Unit -3: Advanced Server Side Scripting

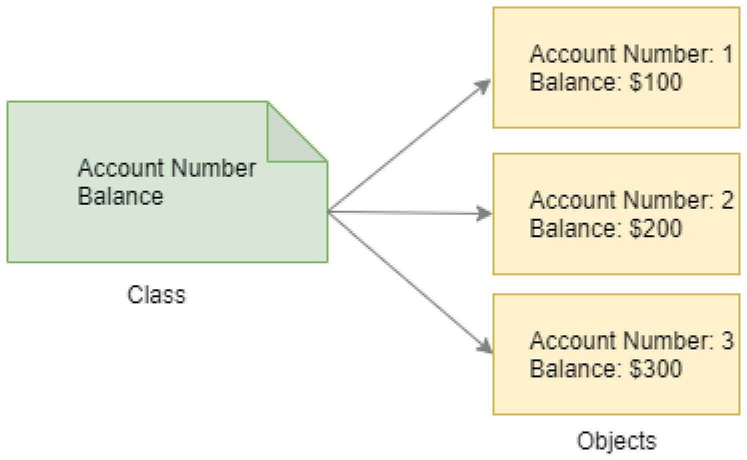
Object Oriented Programming in PHP

- Modern programming languages usually support or even require an object-oriented approach to software development. Object-oriented development attempts to use the classifications, relationships, and properties of the objects in the system to aid in program development and code reuse.
- PHP introduced object-oriented programming features since version 5.0. Object-Oriented programming is one of the most popular programming paradigms based on the concept of objects and classes.

Classes and Objects

- In the context of OO software, an object can be almost any item or concept—a physical object such as a desk or a customer; or a conceptual object that exists only in software, such as a text input area , a file, a bank account ,etc.
- An object has -
  a) **properties (or attributes)** that represent the characteristics of the object (e.g: account number and balance of a bank account)
  b) **behaviors (or operations)** that represent the functionalities and the actions related to the object (e.g: deposit and withdraw related with bank account)
- An object holds its state in variables that are often referred to as properties. An object also exposes its behavior via functions (or methods).
- A class is the blueprint of objects. A class defines how an object will be. Objects are created as per the class definitions. So, a class is a template for objects, and an object is an instance of a class.
- Class is considered as logical entity and object is considered as physical entity.
- In the real world, you can find many same kinds of objects. For example, a bank has many bank accounts. All of them have account numbers and balances. These bank accounts are

created from the same blueprint. In object-oriented terms, we say that an individual bank account is an instance of a Bank Account class.



- When the individual objects are created, they have all the properties and behaviors from the class, but each object will have different values for the properties.

[Recall Buildings and Map example discussed in class]

Define a Class in PHP

A class is defined by using the class keyword, followed by the name of the class and a pair of curly braces ({}). All its properties and methods go inside the braces:

```
class ClassName
{
    //members of a class
  // properties and methods
}
```

```
<?php

class BankAccount{ //class

// properties

      public $accountNumber;

      public $balance;

// methods

      public function deposit($amount){

            if ($amount > 0) {

                  $this->balance += $amount;

            }

      }

      public function withdraw($amount){

            if ($amount <= $this->balance) {

                  $this->balance -= $amount;

                  return true;

            }

                 return false;

      }

      Public function display(){

            echo "Account Number = ".$accountNumber."<br>

                   Balance =   ".$accountBalance";
```

```
        }
    }
?>
```

Define Objects in PHP

- Classes are nothing without objects! We can create multiple objects from a class. Each object has all the properties and methods defined in the class, but they will have different property values.
- Objects of a class are created using the **new** keyword.
  Example:

```
$account1  = new BankAccount;

$account2  = new BankAccount();
```

- In this exxample, the $account1 is a variable that references the object created by the BankAccount class.

Accessing the members of an Object

- Arrow operator (`->`) is used to access the members of an object
  Example:

```
$account1->accountNumber = 100234;
$account1->accountBalance = 100;
$account1->deposit(500);
$account1->display();
```

NOTE: **$this** holds the reference of the current object and is used to refer current object. The $this keyword allows you to access the properties and methods of the current object within the class using the object operator (`->`).The $this keyword is only available within a class. It doesn't exist outside of the class. If you attempt to use the $this outside of a class, you'll get an error.

PHP Access Modifiers

- Properties and methods can have access modifiers which control where they can be accessed.
- Access modifiers are keywords used to define the visibility or accessibility of properties, methods, and constants within a class. PHP provides three access modifiers: public, protected, and private. These modifiers help enforce encapsulation and control the accessibility of class members from outside the class.
- The three access modifiers:

  public - the property or method can be accessed from everywhere. This is default

  protected - the property or method can be accessed within the class and by classes derived from that class

  private - the property or method can only be accessed within the class

Constructor in PHP

- A constructor is a special method of a class which is used to initialize the object during creation.
- A constructor allows to initialize an object's properties upon creation of the object.
- PHP allows us to declare a constructor method for a class with the name **__construct()**. Notice that the construct function starts with two underscores (__)
- The constructor method is used to initialize the object's properties or perform any necessary setup tasks.
- The constructor does not have a return type, not even void.
- We can define parameters within the constructor to accept arguments during object creation. This allows us to pass initial values to the object's properties (which can be different for different objects)
- When we create an instance of a class, PHP automatically calls the constructor method

- Constructors are useful for performing setup tasks, initializing properties, establishing connections to databases or external services, and any other necessary initialization logic when creating an object.

Types of Constructor

In PHP, there are two types of constructors:

Default Constructor

- If we don't define any constructor in a class, PHP automatically provides a default constructor with no arguments.
- A default constructor does not accept any arguments. So, it is parameter less or no-argument constructor.
- We can define a constructor within a class without parameter it is also considered as default constructor.

```
class ClassName{
   function __construct(){
         // default constructor implementation
   }
}
//Example
class BankAccount{
   private $accountNumber;
   private $balance;
   public function __construct(){
        $this->balance = 1200;
    }
}
```

Parameterized Constructor

- A parameterized constructor is a constructor that accepts one or more arguments.
- It allows us to initialize the object's properties with specific values provided during object creation.
- We need to define the parameter list for the constructor and pass the corresponding arguments when creating an object.

```
class ClassName{
     function __construct($param_1, $param_2,...,$param_n ){
          // constructor implementation
     }
}
```

Example:

```
class Person {
    private $name;
    private $age;
    //parameterized constructor
    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }
```

```
        public function getInfo() {
            return "Name: " . $this->name . ", Age: " . $this->age;
        }
    }


    // Creating an object and passing arguments to the constructor
    $person = new Person("John Doe", 25);
    echo $person->getInfo(); // Output: Name: John Doe, Age: 25
```

NOTES:

- You can define any number of parameters in a constructor based on your class's requirements.
- Note that in PHP, unlike some other programming languages, you cannot explicitly define multiple constructors with different parameter lists. However, you can use default values for parameters to achieve similar functionality.
- If you need to have different initialization behavior based on the presence or absence of arguments, you can achieve it by using default parameter values in the constructor:

```
class MyClass {
    public function __construct($param1 = null, $param2 = null) {
        if ($param1 !== null && $param2 !== null) {
            // Parameterized constructor behavior
        } else {
            // Default constructor behavior
        }
        }
    }
```

Destructor in PHP

- A destructor is a special method within a class that is automatically called when an object is no longer referenced or when its script execution ends. The destructor method is used to perform cleanup tasks, such as releasing resources, closing database connections, or freeing up memory.
- The destructor is automatically invoked before an object is deleted. It happens when the object has no reference or when the script ends.
- The destructor method is defined using the __destruct() function name, and it does not accept any arguments. Notice that the destruct function starts with two underscores (__)!

```
class className{
      public function __destruct(){
            //destructor code
      }
}
```

- The destructor method is automatically called by PHP when the object is no longer referenced or when the script execution ends. You don't need to explicitly call the destructor.
- The destructor does not have a return type, not even void.
- You cannot define parameters within the destructor. It does not accept any arguments.
- A class can have only one destructor method.
- Destructors are useful for performing cleanup tasks and releasing resources that an object may have acquired during its lifetime. It's important to note that PHP automatically takes care of calling destructors, so you don't have to explicitly invoke them.

Example

```php
<?php
class Fruit {
  public $name;
  public $color;

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

  //destruct
  function __destruct() {
    echo "The fruit is {$this->name}.";
  }
}
$apple = new Fruit("Apple");
?>
```

Inheritance in PHP

- Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class to inherit properties and methods from another class.
- The class that is being inherited from is called the **parent class, base class,** or **superclass,** while the class that inherits from it is called the **child class, derived class,** or **subclass.**
- Inheritance allows a class to reuse the code from another class without duplicating it.
- To define a class inherits from another class, you use '**extends** ' keyword. The child class inherits all the public and protected properties and methods from the parent class. It can also override or extend the behavior of the parent class by adding its own properties and methods.

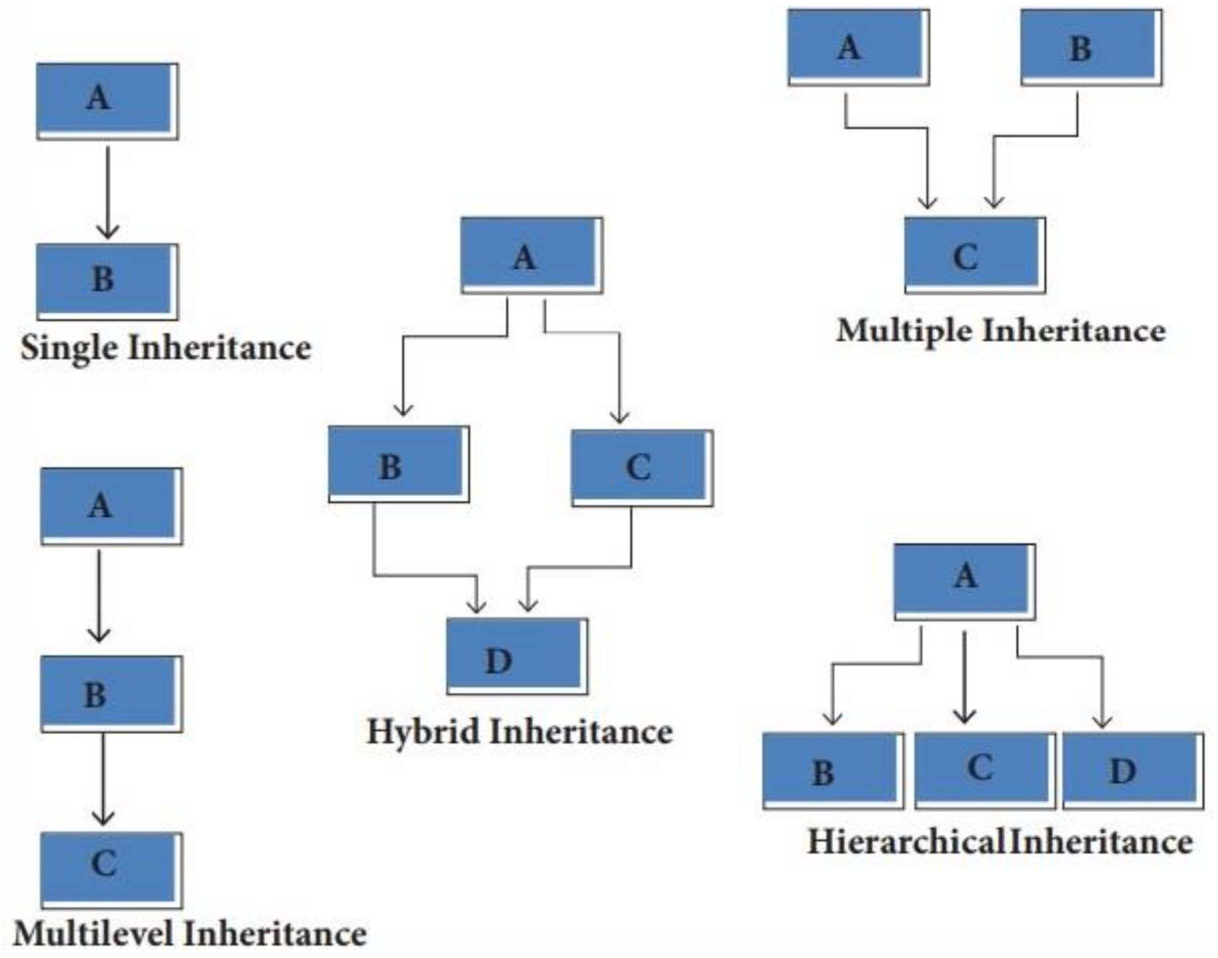
Syntax

```
class Sub extends Super{

     //child class members

}

class Animal {

    public function eat() {

        echo "Animal is eating.";

    }

}


class Cat extends Animal {

    public function meow() {

        echo "Cat is meowing.";

    }

}


// Creating an object of the child class

$cat = new Cat();


// Accessing inherited method from the parent class

$cat->eat();  // Output: Animal is eating.
```

```
// Accessing method specific to the child class

$cat->meow(); // Output: Cat is meowing.
```

- Inheritance allows for code reuse, promoting modularity, and enabling hierarchical relationships between classes. It facilitates the creation of specialized classes that inherit common behaviors and characteristics from a shared parent class.

