XML VS JSON

XML	JSON
• stored as tree structure	• stored as key value pairs
• XML supports UTF-8 and UTF-16 encodings.	• JSON supports UTF as well as ASCII encodings.
• Bulky and slow in parsing, leading to slower data transmission	• very fast as size of file is small
• no suport for arrays use tags as alternative	• supports arrays,strings,bools but not complex types like charts nd images
• prone to attacks cuz of entity expansion and DTD validation turned on by default	• safe unless JSONP used which leads to CSRF (cross site request forgery) attack

- to convert DOM string to XML DOM STRUCTURE

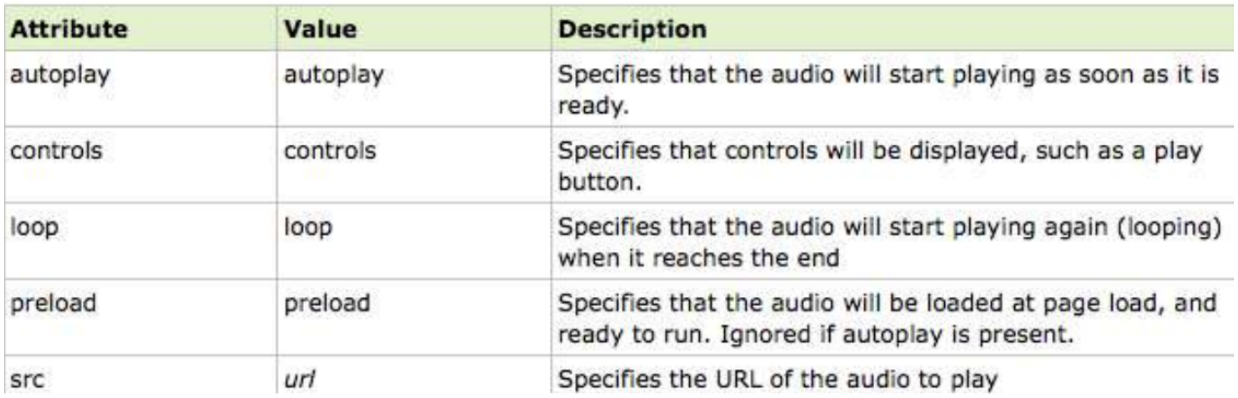
```
var parser = new DOMParser();
var xmldoc = parser.parseFromString(text,"text/xml");
// text is dom string like "<ul><li>hello</li></ul>"
//document.getElementById("demo").innerHTML =
xmldoc.getElementsByTagName("name")[0].childNodes[0].nodeValue;
```

- JSON uses JavaScript syntax, but the JSON format is text only.to convert data stored in Javascript object into json , we use JSON.stringify(obj)

```
var obj = { name: "John", age: 30, city: "New York" };
var myJSON = JSON.stringify(obj);
```

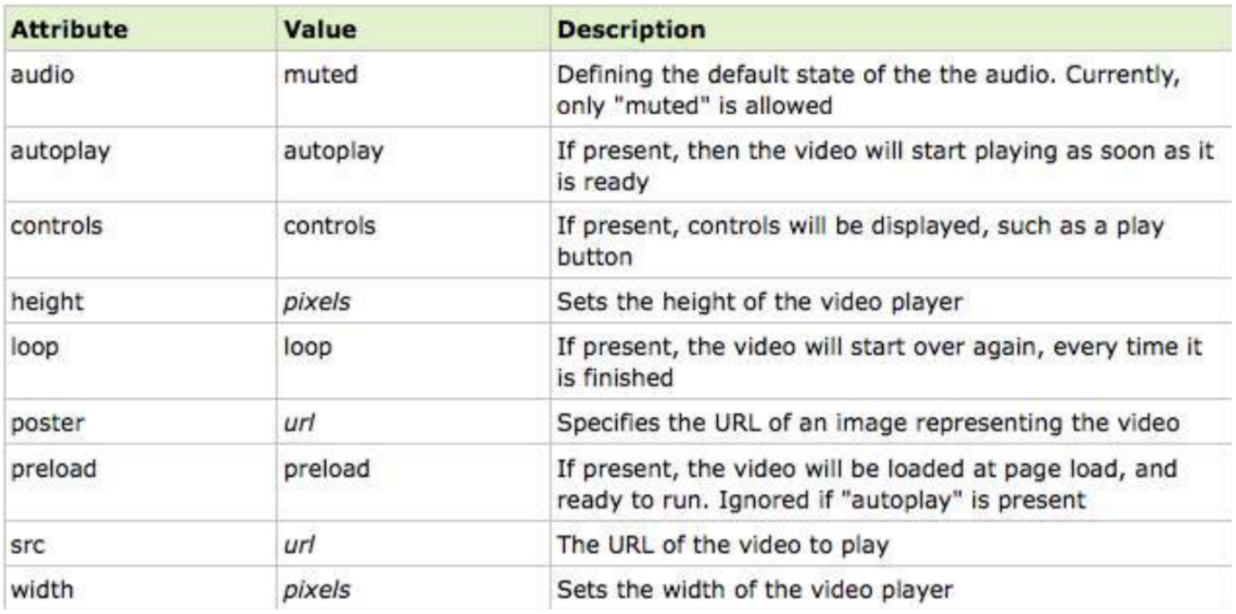
AUDIO,VIDEO,PROGRESS,CANVAS,SVG & GEOLOCATION

- HTML5 <audio> tag
  - format is `<audio src="my_music.mp3" controls></audio>`
  - AUDIO TAG ATTRIBUTES



