IAD BLANKS

Lec # 3

1. The body indeed contains two main elements in our example:
(i) Header : Displays the heading at the top.
(ii) A div containing button element is used to group and structure the button independently.

2. The variables defined under the :root selector are global .(local/global)

3. --button-text-color would be a local variable, available only within the button selector and its children.

4. Just like http://, the file:/// protocol is also a scheme in the broader URI structure.

5. Just as HTTP URLs point to resources on the web , file URLs point to resources on the local system .

6. The default security policy enforced by browsers is called the Same-Origin Policy,which blocks cross-origin requests between different origins. Same-Origin Policy
prevents scripts on one website from making requests to another website's domain unless explicitly allowed by the server.

7. CORS stands for Cross-Origin Resource Sharing, a mechanism that allows or restricts sharing between different domains. CORS ensures that a client (like a browser) can securely request resources (data, scripts, APIs) from a server hosted on a different origin.

8. If you see a browser error like: “Access to fetch at 'https://api.example.com/data' from origin 'http://localhost:3000' has been blocked by CORS policy”. This indicates that the

backend server does not include the appropriate CORS header in its response.

9. Use a lightweight development server (VSCode Live Server, http-server, or built-in Python server python -m http.server, Node.js, Python (Django/Flask), PHP, Ruby on Rails).
(i) Then Access the app at http://localhost:8080.
(ii) http://localhost:8080 is a local server where your app is served(via a server running on port 8080).

10. React (aka React.js or ReactJS) is an open-source front-end JavaScript library that is used for building composable user interfaces, especially for single page application.

11. It is used for handling view layer for web and mobile apps based on components in a declarative manner.

12. React was created by Jordan Walke, a software engineer working for Facebook. React was first deployed on a facebook news feed. in 2011 and on a mobile weight in 2012.

13. Virtual DOM is a light weight copy of the original DOM

- Virtual DOM is maintained by React Libraries.
- After manipulation Virtual DOM only re-renders changed elements.
- In Virtual DOM updates are lightweight.
- In Virtual DOM performance is optimized and UX is optimised
- Virtual DOM is highly efficient as it performs diffing .

14. Real DOM is a true representation of HTML elements

- Real DOM is maintained by the browser after parsing HTML elements
- After manipulation Real DOM, it re-render the entire DOM .
- In Real DOM Updates are heavyweight
- In Real DOM performance is slow and the UX quality is low.

- Real DOM less efficient due to re-rendering of DOM After every change.

15. props (short for "properties") are passed to a component by its parent component and are immutable meaning that they cannot be modified by the own component itself. props acts as an argument for a function. Also,props can be used to customize the behavior of a component and to pass data between components. The components become dynamic with the usage of props.

16. The state entity is managed by the component itself and can be modified using the setter(setState() for class components) function. Unlike props, state can be modified by the component and is used to manage the internal state of the component. i.e. state acts as a component's memory. Moreover, changes in the state trigger a re-render of component.The components become interactive with the usage of state alone.

Lec # 4

1. V8 is Google/runtime JavaScript engine (used in Chrome and other browsers).

2. SpiderMonkey is Mozella engine and used in Firefox.

3. Chakra is a Microsoft Edge runtime engine. It was originally used in browser. In December 2018,Microsoft Edge decided to adopt Chromium (Google's open- source browser project) as its JavaScript runtime in the browser. Chakra is still powering Windows applications that are written in HTML/CSS and JavaScript.

4. While the browser is the most obvious usage scenario for JavaScript runtime environments, they

are also used in other areas such as microcontrollers. Most importantly for us: the Node.js platform we cover soon is built on top of V8 .

5. Today's JavaScript engines interpreter and compile by employing so-called just-in-time (JIT) compilation. This means that JavaScript code that is run repeatedly such as often-called functions is eventually compile and no longer interpreted. This article by Lin Clark explains this in more detail for those that want to know more.

6. It is worthwhile to know that many languages compile into JavaScript. Three of the most well-known languages are Type-script , Dart and Cofee-script ; all three fill one or more gaps of the original JavaScript language. Here is one example of what TypeScript offers: JavaScript is a dynamic language , this means that you have no way of enforcing a certain type on a variable. Instead, a variable can hold any type, a String, a Number, an Array ... but of course often you know what you want the type to be (for instance function parameters). It is useful to provide this knowledge upfront. TypeScript allows you to do that, by enabling static binding(static/dynamic) type checking .

7. Scoping is the context in which values and expressions are accesible/visible. In contrast to other languages, JavaScript has very few scopes:local ,global and block.
A block is used to group a number of statements together with a curly braces{}(Hint: syntax) .

8. The difference between let and const is that const does not allow the reassignment or redeclaration of a variable. The originally assigned element though can change.

9. ANSWER:

- Local
- Global

here/how	Scope
var declared within a function	?
var declared outside of a function	?
let (ES6)	?
const (ES6)	?
variable declaration without var/let/const	?