Types of inheritance



NOTE: PHP does not allow multiple inheritance directly.

Calling constructor of parent class from child class

- In PHP, We can access the constructor of the parent class from the child class using the parent keyword. The parent keyword allows us to refer to the parent class and its members, including the constructor.
- To access the constructor of the parent class in the child class, we can use the **parent::__construct()** syntax within the child class's constructor. This calls the constructor of the parent class and allows you to perform any additional initialization specific to the child class.
- Here's an example that demonstrates how to access the parent class constructor in PHP:

```
class ParentClass {
    protected $name;

    public function __construct($name) {
        $this->name = $name;
        echo "Parent constructor called. Name: " . $this->name . "<br>";
    }
}

class ChildClass extends ParentClass {
    public function __construct($name) {
        parent::__construct($name);
        echo "Child constructor called. Name: " . $this->name . "<br>";
    }
}

// Creating an object of the child class
$child = new ChildClass("John");
```

- By using the **parent::__construct()** syntax, you can access and invoke the constructor of the parent class within the child class, allowing you to perform any necessary initialization and utilize the functionality provided by the parent constructor.

Method overriding in PHP

- Method overriding is a feature of object-oriented programming that allows a child class to provide its own implementation of a method defined in its parent class.
- Inherited methods can be overridden in child class by redefining the methods defined in the parent class.

- Method overriding allows a child class to provide a specific implementation of a method already provided by its parent class.
- To override a method, you redefine that method in the child class with the same name, parameters, and return type.
- The method in the parent class is called overridden method, while the method in the child class is known as the overriding method. The code in the overriding method overrides (or replaces) the code in the overridden method.
- PHP will decide which method (overridden or overriding method) to call based on the object used to invoke the method.
- If an object of the parent class invokes the method, PHP will execute the overridden method. But if an object of the child class invokes the method, PHP will execute the overriding method.

  Example

```
class Animal {
    public function makeSound() {
        echo "Animal is making a sound.";
    }
}

class Cat extends Animal {
    public function makeSound() {
        echo "Cat is meowing.";
    }
}

// Creating objects
$animal = new Animal();
$cat = new Cat();

// Calling the makeSound() method on objects
$animal->makeSound();  // Output: Animal is making a sound.
```

```
$cat->makeSound();     // Output: Cat is meowing.
```

- Method overriding allows child classes to specialize and modify the behavior of inherited methods from the parent class. It provides flexibility in implementing polymorphism, where objects of different classes can respond to the same method name differently based on their specific implementations.
- It's worth noting that method overriding only occurs when the child class declares a method with the same name as the one in the parent class. If a method with the same name doesn't exist in the child class, the parent class method will be used when calling the method on objects of the child class.

NOTE:

To call the overridden method within the overriding method in PHP, we can use the parent keyword followed by scope resolution operator (::) and the method name. The parent keyword allows you to refer to the parent class and its members, including the overridden method.

```
public function makeSound() {
       parent::makeSound(); // Calling the overridden method in the parent class
        echo " Meow!";
    }
```

By using the **parent::methodName() syntax**, you can explicitly call the overridden method from the parent class within the overriding method of the child class. This allows you to extend the functionality of the parent class's method while incorporating its original beha Encapsulation is an important concept in object-oriented programming (OOP), including PHP. It refers to the bundling of data and methods within a class, and controlling access to that data from outside the class. Encapsulation helps in organizing code, promoting data integrity, and providing abstraction.

NOTE: The **final** keyword can be used to prevent class inheritance or to prevent method overriding.

Encapsulation in PHP

- Wrapping up data member and method together into a single unit is called encapsulation.
- By using access modifiers, we can control the visibility and accessibility of properties and methods in a class, which is crucial for encapsulation. The idea is to make the internal workings of a class hidden from the outside, and only allow controlled access through designated methods.
- Additionally, getter and setter methods, also known as accessors and mutators, are used to provide controlled access to class properties. Getter methods retrieve the value of a property, while setter methods set the value of a property. This allows for encapsulation of data, as the properties themselves are kept private or protected, and access is provided through the getter/setter methods.

Here's an example to illustrate encapsulation in PHP:

```
class Employee {

    private $name;

    private $salary;

    public function getName() {

        return $this->name;

    }

    public function setName($name) {

        $this->name = $name;

    }
    public function getSalary() {

        return $this->salary;

    }

    public function setSalary($salary) {

        if ($salary >= 0) {

            $this->salary = $salary;

        }

    }

}

// Creating an object of the class

$employee = new Employee();

// Setting property values using setter methods

$employee->setName("John Doe");

$employee->setSalary(5000);

// Accessing property values using getter methods

echo $employee->getName();    // Output: John Doe

echo $employee->getSalary();  // Output: 5000
```

- In the above code, the "Employee" class encapsulates the "name" and "salary" properties. They are kept private, preventing direct access from outside the class. Access to these properties is provided through the getter and setter methods, which allow controlled manipulation of the data.

- By encapsulating the properties and controlling access through methods, you can enforce data integrity, hide implementation details, and provide a well-defined interface for interacting with the class, promoting better code organization and maintainability.

Polymorphism in PHP

- Polymorphism is derived from two Greek words. Poly (meaning many) and morph (meaning forms). In general, polymorphism means the ability to have many forms.
- Polymorphism is one of the important features of OOP.
- Polymorphism is of two types: Compile-time polymorphism and Run-time polymorphism. PHP doesn't support compile-time polymorphism which means constructor overloading, function overloading and operator overloading are not supported in PHP.
- Runtime polymorphism, also known as dynamic polymorphism or late binding, is a feature of object-oriented programming that allows the selection of a specific method implementation at runtime based on the actual type of the object. It enables different objects to respond differently to the same method call based on their specific implementations.
  - ☞ Runtime polymorphism is closely related to inheritance and method overriding.
- In PHP, runtime polymorphism is achieved through method overriding and inheritance. When a child class overrides a method defined in its parent class, the overridden method in the child class is dynamically selected and executed at runtime based on the type of the object.
- Runtime polymorphism allows for code flexibility and extensibility. It allows you to write code that can work with different types of objects as long as they adhere to a common interface or inherit from a common parent class. This feature enables you to write more generic code that can handle varying object behaviors at runtime, enhancing the versatility and adaptability of your programs.

- Runtime Polymorphism in PHP can be implemented by using of **interfaces** and **abstract classes**

Polymorphism in PHP by using an abstract class

Abstract methods and abstract class:

- A method declared using the abstract keyword and does not have a definition (implementation) is called an **abstract method.**
  - ☞ Methods defined as abstract simply declare the method's signature; they cannot define the implementation.
- A class is declared using the abstract keyword and has zero or more abstract method is called **abstract class**.
  - ▪ Classes defined as abstract cannot be instantiated, and any class that contains at least one abstract method must also be abstract.
  - ▪ An abstract class can contain both abstract methods and regular methods.
  - ▪ Abstract class is used as base class.
  - ▪ Objects cannot be created directly from abstract class.
  - ▪ Child class is derived from abstract class implementing all the abstract methods in parent class and the object of the child class can be created.
  - ▪ When inheriting from an abstract class, all methods marked abstract in the parent's class declaration must be defined by the child class, and follow the usual inheritance and signature compatibility rules.
  - ▪ When a child class is inherited from an abstract class, we have the following rules:
    - ▪ The child class method must be defined with the same name and it re-declares the parent abstract method
    - ▪ The child class method must be defined with the same or a less restricted access modifier
    - ▪ The number of required arguments must be the same. However, the child class may have optional arguments in addition

Example:

```
abstract class Shape {
    abstract public function calculateArea();
}

class Circle extends Shape {
    private $radius;

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

    public function calculateArea() {
        return 3.14 * $this->radius * $this->radius;
    }
}

class Rectangle extends Shape {
    private $length;
    private $width;

    public function __construct($length, $width) {
        $this->length = $length;
        $this->width = $width;
    }

    public function calculateArea() {
        return $this->length * $this->width;
    }
}

// Create an array of shapes
$shapes = [
    new Circle(5),
    new Rectangle(4, 6)
];

// Calculate and display the areas of the shapes
foreach ($shapes as $shape) {
    echo "Area: " . $shape->calculateArea() . "<br>";
}
```

In this example, the Shape abstract class provides a common interface for calculating the area of various shapes, and the subclasses Circle and Rectangle implement the specific logic for their respective shapes. By leveraging the abstract class, we achieve polymorphism by treating different objects as instances of a shared type and invoking their specific implementations of the abstract method.

Polymorphism in PHP by using interface

- Interface is defined as blueprint of classes.
- It defines a set of method signatures without providing any implementation details. It means interface contains only abstract methods.
- An interface serves as a contract that specifies the methods that implementing classes must implement. It establishes a set of rules and requirements for classes that want to adhere to the interface.
- An interface can contain constants. Constants in php are declared using const keyword.
- All methods in interfaces are implicitly public and abstract.
- Interfaces are declared with the interface keyword
  - ☞ Interfaces are defined in the same way as a class, but with the interface keyword replacing the class keyword and without any of the methods having their contents defined.
- To implement an interface, a class must use the implements keyword (operator).
- A class that implements an interface must implement all of the interface's methods (if not ,the class should be declared abstract)
- Classes may implement more than one interface if desired by separating each interface with a comma.
- In practice, interfaces serve two complementary purposes:
  - ❖ To allow developers to create objects of different classes that may be used interchangeably because they implement the same interface or interfaces.
  - ❖ To allow a function or method to accept and operate on a parameter that conforms to an interface, while not caring what else the object may do or how it is implemented.

```
interface Animal {

    public function makeSound();

}

class Cat implements Animal {

    public function makeSound() {

        echo "Cat is meowing.";

    }

}

class Dog implements Animal {

    public function makeSound() {

        echo "Dog is barking.";

    }

}

// Function that accepts any object implementing the Animal
interface

function animalSound(Animal $animal) {

    $animal->makeSound();

}






// Create objects of different classes implementing the Animal
interface

$cat = new Cat();
```

```
$dog = new Dog();

// Call the function with different objects

animalSound($cat); // Output: Cat is meowing.

animalSound($dog); // Output: Dog is barking.
```

In the above code, we have an Animal interface that declares a single method makeSound(). The Cat and Dog classes implement the Animal interface and provide their own implementations of the makeSound() method.

We also have a function animalSound() that accepts an object of type Animal. This function can accept any object that implements the Animal interface, regardless of the specific class.

By creating objects of the Cat and Dog classes and passing them to the animalSound() function, we achieve polymorphism. The function treats the objects as instances of the Animal interface, allowing them to be used interchangeably. The appropriate implementation of the makeSound() method is called based on the actual type of the object, resulting in different sounds being output.

By programming to the interface rather than the specific classes, we can write more flexible and reusable code that can work with different objects that implement the interface.

NOTE: Interfaces can be extended like classes using the extends operator.

Static Members in PHP

- In PHP, static members refer to properties and methods that are associated with a class rather than with instances of that class. These members are shared among all instances of the class and can be accessed directly using the class name without needing to create an object.
- PHP allows you to access the methods and properties in the context of a class rather than an object. Such methods and properties are class methods and properties. Class methods and class properties are called **static methods** and **static properties.**

Static methods

- To define a static method, we place the static keyword in front of the function keyword as follows:

```
class MyClass{

      //static method

      public static function staticMethod(){

      // method body

      }

}
```

- Since a static method is bound to a class, not an individual instance of the class, you cannot access **$this** inside the method. However, you can access a special variable called self. The **self variable** means the current class.
- Static methods are associated with the class itself rather than with instances of the class. They can be called without creating an object of the class. Static methods are declared using the static keyword before the method name.
- The following shows how to call a static method from the inside of the class:

  self::staticMethod(arguments);
- To call a static method from the outside of the class, you use the following syntax:

```
className::staticMethod(arguments)

class MathUtils {

    public static function add($num1, $num2) {

        return $num1 + $num2;

    }

}

echo MathUtils::add(5, 3);  // Output: 8
```

In the example above, the add method is declared as static. It can be called directly using the class name (MathUtils::add()) without creating an object. Static methods are useful when you need to perform operations that are not dependent on any specific instance of the class.

Static properties (variables)

- A static property is a variable that is shared among all instances of a class. It is declared using the keyword static before the property name.

  public static $staticProperty;

Example: class MyClass {

```
    public static $count = 0;

    public function __construct() {

        self::$count++;

    }

}
```

```
$obj1 = new MyClass();

echo MyClass::$count;  // Output: 1

$obj2 = new MyClass();

echo MyClass::$count;  // Output: 2
```

In the example above, the $count property is declared as static. It keeps track of the number of instances created from the MyClass class. Each time a new instance is created, the constructor increments the $count property using the self::$count syntax, where self refers to the class itself.

☑ Static members provide a way to define properties and methods that are shared among all instances of a class or are relevant to the class itself rather than individual objects. They are commonly used for utility functions, counters, configuration settings, and other scenarios where you don't need instance-specific behavior.

Rules to work with static members in PHP

When working with static members in PHP, there are some rules and considerations to keep in mind:

☑ **Declaration:** Static members, both properties, and methods, are declared using the static keyword before their respective declarations.

☑ **Access:** Static members can be accessed using the class name followed by :: (scope resolution operator) without the need to create an instance of the class.

☑ **Instance Independence:** Static members are shared among all instances of the class. Changes made to a static property or by calling a static method will affect all instances and will persist across different instances.

