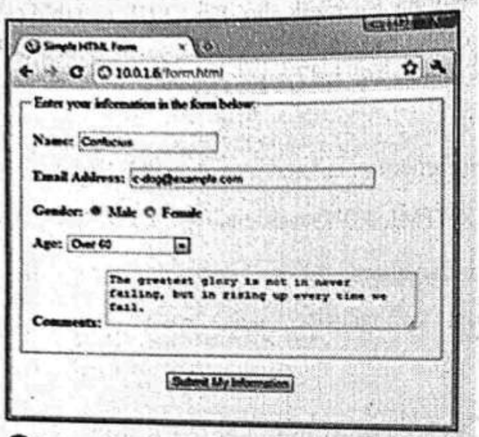
<input type="text" name="city" />

Whatever the user writes into that input field will be accessible through a PHP variable called **REQUEST['city']**. It is critical that the spelling and capitalization are identical! Because PHP ase-sensitive when it comes to variable names, **$_REQUEST['city']** will work but **REQUEST['City']** will not.

mple: This script receives and prints out the information entered into an HTML form.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/ xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/ xhtml" xml:lang="en" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Form Feedback</title>
</head>
<body>
<?php # Script 2.2 - handle_form.php
   // Create a shorthand for the form data:
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$comments = $_REQUEST['comments'];
   /* Not used:
   $_REQUEST['age']
   $_REQUEST['gender']
   $_REQUEST['submit']
   */
      // Print the submitted information:
   echo "<p>Thank you, <b>$name</b>, for the following comments:<br />
      <tt>$comments</tt></p>
      <p>We will reply to you at        <i>$email</i>.</p>\n";
```

```
?>
</body>
</html>
```



Enter your information in the form below:

Name: Confucius

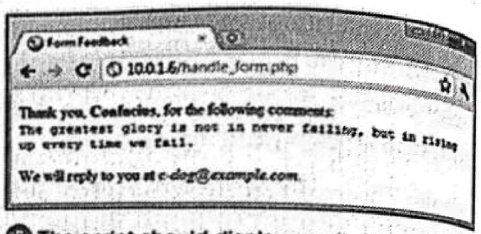
Email Address: c-dog@example.com

Gender: ● Male ○ Female

Age: Over 60

Comments: The greatest glory is not in never failing, but in rising up every time we fail.

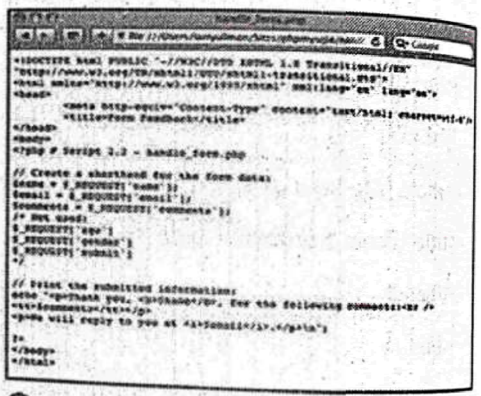
A To test **handle_form.php**, you must load the form through a URL, then fill it out and submit it.



Thank you, Confucius, for the following comments:
The greatest glory is not in never failing, but in rising up every time we fail.

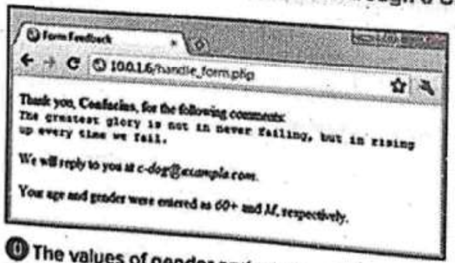
We will reply to you at *c-dog@example.com*.

B The script should display results like this.



```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
        <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
        <title>Form Feedback</title>
</head>
<body>
<?php # Script 2.3 - handle_form.php

// Create a shorthand for the form data:
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$comments = $_REQUEST['comments'];
/* Not used:
$_REQUEST['age']
$_REQUEST['gender']
$_REQUEST['submit']
*/

// Print the submitted information:
echo "<p>Thank you, <b>$name</b>, for the following comments:<br />
<tt>$comments</tt></p>
<p>We will reply to you at <i>$email</i>.</p>\n";

?>
</body>
</html>
```

C If you see the PHP code after submitting the form, the problem is likely that you did not access the form through a URL.



Thank you, Confucius, for the following comments:
The greatest glory is not in never failing, but in rising up every time we fail.

We will reply to you at *c-dog@example.com*.

Your age and gender were entered as *60+* and *M*, respectively.

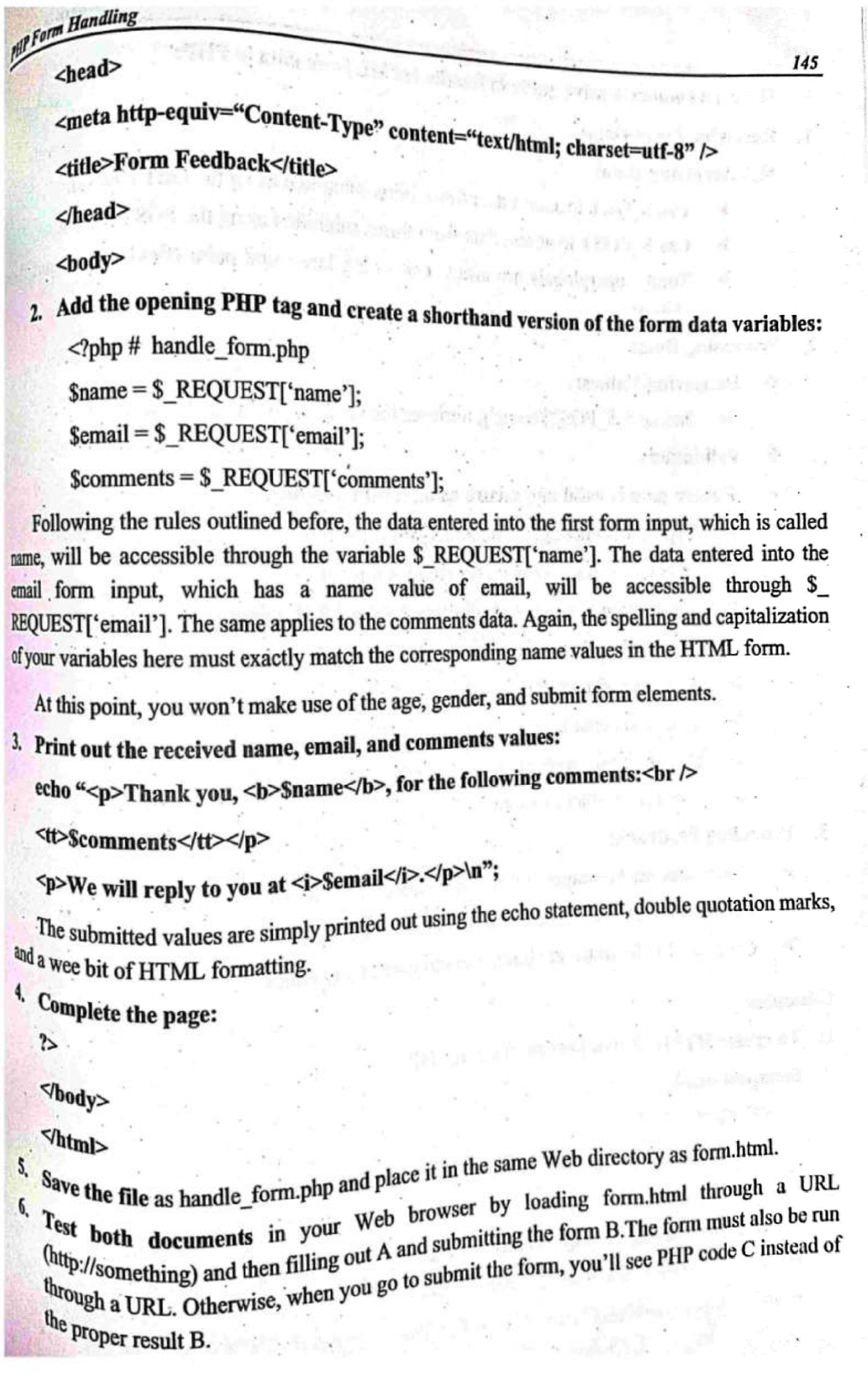
D The values of gender and age correspond to those defined in the form's HTML.

To Handle an HTML Form:

1. **Begin a new PHP document** in your text editor or IDE, to be named handle_form.php starting with the HTML.

   <!DOCTYPE html PUBLIC "-//W3C// DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/ xhtml1-transitional.dtd">

   <html xmlns="http://www.w3.org/ 1999/xhtml" xml:lang="en" lang="en">