Attribute	Value	Description
autoplay	autoplay	Specifies that the audio will start playing as soon as it is ready.
controls	controls	Specifies that controls will be displayed, such as a play button.
loop	loop	Specifies that the audio will start playing again (looping) when it reaches the end
preload	preload	Specifies that the audio will be loaded at page load, and ready to run. Ignored if autoplay is present.
src	*url*	Specifies the URL of the audio to play

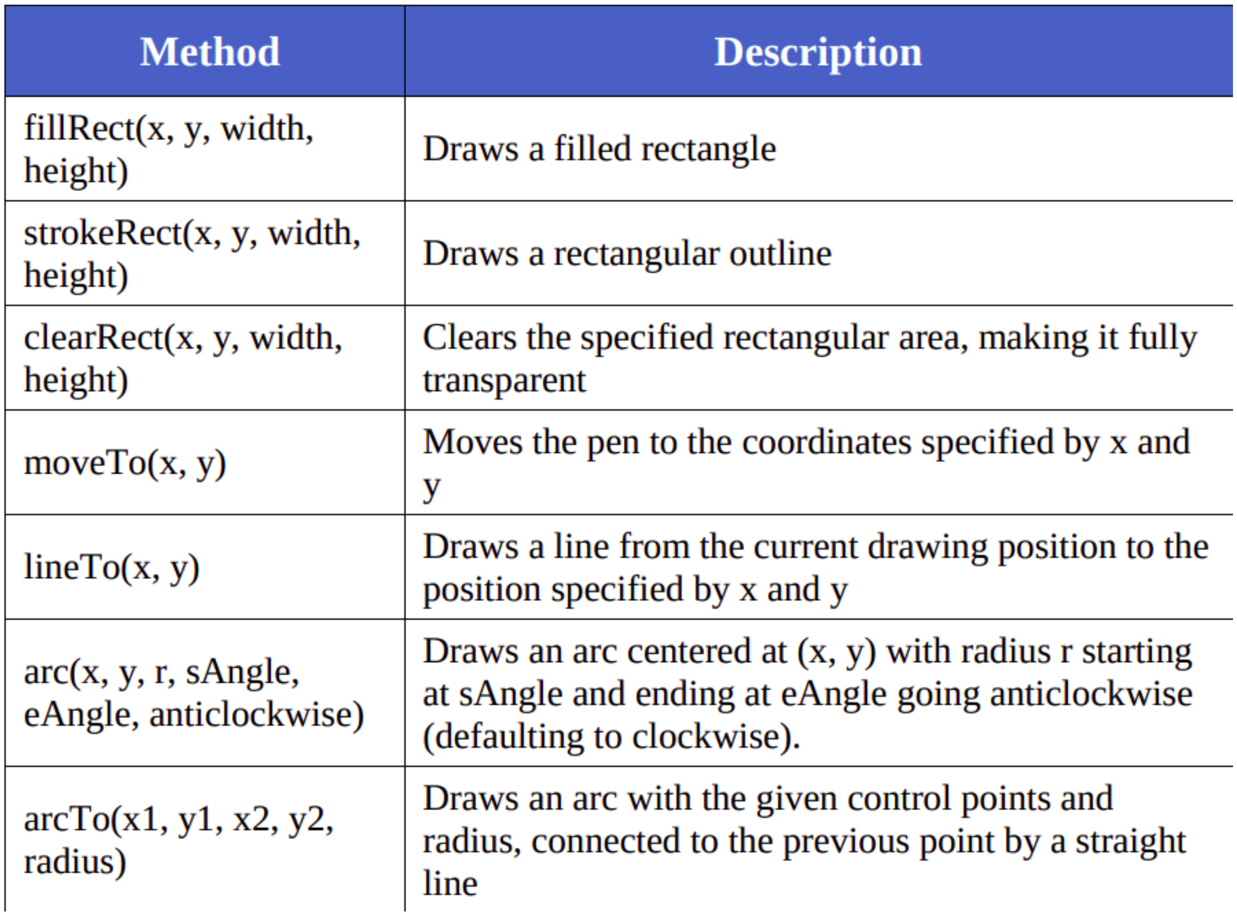
- HTML5 <video> tag
  - Any content between the opening and closing tags is fallback content. This content is displayed only by browsers that don't support the tag.
  - attributes :



