DWA_01.3 Knowledge Check_DWA1

---

1. Why is it important to manage complexity in Software?

It is important to manage complexity in Software because programming is complex and if not managed it could lead to unreadable code, bugs that take forever to fix and code that is not easily maintainable. Also bugs in software can have catastrophic implications such as a system crashing, so, managing complexity in software helps to avoid such. In the long run managing complexity helps in code maintainability and reuse.

---

2. What are the factors that create complexity in Software?

- **How large the code base is.**
- **How well the code style is.**
- **The naming of variables, if not clear it can contribute to complexity.**
- **Programming is complex: it is very hard to learn and master and requires a lot of precision.**
- **Programming is interrelated, it has a lot of people working on it and a lot of moving parts.**
- **Software is ever evolving and changing, that creates complexity.**

---

3. What are ways in which complexity can be managed in JavaScript?

- **Having descriptive variable and function names**
- **Paying attention to code style, using style guides to improve code style.**
- **Documenting your code using JSDOC to get people reading your code to understand it. Also use JSDOC to define your functions more clearly and avoid errors.**
- **Make your code more to so that functions can be exported and used in a different module, should your code become too clustered.**

---

4. Are there implications of not managing complexity on a small scale?

Yes, if unmanaged on a small scale the code might not run at all and might take time to fix bugs due to not managing the complexity. But not as catastrophic as not managing complexity on a larger scale.

---

5. List a couple of codified style guide rules, and explain them in detail.

- Destructuring:
Instead of creating temporary references for accessing properties of an object it is better to use object destructuring when accessing or using multiple properties of an object. This helps in avoiding repetitive code. Take for example:

```
const getFullName => (user) {
const firstName = user.firstname
const lastName = user.lastname
return `${firstName} ${lastName`
}
```

The above can become very verbose if the code were to get longer. A better way to get the same result in a shorter way is through
destructuring:

```
const getFullName => (user) {
const {firstName, lastName} = user
return `${firstName} ${lastName}`
}
```

The above correction is more concise and much more readable. There is though a much better way, which is to destructure the object parameter in the function like so:

```
const getFullName => ({firstName, lastName}) {
return `${firstName} ${lastName}`
```

}

- Strings:
When adding or building up strings, it is best practice to use template strings instead of concatenation. This is because template strings provide you with more readability and more proper new lines. Plus, with template strings you can nest code which is impossible with concatenation.

For example:
This is bad:
const name = “Mihlali”
const age = 55
console.log(“My name is” + name + “,” + “I am” + age + “.”)

This is more concise and readable:
const name = “Mihlali”
const age = 55
console.log(`My name is ${name}, I am ${age}.`

Here is an example of new lines with concatenation:

const errorMessage = 'This is a super long error that was thrown because \
of Batman. When you stop to think about how Batman had anything to do \
with this, you would get nowhere \
fast.'

But here is a better way with template literals

const errorMessage = 'This is a super long error that was thrown because of Batman. When you stop to think about how Batman had anything to do with this, you would get nowhere fast.'
This example shows how less clustered template strings make your code look, unlike with concatenation.

- Functions:
It is better to use name function expressions than function declarations.
Function declaration are hoisted which means they can be called before being defined. This is bad for readability and maintainability.

For example:

```
hoisted() // Logs "foo"

function hoisted() {
  console.log("foo")
}
```

Whereas with function expression you will get an error if you call it before definition:

```
notHoisted() // error message

const notHoisted = funtion(){
    console.log(“foo”)
}
```

---

6. To date, what bug has taken you the longest to fix - why did it take so long?

In the capstone assignment for IWA, I struggled to get the background of my books to change as the theme change. I realized later (a week later) that I was working in the wrong lexical scope and my variable names were so similar I got them mixed up. You could say I didn’t manage the complexity of my code.
