1. Call Stack, Callback Queue & Event Loop

Explanation:
JavaScript runs code synchronously using the Call Stack.
Async functions like setTimeout are handled by browser APIs.
The Event Loop checks when the Call Stack is empty and then executes async callbacks.

Code:
```
function runCode() {
  console.log("Start");

  setTimeout(() => {
    console.log("Async Task");
  }, 0);

  console.log("End");
}
runCode();
```

2. Callback Queue (setTimeout 0ms)

Explanation:
Even if delay is 0ms, JavaScript first completes main code.
Async callbacks wait in the Callback Queue until the Call Stack is empty.

Code:
```
console.log("Main Content");

setTimeout(() => {
  console.log("Ads Loaded");
}, 0);
```

3. JSON.parse()

Explanation:
API responses come as JSON strings.
JSON.parse converts a string into a usable JavaScript object.

Code:

```
const response = '{"id":101,"name":"Laptop","price":50000}';
const data = JSON.parse(response);
console.log(data.name);
```

4. JSON.stringify()

Explanation:
localStorage and APIs accept only strings.
JSON.stringify converts JavaScript objects into JSON strings.

Code:
```
const user = { username: "admin", role: "manager" };
const jsonData = JSON.stringify(user);
console.log(jsonData);
```

5. Array forEach()

Explanation:
forEach is used to perform an action on every element.
It does not return a new array.

Code:
```
const users = ["Amit", "Ravi", "Neha"];
users.forEach(user => {
  console.log("Welcome " + user);
});
```

6. Array map()

Explanation:
map creates a new array by modifying each element.
Original array remains unchanged.

Code:
```
const prices = [1000, 2000, 3000];
const discounted = prices.map(price => price * 0.9);
console.log(discounted);
```

7. Array filter()

Explanation:
filter returns only elements that satisfy a condition.

Code:
```
const products = [
  { name: "Phone", inStock: true },
  { name: "Laptop", inStock: false }
];

const available = products.filter(p => p.inStock);
console.log(available);
```

8. Array reduce()

Explanation:
reduce combines all values into a single value.
Commonly used for totals.

Code:
```
const cart = [500, 1000, 1500];
const total = cart.reduce((sum, price) => sum + price, 0);
console.log(total);
```

9. Function Declaration vs Arrow Function

Explanation:
Arrow functions have shorter syntax but do not have their own 'this' keyword.

Code:
```
function gstNormal(amount) {
  return amount * 0.18;
}

const gstArrow = amount => amount * 0.18;

console.log(gstNormal(1000));
console.log(gstArrow(1000));
```

10. Closures

Explanation:
A closure allows a function to remember variables from its outer scope.
Used for data security.

Code:
```
function createCounter() {
  let count = 0;
  return function () {
    count++;
    return count;
  };
}

const counter = createCounter();
console.log(counter());
console.log(counter());
```

11. Callback Function

Explanation:
A callback is a function passed as an argument to another function.
It executes after an operation completes.

Code:
```
function validateUser(username, callback) {
  if (username === "admin") {
    callback("Login Successful");
  } else {
    callback("Login Failed");
  }
}

validateUser("admin", msg => console.log(msg));
```

12. Callback Hell

Explanation:
Multiple nested callbacks make code unreadable and hard to manage.
This is known as Callback Hell.

Code:

```
validate(() => {
  payment(() => {
    invoice(() => {
      console.log("Done");
    });
  });
});
```

13. Async / Await

Explanation:
async/await makes asynchronous code look synchronous.
Improves readability and error handling.

Code:
```
async function fetchData() {
  const response = await fetch(url);
  const data = await response.json();
  console.log(data);
}
```

14. DOM Manipulation

Explanation:
JavaScript can dynamically create and add HTML elements to the page.

Code:
```
const li = document.createElement("li");
li.innerText = "New Item";
document.querySelector("ul").appendChild(li);
```

15. Event Listener

Explanation:
addEventListener listens for user actions like submit or click.

Code:
```
form.addEventListener("submit", e => {
  e.preventDefault();
  alert("Form Submitted");
});
```