☑ **Instance References:** Static members cannot directly access non-static (instance) members or methods. They can only access other static members or methods.

☑ **Initialization:** Static properties can be initialized directly at the point of declaration or within a static initialization methods.

☑ **Inheritance:** Static members can be inherited by child classes. If a child class re-declares a static member with the same name, it will override the parent class's static member.

Exception Handling in PHP

- An exception is an unexpected outcome of a program, which can be handled by the program itself. Basically, an exception disrupts the normal flow of the program. But it is different from an error because an exception can be handled, whereas an error cannot be handled by the program itself.
    - ☞ An unexpected result of a program is an exception, which can be handled by the program itself. Exceptions can be thrown and caught in PHP
    - ☞ Technically speaking, an exception is an instance of the ***Exception*** class, which implements the ***Throwable*** interface.
- Exception handling is a powerful mechanism of PHP, which is used to handle runtime errors (runtime errors are called exceptions). So that the normal flow of the program can be maintained.
- The main purpose of using exception handling is to maintain the normal execution of the application.
    - ☞ Instead of halting the script, you can handle the exceptions gracefully. This is known exception handling.
- ▪ Exception handling is used to change the normal flow of the code execution if a specified error condition occurs. This condition is called an exception.

When an exception is triggered, following things happen:

- The current code state is saved
- The code execution will switch to a predefined (or custom) exception handler function

- Depending on the situation, the handler may then resume the execution from the saved code state, terminate the script execution or continue the script from a different location in the code

Keywords used in Exception Handling in PHP

- **try, catch ,finally** and **throw** are the important keywords used in PHP.
  - The **try** block is used to enclose the code that may throw an exception.
  - The **catch** block is used to catch and handle exceptions that are thrown within the try block.
  - The **finally** block is optional and is used to specify code that should be executed regardless of whether an exception was thrown or caught. It is commonly used for cleanup operations.
  - The **throw** statement is used to explicitly throw an exception. It allows you to specify the exception object that represents the error or exceptional condition.

```
try {

    // Code that may throw an exception

} catch (ExceptionType $e) {

    // Code to handle the exception

}

finally {

    // Code that will always be executed

}
```

try

- The try block contains the code that may have an exception or where an exception can arise.
- When an exception occurs inside the try block during runtime of code, it is caught and resolved in catch block.
- The try block must be followed by catch or finally block.
- A try block can be followed by minimum one and maximum any number of catch blocks.

catch

- The catch block contains the code that executes when a specified exception is thrown.
- It is always used with a try block, not alone.
- When an exception occurs, PHP finds the matching catch block.
- This block is not executed if matching exception is not thrown and caught by it.

throw

- It is a keyword used to throw an exception.
- It also helps to list all the exceptions that a function throws but does not handle itself.
- Remember that each throw must have at least one "catch".

finally

- The finally block contains a code, which is used for clean-up activity in PHP.
- Basically, it executes the essential code of the program.
- This block is always executed.

- ❖ When an exception is thrown, the code following it will not be executed, and PHP will try to find the matching "catch" block.
- ❖ If an exception is not caught, a fatal error will be issued with an "Uncaught Exception" message.

Note: Exceptions should only be used with error conditions, and should not be used to jump to another place in the code at a specified point.

Predefined Exceptions in PHP

- ❖ PHP provides several built-in exception classes that you can use for specific types of errors or exceptional conditions. These classes are part of the PHP Standard Library and can be used directly or extended to create custom exceptions.
- ❖ Here are some of the commonly used built-in exception classes in PHP:
    - **Exception:** This is the base class for all exceptions in PHP. It represents a generic exception and can be used when no specific exception class is appropriate.
    - **ErrorException:** This exception is thrown when a PHP error occurs. It is generated by the error handling system when a PHP error is converted to an exception. It extends the Exception class.
    - **InvalidArgumentException:** This exception is thrown when an invalid argument is passed to a function or method. It indicates that the value or type of the argument is not as expected.
    - **RuntimeException:** This exception is a generic exception class for runtime errors. It is usually used for exceptional conditions that occur during the execution of a script.
    - **LogicException:** This exception is a base class for exceptions that indicate logical errors in the code. It is typically used when there is a problem with the code's logic or flow.
    - **OutOfBoundsException:** This exception is thrown when an index or offset is outside the valid range. It is commonly used with arrays or other data structures.

- **FileNotFoundException:** This exception is thrown when a file cannot be found or accessed.
- **PDOException:** This exception is specific to database-related errors and is thrown by the PHP Data Objects (PDO) extension when there is an error in database operations.
- **DivisionByZeroError:** This exception is thrown when a division or modulus operation is performed with a divisor of zero.

Some methods of Exception class in PHP

The Exception class in PHP provides several methods that can be used to work with exceptions and retrieve information about the exception that was thrown. Here is a list of commonly used methods of the Exception class along with their descriptions:

getMessage(): Returns the error message associated with the exception.

getCode(): Returns the code associated with the exception. This code can be used to identify different types of exceptions or error conditions.

getFile(): Returns the name of the file in which the exception was thrown.

getLine(): Returns the line number in the file at which the exception was thrown.

getTrace(): Returns an array of the function or method calls that led to the throwing of the exception.

getTraceAsString(): Returns a string representation of the exception's stack trace.

getPrevious(): Returns the previous exception in the chain, if any. This method is useful when exceptions are chained together.

getCode(): Returns the exception code.

Example1: Exception handling

```
<?php
function divide($dividend, $divisor) {
  if($divisor == 0) {
    throw new Exception("Division by zero");
   }
  $r = $dividend/$divisor;
  echo "Result = ".$r."<br>";
}
try {
   divide(20,5);
   divide(5, 0);

} catch(Exception $e) {
  echo "Unable to divide! ".$e->getMessage();
}
```

Example2: Exception handling

```
<?php
class test{
    private $n1,$n2;
    function __construct($num1,$num2){
        $this->n1=$num1;
        $this->n2=$num2;
    }
    function calculate(){
        try{
            $result = ($this->n1+$this->n2)/(5-$this->n2);
            echo"The result is ".$result."<br>";
        }
        catch(DivisionByZeroError $ex){
            echo"Exception!! ".$ex->getMessage()."<br>";
        }
        finally{
            echo"Hello  from finally block <br>";

        }

    }
}
$obj1 = new Test(3,4);
$obj2 = new Test(3,5);
```

```
//no exception
$obj1->calculate();
//exception here
$obj2->calculate();
```

Multiple catch blocks

- ☞ PHP allows a series of catch blocks following a try block to handle different exception cases. Various catch blocks may be employed to handle predefined exceptions and errors as well as user defined exceptions

Example:

```
<?php
function calculate($a, $b)
{
    try {
        if (count($b) < 6) {
            throw new OutOfBoundsException("Index is out of bound!");
        }
        if ($b[5] == 0) {
     throw new DivisionByZeroError("Division by zero is not allowed.");
        }
        $result = $a / $b[5];
        echo "Result: {$result} <br>";
    } catch (DivisionByZeroError $e) {
        echo "Error: " . $e->getMessage() . "<br>";
    } catch (OutOfBoundsException $e) {
        echo "Error: " . $e->getMessage() . "<br>";
    }
}
// index out of bounds
calculate(10, [3, 6, 7, 8]);
//division by zero
calculate(10, [3, 6, 7, 8, 9, 0, 45]);
//no exception
calculate(10, [3, 6, 7, 8, 9, 5, 45]);
```

User defined / Custom Exception

☞ In PHP, we can create user defined exception (a class) by extending it with the built-in Exception class or any other exception class provided by PHP. Custom exception classes allow you to define and handle specific types of errors or exceptional situations in your code.

Syntax:

```
class MyException extends Exception {
//code
}
```

If required, we can define constructor and custom functions for this class.

Custom exception classes provide flexibility in handling different types of errors or exceptional situations in your code. You can define your own methods and properties in the custom exception class to add specific behavior or data related to the exception.

Advantages:

☞ Built-in exceptions are good but custom exceptions have more importance from a developer's point of view since it can target and catch exception wherever he wants.

☞ Easy to debug as the developer can define custom exceptions at multiple points and handle the same.

☞ Can easily modify the existing Exception class and use it in a more efficient way by extending it.

☞ Is useful for catching "uncaught" exception.

Example: Custom Exception in PHP

```
<?php
class InvalidAgeException extends Exception{

}
function getAge($a){
    if ($a <0 || $a>160){
        throw new InvalidAgeException("Invalid Age!!");
    }
    echo "<br>Age = {$a}<br>";
}
try{
//no exception
getAge(23);
//exception
getAge(-3);
}catch(InvalidAgeException $e){
    echo"Error: {$e->getMessage()}";
}
```

NOTES:

- All types of exceptions can be caught by specifying a single catch block with Exception as exception type .

```
catch(Exception $e){

//handler code

}
```

- In PHP, you can catch multiple exceptions through a single catch block by using the pipe (|) symbol to specify multiple exception types. This is known as a multi-catch block.

```
catch (ExceptionType1 | ExceptionType2 | ExceptionType3 $e) {

    // Handle the caught exceptions

}
```

jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript. - jQuery

- jQuery is a fast, small, and feature-rich JavaScript library. This library was created by John Resig in 2006.
    - ☞ jQuery is a lightweight, "write less, do more", JavaScript library.
- jQuery has been designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax.
- The purpose of jQuery is to make it much easier to use JavaScript on your web applications.
    - ☞ jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish, and wraps them into methods that you can call with a single line of code.
    - ☞ jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.
- jQuery can be used to find a particular HTML element in the HTML document with a certain ID, class or attribute and later we can use jQuery to change one or more of attributes of the same element like color, visibility etc. jQuery can also be used to make a webpage interactive by responding to an event like a mouse click.

jQuery Important Features

- *DOM Selection:* jQuery provides Selectors to retrieve DOM element based on different criteria like tag name, id, css class name, attribute name, value, nth child in hierarchy etc.
- *DOM Manipulation:* You can manipulate DOM elements using various built-in jQuery functions. For example, adding or removing elements, modifying html content, css class etc.
- *Special Effects:* You can apply special effects to DOM elements like show or hide elements, fade-in or fade-out of visibility, sliding effect, animation etc.

- *Events:* jQuery library includes functions which are equivalent to DOM events like click, dblclick, mouseenter, mouseleave, blur, keyup, keydown etc. These functions automatically handle cross-browser issues.
- *AJAX:* jQuery also includes easy to use AJAX functions to load data from servers without reloading whole page.
- *Cross-browser support:* jQuery library automatically handles cross-browser issues, so the user does not have to worry about it.

Adding jQuery to Web Pages

- ☞ There are several ways to use jQuery library on your web application.
- ☞ You can:
  - ☑ Download the jQuery library from jQuery.com
  - ☑ Include jQuery from a CDN, like Google

Downloading jQuery

There are two versions of jQuery available for downloading:

- Production version - this is for your live website because it has been minified and compressed
- Development version - this is for testing and development (uncompressed and readable code)

The jQuery library is a single JavaScript file, and you reference it with the HTML <script> tag:

```
<head>
<script src="jquery.js"></script>
</head>
```

Tip: Place the downloaded file in the same directory as the pages where you wish to use it.

jQuery CDN

- ☑ If you don't want to download and host jQuery yourself, you can include it from a CDN (Content Delivery Network).
- ☑ You can reference jQuery library from public CDN such as Google, Microsoft, CDNJS, jsDelivr etc.

Using jQuery from Google CDN:

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.
0/jquery.min.js"></script>
</head>
```

One big advantage of using the hosted jQuery from Google:

Many users already have downloaded jQuery from Google when visiting another site. As a result, it will be loaded from cache when they visit your site, which leads to faster loading time. Also, most CDN's will make sure that once a user requests a file from it, it will be served from the server closest to them, which also leads to faster loading time.

jQuery Syntax

- ☑ With jQuery we select (query) HTML elements and perform "actions" on them.
- ☑ The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).
- ☑ Basic syntax is:

$(selector).action()

- A **$** sign to define/access jQuery
- A **(selector)** to "query (or find)" HTML elements
- A jQuery **action()** to be performed on the element(s)

NOTE: In jQuery, the dollar sign ($) is an alias for the jQuery function. The jQuery function is a core feature of the jQuery library and is used to select and manipulate HTML elements on a web page.

Examples:

`$(this).hide()` - hides the current element.

`$("p").hide()` - hides all <p> elements.

`$(".test").hide()` - hides all elements with class="test".

`$("#test").hide()` - hides the element with id="test".

The Document Ready Event

jQuery methods are generally placed inside a document ready event. This is to prevent any jQuery code from running before the document is finished loading (is ready).

```
$(document).ready(function(){

  // jQuery methods go here...