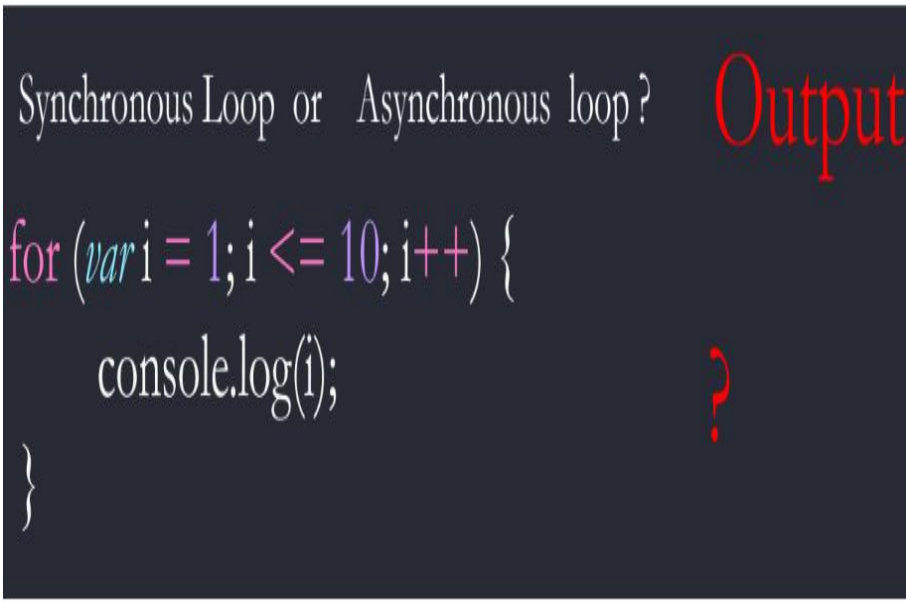
- Block
- Block
- Global

10. ANSWER
- Ok
- Ok
- Not ok
- Ok
- Ok

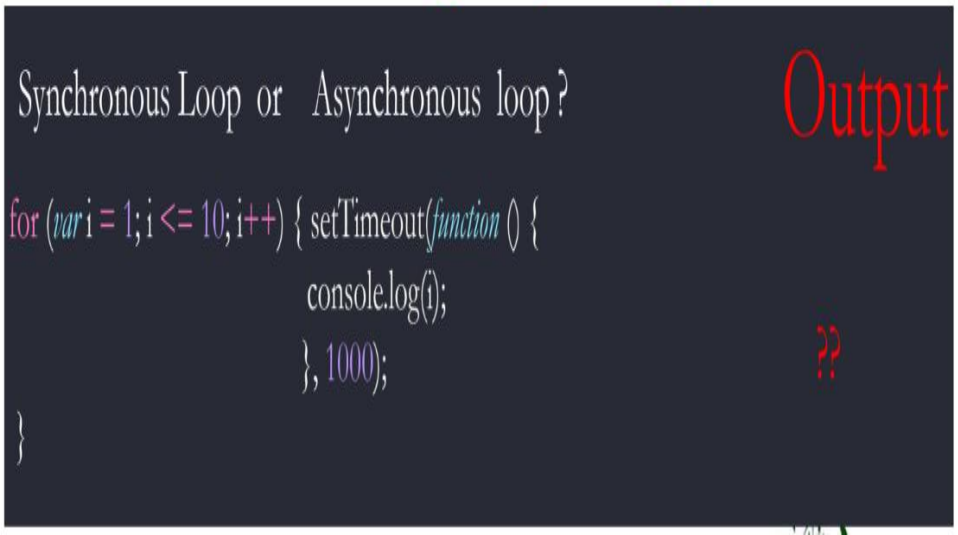
Code	COMMENT or ERROR
let a = [1 2 3]; const b = [4 5 6]	//array with 3 numbers //array with 3 numbers
a = "hello world";	?
b = "hello world";	?
b[0] = -1;	?
console.log(b);	?

11. Before ES6 there was no block scope, means there is no block scope (refer to previous scope table) we only had two scopes available: local and global. Having only two scopes available resulted in code behavior that does not always seem intuitive.

12. Output:
1
2
3
4
5
6
7
8
9
10



13. Output:
11
11
11
11
11
11



11
11
11
11

14. In the code above, var i has scope function level, but we actually need it to be of block level scope such that every function has its own independent copy of it.

15. printing undefined instead of null.

16. Waiting for set timeout between print outs one by one.

17. A stack is a data structure that JavaScript uses to store static data. Static data is data where the engine knows the size at compile time. In JavaScript, this includes primitive values (strings, numbers, booleans, undefined, and null) and references, which point to objects and functions.

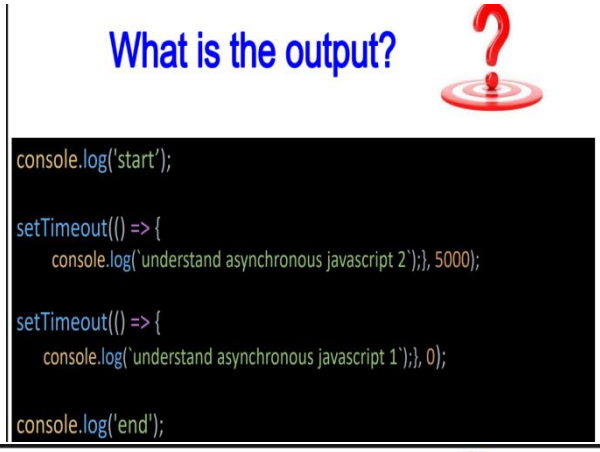
18. The heap is a different space for storing data where JavaScript stores objects and arrays .Unlike the stack, the engine doesn't allocate a fixed amount of memory for these objects. Instead, more space will be allocated as needed.

19. All variables first point to the stack. In case it's a non-primitive value, the stack contains a reference to the object in the heap. The memory of the heap is not ordered in any particular way, which is why we need to keep a reference to it in the stack. You can think of references as addresses and the objects in the heap as houses that these addresses belong to.

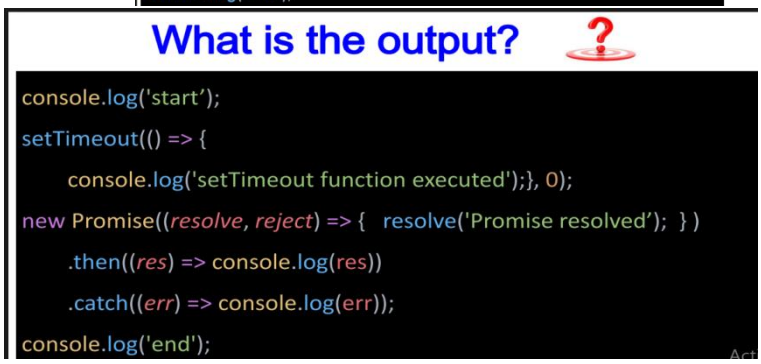
20. Node is all about asynchronous function execution. All these async callbacks doesn’t run immediately and are going to run some time later , so can’t be pushed immediately inside the callstack unlike synchronous
functions like console.log(),mathematical operations.

21. output

start
end
understand asynchronous javascript 1
understand asynchronous javascript 2



22. Output
start
end
Promise resolved
setTimeout function executed



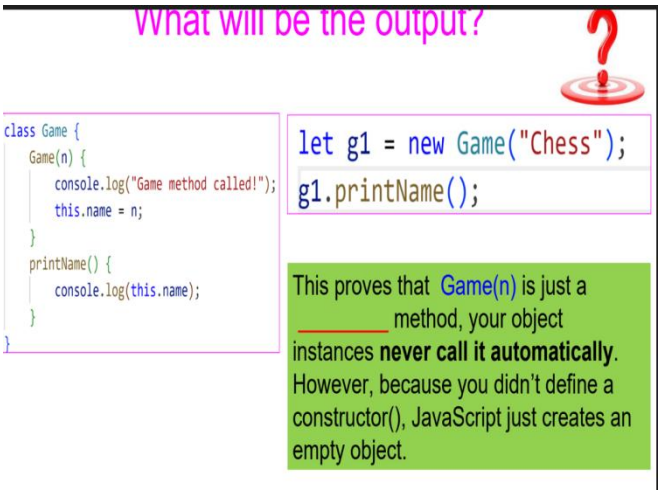
Lec # 5

1. Node.js is an open source, cross platform JavaScript runtime environment that allows developers to execute JavaScript code on the server side. It was released in 2009 by Ryan Dahl and is built on the Chrome V8 JavaScript engine. Node.js enables the development of scalable and efficient network applications by allowing JavaScript to run outside of a browser .

2. The function inside a class after ES6 allow multiple instances to share the same method, improving memory efficiency.

3. Output

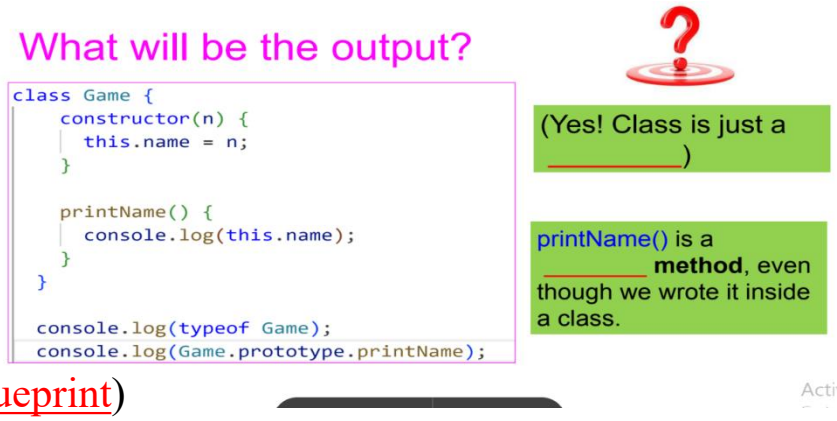
Undefined



4. This proves that Game(n) is just a regular method, your object instances never call it automatically.However, because you didn't define a constructor(), JavaScript just creates an empty object.

