UNIT-2

Introduction to JavaScript

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities. JavaScript was first known as Live Script, but Netscape changed its name to JavaScript, possibly because of the excitement being generated by Java. JavaScript made its first appearance in Netscape 2.0 in 1995 with the name Live Script. The general-purpose core of the language has been embedded in Netscape, Internet Explorer, and other web browsers.

Advantages of JavaScript:

The merits of using JavaScript are:

- Less server interaction: You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- Immediate feedback to the visitors: They don't have to wait for a page reload to see if they have forgotten to enter something.
- Increased interactivity: You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- Richer interfaces: You can use JavaScript to include such items as drag-and drop components and sliders to give a Rich Interface to your site visitors.

Variables in JavaScript

In JavaScript, variables can be thought of as containers that hold values, and which can be retrieved later on in the application. There are several ways to define a variable, each with its own syntax and use cases. Let’s understand different methods of doing so with the help of some examples to understand the concept better.

1. **Using the var keyword**

The “var” variable has function scope, which means it is accessible within the function where it is defined or globally if defined outside of any function. In this example, the variables' name and age are defined using the var keyword. They are accessible within the sayHello() function and can also be accessed globally (outside the function) due to their global scope.

Example-1

```
<html>
<head>
  <title>Various ways to define a variable in JavaScript</title>
  <script>
    var name = "John";
    var age = 25;
    function sayHello() {
      var message = "Hello, " + name + "! You are " + age + " years old.";
      console.log(message);
    }
   sayHello();
  </script>
</head>
<body>
  <h1>Variable Example - var</h1>
</body>
</html>
```

2. Using the let Keyword

Introduced in ECMAScript 2015 (ES6) version update, the let keyword provides block scope for variables, i.e they are limited to the scope of a block only (i.e., within curly braces) where they are declared.In this example, the variables name and age are defined using let. They are accessible within the sayHello() function due to function scope, but they cannot be accessed outside the function because of block scope.

Example-2

```
<html>
<head>
  <title>Various ways to define a variable in JavaScript</title>
  <script>
    let name = "John";
    let age = 25;
    function sayHello() {
      let message = "Hello, " + name + "! You are " + age + " years old.";
      console.log(message);
    }
    sayHello();
  </script>
</head>
<body>
  <h1>Variable Example - let</h1>
</body>
</html>
```

3. Using the const Keyword

The const keyword is used to define variables that are constants, meaning their value cannot be reassigned once initialized, but if the variable is an object or an array, its properties or elements can still be modified. Constants have block scope, like variables defined with let.In this example, the variables PI and maxAttempts are defined as constants using the const keyword. The area variable within the calculate Area () function is also defined as a constant, ensuring it does not change within the function.

Example-3:

```
<html>
<head>
  <title>Various ways to define a variable in JavaScript</title>
  <script>
    const PI = 3.14159;
    const maxAttempts = 3;
    function calculateArea(radius) {
      const area = PI * radius * radius;
      console.log("The area is: " + area);
    }
    calculateArea(5);
  </script>
</head>
<body>
<h1>Variable Example - const</h1>
</body>
</html>
```

JavaScript Variable Scope

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

- **Global Variables:** A global variable has global scope which means it can be defined anywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

```
<script type="text/javascript">
<!--
var myVar = "global"; // Declare a global variable
function checkscope( ) {
   var myVar = "local";  // Declare a local variable
   document.write(myVar);
}
//-->
</script>
```

It will produce the following result:

```
Local
```

JavaScript Data Type

There are two kinds of data types as mentioned below

- Primitive Data Type
- Non Primitive Data Type

The following table describes **Primitive Data** Types available in JavaScript:

Sr.No.	Datatype Description
1.	**String** Can contain groups of character as single value. It is represented in double quotes.E.g. var x= "tutorial".
2.	**Numbers** Contains the numbers with or without decimal. E.g. var x=44, y=44.56;
3.	**Booleans** Contain only two values either true or false. E.g. var x=true, y= false.
4.	**Undefined** Variable with no value is called Undefined. E.g. var x;
5.	**Null** If we assign null to a variable, it becomes empty. E.g. var x=null;

The following table describes **Non-Primitive** Data Types in java Script.

Sr.No.	Datatype Description
1.	**Array** Can contain groups of values of same type. E.g. var x= {1,2,3,55};
2.	**Objects** Objects are stored in property and value pair. E.g. var rectangle = { length: 5, breadth: 3};

Arrays in JavaScript

The Array object lets you store multiple values in a single variable. It stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Example:

```
<!DOCTYPE html>
<html>
<body>
<h1>JavaScript Arrays</h1>
<p id="demo"></p>
<script>
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById ("demo").innerHTML = cars;
</script>
</body>
</html>
```

Arrays Methods:

1) push()
2) pop()
3) unshift()
4) shift()
5) indexOf()
6) slice()

1. **Push**

We can use push() method to add elements to the end of array. After adding element this method returns length of the array.

Ex:

```
<script>
var numbers=[10,20,30,40] //push
numbers.push(50)
 console.log(numbers)
</script>
```

2. **Pop()**

We can use pop() method to remove and return last element of the array
Ex:

```
<script>
var numbers=[10,20,30,40]
console.log(numbers.pop())// 40
console.log(numbers.pop())// 30
console.log(numbers)// [10,20]

</script>
```

3. **Unshift()**

We can use unshift() method to add element in the first position. It is counter part of push() method.