```
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Form Feedback</title>
</head>
<body>
```

2. **Add the opening PHP tag and create a shorthand version of the form data variables:**

```
<?php # handle_form.php
$name = $_REQUEST['name'];
$email = $_REQUEST['email'];
$comments = $_REQUEST['comments'];
```

Following the rules outlined before, the data entered into the first form input, which is called name, will be accessible through the variable $_REQUEST['name']. The data entered into the email form input, which has a name value of email, will be accessible through $_REQUEST['email']. The same applies to the comments data. Again, the spelling and capitalization of your variables here must exactly match the corresponding name values in the HTML form.

At this point, you won't make use of the age, gender, and submit form elements.

3. **Print out the received name, email, and comments values:**

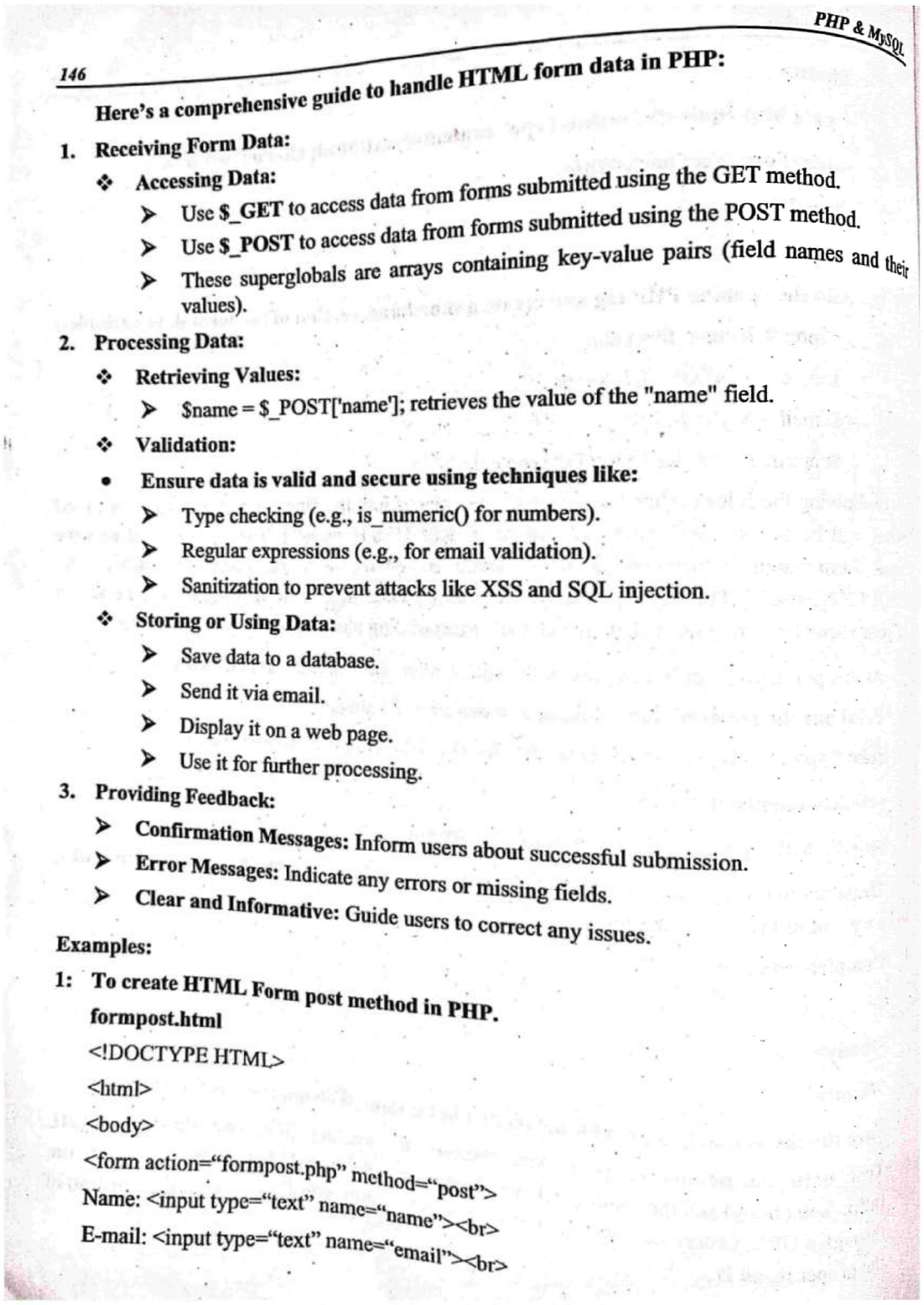
```
echo "<p>Thank you, <b>$name</b>, for the following comments:<br />
<tt>$comments</tt></p>
<p>We will reply to you at <i>$email</i>.</p>\n";
```

The submitted values are simply printed out using the echo statement, double quotation marks, and a wee bit of HTML formatting.

4. **Complete the page:**