5. Output

function
[Function: printName]



6. (Yes! Class is just a blueprint)

7. printName() is a regular method, even though we wrote it inside a class.

8. Before ES6 (2015), developers had to explicitly use prototype to share methods.

9. ES6 class introduced a more readable way to define class, making prototype less visible.

10. But internally, JavaScript still uses prototype and ES6 class update just hides the complexity .

11. Understanding prototypes helps you debug and optimize JS code better.

12. Prototypes allow method sharing & memory efficiency. Without prototypes, every object would have its own copy of methods, leading to huge memory waste.

13. Three (3) main pillars around which the JS language is organized:scope/closure and prototype/objects and type/coercion.

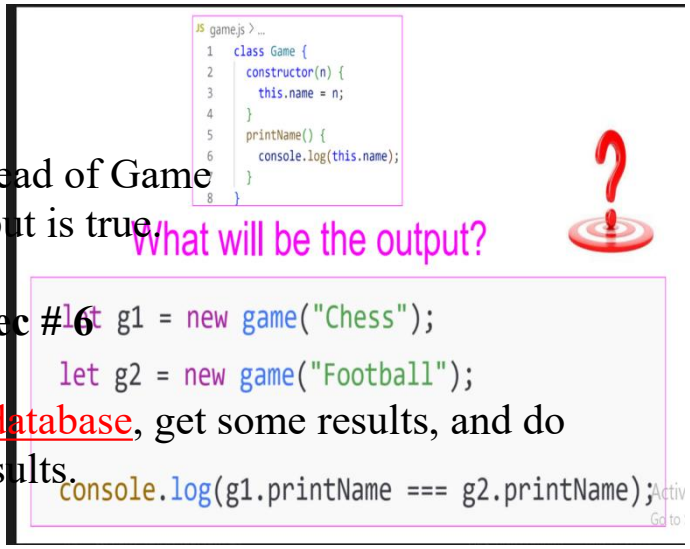
14. printName is (shared /not shared)? not shared

15. Every time new Game(...) is called, a new object is created in memory.

16. What is a Shared Function (Prototype Method)?
Instead of creating a new function for every object,we can store it once in the memory and let all instances access the same function.

17. Output

Undefine because game is
Not defined we use game .Instead of Game
But if there is Game then output is true.



Lec #6

1. Back-end will connect to a database, get some results, and do some processing with those results.

2. Back-end will expose restful API that front-end will use to interact with the DB
3. Server will accept HTTP requests from client/frontend app and use crud operations to interact with the DB.

4. Express is a minimal and flexible web application framework for Node.js. It is designed for building web applications and web services .

5. REST stands for Representation State Transfer.

6. It is an architectural style for designing networked

applications that relies on stateless (Hint: stateless/stateful) communication and standard https methods like GET, POST, PUT, DELETE, PATCH, etc. RESTful APIs follow a set of principles that make applications scalable, flexible,and easy to use.

7. Stateless : Each request from a client contains all the information needed for processing, and the server does not store client state.

8. Client server Architecture: The client and server are independent; they communicate via HTTP requests and responses.

9. Uniform interface: Consistent resource identification (URLs), resource manipulation via representations (JSON, XML), and standard status codes.

10. Cache ability: Responses can be cached to improve performance.
11. Layer architecture System: The client does not need to know whether it is directly communicating with the server or an intermediary.

12. The concept of a RESTful API is an architectural style—it's a set of design principles and constraints for how web service (hint: app/service) should be built—while Express is just one of many tools (framework) you can use to implement that design. In other words, RESTful API design is independent of the technology used to create/built the server. You can build a RESTful API using Express, Django, Flask, or any other framework; the key is that you follow REST principles such as stateless server, a uniform interface, and proper use of https methods and codes. Express doesn't force you to build a RESTful API—it merely provides a environment to implement RESTful design if you choose to do so.

13. RESTful APIs are web services in their purest form means the service exposes an interface (usually via HTTP and data formats like JSON) for other software to consume.

14. Web applications can—and often do—use RESTful APIs as the communication/interface layer between the client and the server. This separation aligns with the client-server architecture, ensuring clarity and maintainability in your project's design.

15. All web applications have a server side (which behaves like a (web service), not every web service is a complete web application .
16. Name few real-world examples where you have a web service that isn't a full web application—meaning it exposes functionality via APIs without providing a complete interface (UI).

17. Payment gateways, weather APIs, geocoding APIs, and cloud storage APIs) are considered web services because they make HTTP requests to their API endpoint/interface for data exchange without a built-in user interface. They often use JSON because it's popular today, but JSON is not an inherent requirement of the web service model. JSON is not an inherent requirement of the web service model

18. Endpoints are Fundamental: Every web API exposes endpoints (specific URLs) that represent resources or services. Clients make http request (using methods like GET, POST, etc.) to these endpoints, and the server responds with data (commonly in JSON, XML, etc.).

19. In many cases, an API is a web service without a user interface.

20. Additionally, the term "API" can also refer to function libraries or SDKs that expose a set of function calls. These aren't necessorily part of a networked

client-server architecture but are still considered API's because they define how different software components interactly .

21. When discussing the server side alone, the focus is on REST principles—defining uniform interface, stateless ,JSON , and ensuring stateless interactions. Later, when you integrate the front-end, you'll see how the REST API serves as the communication bridge between the server and the React application. This separation lets you build and test the backend independently before connecting it with the client side.

22. The base URL is a key component in our RESTful API design. It ensures that both the front-end and back-end consistently generate/create/build and resolve URLs. For instance, our modern front-end (built with frameworks like React or Angular) will use this base URL to construct API request, while our Express server uses it to define its routing logic. This uniformity is crucial for seamless communication between client and server.

23. In our project, the base URL (stored in .env files) is fixed office address for our server (Hint: server/client ?) .
The RESTful API then provides the "rooms" or endpoint inside that building, which the front-end (like a React app) use/call to retrieve data. This clear separation ensures that even as you build and deploy your server-side logic in a RESTful style, both components—front-end and back-end—consistently know where to send request (Hint: data/request ?) and how to URL URLs.

24. A model represents the data for the application.

25. The view is the visual/render representation of that data.

26. A controller request/input data takes on the view and translates that to changes in the model."

27. In a traditional backend MVC setup, the **View (V)** is responsible for rendering UI using templating engines like **EJS (*Embedded JavaScript*), Pug**, or **Handlebars**.

28. But when using **React for the frontend**, React itself handles UI **rendering.**

29. The backend now **only provides data via APIs (JSON responses)** instead of rendering HTML.

30. In a **React + Express** setup, the **backend only serves data (Model + Controller)**, while React takes over the **View layer**.

Lec # 7

1. app.get("/api/projects(i.e. root route)", (req, res) => res.json("Server is running! Welcome to the Capstone Project API."););

2. app.get("/api/projects", **callback function**); // General structure

Correct option: B. callback function

Why? → Because it is executed **later** (when an HTTP request is received).