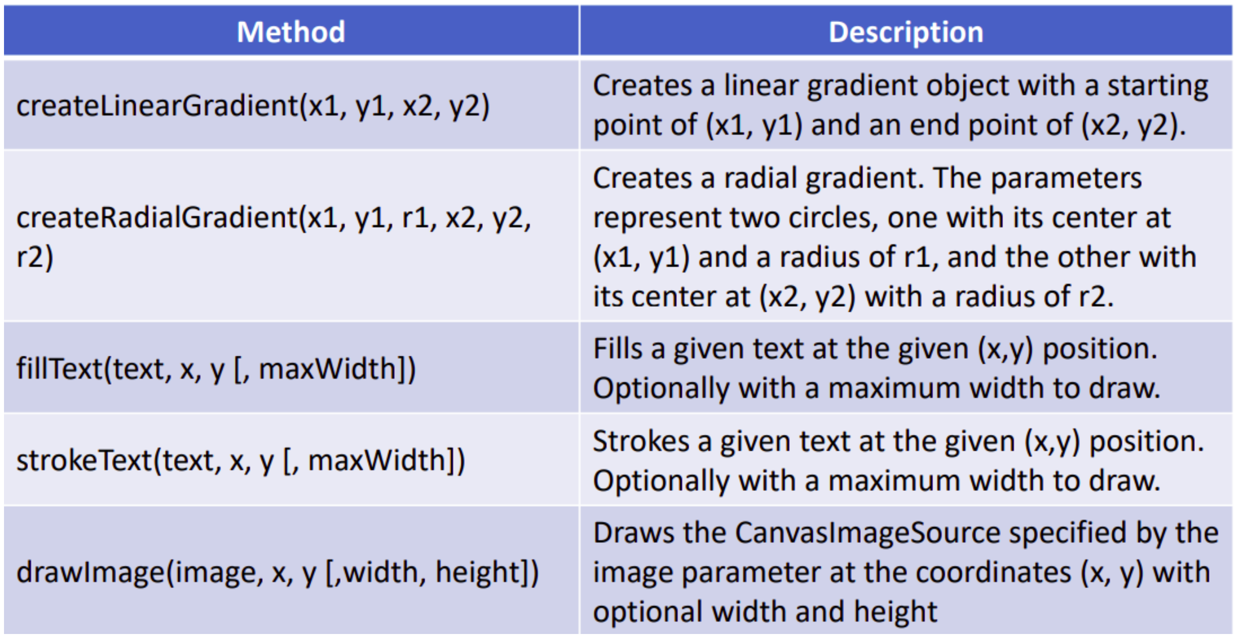
Attribute	Value	Description
audio	muted	Defining the default state of the the audio. Currently, only "muted" is allowed
autoplay	autoplay	If present, then the video will start playing as soon as it is ready
controls	controls	If present, controls will be displayed, such as a play button
height	*pixels*	Sets the height of the video player
loop	loop	If present, the video will start over again, every time it is finished
poster	*url*	Specifies the URL of an image representing the video
preload	preload	If present, the video will be loaded at page load, and ready to run. Ignored if "autoplay" is present
src	*url*	The URL of the video to play
width	*pixels*	Sets the width of the video player

- HTML5 <progress> tag
  - creates progress bar
  - attribute value = "" : percentage of progress
  - attribute max = ""
- HTML5 <canvas> tag

- methods

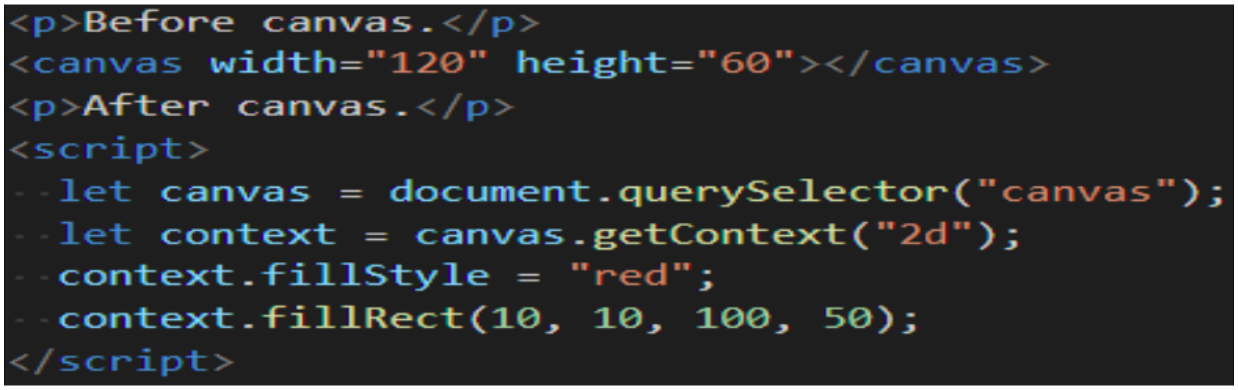


Method	Description
fillRect(x, y, width, height)	Draws a filled rectangle
strokeRect(x, y, width, height)	Draws a rectangular outline
clearRect(x, y, width, height)	Clears the specified rectangular area, making it fully transparent
moveTo(x, y)	Moves the pen to the coordinates specified by x and y
lineTo(x, y)	Draws a line from the current drawing position to the position specified by x and y
arc(x, y, r, sAngle, eAngle, anticlockwise)	Draws an arc centered at (x, y) with radius r starting at sAngle and ending at eAngle going anticlockwise (defaulting to clockwise).
arcTo(x1, y1, x2, y2, radius)	Draws an arc with the given control points and radius, connected to the previous point by a straight line



