WEEK 4 RESEARCH

Rachel VanHorn

What are the differences between var, let and const?

The declarations **var**, **let** and **const** are all statements that declare variables. **Const** declares a variable with a value that cannot be changed and cannot be redeclared or updated. **Let** declares a variable locally, so it is limited to the block or expression in which it is used. **Let** can be updated, but not redeclared. Whereas, **var** declares a variable globally or locally, depending where in the code it is declared. **Var** can be both redeclared and updated. **Var** also differs from **let** and **const** in that it is hoisted, or read before all other code regardless of where it exists in the code. This means you can call on the declared variable in code that comes before the line it is declared on, it will return undefined until the line it is declared. **Let** and **const** do not exist in the code until they are read on the line in which they are declared, if they are referenced before they are declared you will receive a reference error.

Sources:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/let

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/const

https://developer.mozilla.org/en-US/docs/Glossary/Hoisting

https://www.w3schools.com/js/js_variables.asp

https://www.freecodecamp.org/news/var-let-and-const-whats-the-difference/

How does a promise work?

A promise in Javascript is a way to manage asynchronous functions so that code can run even when you don't yet have the value the promise is referencing. A promise has 4 states.

1. A promise is pending when it has been created but you do not have the value needed to continue.
2. A promise is fulfilled when the promise succeeded.
3. A promise is rejected when there was a failure because of an error.
4. A promise is settled once a decision has been made, regardless of if It is fulfilled or rejected.

A promise takes in a callback that has two arguments: an argument for the resolution of the promise, and an argument for the rejection of the promise. Depending on what the awaited for value comes back with, one of those two arguments are called on. We can use then() and catch() to tell us what will happen after we receive the value that the promise is waiting for. If the promise has been fulfilled then() {some code for when the promise was successful}, we can also use then() for if our value was not successful. If the promise was rejected or failed because of an error, we can use catch() {some code for what happens when there is an error}. A promise allows us to keep out code running while we wait for values to be returned that are outside our control, such as user input, an html request or some other

reason. Using promises means we don’t have to put breaks into our code to wait for outside input or delayed input.

Sources:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

https://www.freecodecamp.org/news/javascript-promise-methods/

https://dillionmegida.com/p/javascript-promises/