```
?>
</body>
</html>
```

5. **Save the file** as handle_form.php and place it in the same Web directory as form.html.
6. **Test both documents** in your Web browser by loading form.html through a URL (http://something) and then filling out A and submitting the form B.The form must also be run through a URL. Otherwise, when you go to submit the form, you'll see PHP code C instead of the proper result B.



Here's a comprehensive guide to handle HTML form data in PHP:

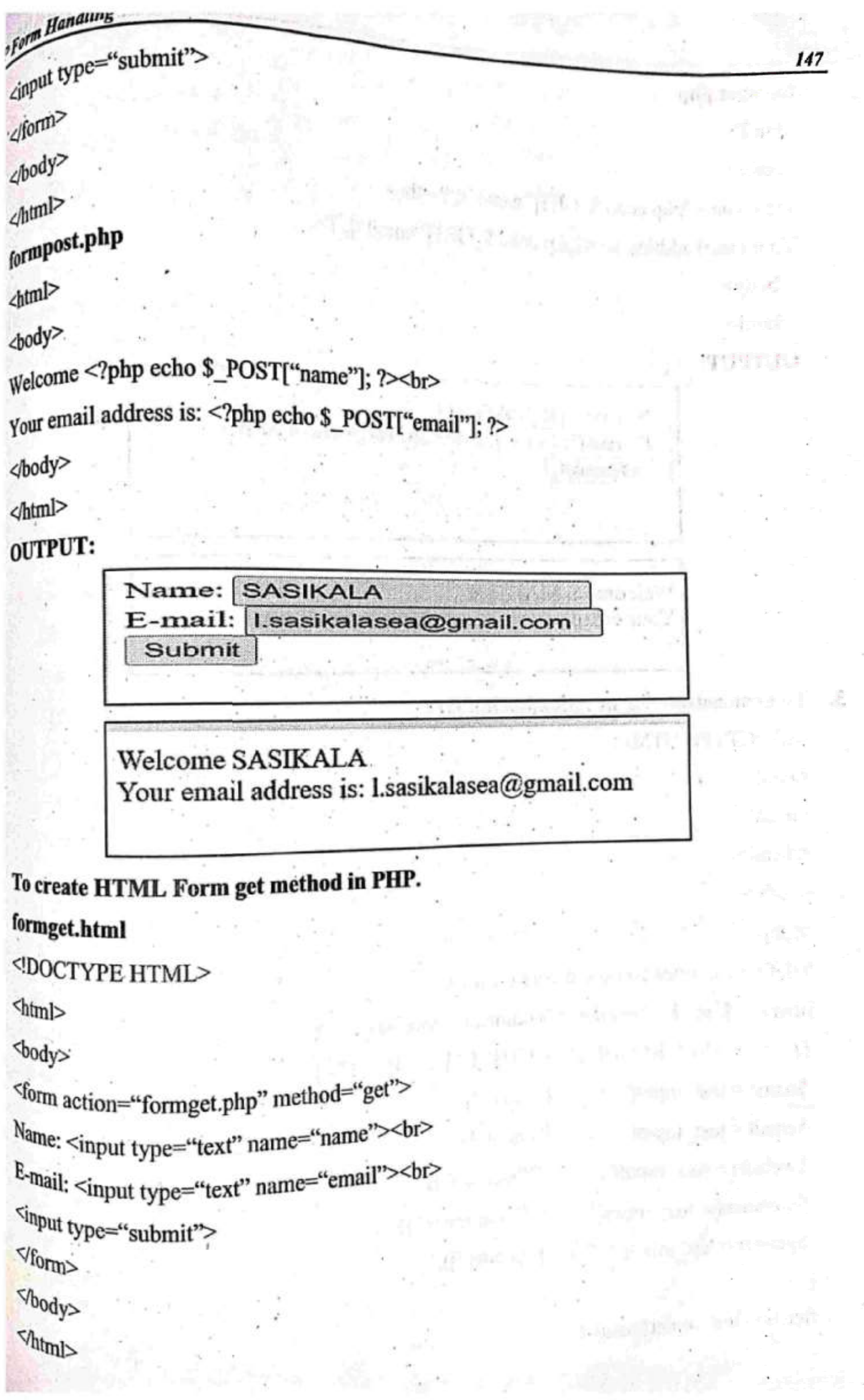
1. **Receiving Form Data:**
   - ❖ **Accessing Data:**
     - ➤ Use **$_GET** to access data from forms submitted using the GET method.
     - ➤ Use **$_POST** to access data from forms submitted using the POST method.
     - ➤ These superglobals are arrays containing key-value pairs (field names and their values).
2. **Processing Data:**
   - ❖ **Retrieving Values:**
     - ➤ $name = $_POST['name']; retrieves the value of the "name" field.
   - ❖ **Validation:**
   - • **Ensure data is valid and secure using techniques like:**
     - ➤ Type checking (e.g., is_numeric() for numbers).
     - ➤ Regular expressions (e.g., for email validation).
     - ➤ Sanitization to prevent attacks like XSS and SQL injection.
   - ❖ **Storing or Using Data:**
     - ➤ Save data to a database.
     - ➤ Send it via email.
     - ➤ Display it on a web page.
     - ➤ Use it for further processing.
3. **Providing Feedback:**
   - ➤ **Confirmation Messages:** Inform users about successful submission.
   - ➤ **Error Messages:** Indicate any errors or missing fields.
   - ➤ **Clear and Informative:** Guide users to correct any issues.

Examples:

1: To create HTML Form post method in PHP.

formpost.html

```
<!DOCTYPE HTML>
<html>
<body>
<form action="formpost.php" method="post">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
```



```
<input type="submit">
</form>
</body>
</html>
```

formpost.php

```
<html>
<body>
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
</body>
</html>
```

OUTPUT:

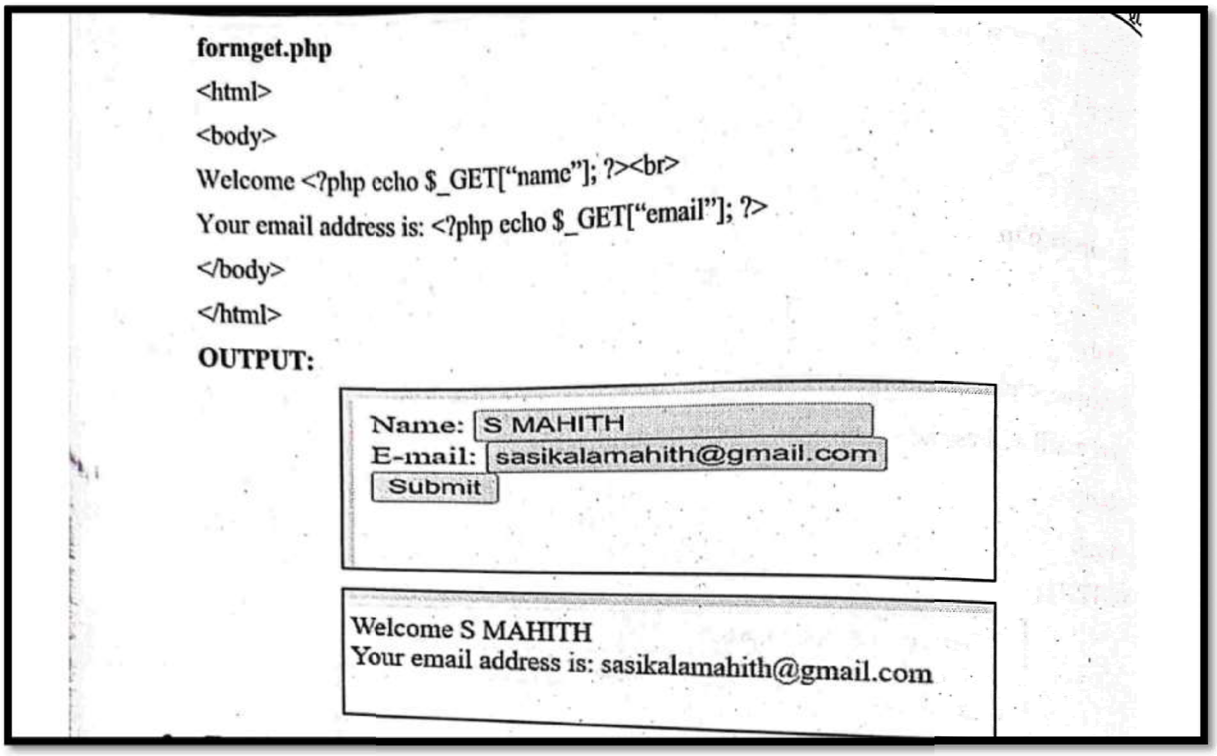
```
Name: SASIKALA
E-mail: l.sasikalasea@gmail.com
Submit
```

