



WEEK-9 NOTES

▼ Callback Functions in jQuery

A **callback function** is a function that is **passed as an argument** to another function and is **executed later**.

Example (Plain JavaScript)

```
function greet(callback) {  
    callback();  
}  
  
greet(function () {  
    console.log("Hello");  
});
```

Why Callbacks Are Needed in jQuery

jQuery deals with:

- User actions (click, keypress)
- Animations
- AJAX requests
- Timed operations

These **do not finish immediately**, so callbacks tell jQuery:

“Run this code after something happens.”

Basic Callback Structure in jQuery

```
$(selector).action(function () {  
    // callback code  
});
```

Example:

```
$("#btn").click(function () {  
    alert("Button clicked");  
});
```

`function () {}` is the **callback**

▼ jQuery Deferred & Promise

Why Deferred / Promise Exists

Old Way: Callback Hell

```
loadUser(function () {  
    loadOrders(function () {  
        loadPayments(function () {  
            console.log("Done");  
        });  
    });  
});
```

Problems:

- Hard to read
- Hard to manage errors
- No clean success/failure flow

Deferred represents a task that will finish in the future

Promise is a read-only view of that task

- **Deferred** → controller (resolve / reject)
- **Promise** → listener (then / done / fail)

Deferred States

A `Deferred` object can be in **only ONE state at a time**:

State	Meaning
<code>pending</code>	Still running
<code>resolved</code>	Success
<code>rejected</code>	Failure

Creating a Deferred Object

```
let dfd = $.Deferred();
```

This gives you:

- `dfd.resolve()` → success
- `dfd.reject()` → failure
- `dfd.promise()` → safe promise

Deferred Methods (Controller Side)

Method	Purpose
<code>resolve(value)</code>	Mark success
<code>reject(error)</code>	Mark failure
<code>notify(data)</code>	Progress update
<code>state()</code>	Get current state
<code>promise()</code>	Return promise only

Promise Methods (Listener Side)

Method	Purpose
<code>.done()</code>	Success handler
<code>.fail()</code>	Error handler
<code>.always()</code>	Runs in both
<code>.then()</code>	Success + failure
<code>.progress()</code>	Progress updates

▼ AJAX

AJAX = Asynchronous JavaScript And XML

AJAX allows a web page to **send and receive data from a server without reloading the page.**

How AJAX Works



Before jQuery, AJAX was done using:

```
let xhr = new XMLHttpRequest();
xhr.open("GET", "data.txt", true);
xhr.send();

xhr.onload = function () {
  console.log(xhr.responseText);
};
```

AJAX Using jQuery

asic Syntax

```
$.ajax({
  url: "url",
  method: "GET",
```

```

success:function (response) {},
error:function () {}
});

```

Important AJAX Options

Option	Purpose
url	Server endpoint
method / type	GET, POST, PUT, DELETE
data	Data sent to server
dataType	Expected response
contentType	Format sent
success	Success callback
error	Error callback
beforeSend	Before request
complete	After request

AJAX Returns a Promise

```

$.ajax("/user")
.done(function (data) {
console.log("Success");
})
.fail(function () {
console.log("Error");
});

```

jQuery AJAX uses **Deferred/Promise**

Shortcuts of AJAX

Method	Purpose
\$.get()	GET request
\$.post()	POST request
\$.getJSON()	JSON GET

Method	Purpose
<code>\$.load()</code>	Load HTML

▼ GET vs POST vs PUT vs DELETE

GET – Fetch Data

Retrieve data from the server

Example

```
GET /users
GET /users/10
```

AJAX Example

```
$.ajax({
  url: "/users",
  method: "GET",
  success: function (data) {
    console.log(data);
  }
});
```

POST – Create Data

Send data to the server to create a new resource

Exsample

```
POST /users
```

AJAX Example

```
$.ajax({
  url: "/users",
  method: "POST",
```

```
data: {  
  name:"Nisharg",  
  email:"test@gmail.com"  
}  
});
```

PUT – Update (Replace) Data

Update an existing resource completely

Example

```
PUT /users/10
```

AJAX Example

```
$.ajax({  
  url:"/users/10",  
  method:"PUT",  
  data: {  
    name:"Updated Name",  
    email:"updated@gmail.com"  
  }  
});
```

DELETE – Remove Data

Delete a resource from the server

Example

```
DELETE /users/10
```

AJAX Example

```
$.ajax({  
  url:"/users/10",  
  method:"DELETE",  
});
```

```
success:function () {
  alert("User deleted");
}
});
```

▼ Working with JSON Data

JSON (JavaScript Object Notation) is a **lightweight data-interchange format** used to **store and exchange data** between client and server.

JSON Structure Examples

Simple JSON

```
{
  "id":1,
  "name":"Laptop",
  "price":50000
}
```

JSON Array

```
[
  {"id":1,"name":"Laptop"},
  {"id":2,"name":"Mobile"}
]
```

Nested JSON

```
{
  "user":{
    "name":"Nisharg",
    "address":{
```

```
"city":"Ahmedabad",
"pin":380001
}
}
}
```

Converting JSON

JSON.parse() → JSON → JavaScript Object

Used when **data comes from server**.

```
let jsonString ='{"name":"Nisharg","age":21};

let obj =JSON.parse(jsonString);

console.log(obj.name);// Nisharg
```

JSON.stringify() → JavaScript Object → JSON

Used when **sending data to server**.

```
let user = {name:"Nisharg",age:21 };

let jsonData =JSON.stringify(user);

console.log(jsonData);
// '{"name":"Nisharg","age":21}'
```

▼ Serialization & Deserialization

.serialize() (jQuery)

- Converts **form data into a URL-encoded string**
- Output format: `key=value&key=value`

Syntax

```
$("#formId").serialize();
```

Demo

```
<form id="myForm">
<input type="text" name="username" value="Nisharg">
<input type="email" name="email" value="nisharg@gmail.com">
</form>

<script>
let data = $("#myForm").serialize();
console.log(data);
</script>
```

Output

```
username=Nisharg&email=nisharg@gmail.com
```

.`serializeArray()` (jQuery)

- Converts form data into **array of objects**
- Each object has:
 - `name`
 - `value`

Syntax

```
$("#formId").serializeArray();
```

Demo

```
let arr = $("#myForm").serializeArray();
console.log(arr);
```

Output

```
[  
  {name:"username",value:"Nisharg"},  
  {name:"email",value:"nisharg@gmail.com"}  
]
```

JSON.stringify() (JavaScript)

- Converts **JavaScript object** → **JSON string**
- Used for:
 - Sending data to server
 - Storing in `localStorage`
- This is **Serialization**

Syntax

```
JSON.stringify(object);
```

Demo

```
let user = {  
  name:"Nisharg",  
  age:21  
};  
  
let jsonStr = JSON.stringify(user);  
console.log(jsonStr);
```

Output

```
{"name":"Nisharg","age":21}
```

```
JSON.parse() (JavaScript)
```

- Converts **JSON string** → **JavaScript object**

- Used when:
 - Receiving server response
 - Reading from storage
- This is **Deserialization**

Syntax

```
JSON.parse(jsonString);
```

Demo

```
let jsonStr ='{"course":"jQuery","level":"Beginner"}';  
  
let obj =JSON.parse(jsonStr);  
console.log(obj.course);
```

Output

```
jQuery
```