

Full Stack exam:

1.1 The use of the countReducer function returns state data triggered by the action with the type 'increment' in this code snippet, when it is triggered the 'value' is then increased by 1.

1.2

```
function countReducer(state = initialState, action) {
  if (action.type === 'increment') {
    return {
      value: state.value + 1
    };
  } else if (action.type === 'decrement') {
    return {
      value: state.value - 1
    };
  }
  return state;
}
```

1.3

```
function countReducer(state = initialState, action) {
  if (action.type === 'increment') {
    return {
      value: state.value + 1
    };
  } else if (action.type === 'decrement') {
    return {
      value: state.value - 1
    };
  } else if (action.type === 'reset') {
    return initialState;
  }
  return state;
}
```

2.1. Within the code we have the functional component called `classInfo` which is using the `useState` hook to initialise the `studentsCount` at 0 and the `setStudentsCount` function is there to update the state variable. As we get further down the code we then have the return of what the user will interact with where they should be able to view the amount of students within the classroom and interact with a button to add another student.

2.2.a

```
const students = [
  { name: "Nrupul", present: false },
  { name: "Prateek", present: true },
  { name: "Jane", present: true },
  { name: "Paul", present: false },
  { name: "Luke", present: true }
];

const classInfo = () => {
  let [studentsCount, setStudentsCount] = useState(0);

  const addStudent = () => {
    const presentStudentsCount = students.filter(student => student.present).length;
    setStudentsCount(presentStudentsCount);
  };

  return (
    <div>
      <p>Number of students in class room: {studentsCount}</p>
      <button onClick={addStudent}> Add student </button>
    </div>
  )
}
```

b. I ensure that the function is triggered when the button is clicked by calling on the `addStudent` function within the button, making sure that every time the button is clicked the state will be recalled.

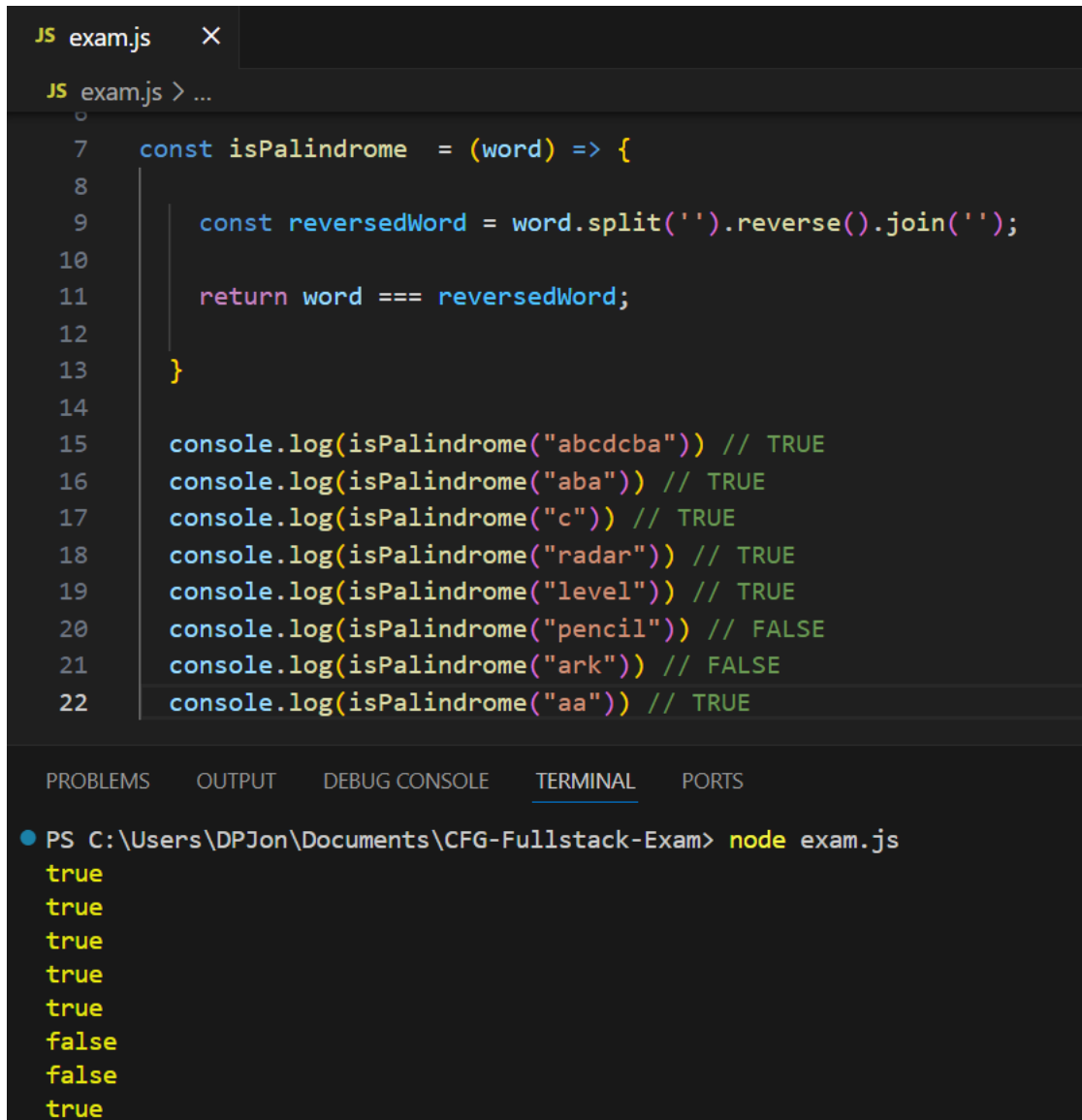
c. I make sure that the state is updated with the use of the `addStudent` function, by creating an array to pull through the list of present students and then calling on the `setStudentsCount` to then update the `studentsCount` state.