```
Welcome SASIKALA
Your email address is: l.sasikalasea@gmail.com
```

To create HTML Form get method in PHP.

formget.html

```
<!DOCTYPE HTML>
<html>
<body>
<form action="formget.php" method="get">
Name: <input type="text" name="name"><br>
E-mail: <input type="text" name="email"><br>
<input type="submit">
</form>
</body>
</html>
```



formget.php

```
<html>
<body>
Welcome <?php echo $_GET["name"]; ?><br>
Your email address is: <?php echo $_GET["email"]; ?>
</body>
</html>
```

OUTPUT:

```
Name: S MAHITH
E-mail: sasikalamahith@gmail.com
Submit
```

```
Welcome S MAHITH
Your email address is: sasikalamahith@gmail.com
```

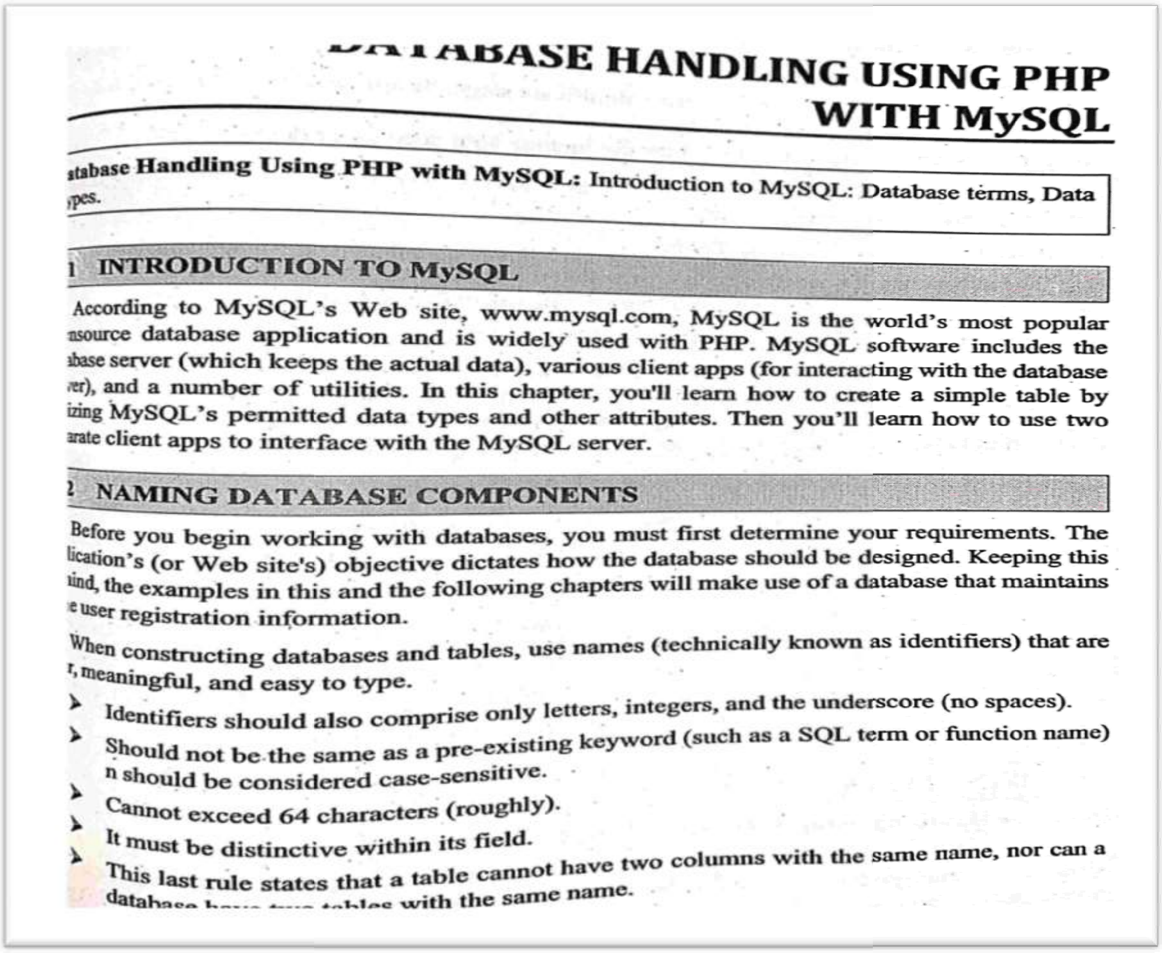
DATABASE HANDLING USING PHP WITH MYSQL: INTRODUCTION TO MYSQL

MySQL is one of the most popular relational database management systems (RDBMS) used in web development. It's open-source, reliable, and widely supported, making it a preferred choice for many developers and organizations. MySQL is used to store and manage structured data, allowing applications to interact with data efficiently.

Here's an introduction to some key concepts of MySQL:

1. **Database**: A database is a collection of related data organized in a structured format. MySQL allows you to create multiple databases, each containing one or more tables.
2. **Table**: A table is a collection of data organized into rows and columns. Each row represents a record, and each column represents a specific attribute of the record. Tables define the structure of the data stored in the database.
3. **Column**: A column (or field) is a vertical entity in a table that represents a specific attribute of the data being stored. Each column has a data type that defines the kind of data it can hold, such as text, number, date, etc.
4. **Row**: A row (or record) is a horizontal entity in a table that represents a single instance of data. Each row contains values corresponding to the columns defined in the table.

5. **Primary Key**: A primary key is a unique identifier for each row in a table. It ensures that each row can be uniquely identified and accessed. Primary keys are typically used to enforce data integrity and enable efficient data retrieval.
6. **SQL (Structured Query Language)**: SQL is a domain-specific language used to interact with databases. It provides a set of commands for creating, querying, updating, and managing databases and their objects (such as tables, indexes, etc.). MySQL supports SQL for performing various database operations.
7. **Query**: A query is a request for data or information from a database. It's written in SQL and can perform tasks such as retrieving data, inserting new records, updating existing records, and deleting records.