Method	Description
createLinearGradient(x1, y1, x2, y2)	Creates a linear gradient object with a starting point of (x1, y1) and an end point of (x2, y2).
createRadialGradient(x1, y1, r1, x2, y2, r2)	Creates a radial gradient. The parameters represent two circles, one with its center at (x1, y1) and a radius of r1, and the other with its center at (x2, y2) with a radius of r2.
fillText(text, x, y [, maxWidth])	Fills a given text at the given (x,y) position. Optionally with a maximum width to draw.
strokeText(text, x, y [, maxWidth])	Strokes a given text at the given (x,y) position. Optionally with a maximum width to draw.
drawImage(image, x, y [,width, height])	Draws the CanvasImageSource specified by the image parameter at the coordinates (x, y) with optional width and height

    - sample code



```
<p>Before canvas.</p>
<canvas width="120" height="60"></canvas>
<p>After canvas.</p>
<script>
  let canvas = document.querySelector("canvas");
  let context = canvas.getContext("2d");
  context.fillStyle = "red";
  context.fillRect(10, 10, 100, 50);
</script>
```

- HTML5 <svg> tag
    - xmlns = "link" : changes element to different XML namespace s
    - code

```
<svg width="100" height="100">
     <circle cx="50" cy="50" r="40" stroke="green" stroke-width="4"
   fill="yellow" />
</svg>
```

```
-  <rect width="300" height="100" style = "fill:rgb(0,0,255); stroke-
   width:3; stroke:rgb(0,0,0)" />
-  <circle cx="50" cy="50" r="40" stroke="black" stroke-width="3"
   fill="red" />
-  <ellipse cx="200" cy="80" rx="100" ry="50“ style = "fill:yellow;
   stroke:purple; stroke-width:2" />
-  <polygon points="200,10 250,190 160,210" style = "fill:lime;
   stroke:purple; stroke-width:1" />
-  <text x="0" y="15" fill="red" transform="rotate(30 20,40)">I love
   SVG</text>
```

    - to draw semicircle make svg canvas height = 2* radius of circle
- HTML5 geolocation API
    - identifiies users location (asks user for permission first)
    - accessed through **navigator.geolocation** object
    - functions
        - getCurrentPosition(success callback,error callback , options) : returns location as one time snap
        - watchPosition(success callback,error callback , options) :returns the location of the visitor every time the location changes

    - Success callback function : receives position object with these read only properties
    - double latitude
    - double longitude
    - double accuracy
    - double altitude
    - double altitudeAccuracy
    - double heading (direction)
    - double speed
    - Error callback function :receives error object with these two properties
    - short code
    - 1, meaning PERMISSION_DENIED
    - 2, meaning POSITION_UNAVAILABLE
    - 3, meaning TIMEOUT
    - DOMString message
    - Options object (third parameter to getCurrentPosition or watchPosition)
    - enableHighAccuracy // true or false
    - timeout // milliseconds
    - maximumAge // milliseco.nds
  - example code snippet :

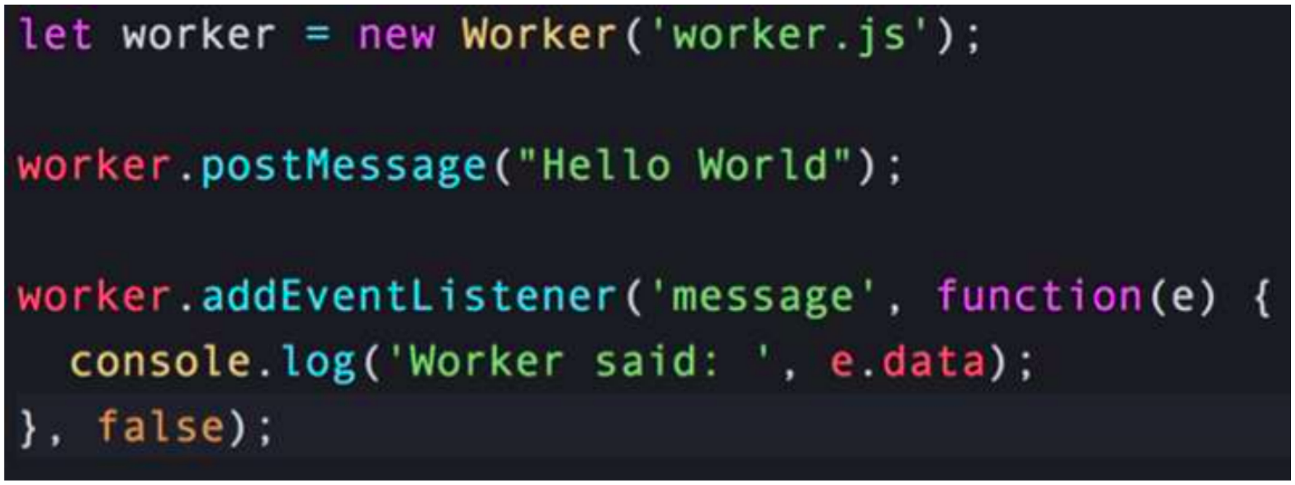
```
navigator.geolocation.getCurrentPosition((position) => {
  doSomething(position.coords.latitude, position.coords.longitude);
});
```