3.1. `action.payload` will be used to update the state, as it can hold the data of the state and when accompanied with an action it is used by the reducer to update the state.

3.2. /

3.3. I believe the better option would be to use figure 4, due to being able to update the state value by adding the new value to it.

Section Two:



The image shows a VS Code editor window with a file named `exam.js`. The code defines a function `isPalindrome` that checks if a word is a palindrome by comparing it to its reverse. Below the function, several `console.log` statements test the function with various words. The terminal at the bottom shows the output of running `node exam.js`, displaying the results of these tests.

```
JS exam.js X
JS exam.js > ...
7  const isPalindrome = (word) => {
8
9      const reversedWord = word.split('').reverse().join('');
10
11     return word === reversedWord;
12
13 }
14
15 console.log(isPalindrome("abcdcba")) // TRUE
16 console.log(isPalindrome("aba")) // TRUE
17 console.log(isPalindrome("c")) // TRUE
18 console.log(isPalindrome("radar")) // TRUE
19 console.log(isPalindrome("level")) // TRUE
20 console.log(isPalindrome("pencil")) // FALSE
21 console.log(isPalindrome("ark")) // FALSE
22 console.log(isPalindrome("aa")) // TRUE

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
● PS C:\Users\DPJon\Documents\CFG-Fullstack-Exam> node exam.js
true
true
true
true
true
false
false
true
```

Section Three:

1. SQL Injection:

Is a type of attack that can take place by a hacker who inject malicious SQL code into a database with the objective of being able to attack and manipulate the data and queries in any which way. There will then be a malformed query execution, meaning that the application doesn't fully clean the input. You can tell if a query has been injected as you will see changes to the queries which break it.

2. Types of SQL Injection Attacks:

Blind SQL Injection: Where the attacker might use a delay within the software to gain information based on the amount of the time it takes for the software to respond

In-band SQL injection: The use of error messages, and detailed ones at that allow the attacker to make intelligent links and maps on the database structure.

3. Impact of SQL Injection:

The consequences can be very severe depending on the database which has been attacked, there can be sensitive data stored within the database for example bank details might be stored in a secured database alongside the individuals personal details, this could cause an individuals money and identity being taken. Not only would data be exploited which could be manipulated in any way the attacker should want, this can also impact the integrity of the business who holds the data, once maybe trusted but no longer due to a SQL injection attack.

4. Mitigation strategies:

First of all we could use an ORM library (Object-Relational Mapping), which convert data between incompatible systems through providing abstraction for interacting with databases. They provide methods to perform the operations within the database rather than raw SQL queries minimising the chance of a SQL injection attack. We could also implement the principle of least privilege, which limits the user permissions within the database, meaning that the attacker wouldn't be able to replicate the permissions of someone with full access as this wouldn't exist.