8. **Index**: An index is a data structure that improves the speed of data retrieval operations on a table. It's created on one or more columns of a table to quickly locate and access rows based on the indexed columns.



ATABASE HANDLING USING PHP WITH MySQL

tabase Handling Using PHP with MySQL: Introduction to MySQL: Database terms, Data ypes.

1 INTRODUCTION TO MySQL

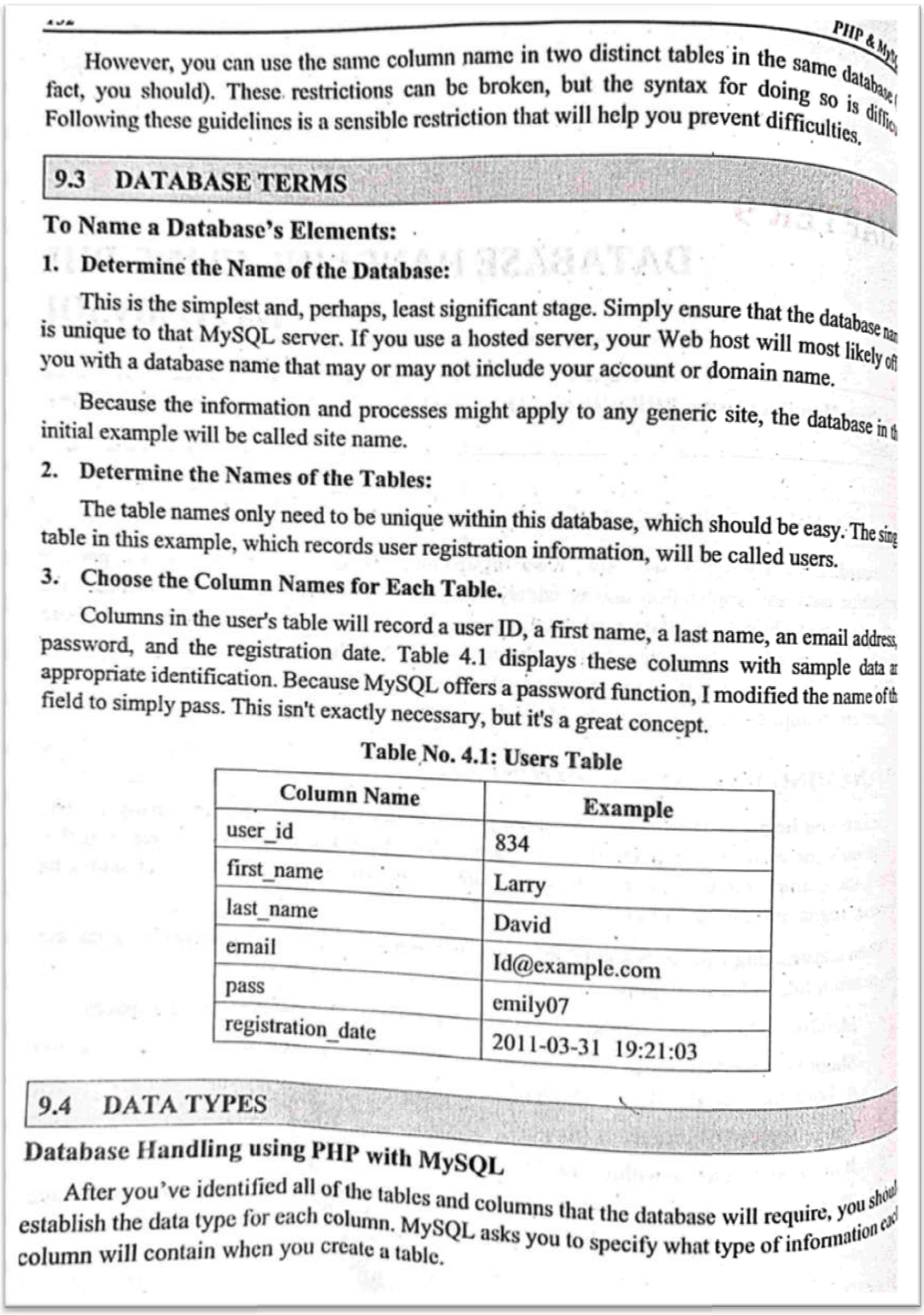
According to MySQL's Web site, www.mysql.com, MySQL is the world's most popular asource database application and is widely used with PHP. MySQL software includes the abase server (which keeps the actual data), various client apps (for interacting with the database ver), and a number of utilities. In this chapter, you'll learn how to create a simple table by izing MySQL's permitted data types and other attributes. Then you'll learn how to use two arate client apps to interface with the MySQL server.

2 NAMING DATABASE COMPONENTS

Before you begin working with databases, you must first determine your requirements. The lication's (or Web site's) objective dictates how the database should be designed. Keeping this ind, the examples in this and the following chapters will make use of a database that maintains e user registration information.

When constructing databases and tables, use names (technically known as identifiers) that are r, meaningful, and easy to type.

- Identifiers should also comprise only letters, integers, and the underscore (no spaces).
- Should not be the same as a pre-existing keyword (such as a SQL term or function name) n should be considered case-sensitive.
- Cannot exceed 64 characters (roughly).
- It must be distinctive within its field.
- This last rule states that a table cannot have two columns with the same name, nor can a database [illegible] with the same name.



However, you can use the same column name in two distinct tables in the same databa fact, you should). These restrictions can be broken, but the syntax for doing so is diffic Following these guidelines is a sensible restriction that will help you prevent difficulties.

9.3 DATABASE TERMS

To Name a Database's Elements:

1. **Determine the Name of the Database:**

This is the simplest and, perhaps, least significant stage. Simply ensure that the database na is unique to that MySQL server. If you use a hosted server, your Web host will most likely of you with a database name that may or may not include your account or domain name.

Because the information and processes might apply to any generic site, the database in th initial example will be called site name.