Ex:

```
<script>

var numbers=[10,20,30,40]
numbers.unshift(50)
console.log(numbers)//[50, 10, 20, 30, 40]

</script>
```

4. **Shift**

We can use shift() method to remove and return first element of the array. It is counter part to pop() method.

Ex:

```
var numbers=[10,20,30,40]
numbers.shift()
console.log(numbers)//[20, 30, 40]
```

5. index Of():

We can use index Of () to find index of specified element. If the element present multiple times then this method returns index of first occurrence. The specified element is not available then we will get -1.

Ex:

```
<script>

var numbers=[10,20,10,30,40];
```

```
console.log(numbers.indexOf(10))//0
console.log(numbers.indexOf(50))// -1
</script>
```

6.Slice

We can use slice operator to get part of the array as slice. slice(begin,end) □ returns the array of elements from begin index to end-1 index. slice(),returns total array.This can be used for cloning purposes.

Ex:

```
var numbers=[10,20,30,40,50,60,70,80]
var num1=numbers. Slice(1,5)
console.log(num1)// [20, 30, 40, 50]
num2=numbers.slice()
console.log(num2)// [10, 20, 30, 40, 50, 60, 70, 80]
```

JavaScript Objects:

- By using arrays we can store a group of individual objects and it is not possible to store key-value pairs.
- If we want to represent a group of key-value pairs then we should go for Objects.
- Array: A group of individual objects
- Object: A group of key-value pairs
- JavaScript objects store information in the form of key-value pairs.
- These are similar to Java Map objects and Python Dictionary objects.

Syntax: var variableName=

```
{
 key1:value1,
key2:value2,
...};
```

Example

```
var Dob={
name:'Ravi',
year: 2000,
 Month:'July'
};
```

JavaScript Loops:

JavaScript Loops are powerful tools for performing repetitive tasks efficiently. Loops in JavaScript execute a block of code again and again while the condition is true.

1. **for Loop**

The JS for loop provides a concise way of writing the loop structure. The for loop contains initialization, condition, and increment/decrement in one line thereby providing a shorter, easy-to-debug structure of looping.

Example:

```
<body>
    <p id="demo">print numbers</p>
<script>
let text = "";
for (let i = 0; i < 5; i++) {
text += "The number is " + i + "<br>";
}
document.getElementById ("demo").innerHTML =
text;
</script>
</body>
</html>
```

2. **while Loop**

While loop starts with checking the condition. If it is evaluated to be true, then the loop body statements are executed otherwise first statement following the loop is executed. For this reason, it is also called the Entry control loop Once the condition is evaluated to be true, the statements in the loop body are executed. Normally the statements contain an updated value for the variable being processed for the next iteration. When the condition becomes false, the loop terminates which marks the end of its life cycle.

Example:

```
<body>
  <p id="demo"></p>
<script>
let text = "";
let i = 0;
while (i < 10) {
```

```
 text += "<br>The number is " + i;
 i++;
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
```

3. do-while Loop

The JS do-while loop is similar to the while loop with the only difference is that it checks for the condition after executing the statements, and therefore is an example of an Exit Control Loop. It executes loop content at least once event the condition is false.

Example:

```
<body>
    <p id="demo"></p>
<script>
let text = ""
let i = 0;
do {
  text += "<br>The number is " + i;
  i++;
}
while (i < 10);
document.getElementById("demo").innerHTML = text;
</script>
</body>
```

Conditional Statements in JavaScript

Conditional statements in JavaScript allow you to execute specific blocks of code based on conditions. If the condition meets then a particular block of action will be executed otherwise it will execute another block of action that satisfies that particular condition. There are several methods that can be used to perform Conditional Statements in JavaScript.

1. if Statement
2. else if Statement
3. if-else Statement
4. switch Statement

1. if Statement

The if statement is used to evaluate a particular condition. If the condition holds true, the associated code block is executed.

Example:

```
<body>
    <h2>JavaScript if</h2>
<p>Display "Good day!" if the hour is less than
18:00:</p>
<p id="demo">Good Evening!</p>
<script>
if (new Date().getHours() < 18) {
  document.getElementById("demo").innerHTML =
"Good day!";
}
</script>
</body>
</html>
```

2. **else if Statement**

The else if statement in JavaScript allows handling multiple possible conditions and outputs, evaluating more than two options based on whether the conditions are true or false.

Example:

```
<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

<p id="demo"></p>

<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML =
greeting;
</script>
```

```
</body>
```

3. If-else Statement

The if-else statement will perform some action for a specific condition. Here we are using the else statement in which the else statement is written after the if statement and it has no condition in their code block.

```
<script>
const num = 0;
if (num > 0) {
        console.log("Given number is positive.");
} else if (num < 0) {
        console.log("Given number is negative.");
} else {
        console.log("Given number is zero.");
};
</script>
```

4. Switch Statement

As the number of conditions increases, you can use multiple else-if statements in JavaScript. but when we dealing with many conditions, the switch statement may be a more preferred option.

```
<body>
    <p id="demo"></p>
<script>
let day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
```

```
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is
" + day;
</script>
```

Functions in JavaScript

A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions. Like any other advanced programming language, JavaScript also supports all the features necessary to write modular code using functions. You must have seen functions like alert() and write() in the earlier chapters. We were using these functions again and again, but they had been written in core JavaScript only once.