WEB WORKERS

- Web Workers are a simple means for web content to run scripts in background threads. The worker thread can perform tasks without interfering with the user interface
- they can make network requests using the `fetch()` or `XMLHttpRequest` APIs.
- Once created, a worker can send messages to the JavaScript code that created it by posting messages to an event handler specified by that code (and vice versa).
- webworkers can send `AJAX` requests using the `XMLHttpRequest` but cant manipulate `DOM` directly inside a worker
- `dedicated workers` : instantiated by main process, and can only communicate/be accessed by script that called it
- `shared workers` : can be reached by all processes running on the same origin (different browser tabs, iframes or other shared workers).

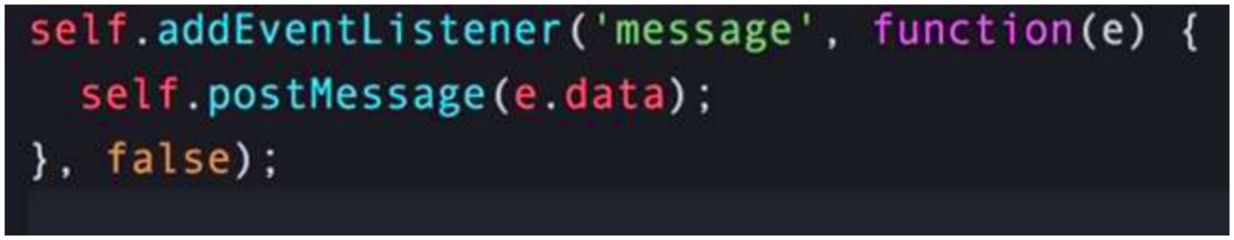
- how it works : parent uses postmessage to post message to the worker , and use onmessage event handler which gets triggered when worker finishes task.worker also has a onmessage eventhandler which waits for a message from parent , does some task and posts message back to parent using postmessage
- code

This is main.js.



```
let worker = new Worker('worker.js');

worker.postMessage("Hello World");

worker.addEventListener('message', function(e) {
  console.log('Worker said: ', e.data);
}, false);
```

This is worker.js



```
self.addEventListener('message', function(e) {
  self.postMessage(e.data);
}, false);
```

- another way is using inline event handlers like onmessage=function(for worker.js) , or worker.onmessage=function(for parent code)

- another code sample for web workers : main.js

```
const first = document.querySelector('#number1');
const second = document.querySelector('#number2');

const result = document.querySelector('.result');

if (window.Worker) {
  const myWorker = new Worker("worker.js");

  [first, second].forEach(input => {
    input.onchange = function() {
      myWorker.postMessage([first.value, second.value]);
      console.log('Message posted to worker');
    }
  })

  myWorker.onmessage = function(e) {
    result.textContent = e.data;
    console.log('Message received from worker');
  }
} else {
  console.log('Your browser doesn\'t support web workers.');
}
```

- worker.js

```
onmessage = function(e) {
  console.log('Worker: Message received from main script');
  const result = e.data[0] * e.data[1];
  if (isNaN(result)) {
    postMessage('Please write two numbers');
  } else {
    const workerResult = 'Result: ' + result;
    console.log('Worker: Posting message back to main script');
    postMessage(workerResult);
  }
}
```