2. **Determine the Names of the Tables:**

The table names only need to be unique within this database, which should be easy. The sing table in this example, which records user registration information, will be called users.

3. **Choose the Column Names for Each Table.**

Columns in the user's table will record a user ID, a first name, a last name, an email address, password, and the registration date. Table 4.1 displays these columns with sample data a appropriate identification. Because MySQL offers a password function, I modified the name of th field to simply pass. This isn't exactly necessary, but it's a great concept.

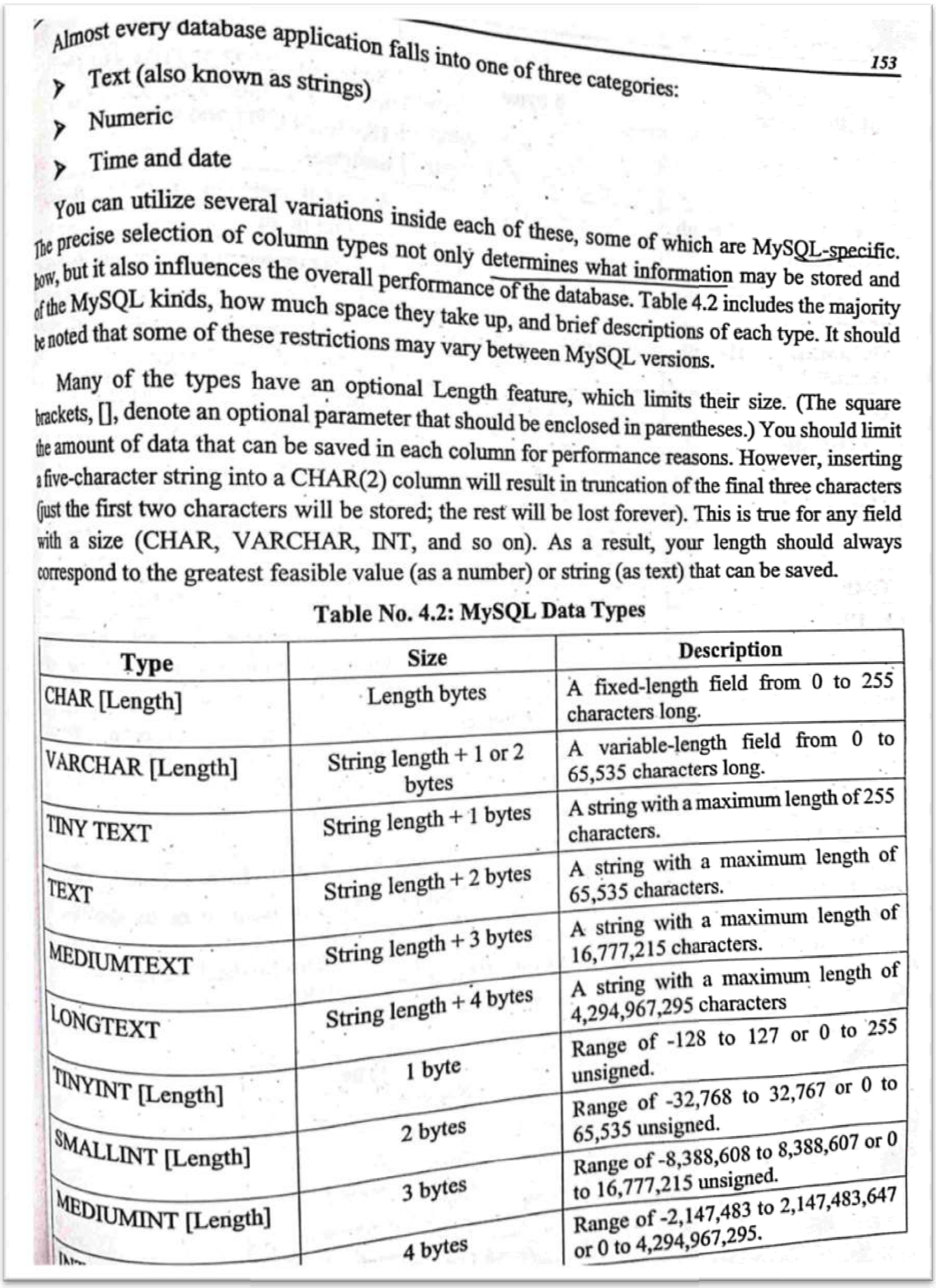
Table No. 4.1: Users Table

Column Name	Example
user_id	834
first_name	Larry
last_name	David
email	ld@example.com
pass	emily07
registration_date	2011-03-31 19:21:03

9.4 DATA TYPES

Database Handling using PHP with MySQL

After you've identified all of the tables and columns that the database will require, you shoul establish the data type for each column. MySQL asks you to specify what type of information eac column will contain when you create a table.



Almost every database application falls into one of three categories:

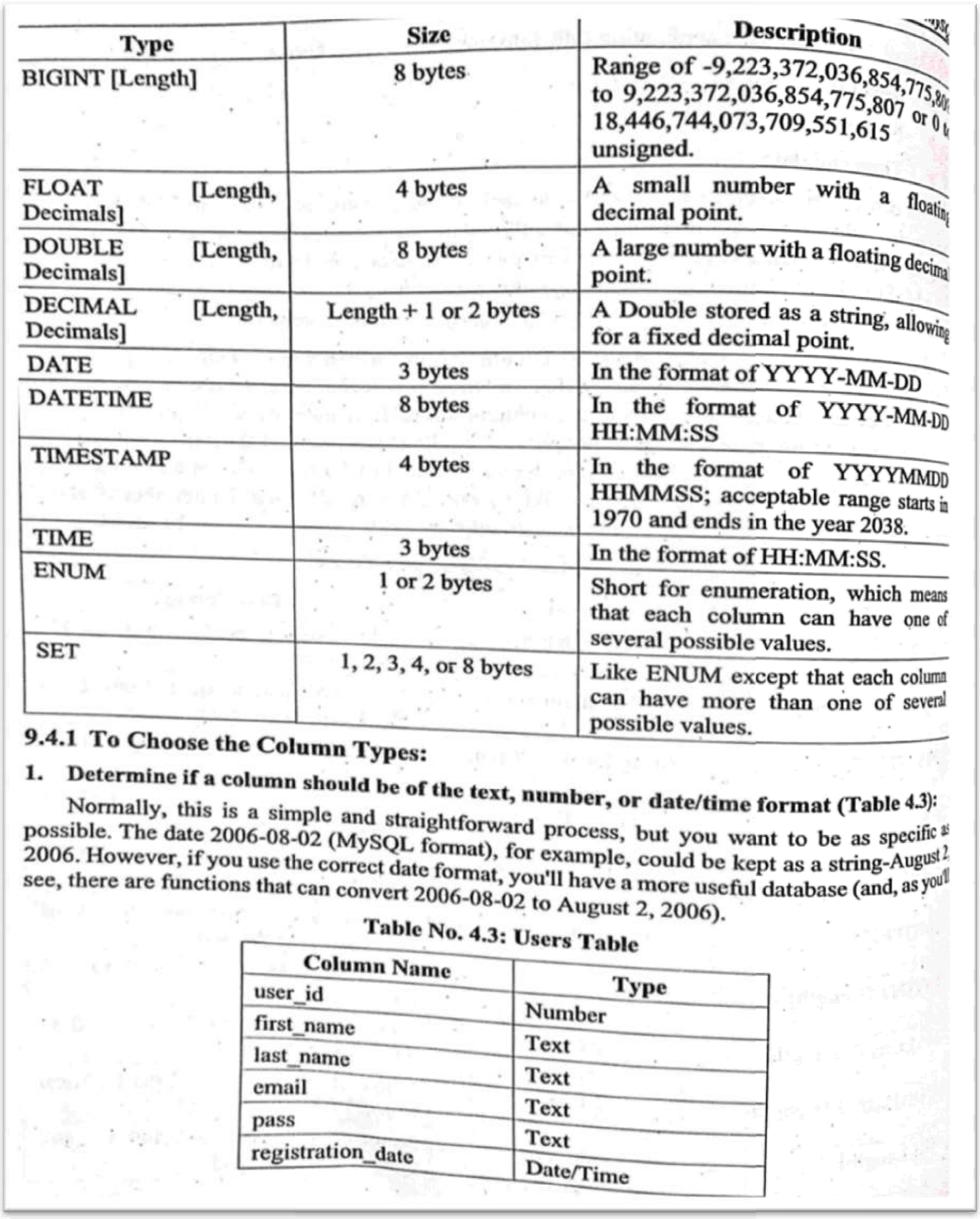
- Text (also known as strings)
- Numeric
- Time and date

You can utilize several variations inside each of these, some of which are MySQL-specific. The precise selection of column types not only determines what information may be stored and how, but it also influences the overall performance of the database. Table 4.2 includes the majority of the MySQL kinds, how much space they take up, and brief descriptions of each type. It should be noted that some of these restrictions may vary between MySQL versions.

Many of the types have an optional Length feature, which limits their size. (The square brackets, [], denote an optional parameter that should be enclosed in parentheses.) You should limit the amount of data that can be saved in each column for performance reasons. However, inserting a five-character string into a CHAR(2) column will result in truncation of the final three characters (just the first two characters will be stored; the rest will be lost forever). This is true for any field with a size (CHAR, VARCHAR, INT, and so on). As a result, your length should always correspond to the greatest feasible value (as a number) or string (as text) that can be saved.

Table No. 4.2: MySQL Data Types

Type	Size	Description
CHAR [Length]	Length bytes	A fixed-length field from 0 to 255 characters long.
VARCHAR [Length]	String length + 1 or 2 bytes	A variable-length field from 0 to 65,535 characters long.