Example:

```
<script type = "text/javascript">
   <!--
      function sayHello() {
         alert("Hello there");
      }
   //-->
</script>
```

Calling a Function

To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

Example:

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello() {
            document.write ("Hello there!");
         }
      </script>

   </head>

   <body>
      <p>Click the following button to call the
function</p>
      <form>
         <input type = "button" onclick = "sayHello()"
value = "Say Hello">
      </form>
      <p>Use different text in write method and then
try...</p>
   </body>
</html>
```

Function Parameters

Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example:

```
<html>
   <head>
      <script type = "text/javascript">
         function sayHello(name, age) {
            document.write (name + " is " + age + " years
old.");
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the
function</p>
      <form>
         <input type = "button" onclick =
"sayHello('Zara', 7)" value = "Say Hello">
      </form>
      <p>Use different parameters inside the function and
then try...</p>
   </body>
</html>
```

The return Statement

A JavaScript function can have an optional **return** statement. This is required if you want to return a value from a function. This statement should be the last statement in a function. For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example:

```
<html>
   <head>
      <script type = "text/javascript">
         function concatenate(first, last) {
            var full;
            full = first + last;
            return full;
         }
         function secondFunction() {
            var result;
            result = concatenate('Zara', 'Ali');
            document.write (result );
         }
      </script>
   </head>

   <body>
      <p>Click the following button to call the
function</p>
      <form>
         <input type = "button" onclick =
"secondFunction()" value = "Call Function">
      </form>
      <p>Use different parameters inside the function and
then try...</p>
  </body>
</html>
```

JavaScript Events

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution. This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers.

1. **Mouse events**

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Example click Event

```
<body>
    <script>
    function clickevent()
    {
        document.write("click Event");
    }
    </script>
    <form>
        <input type="button"
onclick="clickevent()" value="Who's this?"/>
```

```
        </form>
</body>
```

Example Mouse hover:

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>
<script>
    function mouseoverevent()
    {
        alert("Mouseover");
    }
</script>
<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
</body>
</html>
```

2. **Keyboard Events:**

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Example:

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1"
onkeydown="keydownevent()"/>
<script>
    function keydownevent()
    {
        document.getElementById("input1");
        alert("Pressed a key");
    }
</script>
</body>
</html>
```

3. **Form Event:**

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Example

```
<body>
<h2> Enter something here</h2>
<input type="text" id="input1"
onfocus="focusevent()"/>
<script>
    function focusevent()
    {
        document.getElementById("input1").style
.background=" aqua";
    }
</script>
</body>
```

4. **Windows/Document Event**

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Example

```
<html>
<head>Javascript Events</head>
</br>
<body onload="window.alert('Page successfully
loaded');">
<script>
document.write("The page is loaded
successfully");
</script>
</body>
</html
```

JavaScript Form Validation

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user. JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation. Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.

Example:

```
<body>
<script>
function validateform(){
var name=document.myform.name.value;
var password=document.myform.password.value;
if (name==null || name==""){
  alert("Name can't be blank");
  return false;
```

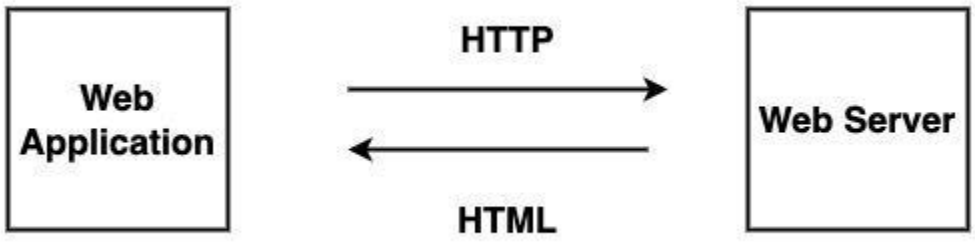
```
}else if(password.length<6){
  alert("Password must be at least 6 characters long.");
  return false;
  }
}
</script>
<body>
<form name="myform" method="post" action="http://www.javatpoint.com/javascriptpages/valid.jsp" onsubmit="return validateform()" >
Name: <input type="text" name="name"><br/>
Password: <input type="password" name="password"><br/>
<input type="submit" value="register">
</form>
</body>
```

What is AJAX?

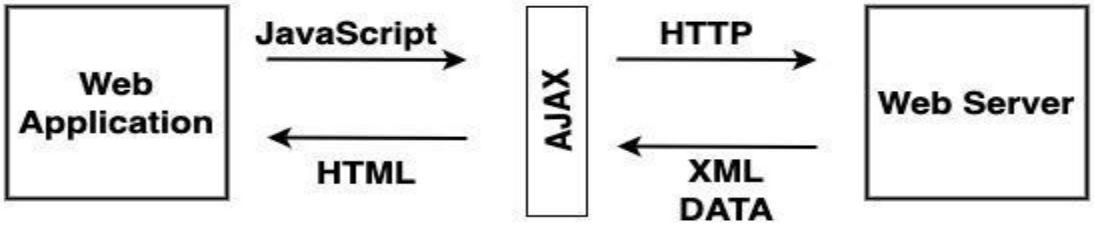
AJAX stands for asynchronous Javascript and XML. AJAX is not a programming language or technology, but it is a combination of multiple web-related technologies like HTML, XHTML, CSS, JavaScript, DOM, XML, XSLT and XMLHttpRequest object. The AJAX model allows web developers to create web applications that are able to dynamically interact with the user. It will also be able to quickly make a background call to web servers to retrieve the required application data. Then update the small portion of the web page without refreshing the whole web page. AJAX applications are much more faster and responsive as compared to traditional web applications. It creates a great balance between the client and the server by allowing them to communicate in the background while the user is working in the foreground. In the AJAX applications, the exchange of data between a web browser and the server is asynchronous means AJAX applications submit requests to the web server without pausing the execution of the application and can also process the requested data whenever it is returned. For example, Facebook uses the AJAX model so whenever we like any post the count of the like button increase instead of refreshing the whole page.

Working of Ajax

Traditional web applications are created by adding loosely web pages through links in a predefined order. Where the user can move from one page to another page to interact with the different portions of the applications. Also, HTTP requests are used to submit the web server in response to the user action. After receiving the request the web server fulfills the request by returning a new webpage which, then displays on the web browser. This process includes lots of pages refreshing and waiting.



AJAX change this whole working model by sharing the minimum amount of data between the web browser and server asynchronously. It speedup up the working of the web applications. It provides a desktop-like feel by passing the data on the web pages or by allowing the data to be displayed inside the existing web application. It will replace loosely integrated web pages with tightly integrated web pages. AJAX application uses the resources very well. It creates an additional layer known as AJAX engine in between the web application and web server due to which we can make background server calls using JavaScript and retrieve the required data, can update the requested portion of a web page without casing full reload of the page. It reduces the page refresh timing and provides a fast and responsive experience to the user. Asynchronous processes reduce the workload of the web server by dividing the work with the client computer. Due to the reduced workload web servers become more responsive and fast.



Advantages of AJAX

The following are the advantages of AJAX –

- It creates responsive and interactive web applications.
- It supports the development of patterns and frameworks that decrease the development time
- It makes the best use of existing technology and feature instead of using some new technology.
- It makes an asynchronous call to the web server which means the client doesn't have to wait for the data to arrive before starting rendering.

Disadvantages of AJAX

The following are the disadvantages of AJAX –

- AJAX is fully dependent on Javascript. So if anything happens with javascript in the browser AJAX will not support.
- The debugging of AJAX applications is difficult.
- Bookmarking of AJAX-enabled pages required pre-planning.
- If one request can fail then it can fail the load of the whole webpage.
- If JavaScript is disabled in your web browser then you are not able to run the AJAX webpage.

Introduction to JQuery

JQuery is a lightweight Javascript library which is blazing fast and concise. This library was created by John Resig in 2006 and jQuery has been designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax. 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 Features

jQuery simplifies various tasks of a programmer by writing less code. Here is the list of important core features supported by jQuery

- **DOM manipulation** − the jQuery made it easy to select DOM elements, negotiate them and modifying their content by using cross-browser open source selector engine called **Sizzle**.
- **Event handling** − The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support** − The jQuery helps you a lot to develop a responsive and feature rich site using AJAX technology.
- **Animations** − The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight** − The jQuery is very lightweight library - about 19KB in size (Minified and gripped).
- **Cross Browser Support** − The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology** − The jQuery supports CSS3 selectors and basic XPath syntax.

JQuery has been developed with the following principles:

- **Separation of JavaScript and HTML**, which encourages developers to completely separate JavaScript code from HTML markup.
- **Brevity and clarity** promotes features like **chainable** functions and shorthand function names.
- **Eliminates of cross-browser incompatibilities**, so developers does not need to worry about browser compatibility while writing code using jQuery library.
- **Extensibility**, which means new events, elements, and methods can be easily added in jQuery library and then reused as a plugin.

JQuery Selectors

jQuery Selectors are used to select HTML element(s) from an HTML document. Consider an HTML document is given and you need to select all the from this document. This is where jQuery Selectors will help.

jQuery Selectors can find HTML elements (ie. Select HTMLelements) based on the following:

- ID Selectors
- Class Selectors
- Element attribute name
- Element attribute value

1. **ID Selectors:**

The element ID selector selects a single element with the given id attribute. This would be an element ID. If the id contains any special characters like periods or colons you have to escape those characters with backslashes.

Syntax:

$('#elementid')

Example:

```
<html>
<head>
   <script src
="https://ajax.googleapis.com/ajax/libs/jquery/3.4.0/jquery.min.
js"></script>
   <style>
      #myColor
{color: white;
background: black;
padding:30px;
height: 90px;
width: 400px;
      }
 #newColor
 {background: pink;
width: 650px;
color: white;
padding:30px;
height: 90px;
      }
   </style>
</head>
<body>
   <div id="myColor"><p style="text-align:center;">Changing the
Element ID</p>
</div>
<br>
   <button onclick = "myFuntion()"> Click here </button>
   <script>
      function myFuntion() {
         $("div").attr('id', 'newColor');
      }
   </script>
   </center>
</body>
</html>
```

2. **Class Selectors**

The .class selector in jQuery is used to select all elements with the specific class.

Syntax:

$(".class")

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
   .one {
      background-color: blue;
      color: white;
      font-size: 18px;
      border: 2px blue dashed;
   }
</style>
<script src="https://code.jquery.com/jquery-
3.4.1.js"></script>
<script>
   $(document).ready(function(){
      $(".demo").addClass("one");
   });
</script>
</head>
<body>
<h1>Heading One</h1>
<h2>Heading Two</h2>
<p class="demo">Demo text 1...</p>
<p>Demo text 2...</p>
<p class="demo">Demo text 3...</p>
<p>Demo text 4...</p>
</body>
</html>
```

3. **Attribute name**

The [attribute|=value] selector selects each element with a specified attribute, with a value equal to a specified string (like "en") or starting with that string followed by a hyphen (like "en-us").