JQUERY

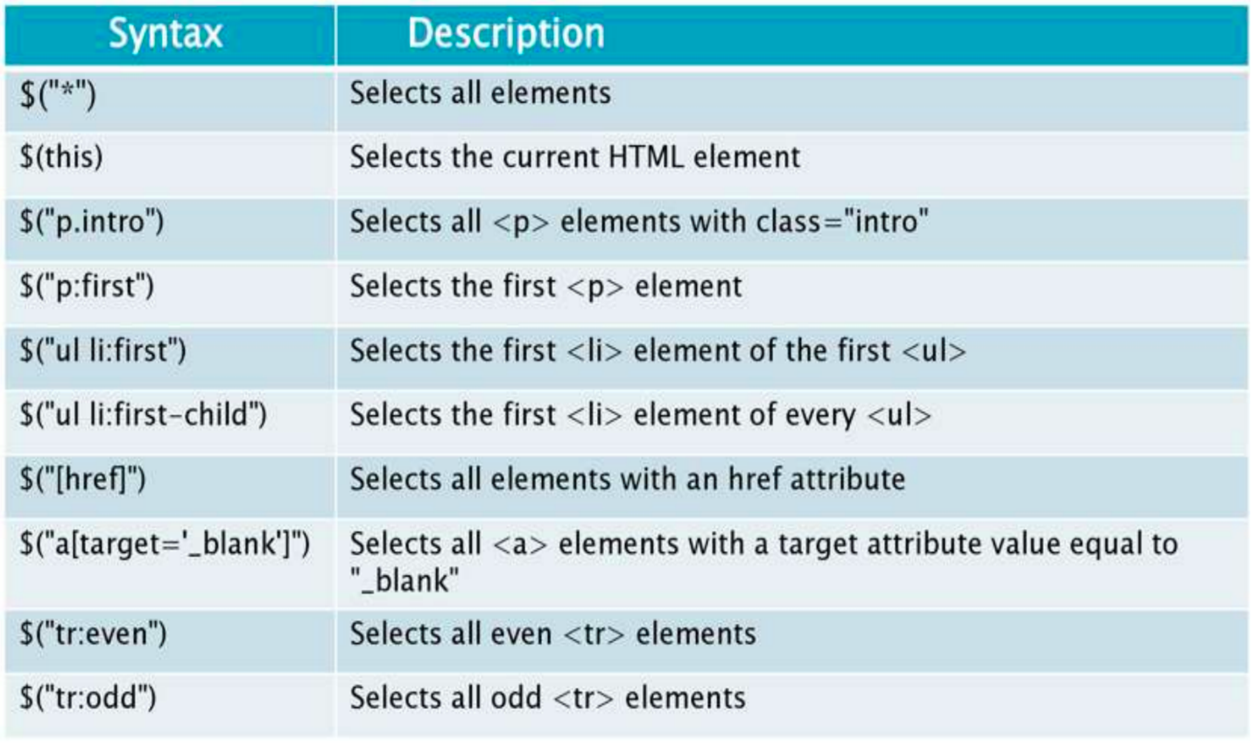
- With jQuery you select (query) HTML elements and perform "actions" on them.
- syntax : `$(selector).action()`

- document ready event : prevents jquery code form running before document finishes loading

```
$(document).ready(function(){ // jQuery methods go here... }
Alternate Syntax : $(function(){ // jQuery methods go here... }
```

- jquery selectors : without prefix -> tag name, with prefix '#' -> id name , with prefix '#' -> class name
- more jquery selectors :

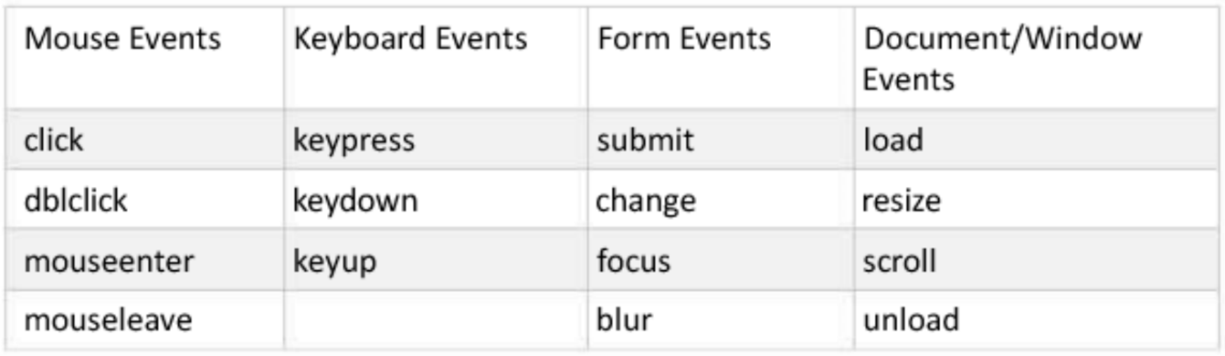
More jquery Selectors :



Syntax	Description
$("*")	Selects all elements
$(this)	Selects the current HTML element
$("p.intro")	Selects all <p> elements with class="intro"
$("p:first")	Selects the first <p> element
$("ul li:first")	Selects the first <li> element of the first <ul>
$("ul li:first-child")	Selects the first <li> element of every <ul>
$("[href]")	Selects all elements with an href attribute
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"
$("tr:even")	Selects all even <tr> elements
$("tr:odd")	Selects all odd <tr> elements