TINY TEXT	String length + 1 bytes	A string with a maximum length of 255 characters.
TEXT	String length + 2 bytes	A string with a maximum length of 65,535 characters.
MEDIUMTEXT	String length + 3 bytes	A string with a maximum length of 16,777,215 characters.
LONGTEXT	String length + 4 bytes	A string with a maximum length of 4,294,967,295 characters
TINYINT [Length]	1 byte	Range of -128 to 127 or 0 to 255 unsigned.
SMALLINT [Length]	2 bytes	Range of -32,768 to 32,767 or 0 to 65,535 unsigned.
MEDIUMINT [Length]	3 bytes	Range of -8,388,608 to 8,388,607 or 0 to 16,777,215 unsigned.
IN	4 bytes	Range of -2,147,483 to 2,147,483,647 or 0 to 4,294,967,295.



Type	Size	Description
BIGINT [Length]	8 bytes	Range of -9,223,372,036,854,775,80 to 9,223,372,036,854,775,807 or 0 t 18,446,744,073,709,551,615 unsigned.
FLOAT [Length, Decimals]	4 bytes	A small number with a floatin decimal point.
DOUBLE [Length, Decimals]	8 bytes	A large number with a floating decima point.
DECIMAL [Length, Decimals]	Length + 1 or 2 bytes	A Double stored as a string, allowin for a fixed decimal point.
DATE	3 bytes	In the format of YYYY-MM-DD
DATETIME	8 bytes	In the format of YYYY-MM-DD HH:MM:SS
TIMESTAMP	4 bytes	In the format of YYYYMMDD HHMMSS; acceptable range starts in 1970 and ends in the year 2038.
TIME	3 bytes	In the format of HH:MM:SS.
ENUM	1 or 2 bytes	Short for enumeration, which means that each column can have one of several possible values.
SET	1, 2, 3, 4, or 8 bytes	Like ENUM except that each column can have more than one of several possible values.

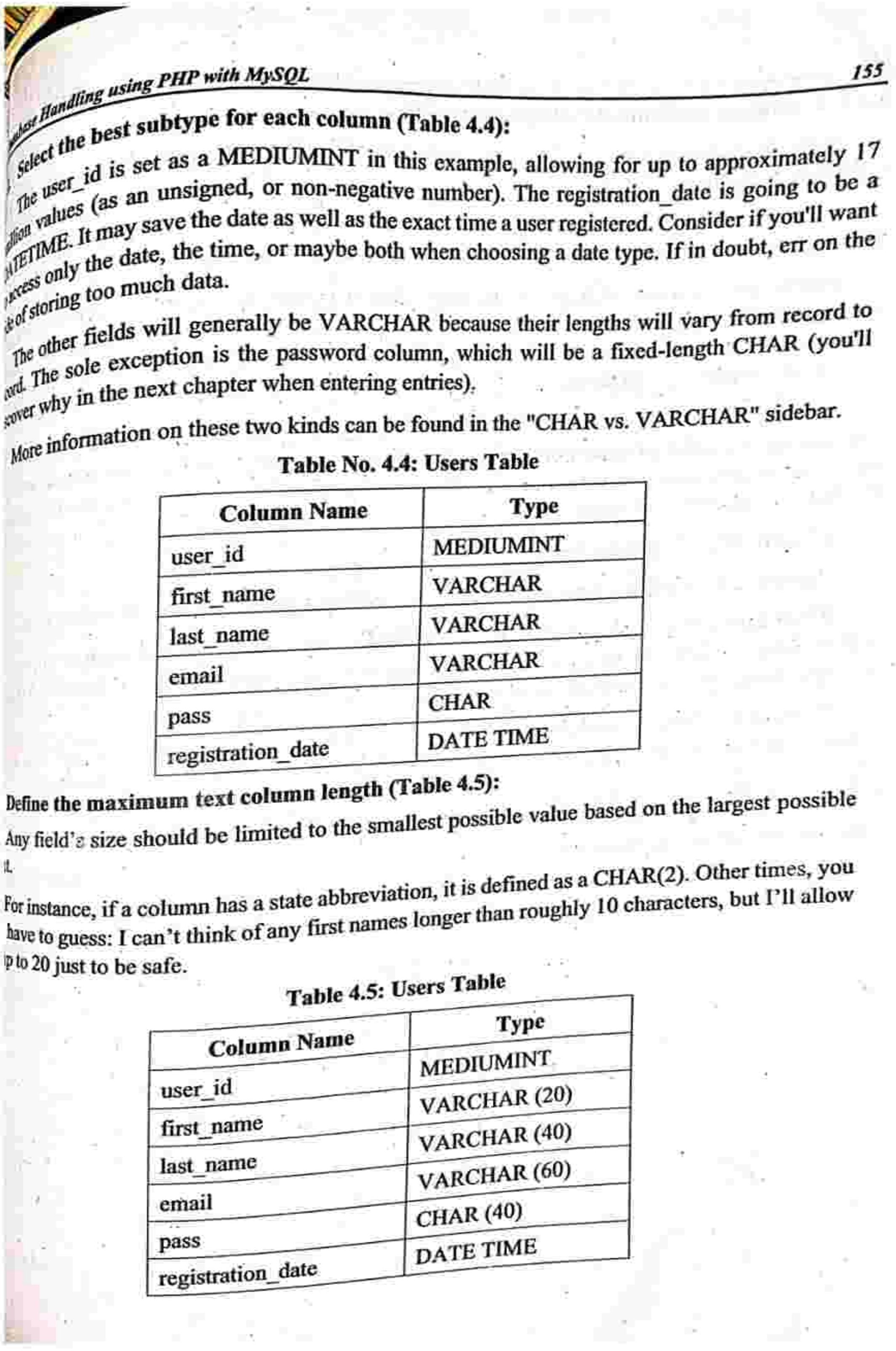
9.4.1 To Choose the Column Types:

1. Determine if a column should be of the text, number, or date/time format (Table 4.3):

Normally, this is a simple and straightforward process, but you want to be as specific as possible. The date 2006-08-02 (MySQL format), for example, could be kept as a string-August 2 2006. However, if you use the correct date format, you'll have a more useful database (and, as you'l see, there are functions that can convert 2006-08-02 to August 2, 2006).

Table No. 4.3: Users Table

Column Name	Type
user_id	Number
first_name	Text
last_name	Text
email	Text
pass	Text
registration_date	Date/Time



Select the best subtype for each column (Table 4.4):

The user_id is set as a MEDIUMINT in this example, allowing for up to approximately 17 illion values (as an unsigned, or non-negative number). The registration_date is going to be a ATETIME. It may save the date as well as the exact time a user registered. Consider if you'll want access only the date, the time, or maybe both when choosing a date type. If in doubt, err on the de of storing too much data.