3. The Arrow Functions in JavaScript helps us to create anonymous function or methods i.e.functions without name As they do not have any names, the arrow makes the syntax shorter/more readable

4. ()=>{} are a concise way of writing anonymous, lexically scoped functions in ES6.

5. The () =>{} can contain other ()=>{} or also regular functions.

6. The () =>{} accomplishes the same result as regular function with fewer lines of code/statement.

7. The () =>{} automatically binds this object to the surrounding code's context.

8. The value of this keyword inside the () =>{} is not dependent on how they are called or how they are defined. It depends only on its enclosing context.

9. If the () =>{} is used as an inner functions (hint: inner/outer) this refers to the parent (hint: parent/global/surrounding) scope in which it is defined.

10. The primary use of arrow functions in the frontend is to attach functionality to UI interactions, such as: clicks, form submissions, and input changes.

11. The error **"Cannot GET /"** happens because your server does not define a route for the root path.

12. app.get("/(route)", (req, res) => {
  res.send("Welcome to the Projects API! Use /api/projects to fetch data.");
});

13. This is the arrow function (also called a request handler) that gets executed when a client , such as a frontend app making a request to " https://localhost:5000 /api/projects", hits the API.

14. Say a React frontend makes a request like this:
fetch("**full url**")
.then(**res => res.json()**) // is a cb

.then(**data => console.log(data)**) // is a cb

.then(res => res.json()): The first .then() is a **callback** to convert the response to JSON.

.then(data => console.log(data)): The second .then() handles the actual data received from the backend.

15. A React component is a function that returns a piece of html, which can be as straightforward as a fragment of user-interface. Consider the creation of a component that renders a navigation bar.

16. The mixture of JavaScript with HTML tags might seem strange (it's called jsx, a syntax extension to JavaScript. For those using typescript, a similar syntax called TSX is used). To make this code functional, a compiler is required to translate the JSX into valid Javascript code.

17. It's important to note that when your component first renders and invokes useState, the initial State is the returned state from useState .

18. What will be the output and WHY?

```
const wizard = {
    magicNumber: 50,
    castSpell: () => {
        console.log(this.magicNumber);
    }
};

wizard.castSpell(); //output :Undefined
```

Undefined because Arrow functions do not have their own “this” they inherit this from the outer scope (where the object is defined), not from the wizard object.

This is the correct way

```
const wizard = {
    magicNumber: 50,
    castSpell: function () {
        console.log(this.magicNumber);
    }
};

wizard.castSpell(); // ✓ 50
```

19. What will be the output and Why?

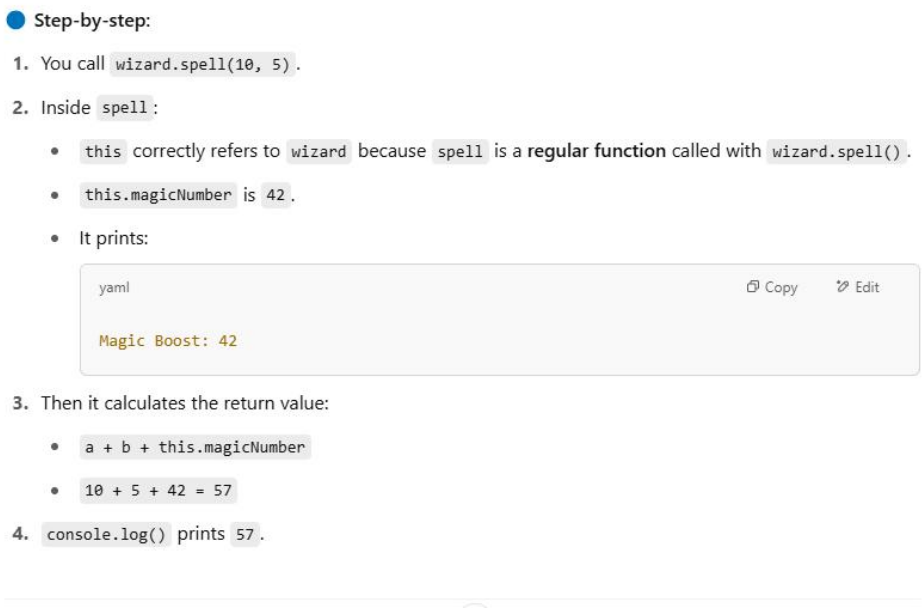
```
const hero ={
    name:"Thor",
    greet:function(){
        const inner = function(){
            console.log(`Hello,I am ${this.name}`);
        }
        inner();
    }
};
hero.greet();  //output Hello I am undefined
```

🔵 **Why it prints** `Hello, I am undefined`:

- `greet` is a **regular function** — so `this` inside `greet` points to `hero` ✅.
- But `inner` is **also a regular function**, and when **you call** `inner()` **directly**, `this` **inside** `inner` **becomes global** (i.e., `window` in browser, `undefined` in strict mode).
- `this.name` **is** `undefined` because the global object doesn't have a `name` property.

20. Whatt will be th output and Why?

```
const wizard={
   magicNumber:42,

   spell:function(a,b){
         console.log(`Magic Boost: ${this.magicNumber}`);
         return a+b+this.magicNumber

   }
};
console.log(wizard.spell(10,5));
```



🔵 **Step-by-step:**

1. You call `wizard.spell(10, 5)`.
2. Inside `spell`:
   - `this` correctly refers to `wizard` because `spell` is a **regular function** called with `wizard.spell()`.
   - `this.magicNumber` is `42`.
   - It prints:

     yaml — Copy — Edit

     ```
     Magic Boost: 42
     ```
3. Then it calculates the return value:
   - `a + b + this.magicNumber`
   - `10 + 5 + 42 = 57`
4. `console.log()` prints `57`.

Lec # 8

1. Props are basically data that flows from one to another component as a parameters. Props can not be modified means in a component.

2. React Props are like function arguments in JavaScript and attributes in HTML. Component use this data to generate dynamic html elements.

3. Props are passed to components via HTML components (Hint: JSX/HTML) attributes .

4. React components has a built-in state object which is private (Hint: public/private) to a component. State can not be accessed from outside of the class.However it can be passed as an argument to another component.

5. Whenever state is changed component calls the render function to render html elements.

6. props are passed in, and they cannot change (read only)

7. props can be passed to component from outside

8. props are public(Hint: public/private)

9. Props have better performance

10. State can be changed mutable(writable/mutable) using events or lifecycle methods within a component

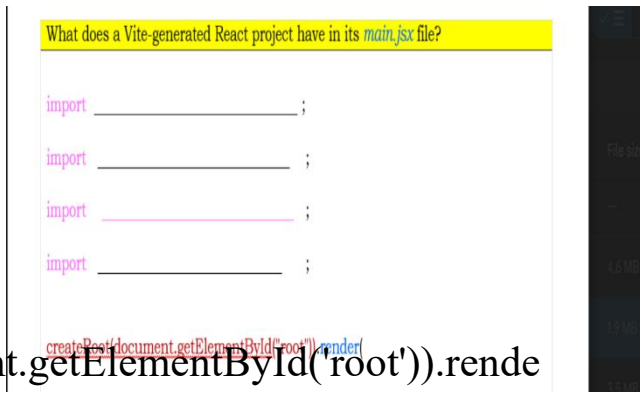
11. State can be passed to child components from inside a component but as prop

12. State is private. (Hint: public/private)

13. State has worse performance

14.

```
import React from 'react';
import ReactDOM from
'react-dom/client';
import App from './App.jsx';
import './index.css';

ReactDOM.createRoot(document.getElementById('root')).rende
r(
  <React.StrictMode>
    <App />
```



```
  </React.StrictMode>,
);
```

15. JavaScript's this is not automatically bound inside class (Hint: class/object) methods.

16. **When customizing a Vite-generated React app, what are some things you need to do?**
Change the title in index.html, remove the React SVG file, clean up the App.css file, replace the contents of the App.jsx file, and update the main.jsx file.