Syntax:

$("[attribute|='value']")

Example

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.7.1/jquery.min.js"></script>
<script>
$(document).ready(function(){
  $("p[title|='Tomorrow']").css("background-color", "red");});
</script>
</head>
<body>
<p title="Tomorrow">This is a paragraph.</p>
<p title="tomorrow">This is a paragraph.</p>
<p title="Tom">This is a paragraph.</p>
<p title="See You Tomorrow">This is a paragraph.</p>
<p title="Tomorrow-the day after today">This is a paragraph.</p>
</body>
</html>
```

4. **Attribute value:**

The [attribute!=value] selector in jQuery is used to select each element that isn’t having the specified attribute and value.

Syntax:

$("[attribute!='value']")

Example:

```
<!DOCTYPE html>
<html>
<head>
<style>
   .one {
      color: white;
      background-color: orange;
      font-size: 16px;
      border: 2px blue dashed;
```

```
    }
</style>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/j
query.min.js"></script>
<script>
    $(document).ready(function(){
        $("p[class!='demo']").addClass("one");
    });
</script>
</head>
<body>
<h1>Car Details</h1>
<p>Car is XUV500</p>
<p class="demo">2179 cc</p>
<p>Independent Suspension</p>
<p>Fuel Tank Capacity: 70 litres</p>
</body>
</html>
```

JQuery Events?

A jQuery Event is the result of an action that can be detected by jQuery (JavaScript). When these events are triggered, you can then use a custom function to do pretty much whatever you want with the event. These custom functions are called **Event Handlers**. 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

1. **Example Mouse Click Event**

Click event with where an alert box is displayed whenever you click on any of the divs.

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
   $(document).
ready(function() {
      $("div").click(function(){
         alert('Hi there!');
      });
   });
</script>
<style>
   div{ margin:10px;padding:12px; border:2px solid #666;
width:60px;cursor:pointer}
</style>
</head>
<body>
   <p>Click on any of the squares to see the result:</p>
   <div>One</div>
   <div>Two</div>
   <div>Three</div>
</body>
```

```
</html>
```

2. **Example DB click Mouse Event**

To bind a dblclick event with where an alert box is displayed whenever you double click on any of the divs.

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
   $(document).ready(function() {
      $("div").dblclick(function(){
         alert('Hi there!');
      });
   });
</script>
<style>
   div{ margin:10px;padding:12px; border:2px solid #666;
width:60px;cursor:pointer}
</style>
</head>
<body>
   <p>Double click on any of the squares to see the result:</p>
   <div>One</div>
   <div>Two</div>
   <div>Three</div>
</body>
</html>
```

3. **Example Mouse enter Event example**

Example:

To bind a mouse enter event with where an alert box is displayed whenever you bring cursor over any of the divs.

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
```

```
   $(document).ready(function() {
      $("div").mouseenter(function(){
         alert('Cursor is in!');
      });
   });
</script>
<style>
   div{ margin:10px;padding:12px; border:2px solid #666;
width:60px;cursor:pointer}
</style>
</head>
<body>
   <p>Bring cursor over any of the squares to see the
result:</p>
   <div>One</div>
   <div>Two</div>
   <div>Three</div>
</body>
</html>
```

4. Mouse leave Event

To bind a mouse leave event with where an alert box is displayed whenever you take cursor out of the div.

Example:

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
   $(document).ready(function() {
      $("div").mouseleave(function(){
         alert('Curosr is out!');
      });
   });
</script>
<style>
   div{
margin:10px;
padding:12px;
border:2px solid #666;
width:60px;
cursor: pointer;
}
</style>
```

```
</head>
<body>
   <p>Take cursor out any of the squares to see the result:</p>
   <div>One</div>
   <div>Two</div>
   <div>Three</div>
</body>
</html>
```

5. **Mouse Down Event**

To bind a **mouse down** event with where an alert box is displayed whenever the left, middle or right mouse button is pressed down over any of the divs.

Example:

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
   $(document).ready(function() {
      $("div").mousedown(function(){
         alert('Mouse button is down!');
      });
   });
</script>
<style>
   div{ margin:10px;
padding:12px;
border:2px solid #666;
width:60px;
cursor:pointer}
</style>
</head>
<body>
   <p>Press mouse button down over any of the squares to see the
result:</p>
   <div>One</div>
   <div>Two</div>
   <div>Three</div>
</body>
</html>
```

6. **Mouse up Event**

To bind a **mouse up** event with where an alert box is displayed whenever the left, middle or right mouse button is released over any of the divs.

