# jQuery Callback Functions — Notes

---

1. Definition:
- A callback is a function passed into another function to be executed after a task completes.
- Very common in jQuery for animations, events, AJAX, effects, etc.

2. Why use callbacks?
- JavaScript is asynchronous; callbacks let you run code after another action finishes.

3. Typical syntax:
```
$(selector).method(params, callbackFunction);
```

4. Effect callbacks:
- Most jQuery effect/animation methods accept a callback as the last argument.
- Example: .hide(), .show(), .slideUp(), .fadeOut().
5. Animation callbacks:
- .animate() supports step, complete, done, fail, always callbacks.

6. jQuery Callback list:
- $.Callbacks() gives a managed list of multiple callback functions.
- Methods: add(), remove(), fire(), fireWith().
7. Important:
- A callback runs only after the original task is complete.
- Without callbacks, code may run before the task finishes.

# jQuery Deferred & Promise — Notes

---

1. Deferred Object:

- Created using $.Deferred().
- Represents asynchronous work that may succeed or fail.
- Methods to change state:
  - deferred.resolve(value) → marks success and runs success callbacks.
  - deferred.reject(error) → marks failure and runs failure callbacks.
- Use deferred.state() to check "pending", "resolved", or "rejected".

2. Promise Object:
- Obtained from a Deferred via deferred.promise().
- Only allows attaching callbacks.
- Cannot resolve or reject — prevents outside interference.
- Promise methods:
  done(), fail(), always(), then(), progress(), state().

3. Callback Registration:
- .done(fn) — success handlers.
- .fail(fn) — failure handlers.
- .always(fn) — always run regardless of success/failure.
- .then(success, fail) — both in one call.

4. Usage pattern:
- Producer (code that performs async task) creates and resolves/rejects the deferred.
- Consumer (caller code) receives a promise and attaches callbacks to react.

5. Common uses:
- AJAX calls (.ajax returns promise).
- Delayed actions or animations.
- Chaining asynchronous tasks.

## AJAX & jQuery — Notes

---

1. What is AJAX?

# Ajax

Asynchronous JavaScript and XML (**Ajax**, or **AJAX**) is a web development technique in which a web app fetches content from the server by making asynchronous HTTP requests, and uses the new content to update the relevant parts of the page without requiring a full page load. This can make the page more responsive, because only the parts that need to be updated are requested.

Ajax can be used to create single-page apps, in which the entire web app consists of a single document, which uses Ajax to update its content as needed.

Initially Ajax was implemented using the `XMLHttpRequest` interface, but the `fetch()` API is more suitable for modern web applications: it is more powerful, more flexible, and integrates better with fundamental web app technologies such as service workers. Modern web frameworks also provide abstractions for Ajax.

This technique is so common in modern web development that the specific term "Ajax" is rarely used.

2. How AJAX works:
- Trigger an event (like button click)
- Create an async request
- Send to server
- Server processes and returns data
- Response is used to update part of the page.

3. jQuery AJAX:
- jQuery provides easy methods for AJAX requests.
- Methods include $.ajax(), $.get(), $.post(), .load(), etc.

4. $.ajax() method:
- Most powerful jQuery AJAX method.
- Accepts a settings object with options like url, type, data, dataType, success, error, etc.

5. Shorthand methods:
- $.get() — simple GET request.
- $.post() — POST request.
- $.getJSON() — GET request expecting JSON.

- .load() — loads HTML into a selected element.

6. AJAX callbacks:
- success — runs when request succeeds.
- error — runs when request fails.
- done(), fail(), always() can be used with the returned jqXHR.

7. Returning data:
- Server can return JSON, text, HTML, XML, etc.

Working with JSON in AJAX (jQuery)

1. JSON Basics:
- JSON (JavaScript Object Notation) is a lightweight text format for structuring data.
- AJAX with JSON is common for API communication.

2. Receiving (GET) JSON:
- Use $.ajax with dataType:"json" to automatically parse response into JS object.
- Shorthand: $.getJSON(url, callback).

3. Sending JSON to Server (POST/PUT):
- Convert JS object to JSON string using JSON.stringify().
- Set contentType:"application/json; charset=utf-8" to tell server you're sending JSON.
- Example:

```
$.ajax({
  url, type:"POST", contentType:"application/json",
  data: JSON.stringify({...}), dataType:"json"
});
```

4. Default Behavior:
- Without JSON.stringify(), jQuery URL-encodes data (form style), not JSON.
- contentType defaults to "application/x-www-form-urlencoded".

5. Why JSON?
- JSON is efficient, easy to parse, and widely adopted by modern APIs.

# Serialization and Deserialization — Notes

1. Serialization is converting an object's data into a format (string/bytes) that can be stored or transmitted.

2. Deserialization is the reverse — converting the serialized data back into an object.

3. Popular serialization formats:
   - JSON: human-readable and popular for web APIs.
   - XML: verbose and schema-oriented.
   - Binary formats: efficient (e.g., Protobuf, BSON).

4. JSON serialization in JavaScript:
   - Serialize: JSON.stringify(obj)
   - Deserialize: JSON.parse(jsonString)

5. Serialization does not save methods/functions — only data/state.

6. Important considerations:
   - Versioning of serialized format
   - Security risks of untrusted data