17. The React way is to keep the state high up in the tree and pass it down to child components that will do the rendering.

18. The useState line will create a state variable called "projects" and a function called "setProjects".

19. When setProjects updates this array, React will re-render the component to reflect the changes.

20. The stateful array of objects is stored in state.

21. The left-hand side (projects) is just a reference to that stateful data

22.
```
export default function ProjectList({ projects }) {

  let pList;

  if (projects.length > 0) {

    pList = projects.map(project => <div
key={project.id}>{project.name}</div>);

  } else {

    // If no projects exist, show a message

    pList = <p>No projects available.</p>;
```

```
  }

  // Return a <div> containing either the project list or the
message

  return <div>{pList}</div>;

}
```

23. Why do we need a new array each time? What happens if we don’t create a new array?

Creating a new array each time ensures **immutability**, which triggers a re-render in React; if we modify the original array, React won't detect changes and won't re-render the component.

24.Write equivalent (forEach, push) version of ProjectList.jsx. will it work?

```
export default function ProjectList({ projects }) {

  let pList = [];

  projects.forEach(project => {

    pList.push(<div key={project.id}>{project.name}</div>);

  });

  if (pList.length === 0) {

    pList = <p>No projects available.</p>;

  }

  return <div>{pList}</div>;

}
```

24. Why Modifying the original array (forEach, push) is not recommended in React.

Modifying the original array (forEach, push) is not recommended in React because it directly mutates the state, which can prevent proper re-rendering.

25. .map() is used to transform data (convert an array of objects to an array of JSX elements).

26. React's rendering requires a new reference on every update.

27. In functional programming, we avoid mutation (mutating) and instead create new objects/arrays with updates. This is crucial in React because React's state management relies on immutability to detect changes and trigger re-renders efficiently.

28. Let's add delete functionality. We want that when we click on a project, it is deleted.

- One way to do this deletion is for App to pass the deleteProject function as a prop, but this gives the child component too much power.
- Instead, we create a stateful function in App called deleteProject and pass just that down as a prop.

29. what does the body element of the index.html file genertaed by VITE look

```
<body>
  <div id="app"></div>
  <script type="module" src="/main.js"></script>
</body>
```

30. export default function ProjectList({ projects }) {
const pList = projects && projects.length > 0 ? (

```
<ul>
          {projects.map((project, index) => (
       <li key={index}>{project}</li>
     ))}
   </ul>
```

```
  ) : (
    <p>No Projects left</p>
  );
  return <div>{pList}</div>;
}
```

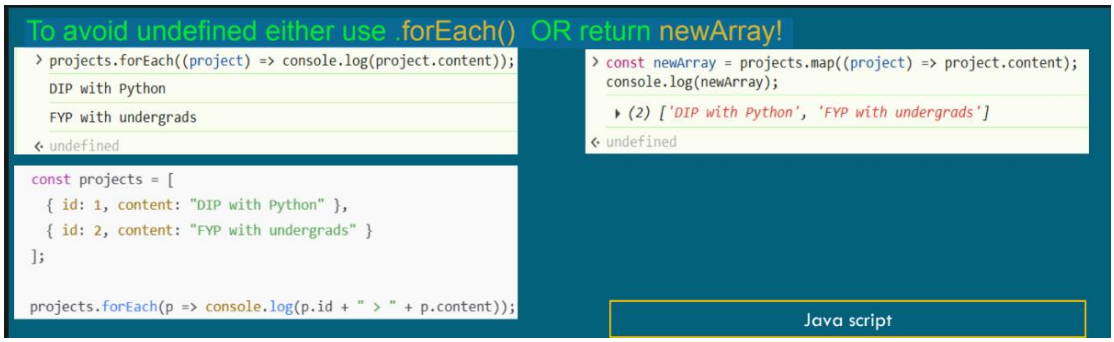
31.
```
import { useState } from "react";
import ProjectList from "./ProjectList";
import DeleteFromList from "./DeleteFromList";

export default function App() {
  // Create a state variable called "projects" and a setter function
called "setProjects".
  // This stores an array of objects, each with an id and content.
  const [projects, setProjects] = useState([
    { id: 1, content: "AI Chatbot" },
    { id: 2, content: "Weather App" },
    { id: 3, content: "Book Bazaar" },
  ]);

  // Delete function: Removes a project by filtering out the
matching ID
  function deleteProject(id) {
    setProjects((prevProjects) => prevProjects.filter((p) =>
p.id !== id));
  }

  return (
    <>
      <h1>Capstone Projects List</h1>
      {/* Only Display Projects → Pass only projects.
          Display + Delete Projects → Pass both projects and
deleter */}
      <ProjectList projects={projects} />
      <DeleteFromList projects={projects} deleter={deleteProject}
/>
    </>);}
```

Lec # 9

1. The DeleteFromList component receives a function as a prop from its parent and uses it inside an onClick event handler.

2. Usually what React developer doing? Is write single line here
Answer:
Builds user interfaces using components.
Manages app state and handles interactions.

3. How to avoid undefined ??
To avoid undefined either use .forEach() OR return newArray!



4. export function DeleteFromList({ projects, deleter }) {

```
 return (
   <ul>
     {projects.map((project) => (
       <li key={project.id}>
         {project.content}
         <button                    onClick={()                    =>
deleter(project.id)}>Delete</button>
       </li>
     ))}
   </ul>
 );
}
```

5. The deleteProject function **updates** the state by filtering out the project with the given id. It calls setProjects, passing a new array that excludes the matching project, ensuring (**immutability**) and triggering a **re-render** in React.

Lec # 10

1. Browser communicates with Google's authentication server and uses Token to authenticate user identity.

2. Browser saves the token given by Google .

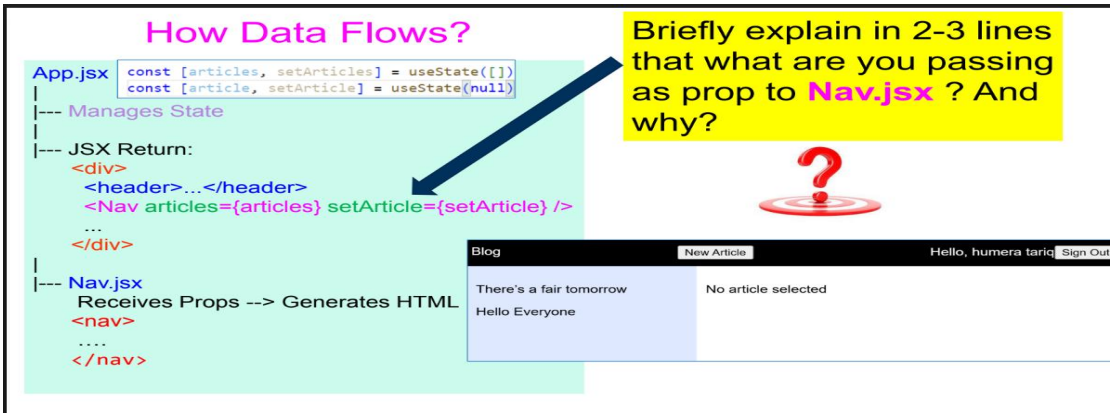
3. You visit a new page → Browser sends the token with the request

4. Google checks the token → Confirms you're logged in

5. The web is built on the same origin policy: a security feature that restrict (Hint: restrict/allow) how documents and scripts can interact with resources from another origin. This principle restricts the ways websites can access resources

6. For example, a document from https://a.example is prevented Hint: prevented/allowed from accessing data hosted at https://b.example.