});
```

It is good practice to wait for the document to be fully loaded and ready before working with it. This also allows you to have your JavaScript code before the body of your document, in the head section.

Here are some examples of actions that can fail if methods are run before the document is fully loaded:

- Trying to hide an element that is not created yet
- Trying to get the size of an image that is not loaded yet

The jQuery team has also created an even **shorter method for the document ready event**:

```
$(function(){
  // jQuery methods go here...
});
```

jQuery Selectors

- jQuery selectors are one of the most important parts of the jQuery library.
- jQuery selectors allow you to select and manipulate HTML element(s).
- jQuery selectors are a powerful feature of the jQuery library that allow you to easily select and manipulate HTML elements on a web page. Selectors are used to target specific elements based on their attributes, tags, classes, or hierarchical relationships. They form the basis for many of the actions you can perform with jQuery, such as modifying content, changing styles, or attaching event handlers.
- jQuery selectors are used to "find" (or select) HTML elements based on their name, id, classes, types, attributes, values of attributes and much more. It's based on the existing CSS Selectors, and in addition, it has some own custom selectors.
- All selectors in jQuery start with the dollar sign and parentheses: $().

Every jQuery selector start with this sign $(). This sign is known as the factory function. It uses the three basic building blocks while selecting an element in a given document.

Selector	**Description**
Tag Name	It represents a tag name available in the DOM. For example: $('p') selects all paragraphs'p'in the document.
Tag ID	It represents a tag available with a specific ID in the DOM. For example: $('#real-id') selects a specific element in the document that has an ID of real-id.
Tag Class	It represents a tag available with a specific class in the DOM. For example: $('real-class') selects all elements in the document that have a class of real-class.

Different ways to use selectors

The jQuery selectors can be used single or with the combination of other selectors. They are required at every steps while using jQuery. They are used to select the exact element that you want from your HTML document.

Selector	Description
Name	It selects all elements that match with the given element name.
#ID	It selects a single element that matches with the given id.
.Class	It selects all elements that matches with the given class.
Universal(*)	It selects all elements available in a DOM.
Multiple Elements A,B,C	It selects the combined results of all the specified selectors A,B and C.

The element Selector

The jQuery element selector selects elements based on the element name. You can select all <p> elements on a page like this: `$("p")`

Example

When a user clicks on a button, all <p> elements will be hidden:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){
  $("button").click(function(){
    $("p").hide();
  });
});
</script>
</head>
<body>
<h2>This is a heading</h2>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<button>Click me to hide paragraphs</button>
</body>
</html>
```

The #id Selector

- The jQuery #id selector uses the id attribute of an HTML tag to find the specific element.
- An id should be unique within a page, so you should use the #id selector when you want to find a single, unique element.
- To find an element with a specific id, write a hash character, followed by the id of the HTML element: `$("#test")`

Example

When a user clicks on a button, the element with id="test" will be hidden:

```
<html>
```

```
<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("#test").hide();

  });

});

</script>

</head>

<body>

<h2>This is a heading</h2>

<p>This is a paragraph.</p>

<p id="test">This is another paragraph.</p>

<button>Click me</button>

</body>

</html>
```

The .class Selector

- The jQuery .class selector finds elements with a specific class.
- To find elements with a specific class, write a period character, followed by the name of the class: `$(".test")`

Example

When a user clicks on a button, the elements with class="test" will be hidden:

```
<html> <head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $(".test").hide();

  });

});

</script> </head>

<body>

<h2 class="test">This is a heading</h2>

<p class="test">This is a paragraph.</p>

<p>This is another paragraph.</p>

<button>Click me</button>

</body>

</html>
```

Select Elements by Attribute

- jQuery also allows you to find an element based on attributes set on it. Specifying an attribute name in square brackets in $ function e.g. $('[class]') will return all the elements that have class attribute irrespective of value.
- You can also specify a specific value of an attribute in attribute selector. For example, $('[class="myCls"]') will return all the elements which have class attribute with myCls as a value.

Example:

In the below example, we are targeting those elements which have the name attribute by $(‘[name]’) and applying the border around them.

```
<html>
<head>
<title>jQuery Atrribute Selector</title>
</head>
<body>
<h3>jQuery Atrribute Selector</h3>
<div>
<form action="">
<input type="text" name="fname" placeholder="Enter Your
first name"> <br/><br/>
<input type="text" name="lname" placeholder="Enter Your
last name"> <br/><br/>
<input type="password" placeholder="Enter the password">
<br/><br/>
</form>
</div>
<button id="demo">Submit</button>
```

```
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/j
query.min.js"></script>
<script>
$(document).ready(function () {
$('[name]').css('border','2px solid black');
});
</script>
</body>
</html>
```

In the below example, we are targeting those input elements have the attribute required

```
<html>
<head>
    <title>jQuery Atrribute Selector</title>
</head>
<body>
    <h3>jQuery Atrribute Selector</h3>
    <div class="test">
        <form action="">
        <input  type="text" name="fname" placeholder="Enter Your
first name" required> </br></br>
        <input type="text" name="lname" placeholder="Enter Your
last name" ></br></br>
        <input type="password" name="password" placeholder="Enter
the password" required> </br></br>
    </form>
    </div>



    <button id="demo">Click Me</button>

    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.mi
n.js"></script>

    <script>

        $(document).ready(function () {

            $('[required]').css('border','2px solid red');

        });

    </script>

</body>

</html>
```

In the below example, we are targeting that element(s) having the name attribute with value fname by $(‘[name=”fname”]’). Notice that other elements with name attribute which don’t have the value fname those are not targeted.

```
<html>

<head>

<title>jQuery Atrribute Selector</title>

</head>

<body>

<h3>jQuery Atrribute Selector</h3>

<div class="test">

<form action="">
```

```
<input type="text" name="fname" placeholder="Enter Your first
name"> </br></br>

<input type="text" name="lname" placeholder="Enter Your last
name"></br></br>

<input type="password" name="password" placeholder="Enter the
password"></br></br>

</form>

</div>

<button id="demo">Click Me</button>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function () {

$('#demo').click(function () {

$('[name="fname"]').css('border','2px solid red');

});

});

</script>

</body>

</html>
```

Different jQuery Selectors

Selector	**Example**	**Description**
*	$("*")	It is used to select all elements.
#id	$("#firstname")	It will select the element with id="firstname"
.class	$(".primary")	It will select all elements with class="primary"
class,.class	$(".primary,.secondary")	It will select all elements with the class "primary" or "secondary"
element	$("p")	It will select all p elements.
el1,el2,el3	$("h1,div,p")	It will select all h1, div, and p elements.
:first	$("p:first")	This will select the first p element
:last	$("p:last")	This will select he last p element
:even	$("tr:even")	This will select all even tr elements
:odd	$("tr:odd")	This will select all odd tr elements
:first-child	$("p:first-child")	It will select all p elements that are the first child of their parent
:first-of-type	$("p:first-of-type")	It will select all p elements that are the first p element of their parent
:last-child	$("p:last-child")	It will select all p elements that are the last child of their parent
:last-of-type	$("p:last-of-type")	It will select all p elements that are the last p element of their parent
:nth-child(n)	$("p:nth-child(2)")	This will select all p elements that are the 2nd child of their parent
:nth-last-child(n)	$("p:nth-last-child(2)")	This will select all p elements that are the

		2nd child of their parent, counting from the last child
:nth-of-type(n)	$("p:nth-of-type(2)")	It will select all p elements that are the 2nd p element of their parent
:nth-last-of-type(n)	$("p:nth-last-of-type(2)")	This will select all p elements that are the 2nd p element of their parent, counting from the last child
:only-child	$("p:only-child")	It will select all p elements that are the only child of their parent
:only-of-type	$("p:only-of-type")	It will select all p elements that are the only child, of its type, of their parent
parent > child	$("div > p")	It will select all p elements that are a direct child of a div element
parent descendant	$("div p")	It will select all p elements that are descendants of a div element
element + next	$("div + p")	It selects the p element that are next to each div elements
element ~ siblings	$("div ~ p")	It selects all p elements that are siblings of a div element
:eq(index)	$("ul li:eq(3)")	It will select the fourth element in a list (index starts at 0)
:gt(no)	$("ul li:gt(3)")	Select the list elements with an index greater than 3
:lt(no)	$("ul li:lt(3)")	Select the list elements with an index less than 3
:not(selector)	$("input:not(:empty)")	Select all input elements that are not empty

:header	$(":header")	Select all header elements h1, h2 ...
:animated	$(":animated")	Select all animated elements
:focus	$(":focus")	Select the element that currently has focus
:contains(text)	$(":contains('Hello')")	Select all elements which contains the text "Hello"
:has(selector)	$("div:has(p)")	Select all div elements that have a p element
:empty	$(":empty")	Select all elements that are empty
:parent	$(":parent")	Select all elements that are a parent of another element
:hidden	$("p:hidden")	Select all hidden p elements
:visible	$("table:visible")	Select all visible tables
:root	$(":root")	It will select the document's root element
:lang(language)	$("p:lang(de)")	Select all p elements with a lang attribute value starting with "de"
[attribute]	$("[href]")	Select all elements with a href attribute
[attribute=value]	$("[href='default.htm']")	Select all elements with a href attribute value equal to "default.htm"
[attribute!=value]	$("[href!='default.htm']")	It will select all elements with a href attribute value not equal to "default.htm"
[attribute$=value]	$("[href$='.jpg']")	It will select all elements with a href attribute value ending with ".jpg"
[attribute|=value]	$("[title|='Tomorrow']")	Select all elements with a title attribute value equal to 'Tomorrow', or starting with 'Tomorrow' followed by a hyphen
[attribute^=value]	$("[title^='Tom']")	Select all elements with a title attribute value starting with "Tom"

[attribute~=value]	$("[title~='hello']")	Select all elements with a title attribute value containing the specific word "hello"
[attribute*=value]	$("[title*='hello']")	Select all elements with a title attribute value containing the word "hello"
:input	$(":input")	It will select all input elements
:text	$(":text")	It will select all input elements with type="text"
:password	$(":password")	It will select all input elements with type="password"
:radio	$(":radio")	It will select all input elements with type="radio"
:checkbox	$(":checkbox")	Itwill select all input elements with type="checkbox"
:submit	$(":submit")	It will select all input elements with type="submit"
:reset	$(":reset")	It will select all input elements with type="reset"
:button	$(":button")	It will select all input elements with type="button"
:image	$(":image")	It will select all input elements with type="image"
:file	$(":file")	It will select all input elements with type="file"
:enabled	$(":enabled")	Select all enabled input elements
:disabled	$(":disabled")	It will select all disabled input elements
:selected	$(":selected")	It will select all selected input elements
:checked	$(":checked")	It will select all checked input elements

jQuery Event Methods

- jQuery is tailor-made to respond to events in an HTML page.
- All the different visitors' actions that a web page can respond to are called events.
- An event represents the precise moment when something happens. Examples:
    - moving a mouse over an element
    - selecting a radio button
    - clicking on an element
- The term "fires/fired" is often used with events. Example: "The keypress event is fired, the moment you press a key".
- jQuery provides simple methods for attaching event handlers to selections. When an event occurs, the provided function is executed. Inside the function, this refers to the DOM element that initiated the event. The event handling function can receive an event object. This object can be used to determine the nature of the event, and to prevent the event's default behavior.
- The jQuery library provides methods to handle all the DOM events and make complete event handling considerably easier than what we have available in JavaScript.
- Following are the examples of some common events −
    - A mouse click
    - A web page loading
    - Taking mouse over an element
    - Submitting an HTML form
    - A keystroke on your keyboard, etc.

The following table lists some of the important DOM events.

Mouse Events	**Keyboard Events**	**Form Events**	**Document Events**
click	keypress	submit	load
dblclick	keydown	change	resize
hover	keyup	select	scroll
mousedown		blur	unload
mouseup		focusin	ready

jQuery Syntax For Event Methods

In jQuery, most DOM events have an equivalent jQuery method.