Example:

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src="https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
   $(document).ready(function() {
      $("div").mouseup(function(){
         alert('Mouse button is released!');
      });
   });
</script>
<style>
   div{ margin:10px;padding:12px; border:2px solid #666;
width:60px;cursor:pointer}
</style>
</head>
<body>
   <p>Release mouse button over any of the squares to see the
result:</p>
   <div>One</div>
   <div>Two</div>
   <div>Three</div>
</body>
</html>
```

Form Events:

A The HTML Form Element interface symbolizes a form element within the DOM, offering access and modification options for various form aspects and components. Form events are standard JavaScript events, augmented by jQuery for added support. They're triggered when form controls gain or lose focus, or when users modify control values, such as typing in a text input or selecting an option in a dropdown.

Examples:

1. Submit
2. Select
3. Focus
4. Blur

1. **submit () event**

The submit event is a commonly used form event, second in popularity to the change event. By using the submit () method, you can attach an event handler that executes a function when the submit event is triggered. This event takes place whenever a form is submitted, enabling you to manage actions like form validation or data processing before the form is actually submitted to a server.

Example:

```
<html>
<head>
<title>jQuery Submit</title>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m
in.js"></script>
<script type="text/javascript">
  $(document).ready(function(){
    $("#myForm").submit(function () {
      alert("Form submitted");
    });
  });
</script>
</head>
<body>
<form id="myForm">
<input name="submit" id="submit" type="submit" />
</form>
</body>
</html
```

2. **Select Event**

In jQuery, there is no select() method that corresponds to a "select" event. Instead, the select() method in jQuery is used to select elements in the DOM based on a given selector. The "select" event in jQuery is used to capture the selection of text within an input or text area element. This event is triggered when a user highlights or selects text within the input or textarea using their mouse or keyboard. Here's an example of how you might use the "select" event in jQuery:

Example:

```
<html>
<head>
<title>jQuery</title>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m
in.js"></script>
<script type="text/javascript">
  $(document).ready(function(){
    $("select").change(function(){
      var selectedColor = $(this).find(":selected").val();
```

```
      alert("You have selected - " + selectedColor);
    });
  });
</script>
</head>
<body>
<select>
  <option>Red</option>
  <option>Blue</option>
  <option>Green</option>
</select>
</body>
</html>
```

3. Focus Event

The focus() method in jQuery serves two purposes:

Triggering the Focus Event: If you call focus() on an element, it triggers the focus event for that element. This event occurs when an element gains focus, usually due to user interaction, like clicking on an input field.

Attaching a Function to the Focus Event: You can also use focus() to attach a function that runs when the focus event is triggered for the specified element. This function executes when the element gains focus, allowing you to perform actions or validations at that point.

Example:

```
<html>
<head>
<title>jQuery</title>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.m
in.js"></script>
<script type="text/javascript">
  $(document).ready(function(){
    $("input").focus(function(){
      $(this).css('background-color', 'pink');
    });
  });
</script>
</head>
<body>
Enter: <input type="text" size="30"/>
</body>
  </html>
```

Dom Manipulation

The Document Object Model (DOM) is a programming interface for HTML and XML (Extensible Markup Language) documents. It specifies the logical structure of documents as well as how they are accessed and manipulated. DOM is a method of representing a webpage in a structured and systematic order, making it easier for programmers and users to navigate the document. Using the Document object's commands or methods, we can easily access and manipulate HTML tags, IDs, classes, Attributes, or Elements. DOM provides JavaScript with access to the HTML and CSS of the web page, as well as the ability to add actions to HTML elements. So, in simple terms, the Document Object Model is an API that represents and interacts with HTML or XML documents.

Properties of DOM

Let's take a look at the document object's properties that can be accessed and modified.

- **Window Object:** A window object is a browser object that is always at the top of the hierarchy. It is similar to an API in that it is used to set and access all of the browser's properties and methods. The browser generates it automatically.
- **Document object:** A document object is created when an HTML document is loaded into a window. The 'document' object has several properties that refer to other objects that allow access to and modification of the web page's content. If we need to access any element in an HTML page, we always begin with the 'document' object. The document object is a window object property.
- **Form Object:** It is defined by form tags.
- **Link Object:** Link tags are used to describe it.
- **Anchor Object:** A href tag is used to represent it.
- **Form Control Elements:** A form may contain a number of control elements, including text fields, buttons, radio buttons, checkboxes, and more.

The following table lists some important methods to add/remove new DOM elements.

Method	Description
append()	Inserts content to the end of element(s) which is specified by a selector.
before()	Inserts content (new or existing DOM elements) before an element(s) which is specified by a selector.
after()	Inserts content (new or existing DOM elements) after an element(s) which is specified by a selector.
prepend()	Insert content at the beginning of an element(s) specified by a selector.
remove()	Removes element(s) from DOM which is specified by selector.
replaceAll()	Replace target element(s) with specified element.
wrap()	Wrap an HTML structure around each element which is specified by selector.

1. after() Method:

The jQuery after() method inserts content (new or existing DOM elements) after target element(s) which is specified by a selector. First of all, specify a selector to get the reference of target element(s) after which you want to add the content and then call after() method. Pass the content string as a parameter. Content string can be any valid HTML element.

Example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
    </script>
    <script>
    $(document).ready(function () {
$('#div1').after('<div style="background-color:yellow"> New div </div>');
});
</script>
    <style>
    div{
    border: 1px solid;
    background-color:red;
    margin: 2px 0 2px 0;
            }
    </style>
</head>
<body>
    <h1>Demo: jQuery after() method </h1>
```

```
    <div id="div1">div 1
    </div>
    <div id="div2">div 2
    </div>
</body>
</html>
```

2. Before Method()

The jQuery before() method inserts content (new or existing DOM elements) before target element(s) which is specified by a selector. Specify a selector to get the reference of target element(s) before which you want to add the content and then call before() method. Pass the content string that can be any valid HTML element as parameter.

