Notes

1. DNS (Domain Name System)

1. DNS translates human-readable domain names into IP addresses for computers to locate servers.
2. It functions like the Internet's phonebook system.
3. It uses hierarchical servers such as root, TLD, and authoritative DNS.
4. Common record types include A, MX, CNAME, and TXT records.
5. DNS improves accessibility and simplifies web navigation for users.

```
Address: 142.250.183.110, Family: IPv4
IPv4 Addresses: [ '142.250.183.110', '142.250.183.102', ... ]
Hostnames for 8.8.8.8: [ 'dns.google' ]
```

---

2. REPL (Read-Eval-Print Loop)

1. REPL reads user input, evaluates it, prints the result, and loops again.
2. It provides an interactive programming environment for testing code snippets.
3. Used in languages like JavaScript (Node.js), Python, and Ruby.
4. Useful for debugging, experimenting, and learning programming interactively.
5. Promotes rapid development by providing immediate feedback on executed commands.

```
$ node
> 2 + 3
5
> console.log("Hello World")
Hello World
```

---

3. React Props

1. Props are short for “properties” used to pass data between React components.
2. They make components dynamic, reusable, and configurable from outside.
3. Props are immutable inside child components, ensuring predictable behavior.
4. They are passed as attributes in JSX during component rendering.
5. Props improve modularity and reusability of UI components efficiently.

```
function Welcome(props) {
  return <h1>Hello, {props.name}!</h1>;
}

function App() {
  return <Welcome name="John" />;
}
```

---

4. Cookies

1. Cookies are small text files stored in the user’s browser by websites.
2. They store session data, preferences, and tracking information temporarily.
3. Cookies can be either session-based or persistent with expiration dates.
4. They are sent with every HTTP request to the server automatically.
5. Used for authentication, personalization, and maintaining user login states.

```
document.cookie = "username=John; expires=Fri, 31 Dec 2025 12:00:00
UTC; path=/";
console.log(document.cookie);
```

---

5. Session

1. A session is a temporary interaction between a client and server.
2. It stores user-specific data during active website usage periods.
3. Sessions end when the user logs out or the browser closes.
4. Session IDs are usually stored in cookies or URLs securely.
5. Sessions help maintain state and user continuity in web applications.

```
// Express.js Example
app.get("/login", (req, res) => {
  req.session.user = "John";
  res.send("Session started!");
});
```

---

6. Arrow Function

1. Arrow functions are a concise syntax for writing JavaScript functions.
2. Introduced in ES6, they use the `=>` arrow symbol.
3. They inherit `this` from their lexical scope, unlike regular functions.
4. Cannot be used as constructors or with `new` keyword.
5. Great for callbacks, array methods, and cleaner functional code.

```
// Normal Function
function multiply(a, b) {
  return a * b;
}

// Arrow Function
const multiplyArrow = (a, b) => a * b;
```

---

8. JSON (JavaScript Object Notation)

1. JSON is a lightweight text format for data exchange between systems.
2. It represents data as key-value pairs in a structured format.
3. JSON is language-independent but easy to use with JavaScript.
4. Commonly used for APIs, configurations, and data storage purposes.
5. It is human-readable and supports arrays, objects, numbers, and strings.

```
{
  "mapName": "Der Eisendrache",
  "releaseYear": 2016,
  "isEasterEggCompleted": true,
  "mainWeapons": ["Wrath of the Ancients", "Death Machine", "Haymaker
12"],
  "characterDetails": {
    "players": ["Dempsey", "Nikolai", "Takeo", "Richtofen"],
    "location": "Griffin Castle, Austria"
  },
  "lastPlayed": null
}
```

---

9. NPM (Node Package Manager)

1. NPM is the default package manager for Node.js applications.
2. It helps install, update, and manage JavaScript libraries easily.
3. Developers use `npm install` to add dependencies in projects.
4. It maintains `package.json` for tracking versions and project dependencies.
5. Enables sharing and reusing of code across multiple projects efficiently.

```
npm init -y         # Create package.json
npm install react   # Install React
npm run start       # Run project script
```

---

10. JSX (JavaScript XML)

1. JSX allows writing HTML-like syntax within JavaScript for React.
2. It simplifies component creation and improves readability significantly.
3. JSX is compiled into regular JavaScript using Babel transpiler.
4. Enables embedding expressions inside `{}` within markup easily.
5. Improves development speed and component-based UI structure consistency.

```
import React, { useState } from "react";

function Counter() {
  const [count, setCount] = useState(0); // state variable
  const increment = () => setCount(count + 1);
  const decrement = () => setCount(count - 1);
  const reset = () => setCount(0);

  return (
    <div>
      <h2>Counter: {count}</h2>
      <button onClick={increment}>+</button>
      <button onClick={decrement}>-</button>
      <button onClick={reset}>Reset</button>
    </div>
  );
}

export default Counter;
```