Example: To assign a click event to all paragraphs on a page, you can do this:

```
$("p").click();
```

The next step is to define what should happen when the event fires. You must pass a function to the event:

```
$("p").click(function(){
  // action goes here!!
});
```

Following is another syntax to bind a click event with any of the DOM elements:

```
$("p").on('click', function(){

   // event handling code goes here

});
```

Example

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("p").click(function(){
    $(this).hide();
  });
});
</script>
</head>
<body>

<p>If you click on me, I will disappear.</p>
<p>Click me away!</p>
<p>Click me too!</p>

</body>
</html>
```

Commonly Used jQuery Event Methods

- **$(document).ready()**
  The $(document).ready() method allows us to execute a function when the document is fully loaded.
- **click()**

  The click() method attaches an event handler function to an HTML element. The function is executed when the user clicks on the HTML element. The following example says: When a click event fires on a <p> element; hide the current <p> element:

```
$("p").click(function(){
  $(this).hide();
});
```

- **dblclick()**

  The dblclick() method attaches an event handler function to an HTML element.The function is executed when the user double-clicks on the HTML element:

```
$("p").dblclick(function(){
  $(this).hide();
});
```

- **mouseenter()**

  The mouseenter() method attaches an event handler function to an HTML element. The function is executed when the mouse pointer enters the HTML element:

```
$("#p1").mouseenter(function(){
  alert("You entered p1!");
});
```

- **mouseleave()**

  The mouseleave() method attaches an event handler function to an HTML element. The function is executed when the mouse pointer leaves the HTML element:

```
$("#p1").mouseleave(function(){
  alert("Bye! You now leave p1!");
});
```

- **mousedown()**

  The mousedown() method attaches an event handler function to an HTML element. The function is executed, when the left, middle or right mouse button is pressed down, while the mouse is over the HTML element:

```
$("#p1").mousedown(function(){
  alert("Mouse down over p1!");
});
```

- **mouseup()**

  The mouseup() method attaches an event handler function to an HTML element. The function is executed, when the left, middle or right mouse button is released, while the mouse is over the HTML element:

```
$("#p1").mouseup(function(){
  alert("Mouse up over p1!");
});
```

- **hover()**

  The hover() method takes two functions and is a combination of the mouseenter() and mouseleave() methods.

  The first function is executed when the mouse enters the HTML element, and the second function is executed when the mouse leaves the HTML element:

```
$("#p1").hover(function(){
  alert("You entered p1!");
},
function(){
  alert("Bye! You now leave p1!");
});
```

- **focus()**

  The focus() method attaches an event handler function to an HTML form field.

  The function is executed when the form field gets focus:

```
$("input").focus(function(){
  $(this).css("background-color", "#cccccc");
});
```

- **blur()**

  The blur() method attaches an event handler function to an HTML form field.The function is executed when the form field loses focus:

```
$("input").blur(function(){
  $(this).css("background-color", "#ffffff");
});
```

The on() Method

The on() method attaches one or more event handlers for the selected elements. Attach a click event to a <p> element:

```
$("p").on("click", function(){
  $(this).hide();
});
```

Attach multiple event handlers to a <p> element:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("p").on({

    mouseenter: function(){

      $(this).css("background-color", "lightgray");

    },

    mouseleave: function(){

      $(this).css("background-color", "lightblue");

    },
    click: function(){
      $(this).css("background-color", "yellow");
    }
  });
});
</script>
</head>
<body>
<p>Click or move the mouse pointer over this paragraph.</p>
</body>
</html>
```

Playing with elements (jQuery DOM Manipulation)

- One very important part of jQuery is the possibility to manipulate the DOM.
- jQuery comes with a bunch of DOM related methods that make it easy to access and manipulate elements and attributes.
- Here are some basic operations which you can perform on DOM elements with the help of jQuery standard library methods −
  - Extract the content of an element
  - Change the content of an element
  - Adding a child element under an existing element
  - Adding a parent element above an existing element
  - Adding an element before or after an existing element
  - Replace an existing element with another element
  - Delete an existing element
  - Wrapping content with-in an element

Getting and Setting Information About Elements

There are many ways to change an existing element. Among the most common tasks is changing the inner HTML or attribute of an element. jQuery offers simple, cross-browser methods for these sorts of manipulations.

Here are a few methods you can use to get and set information about elements:

- **.html()** – Get or set the HTML contents.
- **.text() –** Get or set the text contents; HTML will be stripped.
- **.attr()** – Get or set the value of the provided attribute.
- **.width()** – Get or set the width in pixels of the first element in the selection as an integer.
- **.height()** – Get or set the height in pixels of the first element in the selection as an integer.
- **.position()** – Get an object with position information for the first element in the selection, relative to its first positioned ancestor. This is a getter only.
- **.val()** – Get or set the value of form elements.

Get Content - text(), html(), and val()

Three simple, but useful, jQuery methods for DOM manipulation are:

- **text() -** Sets or returns the text content of selected elements
- **html() -** Sets or returns the content of selected elements (including HTML markup)
- **val() -** Sets or returns the value of form fields

The following **example** demonstrates how to get content with the
jQuery text() and html() methods:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js"></scrip
t>

<script>

$(document).ready(function(){

  $("#btn1").click(function(){

    alert("Text: " + $("#test").text());

  });

  $("#btn2").click(function(){

    alert("HTML: " + $("#test").html());

  });

});

</script>

</head>

<body>

<p id="test">This is some <b>bold</b> text in a paragraph.</p>

<button id="btn1">Show Text</button>

<button id="btn2">Show HTML</button>

</body>

</html>
```

The following **example** demonstrates how to get the value of an input field with the jQuery val() method:

```
<html> <head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">
</script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    alert("Value: " + $("#test").val());

  });

});

</script>

</head>

<body>

<p>Name: <input type="text" id="test" value="Mickey Mouse"></p>

<button>Show Value</button>

</body></html>
```

Get Attributes - attr()

The jQuery **attr()** method is used to get attribute values.

The following example demonstrates how to get the value of the href attribute in a link:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">
</script>

<script>
```

```
$(document).ready(function(){

  $("button").click(function(){

    alert($("#fb").attr("href"));

  });

});

</script>

</head>

<body>

<p><a href="https://www.facebook.com" id="fb">Facebook</a></p>

<button>Show href Value</button>

</body>

</html>
```

Set Content - text(), html(), and val()

We will use the same three methods from the previous page to set content:

- **text() -** Sets or returns the text content of selected elements
- **html() -** Sets or returns the content of selected elements (including HTML markup)
- **val() -** Sets or returns the value of form fields

▪ The following example demonstrates how to set content with the jQuery text(), html(), and val() methods:

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">

</script>
```

```
<script>

$(document).ready(function(){

  $("#btn1").click(function(){

    $("#test1").text("Hello world!");

  });

  $("#btn2").click(function(){

    $("#test2").html("<b>Hello world!</b>");

  });

  $("#btn3").click(function(){

    $("#test3").val("Nepal");

  });

});

</script>

</head>

<body>

<p id="test1">This is a paragraph.</p>

<p id="test2">This is another paragraph.</p>

<p>Input field: <input type="text" id="test3" value="Patan Dubrbar"></p>

<button id="btn1">Set Text</button>

<button id="btn2">Set HTML</button>

<button id="btn3">Set Value</button>

</body>

</html>
```

Set Attributes - attr()

The jQuery attr() method is also used to set/change attribute values. The following example demonstrates how to change (set) the value of the href attribute in a link:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("#social_media").attr("href", "https://www.twitter.com");

  });

});

</script>

</head>

<body>

<p><a   href="https://www.facebook.com"   id="social_media">Social
Media</a></p>

<button>Change href Value</button>

<p>Mouse over the link (or click on it) to see that the value of
the href attribute has changed.</p>

</body>

</html>
```

The attr() method also allows you to set multiple attributes at the same time.

The following example demonstrates how to set both the href and title attributes at the same time:

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">
</script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("#social_media").attr({

      "href" : "https://www.twitter,com",

      "title" : "Social Media"

    });

  });

});

</script></head>

<body>

<p><a href="https://www.facebook.com" title="some title"
id="social_media">Social Media</a></p>

<button>Change href and title</button>

<p>Mouse over the link to see that the href attribute has changed and a
title attribute is set.</p>

</body> </html>
```

Adding Elements

Adding New HTML Content

Following jQuery methods are used to add new content:

- **append() -** Inserts content at the end of the selected elements
- **prepend() -** Inserts content at the beginning of the selected elements
- **after() -** Inserts content after the selected elements
- **before() -** Inserts content before the selected elements

TASK: Explore about **appendTo(), prependTo(), insertAfter() and insertBefore()** methods

append() and prepend() methods to insert a html content

- The jQuery append() method inserts content AT THE END of the selected HTML elements.
- The jQuery prepend() method inserts content AT THE BEGINNING of the selected HTML elements.

Example1

```
<html> <head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">
</script>

<script>

$(document).ready(function(){

  $("#btn1").click(function(){

    $("p").append(" <b>Appended text</b>.");

  });
  $("#btn2").click(function(){

    $("ol").append("<li>Appended item</li>");

  });

});

</script>

</head>

<body>

<p>This is a paragraph.</p>

<p>This is another paragraph.</p>

<ol>

  <li>List item 1</li>

  <li>List item 2</li>

  <li>List item 3</li>

</ol>

<button id="btn1">Append text</button>

<button id="btn2">Append list items</button>

</body> </html>
```

Example2

```
<html> <head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">
</script>

<script>

$(document).ready(function(){
  $("#btn1").click(function(){
    $("p").prepend("<b>Prepended text</b>. ");
  });
  $("#btn2").click(function(){
    $("ol").prepend("<li>Prepended item</li>");
  });
});
</script>
</head>
<body>
<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<ol>
  <li>List item 1</li>
  <li>List item 2</li>
  <li>List item 3</li>
</ol>
<button id="btn1">Prepend text</button>
<button id="btn2">Prepend list item</button>
</body>
</html>
```

append() and prepend() methods to insert several html contents

In both examples above, we have only inserted some text/HTML at the beginning/end of the selected HTML elements.

However, both the append() and prepend() methods can take any number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the examples above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we append the new elements to the text with the append() method (this would have worked for prepend() too) :

```
<html> <head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

function appendText() {

  let txt1 = "<p>Text.</p>";        // Create text with HTML

  let txt2 = $("<p></p>").text("Text.");   // Create text with
jQuery

  let txt3 = document.createElement("p");

  txt3.innerHTML = "Text.";         // Create text with DOM

  $("body").append(txt1, txt2, txt3);   // Append new elements

}

</script>

</head>

<body>

<p>This is a paragraph.</p>

<button onclick="appendText()">Append Text</button>

</body>
```

```
</html>
```

jQuery after() and before() Methods

The jQuery after() method inserts content AFTER the selected HTML elements. The jQuery before() method inserts content BEFORE the selected HTML elements.

Example

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("#btn1").click(function(){

    $("img").before("<b>Before</b>");

  });

  $("#btn2").click(function(){

    $("img").after("<i>After</i>");

  });

});

</script> </head>

<body>

<img
src="https://upload.wikimedia.org/wikipedia/commons/9/9b/Flag_of_
Nepal.svg" width="100" height ="200"><br><br>
```

```
<button id="btn1">Insert before</button>

<button id="btn2">Insert after</button>

</body> </html>
```

Adding Several New Elements With after() and before()

Also, both the after() and before() methods can take an infinite number of new elements as parameters. The new elements can be generated with text/HTML (like we have done in the example above), with jQuery, or with JavaScript code and DOM elements.

In the following example, we create several new elements. The elements are created with text/HTML, jQuery, and JavaScript/DOM. Then we insert the new elements to the text with the after() method (this would have worked for before() too) :

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

function afterText() {

  var txt1 = "<b>I </b>";           // Create element with HTML

  var txt2 = $("<i></i>").text("Love ");  // Create with jQuery

  var txt3 = document.createElement("b");   // Create with DOM

  txt3.innerHTML = "Nepal!";

  $("img").after(txt1, txt2, txt3);    // Insert new elements after
img

}

</script>

</head>
```

```
<body>



<img
src="https://upload.wikimedia.org/wikipedia/commons/9/9b/Flag_of_
Nepal.svg" width="100" height ="200">

<p>Click the button to insert text after the image.</p>

<button onclick="afterText()">Insert After</button>

</body>

</html>
```

Removing Elements

To remove elements and content, there are mainly two jQuery methods:

- **remove() -** Removes the selected element (and its child elements)
- **empty() -** Removes the child elements from the selected element

jQuery remove() Method

The jQuery remove() method removes the selected element(s) and its child elements.

Example:

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("#div1").remove();
  });

});

</script>

</head>

<body>

<div  id="div1"  style="height:100px;width:300px;border:1px  solid
black;background-color:yellow;">

This is some text in the div.

<p>This is a paragraph in the div.</p>

<p>This is another paragraph in the div.</p>

</div>

<br>

<button>Remove div element</button>

</body></html>
```

jQuery empty() Method

The jQuery empty() method removes the child elements of the selected element(s).

Example:

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){
    $("#div1").empty();

  });

});

</script>

</head>

<body>

<div  id="div1"  style="height:100px;width:300px;border:1px  solid
black;background-color:yellow;">

This is some text in the div.

<p>This is a paragraph in the div.</p>

<p>This is another paragraph in the div.</p>

</div>

<br>

<button>Empty the div element</button>

</body>

</html>
```

Filter the Elements to be Removed

The jQuery remove() method also accepts one parameter, which allows you to filter the elements to be removed. The parameter can be any of the jQuery selector syntaxes.

The following example removes all <p> elements with class="test":

```
$("p").remove(".test");
```

Complete example:

```
<html>              <head>                                <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery
.min.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("p").remove(".test");

  });

});

</script>

<style>

.test {

  color: red;

  font-size: 20px;

}

</style>

</head>

<body>

<p>This is a paragraph.</p>

<p class="test">This is another paragraph.</p>
```

```
<p class="test">This is another paragraph.</p>

<button>Remove all p elements with class="test"</button>

</body> </html>
```

This example removes all <p> elements with class="test" and class="demo":

```
$("p").remove(".test, .demo");
```

Complete example:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery
.min.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("p").remove(".test, .demo");

  });

});

</script>

<style>
```

```
.test {

  color: red;

  font-size: 20px;

}

.demo {

  color: green;

  font-size: 25px;

}

</style></head>

<body>

<p>This is a paragraph.</p>

<p class="test">This is p element with class="test".</p>

<p class="test">This is p element with class="test".</p>

<p class="demo">This is p element with class="demo".</p>

<button>Remove   all   p   elements   with   class="test"   and
class="demo"</button>

</body> </html>
```

Replacing and cloning elements

- Methods like replaceAll() and replaceWith() are used to replace elements.
- clone() method is used to copy the existing element.
- The **replaceAll()** method replaces selected elements with new HTML elements.

Example:

Replace all <p> elements with <h2> elements:

```
<html>
<head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.mi
n.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("<h2>Hello world!</h2>").replaceAll("p");
  });
});
</script>
</head>
<body>

<button>Replace all p elements with h2 elements</button><br>

<p>This is a paragraph.</p>
<p>This is another paragraph.</p>
<p>This is another paragraph.</p>

</body>
</html>
```

- The **replaceWith()** method replaces selected elements with new content.