Example:

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
</script>
    <script>
      $(document).ready(function(){
        $('#div1').before('<div style="background-color:yellow"> New div </div>');
      });
    </script>
    <style>
      div{
        border: 1px solid;
        background-color:red;
        margin: 2px 0 2px 0;
      }
    </style>
  </head>

  <body>
    <h1>Demo: jQuery before() method </h1>
    <div id="div1">div 1 </div>
    <div id="div2">div 2</div>
  </body>
</html>
```

3. append() Method

The jQuery append() method inserts content to the end of target element(s) which is specified by a selector. First specify a selector expression to get the reference of an element(s) to which you want to append content, then call append() method and pass content string as a parameter.

Example:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
   </script>
   <script>
      $(document).ready(function () {
        $('p').append('World!');
      });
   </script>
</head>
<body>
<h1>Demo: jQuery append() method </h1>
<p>Hello </p>
</body>
</html>
```

4. prepend() Method

The jQuery prepend() method inserts content at the beginning of an element(s) specified by a selector. First specify a selector expression to get the reference of an element(s) to which you want to prepend the content, then call prepend() method and pass content string as a parameter.

Example:

```
<!DOCTYPE html>
<html>
   <head>
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js"> </script>
      <script>
         $(document).ready(function () {
            $('p').prepend('World');
         });
      </script>
   </head>

   <body>
      <h1>Demo: jQuery prepend() method </h1>
      <p>Hello</p>
   </body>
</html>
```

5. remove() Method

The jQuery remove() method removes element(s) as specified by a selector. First specify a selector expression to get the reference of an element(s) which you want to remove from the document and then call remove() method.

Syntax:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
</script>
<script>
$(document).ready(function () {
$('label').remove ();
});
   </script>
</head>
<body>
<h1>Demo: jQuery remove() method </h1>
<div>This is div.
<label>This is label.</label>
</div>
</body>
</html>
```

6. replace All() Method

The jQuery replaceAll() method replaces all target elements with specified element(s). Here, syntax is different. First specify a content string as replacement element(s) and then call replaceAll() method with selector expression to specify a target element(s).

Example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.2/jquery.min.js">
    </script>
    <script>
     $(document).ready(function () {
        $('<span>This is span</span>').replaceAll('p');
     });
   </script>
</head>
<body>
    <h1>Demo: jQuery replaceAll() method </h1>
```

This is paragraph.

This is another paragraph.

</body>

</html>

JQuery Effects

1. **.animate()**

The .animate() method allows us to create animation effects on any numeric CSS property. The only required parameter is a plain object of CSS properties. This object is similar to the one that can be sent to the .css() method, except that the range of properties is more restrictive.

Example:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>animate demo</title>
  <style>
  div {
    background-color: #bca;
    width: 100px;
    border: 1px solid green;
  }
  </style>
  <script src="https://code.jquery.com/jquery-3.7.0.js"></script>
</head>
<body>

<button id="go">&raquo; Run</button>
<div id="block">Hello!</div>

<script>
// Using multiple unit types within one animation.

$( "#go" ).on( "click", function() {
  $( "#block" ).animate({
    width: "70%",
    opacity: 0.4,
    marginLeft: "0.6in",
    fontSize: "3em",
    borderWidth: "10px"
  }, 1500 );
});
</script>

</body>
</html>
```

2. .fadeIn()

The .fadeIn() method animates the opacity of the matched elements.

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>fadeIn demo</title>
  <style>
  span {
    color: red;
    cursor: pointer;
  }
  div {
    margin: 3px;
    width: 80px;
    display: none;
    height: 80px;
    float: left;
  }
  #one {
    background: #f00;
  }
  #two {
    background: #0f0;
  }
  #three {
    background: #00f;
  }
  </style>
  <script src="https://code.jquery.com/jquery-3.7.0.js"></script>
</head>
<body>

<span>Click here...</span>
<div id="one"></div>
<div id="two"></div>
<div id="three"></div>

<script>
$( document.body ).on( "click", function() {
  $( "div:hidden" ).first().fadeIn( "slow" );
} );
</script>

</body>
</html>
```

3. .fadeOut()

The .fadeOut() method animates the opacity of the matched elements. Once the opacity reaches 0, the display style property is set to none, so the element no longer affects the layout of the page.

Example:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>fadeOut demo</title>
  <style>
  p {
    font-size: 150%;
    cursor: pointer;
  }
  </style>
  <script src="https://code.jquery.com/jquery-3.7.0.js"></script>
</head>
<body>

<p>
  If you click on this paragraph
  you'll see it just fade away.
</p>

<script>
$( "p" ).on( "click", function() {
  $( "p" ).fadeOut( "slow" );
});
</script>

</body>
</html>
```

4. .slide Down()

The .slide Down() method animates the height of the matched elements. This causes lower parts of the page to slide down, making way for the revealed items.

Example:

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>slideDown demo</title>
  <style>
  div {
    background: #de9a44;
    margin: 3px;
    width: 80px;
    height: 40px;
    display: none;
    float: left;
  }
```

```
  </style>
  <script src="https://code.jquery.com/jquery-3.7.0.js"></script>
</head>
<body>
Click me!
<div></div>
<div></div>
<div></div>
<script>
$( document.body ).on( "click", function () {
  if ( $( "div" ).first().is( ":hidden" ) ) {
    $( "div" ).slideDown( "slow" );
  } else {
    $( "div" ).hide();
  }
} );
</script>
</body>
</html>
```