---

11. Buffers

- A temporary memory area used to store and manage binary data efficiently.
- Helps process files, images, and network packets directly in memory.
- Works seamlessly with streams for smooth data transfer operations.
- Stores incomplete data chunks before full data is received.
- Ideal when data arrives gradually from files or network sources.

```
// Create a Buffer from a string
const buf = Buffer.from('Hello');

// Display buffer content
console.log(buf);           // <Buffer 48 65 6c 6c 6f>
console.log(buf.toString()); // Hello
```

---

12. Streams

- A Stream is a continuous flow of data that can be read or written in small chunks instead of loading it all at once.

- Enable efficient reading and writing of large files or data sources.
- Reduce memory usage by processing data piece by piece.
- Commonly used for file handling, video, or network data transfer.

Types of streams:

1. Readable – For reading data (e.g., `fs.createReadStream()`).
2. Writable – For writing data (e.g., `fs.createWriteStream()`).
3. Duplex – Both readable and writable (e.g., network sockets).
4. Transform – Modify data while reading/writing (e.g., compression).

```
const fs = require('fs');

// Create a readable stream (read file in chunks)
const readable = fs.createReadStream('input.txt', 'utf8');

// Create a writable stream (write to another file)
const writable = fs.createWriteStream('output.txt');

// Pipe the data from input.txt to output.txt
readable.pipe(writable);

console.log('File copied using streams!');
```

---

13. Promises

- A Promise is an object in JavaScript that represents the eventual completion (or failure) of an asynchronous operation and its result
- It represents a value that may be available now or in the future.
- Has three states: *pending*, *fulfilled*, and *rejected*.
- It helps to avoid callback hell and makes asynchronous code easier to manage.
- Useful for asynchronous operations such as fetching or database queries.

```
// Creating a Promise
let myPromise = new Promise((resolve, reject) => {
  let success = true;
  setTimeout(() => {
    if (success) {
      resolve("Data fetched successfully!"); // handles success
    } else {
      reject("Error while fetching data.");   // handles failure
    }
  }, 2000);
});

// Consuming a Promise
myPromise
  .then((result) => {
    console.log(result);
  })
  .catch((error) => {
    console.log(error);
  })
  .finally(() => {
    console.log("Operation completed");
  });
```

14. Generator

- Express Generator is a CLI tool for creating Express app boilerplates quickly.
- It sets up the basic folder structure and configuration automatically.
- Saves time by generating routes, views, and public directories instantly.
- Helps developers start coding without manual setup of project files.
- Includes folders like /routes, /views, /public, and app.js.

```
npx express myApp
cd myApp
npm install
npm start
```

15. Authentication

- Authentication verifies a user's identity before granting access to resources.
- Common methods include passwords, tokens, and biometric verification.
- In Node.js, it's often implemented using `JWT` , `Passport.js` , or `OAuth` .
- Ensures that only authorized users can access protected routes or data.
- Helps maintain security and prevent unauthorized access to applications.
- **In Express:** Usually implemented using **middleware** (like `passport.js` , `jsonwebtoken` , or custom logic).
- **Example Flow:**
  1. User logs in → credentials verified.
  2. Server creates a **token** (e.g., `JWT`).
  3. Token is sent with each request to verify the user.

```
const jwt = require("jsonwebtoken");
app.post("/login", (req, res) => {
  const token = jwt.sign({ user: req.body.user }, "secret");
  res.json({ token });
});
```

---

16. HTTP (Hyper-Text Transfer Protocol)

- A set of rules used by browsers and servers to communicate and transfer web content (like HTML pages, images, videos).
- Standard web protocol for sending data between your browser and websites.
- Data is sent in plain text, which is not secure and can be read by others.
- Uses port 80. It is faster but offers no protection for information.
- Suitable for non-sensitive sites like blogs or news portals.
- The lack of encryption makes it vulnerable to attacks.

```
http://example.com
```

---

17. HTTPS (Hyper-Text Transfer Protocol Secure)

- The secure version of HTTP, which adds encryption (SSL/TLS) to protect the data being exchanged.
- Secure version of HTTP that encrypts all data exchanged for safety.
- Uses TLS/SSL encryption to protect data from being intercepted or stolen.
- Uses port 443. It verifies website identity with an SSL certificate and handshake.
- Essential for banks, e-commerce, and any site handling personal data.
- Shows a padlock in the address bar to indicate a secure connection.

```
https://example.com
```

---

18. Event Loop