Example

Replace the first <p> element with new text:

```
<html>

<head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.js">
</script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $("p:first").replaceWith("Hello world!");

  });

});

</script>

</head>

<body>



<p>This is a paragraph.</p>

<p>This is another paragraph.</p>



<button>Replace the first p element with new text</button>



</body>

</html>
```

jQuery Traversing

jQuery traversing, which means "move through", are used to "find" (or select) HTML elements based on their relation to other elements. Start with one selection and move through that selection until you reach the elements you desire.

jQuery provides a variety of methods that allow us to traverse the DOM. The largest category of traversal methods are tree-traversal.

With jQuery you can **traverse up** the DOM tree to find ancestors of an element. An ancestor is a parent, grandparent, great-grandparent, and so on. Some useful jQuery *methods for traversing up* the DOM tree are:

- **parent()-**The parent() method returns the direct parent element of the selected element. This method only traverse a single level up the DOM tree.
- **parents()-**The parents() method returns all ancestor elements of the selected element, all the way up to the document's root element (<html>).
- **parentsUntil()-**The parentsUntil() method returns all ancestor elements between two given arguments

With jQuery you can **traverse down** the DOM tree to find descendants of an element. A descendant is a child, grandchild, great-grandchild, and so on.
Some useful jQuery *methods for traversing down* the DOM tree are:

- **children()-**The children() method returns all direct children of the selected element.This method only traverses a single level down the DOM tree.

- **find()** - The find() method returns descendant elements of the selected element, all the way down to the last descendant.

With jQuery you can **traverse sideways** in the DOM tree to find siblings of an element. Siblings share the same parent. Some useful jQuery methods for *traversing sideways* in the DOM tree:

- **siblings()-**The siblings() method returns all sibling elements of the selected element.
- **next() -** The next() method returns the next sibling element of the selected element.
- **nextAll()-**The nextAll() method returns all next sibling elements of the selected element.
- **nextUntil()** - The nextUntil() method returns all next sibling elements between two given arguments.
- **prev()-** The prev() method returns the previous sibling element of the selected element.
- **prevAll() -** The prevAll() method returns all previous sibling elements of the selected element.
- **prevUntil() -** The prevUntil() method returns all previous sibling elements between two given arguments.

jQuery also has following **filtering methods**

- **first() -** The first() method returns the first element of the specified elements.
- **last() -**The last() method returns the last element of the specified elements.
- **eq() -** The eq() method returns an element with a specific index number of the selected elements.
- **filter()** - The filter() method lets you specify a criteria. Elements that do not match the criteria are removed from the selection, and those that match will be returned.
- **not()** - The not() method returns all elements that do not match the criteria.

Hiding and Unhiding images using jQuery.

With jQuery, we can hide and show HTML elements with the **hide()** and **show()** methods. These methods also work for image tag.

Syntax:

```
$(selector).hide(speed,callback);
$(selector).show(speed,callback);
```

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds. The optional callback parameter is a function to be executed after the hide() or show() method completes.

We can also toggle between hiding and showing an element with the **toggle()** method.

Syntax:

```
$(selector).toggle(speed,callback);
```

The optional speed parameter can take the following values: "slow", "fast", or milliseconds. The optional callback parameter is a function to be executed after toggle() completes.

Example:

```
<html>
<head>
    <title>Document</title>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jquery.min.j
s"></script>
    <script>
        $(document).ready(function () {
            $("#hideButton").click(function () {
                $("#myPhoto").hide();
            });
            $("#showButton").click(function () {
                $("#myPhoto").show();
            });
            $("#toggleButton").click(function () {
                $("#myPhoto").toggle();
            });
        }
        );

    </script>
</head>

<body>
    <div align="center">
        <img src="https://cdn.britannica.com/17/83817-050-
67C814CD/Mount-Everest.jpg?w=300" id="myPhoto">
        <br><button id="showButton">Show Image</button>
        <button id="hideButton">Hide Image</button>
        <button id="toggleButton">Toggle Image</button>
    </div>
</body>

</html>
```

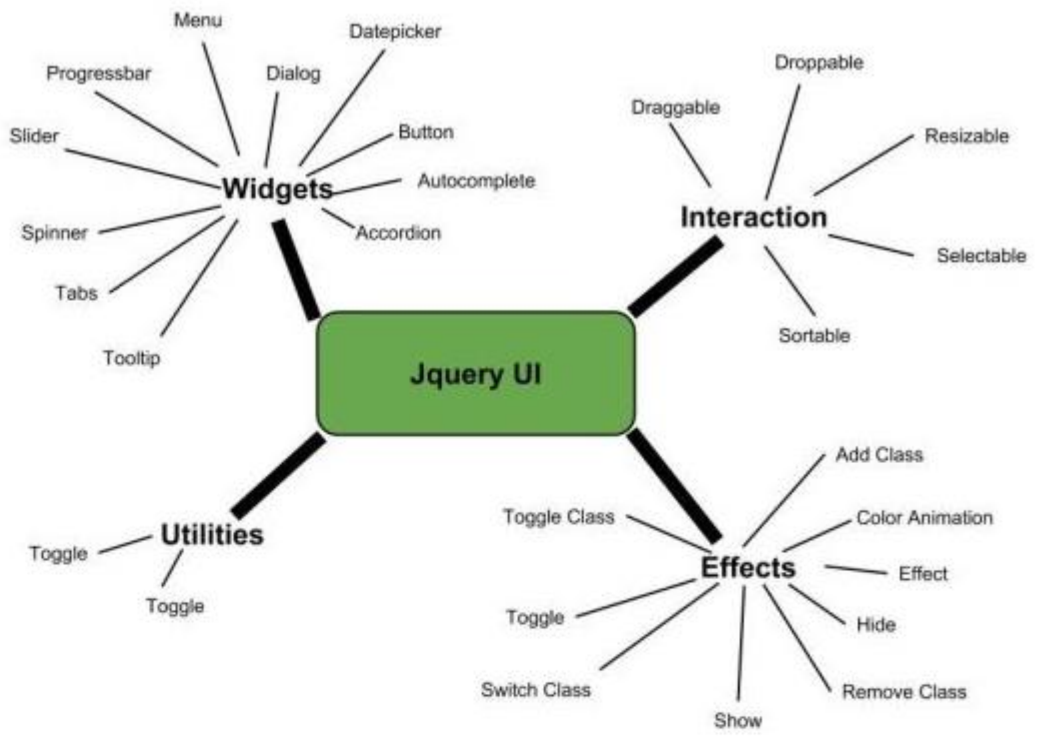
jQuery UI

jQuery UI is a curated set of user interface interactions, effects, widgets, and themes built on top of the jQuery JavaScript Library. Whether you're building highly interactive web applications or you just need to add a date picker to a form control, jQuery UI is the perfect choice.

- jQuery UI is a widget and interaction library built on top of the jQuery JavaScript Library that you can use to build highly interactive web applications. This guide is designed to get you up to speed on how jQuery UI works. Follow along below to get started.
- jQueryUI is a powerful JavaScript library built on top of jQuery JavaScript library. UI stands for User interface, It is a set of plug-ins for jQuery that adds new functionalities to the jQuery core library.
- The set of plug-ins in jQueryUI includes interface interactions, effects, animations, widgets, and themes built on top of jQuery JavaScript Library.

Features of jQuery UI

- JqueryUI is categorized into four groups, interactions, widgets, effects, utilities. These will be discussed in detail in the subsequent chapters. The structure of the library is as shown in the image below –



- **Interactions** – These are the interactive plugins like drag, drop, resize and more which give the user the ability to interact with DOM elements.
- **Widgets** – Using widgets which are jQuery plugins, you can create user interface elements like accordian,datepicker etc.
- **Effects** – These are built on the internal jQuery effects. They contain a full suite of custom animations and transitions for DOM elements.
- **Utilities** – These are a set of modular tools the jQueryUI library uses internally

Advantages of jQuery UI

- Good for highly interactive web applications
- Open source and free to use

- Powerful theme mechanism
- Stable and maintenance friendly
- Extensive browser support

Download & Installation of jQuery UI

- Download from official site (https://jqueryui.com/) and save in required directory.
  Or
  Use CDN link
  NOTE: This library also contains the files other than .js.

Some examples of jQuery UI

jQuery UI Draggable

- jQuery UI draggable() method is used to make any DOM element draggable. Once the element is made draggable, you can move it by clicking on it with the mouse and drag it anywhere within the viewport.
- We can use the draggable () method in two forms:

First Method

The draggable (option) method specifies that an HTML element can be moved in the HTML page. Here, the option parameter specifies the behavior of the elements involved.

Syntax:

$(selector, context).draggable(options);

You can use one or more options at a time using JavaScript object. In the case of multiple options, you should use commas to separate them. For example:

$(selector, context).draggable({option1: value1, option2: value2..... });

Some possible options:

delay	It specifies the delay in milliseconds, after which the first movement of the mouse is taken into account. The displacement may begin after that time. By default its value is "0".
disabled	It disables the ability to move items when set to true. Items cannot be moved until this function is enabled (using the draggable ("enable") instruction). By default its value is "false".
distance	The number of pixels that the mouse must be moved before the displacement is taken into account. By default its value is "1".
opacity	Opacity of the element moved when moving. By default its value is "false".
revert	It indicates whether the element is moved back to its original position at the end of the move. By default its value is "false".
revertduration	It indicates the duration of displacement (in milliseconds) after which the element returns to its original position (see options.revert). By default its value is "500".

Example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link href="jquery-ui/jquery-ui.min.css" rel="stylesheet" type="text/css"/>
    <link href=" jquery-ui/jquery-ui.theme.min.css" type="text/css"/>