5. Show()

Shows the selected elements

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>show demo</title>
  <style>
  div {
    background: #def3ca;
    margin: 3px;
    width: 80px;
    display: none;
    float: left;
    text-align: center;
  }
  </style>
  <script src="https://code.jquery.com/jquery-3.7.0.js"></script>
</head>
<body>

<button id="showr">Show</button>
<button id="hidr">Hide</button>
<div>Hello 3,</div>
<div>how</div>
<div>are</div>
<div>you?</div>

<script>
$( "#showr" ).on( "click", function() {
  $( "div" ).first().show( "fast", function showNext() {
    $( this ).next( "div" ).show( "fast", showNext );
  });
});
```

```
$( "#hidr" ).on( "click", function() {
  $( "div" ).hide( 1000 );
});
</script>

</body>
</html>
```

JQuery animate ()

The jQuery animate() method is used to create custom animations by changing the CSS numerical properties of a DOM element, for example, width, height, margin, padding, opacity, top, left, etc**.**

Syntax:

```
$(selector).animate({ properties }, [speed, callback] );
```

JQuery Properties:

- **Properties** − A required parameter which defines the CSS properties to be animated and this is the only mandatory parameter of the call.
- **speed** − An optional string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **Callback** − An optional parameter which represents a function to be executed whenever the animation completes.

Example:

```
<!doctype html>
<html>
<head>
<title>The jQuery Example</title>
<script src = "https://www.tutorialspoint.com/jquery/jquery-
3.6.0.js"></script>
<script>
$(document).ready(function() {
   $("#right").click(function(){
      $("div").animate({left: '250px'});
   });
   $("#left").click(function(){
      $("div").animate({left: '0px'});
   });
});
</script>
<style>
  #left, #right
{margin:3px;
border:2px solid #666;
height:30px;
```

```
width:100px;
cursor:pointer;
}
   #box{
position:relative;
margin:3px;
padding:12px;
border:2px solid #666;
height:100px;
width:180px;
}
</style>
</head>
<body>
   <p>Click on Left or Right button to see the result:</p>
   <div id="box"  style="background-color:#9c9cff;">This is
Box</div>
   <button id="right" style="background-color:#fb7c7c;">Right
Move</button>
   <button id="left" style="background-color:#93ff93;">Left
Move</button>
</body>
</html>
```

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. The image below illustrates an HTML page as a tree (DOM tree). With jQuery traversing, you can easily move up (ancestors), down (descendants) and sideways (siblings) in the tree, starting from the selected (current) element. This movement is called traversing - or moving through - the DOM tree.

Method	Description
add()	Adds elements to the set of matched elements
add Back()	Adds the previous set of elements to the current set
andSelf()	Deprecated in version 1.8. An alias for add Back()
children()	Returns all direct children of the selected element

closest()	Returns the first ancestor of the selected element
contents()	Returns all direct children of the selected element (including text and comment nodes)
each()	Executes a function for each matched element
end()	Ends the most recent filtering operation in the current chain, and return the set of matched elements to its previous state
eq()	Returns an element with a specific index number of the selected elements
filter()	Reduce the set of matched elements to those that match the selector or pass the function's test
find()	Returns descendant elements of the selected element
first()	Returns the first element of the selected elements
has()	Returns all elements that have one or more elements inside of them
is()	Checks the set of matched elements against a selector/element/jQuery object, and return true if at least one of these elements matches the given arguments
last()	Returns the last element of the selected elements
map()	Passes each element in the matched set through a function, producing a new jQuery object containing the return values
next()	Returns the next sibling element of the selected element
nextAll()	Returns all next sibling elements of the selected element
nextUntil()	Returns all next sibling elements between two given arguments
not()	Returns elements that do not match a certain criteria
offsetParent()	Returns the first positioned parent element
parent()	Returns the direct parent element of the selected element
parents()	Returns all ancestor elements of the selected element

parentsUntil()	Returns all ancestor elements between two given arguments
prev()	Returns the previous sibling element of the selected element
prevAll()	Returns all previous sibling elements of the selected element
prevUntil()	Returns all previous sibling elements between two given arguments
siblings()	Returns all sibling elements of the selected element
slice()	Reduces the set of matched elements to a subset specified by a range of indices

JQuery Filter's

Predefine Filters

These are inbuilt filters in query panel created by the administrator. Predefined filters are created at the Universe level and are directly used in the report from the Universe. Drag an object on which you want to apply filter to query filter pane and drag predefined filters too. When you run the query data w.r.t query filter will be returned in the report.

Custom Filters

These filters are created with the queries in the query panel. Custom filters are created in the Query panel under the query filter tab. Drag the object to the query filter pane and make use of various relational operator to pass the filter condition. You can put a constant value or a list of values in the query filter.

Prompts

They are used to display a question or list of values and are known as dynamic filters.

Filter Condition

Constant option allows you to enter a single value in filter. List of values allow you to choose one value from all the available values for an object. Prompt is used to pass dynamic value to a query filter.