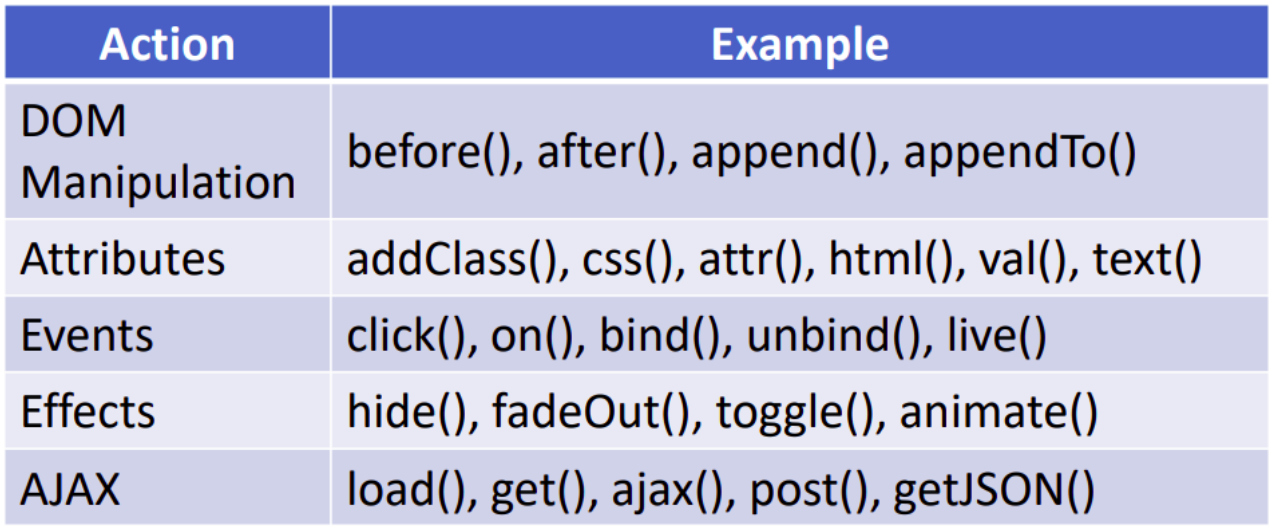
- jquery effects : .hide(),.show(),.toggle() with speed nd callback as optional parameters for all 3
- jquery events

jQueryEvent Methods:



Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

- jquery methods



Action	Example
DOM Manipulation	before(), after(), append(), appendTo()
Attributes	addClass(), css(), attr(), html(), val(), text()
Events	click(), on(), bind(), unbind(), live()
Effects	hide(), fadeOut(), toggle(), animate()
AJAX	load(), get(), ajax(), post(), getJSON()

- example code snippets

```
$(“p”).appendTo(“#contents”); //moves paragraph to div with id contents

Example : Setting
$(“img.logo”).attr(“align”, “left”);
$(“p.copyright”).html(“© 2009 ajaxray”);
$(“input#name”).val(“Spiderman”);

Example : Getting
var allignment = $(“img.logo”).attr(“align”);
var copyright = $(“p.copyright”).html();
var username = $(“input#name”).val();

Events
$(document).ready(function(){
$(“#message”).click(function(){ $(this).hide();
})
});

Effects
$(“a#show-cart”).click(function(){ $(“#cart”).slideToggle(“slow”); })

use the animate method to build custom animations
$("#showdown").click(function(){
$("#my-div").animate({width: "70%", opacity: 0.4, fontSize: "3em" }, 1200 );
});
```

- note that $(“pelem”).append(elements) adds as child elements to pelem while $(“pelem”).after(elements) adds as sibling elements to pelem (after pelem)

- chaining cant be done getattr functions as they usually return strings
- within any event handler function this element refers to the element for which the handler is called

Promises

- A callback function is executed after the current effect is 100% finished.but lot of nesting and callback hell and it isnt async.so use promise
- A promise is used to handle the asynchronous result of an operation.it represents a value not known yet.it has 3 possible states Pending – Fulfilled – Rejected.
- The Promise object is created using the new keyword and contains the an executor function which has a resolve and a reject callback. As the names imply, each of these callbacks returns a value with the reject callback returning an error object

```
let promise = new Promise(function(resolve, reject) {
  // Code to execute
});
```

> A Promise executor should call only one `resolve` or one `reject`. Once one state is changed (pending => fulfilled or pending => rejected), that's all. Any further calls to `resolve` or `reject` will be ignored.

- A consumer function (that uses an outcome of the promise) should get notified when the executor function is done with either resolving (success) or rejecting (error).
- .then is called on promise object to handle resolve/reject (usually used to handle succesful operation) and .catch is used to handle error. .then usually returns promise object so it can be chained
- .finally performs cleanups like stopping a loader, closing a live connection, and will be called irrespective of whether a promise `resolve`s or `reject`
- code sample

```
var weather;
const date = new Promise(
        function(resolve, reject) {
                weather = true; //usually a API call
                if (weather) {
                        const dateDetails = {
                        name: 'Cubana Restaurant',
                        location: '55th Street',
                        table: 5
                        };
                        resolve(dateDetails)
                } else {
                reject(new Error('Bad weather'))
                }
        }
);
date
.then(function(done) {
console.log('We are going on a date!')
console.log(done)
})
.catch(function(error) {
console.log(error.message)
})
```