    <style>
        #draggable { width: 100px; height: 100px; padding: 0.5; background:#7fffd4;}
        </style>
    <script src="jquery-ui/external/jquery/jquery.js"
type="text/javascript"></script>
    <script src="jquery-ui/jquery-ui.min.js" type="text/javascript"></script>
    <script>
        $( function() {
          $( "#draggable" ).draggable();
        } );
        </script>
</head>
<body>
    <div id="draggable" class="ui-widget-content">
        <p>Drag me around</p>
      </div>
```

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

Second Method

The draggable (action, params) method is used to perform an action like prevent displacement. Here action is specified as a string and one or more params can be provided based on the given action.

Syntax:

$(selector, context).draggable ("action", [params]);

Example:

$("#div7 ").draggable ('disable');

Following is a list of actions used with this method:

Action	Description
destroy()	It is used to remove drag functionality completely. The elements are no longer movable. This will return the element back to its pre-init state.
disable()	It is used to disable drag functionality. Elements cannot be moved until the next call to the draggable("enable") method.
enable()	It is used to reactivates drag management. The elements can be moved again.
option(optionname)	It gets the value currently associated with the specified optionname. Here optionname is name of the option to get and is of type string.
option()	It gets an object containing key/value pairs representing the current draggable options hash.
option(optionname, value)	It sets the value of the draggable option associated with the specified optionname. Here optionname is the name of the option to set and value is the value to set for the option.

Some Examples Left

Asynchronous JavaScript and XML (AJAX)

- AJAX is not a technology in itself, but rather an approach to using a number of existing technologies together, including HTML or XHTML, CSS, JavaScript, DOM, XML, XSLT, and most importantly the XMLHttpRequest object.
    - ☑ *AJAX is not a programming language. AJAX is not a technology.*
- When these technologies are combined in the Ajax model, web applications are able to make quick, incremental updates to the user interface without reloading the entire browser page. This makes the application faster and more responsive to user actions.
- Ajax's most appealing characteristic is its "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without having to refresh the page.
- Although X in Ajax stands for XML, JSON is preferred because it is lighter in size and is written in JavaScript. Both JSON and XML are used for packaging information in the Ajax model.

AJAX, which stands for Asynchronous JavaScript and XML, is a web development technique that allows you to update parts of a web page without having to reload the entire page. It enables communication between the web browser and the server in the background, asynchronously, without interfering with the user's interaction with the page.

- Basically, what Ajax does is make use of the browser's built-in XMLHttpRequest (XHR) object to send and receive information to and from a web server asynchronously, in the background, without blocking the page or interfering with the user's experience.
- Ajax has become so popular that you hardly find an application that doesn't use Ajax to some extent. The example of some large-scale Ajax-driven online applications are: Gmail, Google Maps, Google Docs, YouTube, Facebook, Flickr, and so many other applications.

Using AJAX we can:

— Update a web page without reloading the page
— Request data from a server - after the page has loaded
— Receive data from a server - after the page has loaded
— Send data to a server - in the background

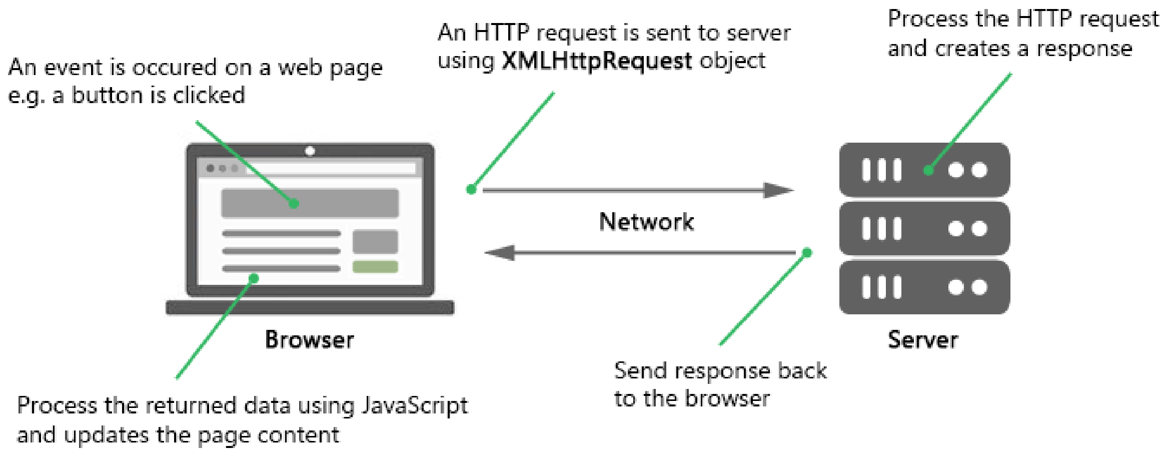
- AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast.
- AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster

NOTE: **Synchronous requests** block the execution until a response is received, whereas **asynchronous requests** allow the execution to continue while waiting for responses. Asynchronous requests can improve the user experience and enable concurrent operations, but they often require more intricate code structures to handle the asynchronous nature of the communication.

Working of AJAX

- To perform Ajax communication JavaScript uses a special object built into the browser—an **XMLHttpRequest** (XHR) object—to make HTTP requests to the server and receive data in response.

  The following illustrations demonstrate how Ajax communication works:


An event is occured on a web page e.g. a button is clicked
An HTTP request is sent to server using **XMLHttpRequest** object
Process the HTTP request and creates a response
Network
Browser
Server
Send response back to the browser
Process the returned data using JavaScript and updates the page content

Since Ajax requests are usually asynchronous, execution of the script continues as soon as the Ajax request is sent, i.e. the browser will not halt the script execution until the server response comes back.

Here's a step-by-step explanation of how AJAX works:

- **User interaction:** The process starts when a user performs an action on a web page, such as clicking a button or submitting a form.
- **JavaScript event handler:** In response to the user's action, a JavaScript event handler is triggered. This event handler initiates an AJAX request.
- **AJAX request creation:** The JavaScript code creates an *XMLHttpRequest* object (or uses the fetch API in modern browsers) to make an HTTP request to the server. This request can be of various types, such as GET or POST, depending on the desired action.
- **Server-side processing:** The server receives the AJAX request and processes it. This may involve fetching data from a database, performing calculations, or executing other server-side operations.
- **Server response:** After processing the request, the server sends a response back to the web browser. The response can be in various formats, such as XML, JSON, HTML, or plain text.

- **DOM manipulation:** Upon receiving the response, the JavaScript code in the web browser can manipulate the Document Object Model (DOM) of the page to update specific parts. It can modify the content, style, or structure of the page based on the data received.
- **User feedback:** The updated portion of the page is displayed to the user, providing a seamless and dynamic user experience. This can be done by inserting new content, updating existing content, or hiding/showing elements.

Working of AJAX

- ➢ User sends a request from the UI and a javascript call goes to XMLHttpRequest object.
- ➢ HTTP Request is sent to the server by XMLHttpRequest object.
- ➢ Server interacts with the database using JSP, PHP, Servlet, ASP.net etc.
- ➢ Data is retrieved.
- ➢ Server sends XML data or JSON data to the XMLHttpRequest callback function.
- ➢ HTML and CSS data is displayed on the browser.

AJAX with PHP

Sending Request and Retrieving the Response in AJAX

- ❖ Before we perform Ajax communication between client and server, the first thing we must do is to instantiate an XMLHttpRequest object, as shown below:
  - ▪ `const httpRequest = new XMLHttpRequest();`
- ❖ Now, the next step in sending the request to the server is to instantiating the newly-created request object using the **open()** method of the **XMLHttpRequest** object.

The open() method typically accepts three parameters— the HTTP request method to use, such as "GET", "POST", etc., the URL to send the request to and the optional third parameter sets whether the request is asynchronous which is true by default

Example:

```
request.open("GET", "info.txt",true);

--or–

request.open("POST", "add-user.php",true);
```

- Tip: The file can be of any kind, like .txt or .xml, or server-side scripting files, like .php or jsp, which can perform some actions on the server (e.g. inserting or reading data from database) before sending the response back to the client.

❖ And finally send the request to the server using the **send()** method of the **XMLHttpRequest** object.

Example:

```
request.send();

 -Or-

 request.send(body);
```

Note: The send() method accepts an optional body parameter which allow us to specify the request's *body*. This is primarily used for HTTP POST requests, since the HTTP GET request doesn't have a request body, just request headers.

The parameter to the **send()** method can be any data you want to send to the server if POST-ing the request. Form data should be sent in a format that the server can parse, like query string or other formats, like multipart/form-data, JSON, XML, and so on.

Performing an Ajax GET Request

- The GET request is typically used to get or retrieve some kind of information from the server that doesn't require any manipulation or change in database, for example, fetching search results based on a term, fetching user details based on their id or name, and so on.
- The following example will show you how to make an Ajax GET request in JavaScript.

```
<!DOCTYPE html>

<html lang="en"><head>

<title>JavaScript Ajax GET Demo</title>

<script>

function displayFullName() {

   // Creating the XMLHttpRequest object

   const request = new XMLHttpRequest();

   // Instantiating the request object

   request.open("GET", "/examples/php/greet.php?fname=John&lname=Clark");

   // Defining event listener for readystatechange event

   request.onreadystatechange = function() {
        // Check if the request is compete and was successful
        if(this.readyState === 4 && this.status === 200) {
            // Inserting the response from server into an HTML element
            document.getElementById("result").innerHTML = this.responseText;
        }
    };
    // Sending the request to the server
    request.send();
}
</script>
</head>
<body>
    <div id="result">
        <p>Content of the result DIV box will be replaced by the server response</p>
    </div>
    <button type="button" onclick="displayFullName()">Display Full Name</button>
</body> </html>
```

Here's the code from our "greet.php" file that simply creates the full name of a person by joining their first name and last name and outputs a greeting message.

```
<?php
if(isset($_GET["fname"]) && isset($_GET["lname"])) {
    $fname = htmlspecialchars($_GET["fname"]);
    $lname = htmlspecialchars($_GET["lname"]);
    // Creating full name by joining first and last name
    $fullname = $fname . " " . $lname;
    // Displaying a welcome message
    echo "Hello, $fullname! Welcome to our website.";
} else {
    echo "Hi there! Welcome to our website.";
}
?>
```

When the request is asynchronous, the send() method returns immediately after sending the request. Therefore you must check where the response currently stands in its lifecycle before processing it using the readyState property of the XMLHttpRequest object.

The **readyState** is an integer that specifies the status of an HTTP request. Also, the function assigned to the ***onreadystatechange*** event handler called every time the **readyStat**e property changes.

The full list of the **readyState** values is documented at [XMLHTTPRequest.readyState]() and is as follows:

Value	State	Description
`0`	`UNSENT`	`An XMLHttpRequest object has been created, but the open() method hasn't been called yet (i.e. request not initialized).`
`1`	`OPENED`	`The open() method has been called (i.e. server connection established).`
`2`	`HEADERS_RECEIVED`	`The send() method has been called (i.e. server has received the request).`
`3`	`LOADING`	`The server is processing the request.`
`4`	`DONE`	`The request has been processed and the response is ready.`

- ***0 (uninitialized) or (request not initialized)***
- ***1 (loading) or (server connection established)***
- ***2 (loaded) or (request received)***
- ***3 (interactive) or (processing request)***
- ***4 (complete) or (request finished and response is ready)***

Note: Theoretically, the readystatechange event should be triggered every time the **readyState** property changes. But, most browsers do not fire this event when **readyState** changes to 0 or 1. However, all browsers fire this event when **readyState** changes to 4 .

The status property returns the numerical HTTP status code of the **XMLHttpRequest**'s response. Some of the common HTTP status codes are listed below:

- **200 — OK**. The server successfully processed the request.
- **404 — Not Found**. The server can't find the requested page.
- **503 — Service Unavailable**. The server is temporarily unavailable.

NOTE: Check the list of codes here [HTTP response status codes]() for the HTTP response.

After checking the state of the request and the HTTP status code of the response, you can do whatever you want with the data the server sent. You have two options to access that data:

- ☞ **httpRequest.responseText** – returns the server response as a string of text
- ☞ **httpRequest.responseXML** – returns the response as an XMLDocument object you can traverse with JavaScript DOM functions

Performing an Ajax POST Request

- ❖ The POST method is mainly used to submit a form data to the web server.
- ❖ The following example will show you how to submit form data to the server using Ajax.

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>JavaScript Ajax POST Demo</title>
<script>
function postComment() {
   // Creating the XMLHttpRequest object
   const request = new XMLHttpRequest();
   // Instantiating the request object
   request.open("POST", "confirmation.php");
   // Defining event listener for readystatechange event
   request.onreadystatechange = function() {
      // Check if the request is compete and was successful
      if(this.readyState === 4 && this.status === 200) {
         // Inserting the response from server into an HTML element
         document.getElementById("result").innerHTML = this.responseText;
      }
   };
```

```
        // Retrieving the form data
        var myForm = document.getElementById("myForm");
        var formData = new FormData(myForm);

        // Sending the request to the server
        request.send(formData);
    }
</script>
</head>
<body>
    <form id="myForm">
        <label>Name:</label>
        <div><input type="text" name="name"></div>
        <br>
        <label>Comment:</label>
        <div><textarea name="comment"></textarea></div>
        <p><button type="button" onclick="postComment()">Post Comment</button></p>
    </form>
    <div id="result">
        <p>Content of the result DIV box will be replaced by the server response</p>
    </div>
</body></html>
```

If you are not using the FormData object to send form data, for example, if you're sending the form data to the server in the query string format, i.e. request.send(key1=value1&key2=value2) then you need to explicitly set the request header using setRequestHeader() method, like this:

request.setRequestHeader("Content-type", "application/x-www-form-urlencoded");

The setRequestHeader() method, must be called after calling open(), but before calling send().

Some common request headers are: application/x-www-form-urlencoded, multipart/form-data, application/json, application/xml, text/plain, text/html, and so on.

Note: The FormData object provides an easy way to construct a set of key/value pairs representing form fields and their values which can be sent using XMLHttpRequest.send() method. The transmitted data is in the same format that the form's submit() method would use to send the data if the form's encoding type were set to multipart/form-data.

Here's the code of our "confirmation.php" file that simply outputs the values submitted by the user.

```
<?php
if($_SERVER["REQUEST_METHOD"] == "POST") {
    $name = htmlspecialchars(trim($_POST["name"]));
    $comment = htmlspecialchars(trim($_POST["comment"]));

    // Check if form fields values are empty
    if(!empty($name) && !empty($comment)) {
        echo "<p>Hi, <b>$name</b>. Your comment has been received 
successfully.<p>";
        echo "<p>Here's the comment that you've entered: 
<b>$comment</b></p>";
    } else {
        echo "<p>Please fill all the fields in the form!</p>";
    }
} else {
    echo "<p>Something went wrong. Please try again.</p>";
}
```

```
?>
```

NOTE: For security reasons, browsers do not allow you to make cross-domain Ajax requests. This means you can only make Ajax requests to URLs from the same domain as the original page, for example, if your application is running on the domain "mysite.com", you cannot make Ajax request to "othersite.com" or any other domain. This is commonly known as same origin policy. However, you can load images, style sheets, JS files, and other resources from any domain.

AJAX with MySQL

- To illustrate the integration of AJAX and MySQL, you would typically need a server-side programming language like PHP to handle the database operations. Here's a simplified example that demonstrates AJAX and MySQL integration using PHP:
  Example:

  In this example, when the user clicks the "Load Employees" button, the loadEmployees() JavaScript function is called. It sends an AJAX GET request to load_employees.php. The PHP script connects to the MySQL database, fetches employee data from the employees table, converts it into a JSON response, and sends it back to the client. The JavaScript function displayEmployees() receives the JSON response, parses it, and dynamically updates the employeeContainer div in the HTML to display the employee data.

  //HTML file

```
<!DOCTYPE html>

<html>

<head>

  <title>AJAX and MySQL Example</title>

  <script src="myscript.js"></script>

</head>

<body>
```

```
  <h1>Employee List</h1>

  <button onclick="loadEmployees()">Load Employees</button>

  <div id="employeeContainer"></div>

</body>

</html>
```

//JavaScript (myscript.js) file

```
function loadEmployees() {

  var xhr = new XMLHttpRequest();

  xhr.open('GET', 'load_employees.php', true);

  xhr.onreadystatechange = function() {

    if (xhr.readyState === 4 && xhr.status === 200) {

      var employees = JSON.parse(xhr.responseText);

      displayEmployees(employees);

    }

  };

  xhr.send();

}

function displayEmployees(employees) {

  var container = document.getElementById('employeeContainer');

  container.innerHTML = '';

  for (let i = 0; i < employees.length; i++) {

    let employee = employees[i];

    let employeeDiv = document.createElement('div');
    employeeDiv.innerHTML = employee.name + ' - ' + employee.position;

    container.appendChild(employeeDiv);

  }

}
```

PHP (load_employees.php):

```
<?php

// Connect to MySQL database

$servername = "localhost";

$username = "root";

$password = "your_password";

$dbname = "your_database";

$conn = new mysqli($servername, $username, $password, $dbname);

// Query employees table

$query = "SELECT * FROM employees";

$result = $conn->query($query);

// Fetch data and store in an array

$employees = array();

while ($row = $result->fetch_assoc()) {

  $employees[] = $row;

}

// Send JSON response

header('Content-Type: application/json');

echo json_encode($employees);
```

// Close database connection

$conn->close();

?>

Another Example

The following example will demonstrate how a web page can fetch information from a database with AJAX:

(when a name of college is selected from combo box, it's detail is fetched from database table and displayed using AJAX call.