7. write 2 3 lines on what you passing as prop to Nav.js and why ?
Answer
You're passing articles and setArticle as props to Nav.jsx so it can display a list of articles and allow users to select one. articles holds the data to show, and setArticle lets Nav update the selected article in the parent (App.jsx).



8. function submit(e) {
  setError(null);
  e.preventDefault();

  if (!title || !body) {
    setError("Both the title and body must be supplied");
  } else {
    // Proceed with form submission logic, e.g., save data or call an API
    console.log("Form submitted:", { title, body });
    // Optionally, clear the form
    setTitle("");
    setBody("");
  }
}

9. What will nav return?

```
export default function Nav({ articles, setArticle }) {
  return (
    <nav>
      <ul>
        {articles.map((article) => (
          <li key={article.id}>
 <button onClick={() => setArticle(article)}>
              {article.title}
            </button>
          </li>
        ))}
```

```
      </ul>
    </nav>
  );
}
```

Lec # 11

1. CORS stands for Cross-origin resource sharing, a security feature built into browsers.

2. CORS enforce security by blocking the request (request/response) from being accessible to your frontend (front-end/back-end) , unless the server (client/server) explicitly allows it via CORS headers .

3. For example:
- Your frontend is hosted at `frontend.com`.
- Your backend API is hosted at `api.backend.com`.

The browser treats these as different origins and blocks the request (requests/response) unless it's explicitly allowed.

4. CORS errors are triggered by the same-origin policy (Same-Origin Policy/Cross-Origin Policy), which prevents malicious websites from making unauthorized API calls using your credentials.

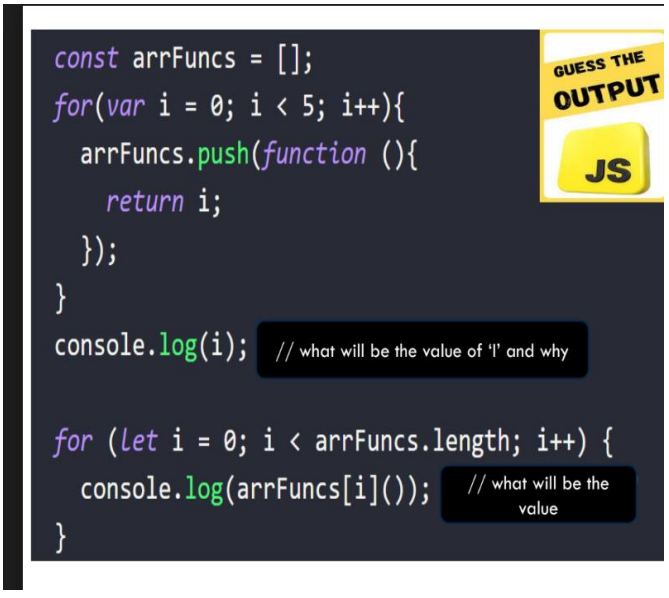
5. When the server (server/client) doesn't include the right CORS headers, the server refuses to share the response (request/response) and throws this error:

Access to resource at 'https://api.backend.com' from origin 'https://frontend.com'has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present.

6. In short, the browser isn't blocking the request , it's blocking the response for security reasons.

7. Every rectangular block on previous slide is an Execution Context(EC). Every execution context creates
a lexical environment for variables. The lexical environment is basically data structure that
keeps the variable and their value reference in memory so that it can easily find it for execution
context. Lexical environment consist of two parts : environment record and reference to outer
lexical environment. While environmental record keeps the local variable data, outer
reference keeps a reference for parents data lexical environment .

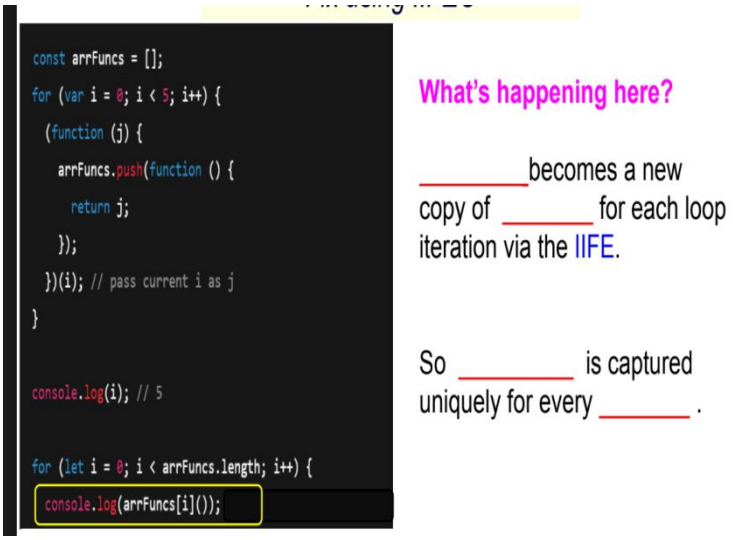
8. (inner functions memory to remember reference and lexical environment or scope on its current scope)



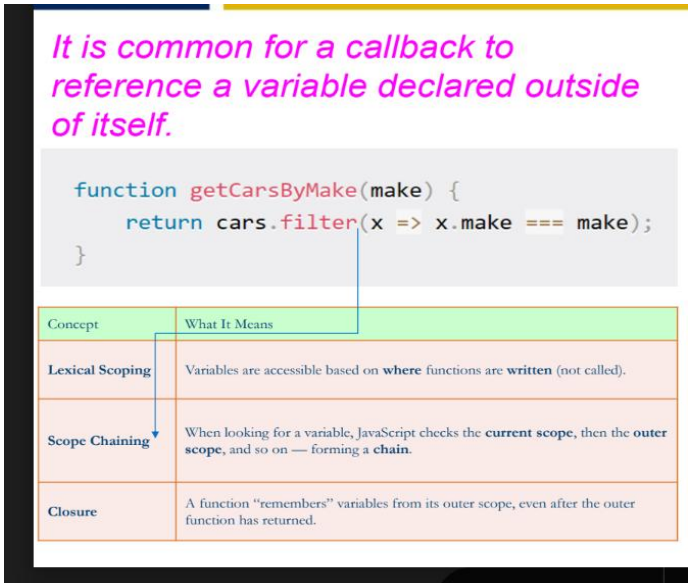
9. This code is not working as we expected because of var keyword/closure .The var keyword makes a Global variable and when we push a function we return the global variable i. So, when we call one of those functions in that array after the loop it logs 5 because we get the current value of i which is 5 and we can access it because it's a global variable. Because closure keeps the reference of that variable not its values at the time of its creation. We can solve this using IIFE(Immediately Invoked Function Execution) or changing the var keyword to LET for block-scoping.