The other fields will generally be VARCHAR because their lengths will vary from record to ord. The sole exception is the password column, which will be a fixed-length CHAR (you'll cover why in the next chapter when entering entries).

More information on these two kinds can be found in the "CHAR vs. VARCHAR" sidebar.

Table No. 4.4: Users Table

Column Name	Type
user_id	MEDIUMINT
first_name	VARCHAR
last_name	VARCHAR
email	VARCHAR
pass	CHAR
registration_date	DATE TIME

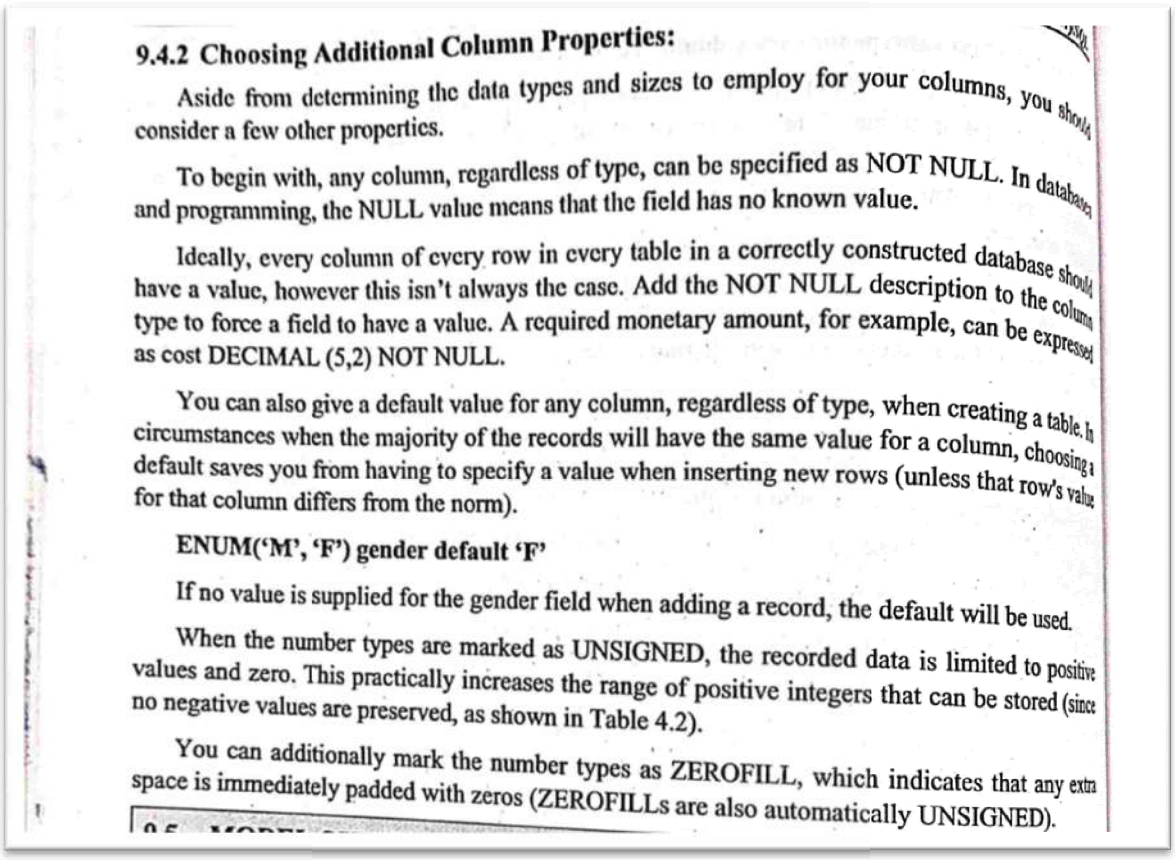
Define the maximum text column length (Table 4.5):

Any field's size should be limited to the smallest possible value based on the largest possible it.

For instance, if a column has a state abbreviation, it is defined as a CHAR(2). Other times, you have to guess: I can't think of any first names longer than roughly 10 characters, but I'll allow up to 20 just to be safe.

Table 4.5: Users Table

Column Name	Type
user_id	MEDIUMINT
first_name	VARCHAR (20)
last_name	VARCHAR (40)
email	VARCHAR (60)
pass	CHAR (40)
registration_date	DATE TIME



9.4.2 Choosing Additional Column Properties:

Aside from determining the data types and sizes to employ for your columns, you should consider a few other properties.

To begin with, any column, regardless of type, can be specified as NOT NULL. In databases and programming, the NULL value means that the field has no known value.

Ideally, every column of every row in every table in a correctly constructed database should have a value, however this isn't always the case. Add the NOT NULL description to the column type to force a field to have a value. A required monetary amount, for example, can be expressed as cost DECIMAL (5,2) NOT NULL.

You can also give a default value for any column, regardless of type, when creating a table. In circumstances when the majority of the records will have the same value for a column, choosing a default saves you from having to specify a value when inserting new rows (unless that row's value for that column differs from the norm).

ENUM('M', 'F') gender default 'F'

If no value is supplied for the gender field when adding a record, the default will be used.

When the number types are marked as UNSIGNED, the recorded data is limited to positive values and zero. This practically increases the range of positive integers that can be stored (since no negative values are preserved, as shown in Table 4.2).

You can additionally mark the number types as ZEROFILL, which indicates that any extra space is immediately padded with zeros (ZEROFILLs are also automatically UNSIGNED).