Good practices, debugging

General good practices

- Minimize the use of global variables. These can be accidentally overwritten by accident, and may cause unintended effects.
- Global declarations (const, let, var) should typically be given at the beginning of your script.
- Local declarations can generally be done close to where they are needed to keep your code readable. Consistency helps with understanding your code, especially if someone else is reading it.
- Use === instead of ==. === is better most of the time, as long as you know the data types you will be working with.

Naming variables and functions

- Variables and functions are good to be named with camelCase. This is a common good practice in JS.
- The more descriptive your variable and function names, the more readable your code is. For example userName and productPrice are better names than data, value or x.
- When you iterate over arrays and other structures using a loop, a variable named i is often used. I stands for index.

Closures

- A closure means that an inner function has access to the outer (meaning enclosing) function's variables.
- A closure is created every time a function is created.
- Closures are a combination of a function and the lexical environment in which it is declared.
- Scope chain: when declaring a function within a function, the inner function has access to the variables and scope of the outer function.
- A closure allows for a function allows a function to access and change variables from an outer function, even after the outer function has completed execution.

Practical uses of closures

- Closures can be used to create de-facto private variables and methods. In JavaScript, private variables don’t exist in the traditional sense.
- Closures are often used in event handlers and callbacks for maintaining state and context.
- A callback is simply a function passed as an argument to another function.

Closure example

```
function outerFunction() {
    let outerVariable = 'I am outside!';

    function innerFunction() {
      console.log(outerVariable);
    }

    return innerFunction;
}

const myInnerFunction = outerFunction();
myInnerFunction(); // Outputs: "I am outside!"
```

Debugging

- Debugging is the act of finding bugs and issues in code
- Code might contain syntax errors or logical errors
- Syntax errors should be dealed with immediately upon noticing them. They are often simple to fix.
- Logical errors can be harder to detect, as they may only appear when certain, very specific conditions are met.
- Thorough testing is important before deeming an application finished to find any logical errors.
- Not all bugs are always found, but try to fix or work around them as you find more.
- Finding the location where the bug occurs is important. You should often try to reproduce the bug repeatedly to pinpoint its location.

Debugging techniques

- One of the most basic and effective debugging techniques in JavaScript is simply using console.log to test variable values at certain points. You can also console.log test messages to see if functions are called when they're supposed to, and not called when they're not supposed to.
- When working with webpage applications, use the console as often as possible to catch bugs early. Console error messages often contain critical information needed for debugging.

```
function addTask(newText) {
  let completed = false;
  let node = document.createElement("li");
  node.addEventListener("click", function () {
    toggleCompleted(node);
    console.log("function added");
  });
```

Debugging techniques

- Ask another person to see your code. Someone else may see a different problem and/or a different solution that will help in fixing your issue.
- Search online for the specific error message you are getting. More often than not, someone else has had the same issue.
- If you get stuck and you're not able to find any solutions, you can ask an AI assistant. However, always relying on AI isn't good for learning.

Common error types

- Error: the generic Error constructor creates and error object. Even errors are objects in JavaScript.
- SyntaxError: occurs when there's a syntactical mistake in the code.
- ReferenceError: occurs when a non-existent variable is referenced.
- TypeError: occurs when an operation is performed on a value of an unexpected type.
- RangeError: occurs when a value is not in the range of allowed values. For example, this can occur when trying to access a non-existent index in an array.

Reading the developer console for errors

- In the dev tools, you can see the row and the column where the error has occurred.
- You can also see the error type, and if you don’t catch it with error handling (try-catch), it will read as uncaught and might stop your program from continuing execution.
- You can also see the file name and type where the error occurred.