XHR & XMLHttpRequests

- Asynchronous applications, upon user action, updates a part of the page without reloading the entire page
- create an XHR object using `var xhr = new XMLHttpRequest();`
- XMLHttpRequest Methods
  - `abort()` : cancels request
  - `getAllResponseHeaders()` : Returns the complete set of HTTP headers as a string
  - `getResponseHeader( headerName )` : Returns the value of the specified HTTP header.
  - `open(method,URL,async=true/false)` : Specifies the method, URL, and other optional attributes of a request.
  - `send(content)` : sends request (for GET content is null as no body to be sent)
  - `setRequestHeader( label, value )` : Adds a label/value pair to the HTTP header to be sent
- XMLHttpRequest Properties
  - `onreadystatechange` : An event handler for an event that fires at every state change.
  - `readyState` : The readyState property defines the current state of the XMLHttpRequest object. possible values of readyState are
    - 0 - request not initialised (created XMLHttpRequest object)
    - 1 - request set up (called open())
    - 2 - request sent (send() called)
    - 3 - request in progress (browser-server connection established but no response yet)
    - 4 - request is completed and response is received

    - `responseText` : Returns the response as a string
    - `responseXML` : Returns the response as XML. This property returns an XML document object,
    - `status` : Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").
    - `statusText` : Returns the status as a string (e.g., "Not Found" or "OK").
- code ex :

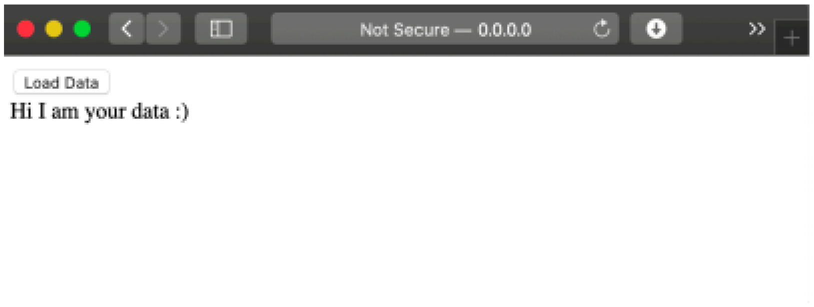
JavaScript

```javascript
function loadData() {
    let xhr = new XMLHttpRequest();

    // true - asynchronous
    xhr.open("get", "sample.txt", true);
    xhr.onreadystatechange = showData;
    // Default - text
    xhr.responseType = 'text';
    xhr.send(null);
}
```

```javascript

function showData() {
    // this refers to xhr object
    if (this.readyState == 4 && this.status == 200) {
        // this.response or responseText or responseXML
        document.querySelector('#container').innerHTML = this.responseText;
    }
}
```

HTML

```html
<button onclick="loadData()">Load Data</button>
<div id="container"></div>
```



Hi I am your data :)

sample.txt

```
Hi I am your data :)
```

AJAX

- load method : loads data from a server and puts the data into the selected element.
  - $(selector).load(URL, data, callback(responseTxt,statusTxt,xhr)) : callback executed after load completed
  - url parameter can also be jquery selector
  - callback function has 3 parameters
    - responseTxt - contains the resulting content if the call succeed
    - statusTXT - contains the status of the call
    - xhr - contains the XMLHttpRequest object
- ajax get method : $.get(URL,callback);
- ajax post method : $.post(URL,data,callback);
- ajax request : $.ajax(settings) or $.ajax(url, settings) Used for sending an Ajax request. The settings is an object of key-value pairs. The frequently-used keys are:
  - url: The request URL, which can be placed outside the settings in the latter form.
  - type: GET or POST.
  - data: Request parameters (name=value pairs). Can be expressed as an object (e.g., {name:"peter", msg:"hello"}), or query string (e.g., "name=peter&msg=hello").
  - dataType: Expected response data type, such as text, xml, json, script or html.
  - headers: an object for request header key-value pairs. The header XRequested-With:XMLHttpRequest is always added.

```
$.ajax({url: "demo_test.txt", success: function(result){
$("#div1").html(result);
}});
 $.ajax('/jquery/submitData', {
        type: 'POST', // http method
        data: { myData: 'This is my data.' }, // data to submit
        success: function (data, status, xhr) { // success callback function
                $('p').append('status: ' + status + ', data: ' + data);
                },
        error: function (jqXhr, textStatus, errorMessage) {
                $('p').append('Error' + errorMessage);
                }
})
```

- fetch(url) : returns promise that rejects when a real failure occurs such as a web browser timeout, a loss of network connection, and a CORS violation.

```
fetch('/readme.txt') .then(response => response.text()) .then(data =>
console.log(data));
using async/await
async function fetchText() {
        let response = await fetch('/readme.txt');
        let data = await response.text();
         console.log(data);
 }
```