10. Blanks

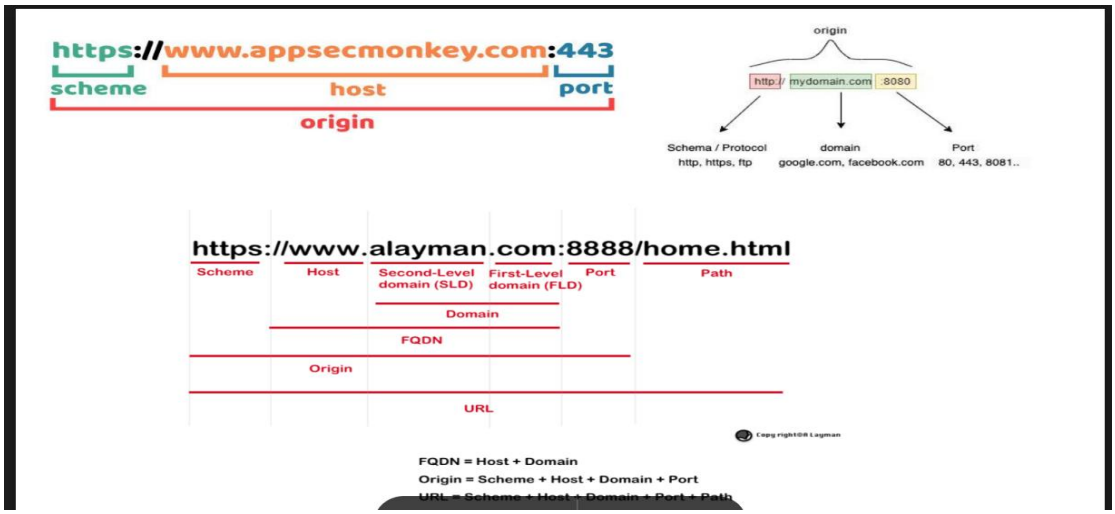
- i
- j
- j
- iteration/function



11. make is available in the callback because of ,lexical scoping (lexical scoping/scope chaining/closure) and the value of make is persisted when the anonymous function is called by filter because of a closure .



12.



Cors is basically Middleware

```
import express from 'express';
import cors from 'cors';

const app = _____ ;

app.use(cors({
  origin: ' _____ ', // frontend origin
  methods: [ _____ ], // Access-Control-Allow-Methods
  allowedHeaders: [ '_____ ', ' _____ '] // Access-Control-Allow-Headers
}));

app.get('/data', (req, res) => {
  res.json({ message: 'Using cors middleware' });
});

app._____ (5000);
```

13.Answer
1. Express()
2. http://localhost:5000
3. GET,POST,PUT,DELETE
4. allowedHeaders: ['Content-Type', 'Authorization']
5. liSTEN

14. Closures do not **copy** the values of variables from a function's **outer** scope during creation.
Instead, they maintain **references** throughout the closure's lifetime."

15. // Fetch articles from API
export async function fetchArticles() {
  Const response = await fetch("http://localhost:3000/articles");
  const articles = await response.json();
  // Combine with response id (if response contains an id)
  return { articles };
}

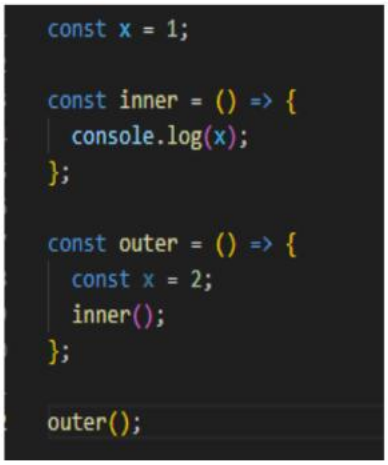
const response = await fetch("http://localhost:3000/articles/create-article", {
  method: "POST", // HTTP method
  headers: {

```
    "Content-Type": "application/json", // Tell server we're sending JSON
    // Add other headers if needed, e.g. Authorization
  },
  body: JSON.stringify({
    title: "Your article title",
    content: "Your article content",
  }),
});
```

Lec # 12

1. When we call and enter a function a new execution context is created on the call/function stack to keep track of state(variable) as we execute the function's code.

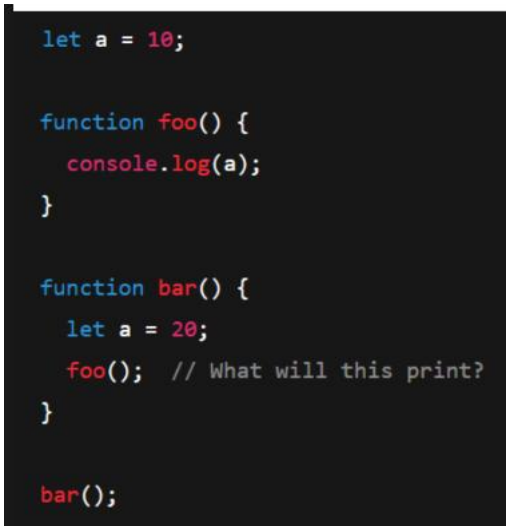
2. The answer is 1. Why? When inner is called, x is defined in outer (outer/inner) EC which sits just below inner's EC, but Javascript doesn't care about that. Javascript Uses static (static/dynamic) scoping, meaning it only cares about which variables are in-scope at the time a function is created.



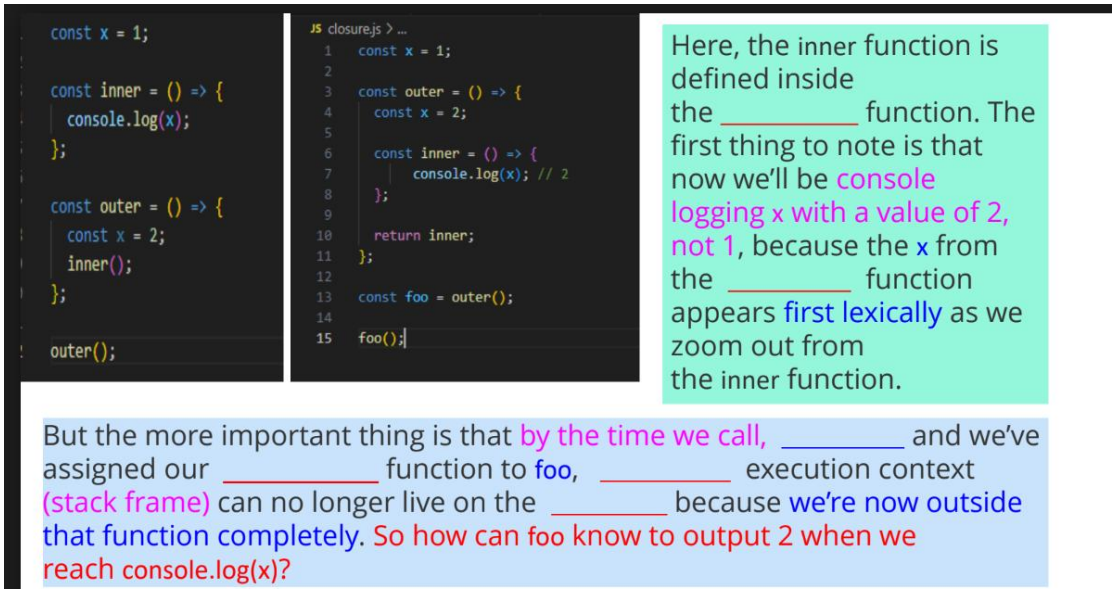
```
const x = 1;

const inner = () => {
  console.log(x);
};

const outer = () => {
  const x = 2;
  inner();
};

outer();
```

3. foo() looks for a where it was **defined**, not where it was **called**.