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <form>
        Select a College <br><br>
        <select name="colleges" id ="collegeId">
            <option value=" ">--Select a College--</option>
            <option value="101">Patan Campus</option>
            <option value="100">RR Campus</option>
            <option value="102">Prime </option>
        </select>
    </form>
    <br>
    <div id="record">--Record will be displayed here--</div>

    <script>
        function getRecord(selectedValue) {
            if (selectedValue == " ") {
                document.getElementById("record").innerHTML = "";
                return;
            }
            const request = new XMLHttpRequest();
            request.open("GET", "getrecord.php?option="+selectedValue, true);
            request.onreadystatechange = function () {

                if (this.readyState == 4 && this.status == 200) {
                    document.getElementById("record").innerHTML =
this.responseText;
                }
            };
            request.send();

        }
        let comboBox = document.getElementById("collegeId");
        comboBox.addEventListener('change',  function() {
        var selectedValue = comboBox.value; // Get the selected value
        getRecord(selectedValue); // Call the event handler method with the
selected value as a parameter
        });
    </script>
</body>
</html>
```

```
<?php
//echo"getrecord.php page";
$option = (int) $_GET["option"];
$con = new mysqli("localhost", "root", "", "bca_lab");
if ($con->connect_error) {
    die('Could not connect: ' . $con->connect_error);
}
// Fetch records for the selected option
$sql = "SELECT * FROM colleges WHERE college_id = $option";
$result = $con->query($sql);

if ($result->num_rows > 0) {
    echo "<table border='1'>
    <th>College Name</th>
    <th>Location</th>
    <th>Phone Number</th>
    <th>WebSite</th>";
    while ($row = $result->fetch_assoc()) {
        echo "<tr>";
        echo "<td>" . $row['college_name'] . "</td>";
        echo "<td>" . $row['location'] . "</td>";
```

```
            echo "<td>" . $row['phone_no'] . "</td>";
            echo "<td>" . $row['website'] . "</td>";
            echo "</tr>";
            ;
        }
} else {
        echo "<li>No records found</li>";
}
echo "</table>";
$con->close();
?>
```

jQuery –AJAX

- jQuery provides several methods for AJAX functionality.
- jQuery provides a simplified and convenient API for working with AJAX requests. It abstracts away some of the complexities of using the XMLHttpRequest object directly, making it easier to send and handle AJAX requests
- To make an AJAX request using jQuery, you use the $.ajax() or $.get() or $.post() methods. These methods provide a simple and consistent way to send different types of requests:
    - ☞ **$.ajax()** allows you to send requests with custom configurations.
    - ☞ **$.get()** is used for making GET requests.
    - ☞ **$.post()** is used for making POST requests.
- In the AJAX request configuration, you specify a success callback function that will be executed when the request succeeds. Inside this function, you can handle the response returned by the server. Similarly, you can define an error callback function to handle any errors that occur during the request
- If you need to send data along with your request (e.g., form data or JSON payload), you can specify the data property in the AJAX configuration object.

jQuery ajxa() Method

- The ajax() method is used to perform an AJAX (asynchronous HTTP) request.
- $.ajax() returns the XMLHttpRequest that it creates. In most cases you won't need that object to manipulate directly, but it is available if you need to abort the request manually.
- Syntax

```
$.ajax({name:value, name:value, ... })
```

The parameters specifies one or more name/value pairs for the AJAX request. Possible names/values in the table below:

Name	Value/Description
async	A Boolean value indicating whether the request should be handled asynchronous or not. Default is true
beforeSend(xhr)	A function to run before the request is sent
cache	A Boolean value indicating whether the browser should cache the requested pages. Default is true
complete(xhr,status)	A function to run when the request is finished (after success and error functions)
contentType	The content type used when sending data to the server. Default is: "application/x-www-form-urlencoded"
context	Specifies the "this" value for all AJAX related callback functions
data	Specifies data to be sent to the server
dataFilter(data,type)	A function used to handle the raw response data of the XMLHttpRequest
dataType	The data type expected of the server response.
error(xhr,status,error)	A function to run if the request fails.
global	A Boolean value specifying whether or not to trigger global AJAX event handles for the request. Default is true
ifModified	A Boolean value specifying whether a request is only successful if the response has changed since the last request. Default is: false.
jsonp	A string overriding the callback function in a jsonp request

jsonpCallback	Specifies a name for the callback function in a jsonp request
password	Specifies a password to be used in an HTTP access authentication request.
processData	A Boolean value specifying whether or not data sent with the request should be transformed into a query string. Default is true
scriptCharset	Specifies the charset for the request
success(result,status,xhr)	A function to be run when the request succeeds
timeout	The local timeout (in milliseconds) for the request
traditional	A Boolean value specifying whether or not to use the traditional style of param serialization
type	Specifies the type of request. (GET or POST)
url	Specifies the URL to send the request to. Default is the current page
username	Specifies a username to be used in an HTTP access authentication request
xhr	A function used for creating the XMLHttpRequest object

Example:

```
<html><head>

<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jque
ry.min.js"></script>

<script>

$(document).ready(function(){

  $("button").click(function(){

    $.ajax({url: "demo_test.txt", success: function(result){

      $("#div1").text(result);

    }});

  });

});

</script>
```

```
</head>

<body>

<div id="div1"><h2>Let jQuery AJAX Change This Text</h2></div>

<button>Get External Content</button>

</body></html>
```

jQuery load() Method in AJAX

- The load() method loads data from a server and puts the returned data into the selected element

  `$(selector).load(url,data,function(response,status,xhr))`

Parameter	Description
url	Required. Specifies the URL you wish to load
data	Optional. Specifies data to send to the server along with the request
function(response,status,xhr)	Optional. Specifies a callback function to run when the load() method is completed. Additional parameters: response - contains the result data from the request status - contains the status of the request ("success", "notmodified", "error", "timeout", or "parsererror") xhr - contains the XMLHttpRequest object

Example

```
<html><head>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.0/jque
ry.min.js"></script>
<script>
$(document).ready(function(){
  $("button").click(function(){
    $("#div1").load("demo_test.txt");
  });
});
</script>
</head>
<body>

<div id="div1"><h2>Let jQuery AJAX Change This
Text</h2></div>

<button>Get External Content</button>

</body>
</html>
```

Content Management System CMS

- A content management system (CMS) is an application that is used to manage content, allowing multiple contributors to create, edit and publish. Content in a CMS is typically stored in a database and displayed in a presentation layer based on a set of templates like a website.
- A content management system (CMS) is software that helps the users to create, manage, and modify content on a website without the need for technical knowledge. In other words, a CMS lets you build a website without needing to write code from scratch (or even know how to code at all).
- A CMS provides a user-friendly interface that simplifies the process of adding, editing, and organizing content, typically through a web-based dashboard.
- CMSs are commonly used for websites, blogs, and online stores, as they provide an interface for users to create and organize content without requiring technical expertise.
- Examples of popular CMSs : WordPress, Joomla, Drupal, Wix

Joomla

- Joomla is a powerful and flexible open-source content management system (CMS) that enables users to build and manage websites and online applications. It was first released in 2005 and has since gained popularity for its versatility and extensive feature set.
- Joomla provides every tool that users need to manage, update, and publish content. Additionally, the platform allows you to build any type of website, be it for a personal or business project.

Features of Joomla

Joomla has its own powerful built-in features (core features) including followings:

- **User Manager** − It allows managing the user information such as permission to edit, access, publish, create or delete the user, change the password and languages. The main part of the user manager is *Authentication*.
- **Content Manager** − It allows managing the content using WYSIWYG editor to create or edit the content in a very simple way.
- **Banner Manager** − It is used to add or edit the banners on the website.
- **Template Manager** − It manages the designs that are used on the website. The templates can be implemented without changing the content structure within a few seconds.
- **Media Manager** − It is the tool for managing the media files and folder in which you can easily upload, organize and manage your media files into your article editor tool.
- **Contact Manager** − It allows to add contacts, managing the contact information of the particular users.
- **Web Link Manager** − The link resource is provided for user of the site and can be sorted into categories.
- **Search** − It allows users to search the appropriate information on the site. You can use smart indexing, advanced search options, auto suggest searches to make Joomla search best.
- **Menu Manager** − It allows to create menus and menu items and can be managed subsequently. You can put menu in any style and in multiple places.
- **RSS** − It stands for Really Simple syndication which helps your site contents and RSS files to be automatically updated.

Advantages of Joomla

- It is an open source platform and available for free.
- Joomla is designed to be easy to install and set up even if you're not an advanced user.
- Since Joomla is so easy to use, as a web designer or developer, you can quickly build sites for your clients. With minimal instructions to the clients, clients can easily manage their sites on their own.

- It is very easy to edit the content as it uses WYSIWYG editor (What You See Is What You Get is a user interface that allows the user to directly manipulate the layout of the document without having a layout command).
- It ensures the safety of data content and doesn't allow anyone to edit the data.
- By default, Joomla is compatible with all browsers.
- The templates are very flexible to use.
- Media files can be uploaded easily in the article editor tool.
- Provides easy menu creation tool.

Joomla is used all over the world to power Web sites of all shapes and sizes. For example:

- Corporate Web sites or portals
- Corporate intranets and extranets
- Online magazines, newspapers, and publications
- E-commerce and online reservations
- Government applications
- Small business Web sites
- Non-profit and organizational Web sites
- Community-based portals
- School and church Web sites
- Personal or family homepages

Extensions in Joomla

- From site management to core enhancement, Joomla also helps extend your site's functionality with the vast amount of extensions available. They fall under these types:
  - **Components** – core extensions that you can modify from the back-end and view on the front-end of your site. Some examples are com_content, com_newsfeeds, and com_weblinks.
  - **Modules** – lightweight extensions or widgets that display certain content types, such as recent articles, article categories, or search boxes.

- **Plugins –** pieces of software that provide additional functions such as adding a music player or social media share buttons to enhance your website's performance.
- **Templates –** sets of design and layouts that control the site's appearance. A template defines how components and Joomla modules are shown on the website.
- **Languages –** offers translation packs for creating a multi-language site.
- **Libraries –** collections of function-related code that are used in modules, components, or plugins.

NOTE: Official documentation of Joomla is found on this link => https://docs.joomla.org/

WordPress

- WordPress is a widely used open-source content management system (CMS) that powers millions of websites worldwide. It was initially developed as a blogging platform in 2003 but has evolved into a versatile CMS capable of supporting various types of websites, from simple blogs to complex e-commerce sites.
- The same WordPress software powers both WordPress.com and WordPress.org sites. One of the main differences is how your website is hosted. Web hosting is a service that allows your website to be published and accessible on the internet.
    - **WordPress.com:** Includes managed hosting, which helps optimize your website for speed, security, and performance. Managed hosting takes care of many technical aspects of your site for you. WordPress.com has many plans with increasingly powerful features for different site types and budgets.
    - **WordPress.org:** You can download the WordPress source code for free and upload it with a hosting provider of your choice. Going this route is also called 'self-hosting' WordPress. As this option doesn't include hosting, you'll need to find and pay for separate hosting and be more involved on the technical side to build and maintain your website effectively.

Features of WordPress

- **User Management** − It allows managing the user information such as changing the role of the users to (subscriber, contributor, author, editor or administrator), create or delete the user, change the password and user information. The main role of the user manager is **Authentication**.

- **Media Management** − It is the tool for managing the media files and folder, in which you can easily upload, organize and manage the media files on your website.
- **Theme System** − It allows modifying the site view and functionality. It includes images, stylesheet, template files and custom pages.
- **Extend with Plugins** − Several plugins are available which provides custom functions and features according to the users need.
- **Search Engine Optimization** − It provides several search engine optimization (SEO) tools which makes on-site SEO simple.
- **Multilingual** − It allows translating the entire content into the language preferred by the user.
- **Importers** − It allows importing data in the form of posts. It imports custom files, comments, post pages and tags.

Advantages of WordPress

- It is an open source platform and available for free.
- CSS files can be modified according to the design as per users need.
- There are many plugins and templates available for free. Users can customize the various plugins as per their need.
- It is very easy to edit the content as it uses WYSIWYG editor (What You See Is What You Get is a user interface that allows the user to directly manipulate the layout of document without having a layout command).
- Media files can be uploaded easily and quickly.
- It offers several SEO tools which makes on-site SEO simple.
- Customization is easy according to the user's needs.
- It allows creating different roles for users for website such as admin, author, editor and contributor.

NOTE: Visit https://developer.wordpress.org/advanced-administration/before-install/howto-install/ for installation guide and https://wordpress.com/support/?aff=15767&cid=1654213 for WordPress support.