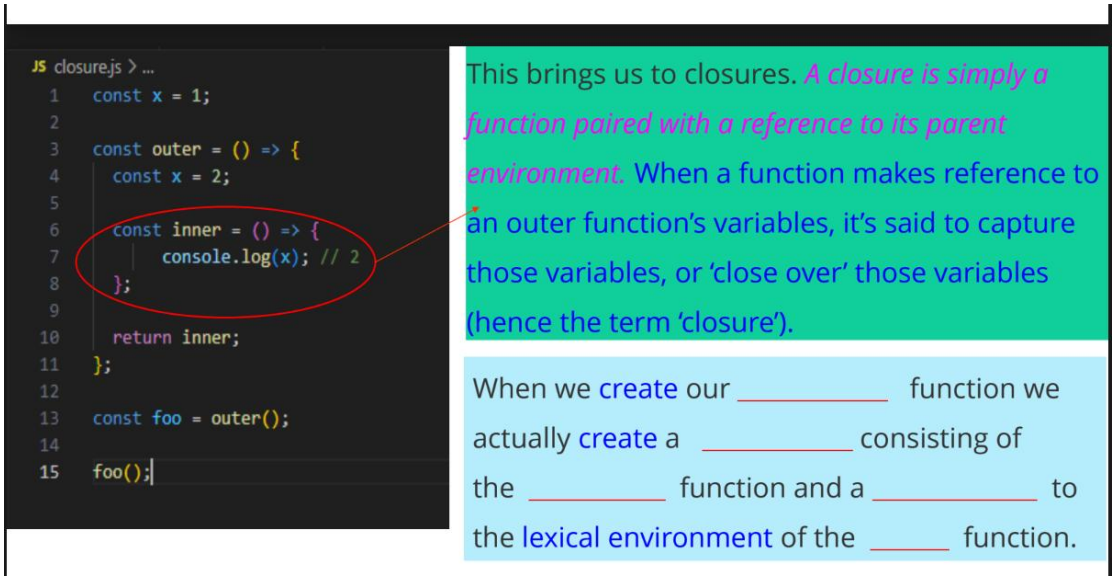
Since foo was **defined** when a = 1 in global scope, it prints **1**, even though bar() has a different a = 2 locally.



```
let a = 10;

function foo() {
  console.log(a);
}

function bar() {
  let a = 20;
  foo();  // What will this print?
}

bar();
```



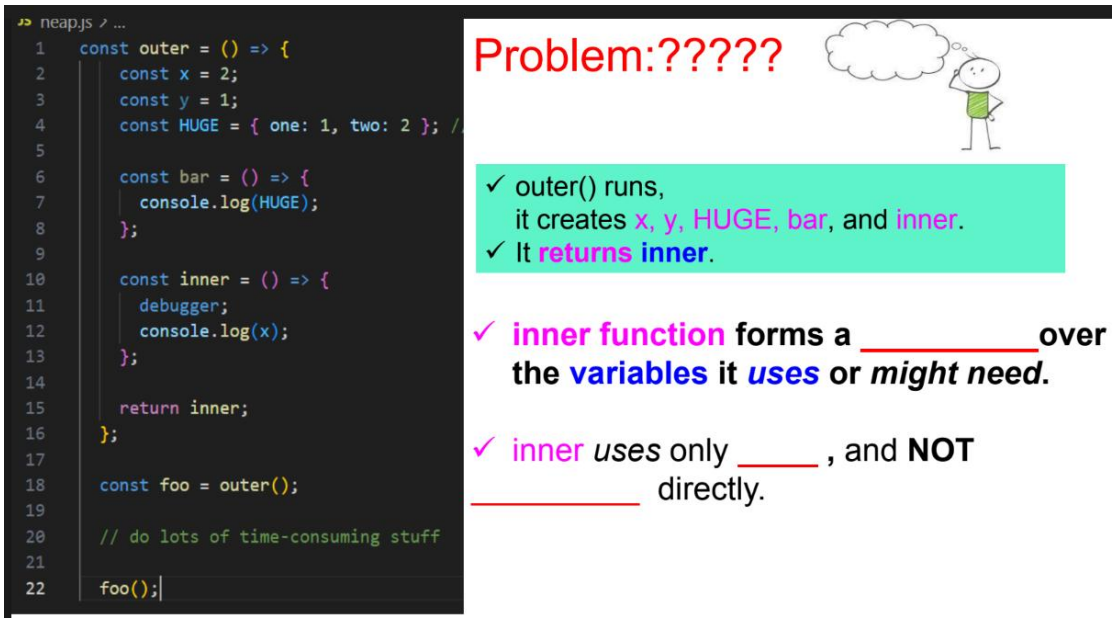
4. Answer
1. Outer
2. Outer
3. Outer
4. Inner
5. Outer
6. Stack



5. Answer
1. Inner
2. Closure
3. Inner
4. Reference

5. Outer

6. When inner is created, it stores an internal [[Scopes]] property which captures the lexical environment of the outer function. Then, when we call foo we use the [[Scopes]] property to traverse the scope chain and find the value of x.



7. Answer
1. Closurre
2. Reference
3. Copies of value

8. Higher-order functions allow for functions to be created **dynamically** at runtime.In languages like JavaScript, functions are first-class objects, meaning they can be saved in variables, passed to other (possibly higher-order) functions, and returned from (possibly higher-order) functions.To enable this, HOFs are typically managed on the **heap** at runtime

9. Higher-order functions can either **accept** functions, **return** functions, or **both**, but not always both!

forEach, map, reduce, and filter accept a **callback** function as an argument and **invoke** that function on **each element** of the array.

10. seen is now **global** or leaked outside the **function scope**.

If you have multiple reduce functions in the same file, they might accidentally clash or **overwrite shared state**.

It breaks **encapsulation**, an OOP principle.

11. import { createUniqueReducer } from '../services/uniqueService.js';

```
export const getSharedPosts = (req, res) => {
  const shares = [
    { userId: 1, postId: 101, action: 'share' },
    { userId: 2, postId: 101, action: 'share' },
    { userId: 1, postId: 101, action: 'share' }, // Duplicate
    { userId: 3, postId: 102, action: 'share' },
    { userId: 2, postId: 102, action: 'share' }, // Duplicate
    { userId: 4, postId: 103, action: 'share' }
  ];

  // Apply the unique reducer to filter out duplicates
  const uniqueShares = shares.reduce(createUniqueReducer(), []);

  res.json(uniqueShares); // Return unique shared posts
};
```

12. app.get('/api/shared-posts', (req, res) => {