Intro

Functional programming is about verbs Object oriented programming is about nouns

Data Structures

Objects

```
var person = {};
person.name = "Mrs. White";

var person = {
    "name": "Mrs. White"
};
```

Functions

Arrow Functions

- Binds context to its parent context
- It doesn't own its own keyword this
- Doesn't have its own value for the arguments keyword

Arguments

function(a, b, c, …d); d => is all leftover args keyword: `arguments` is all values passed into function

Scope

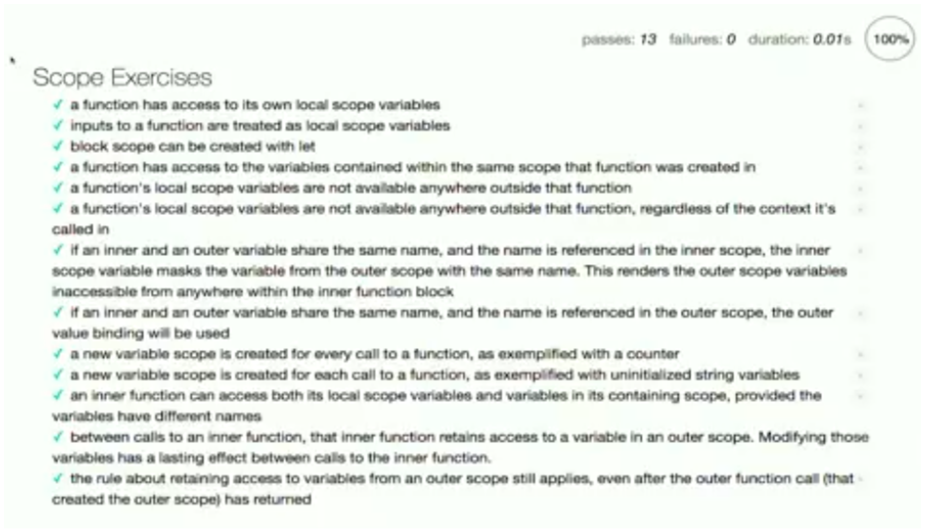
Area where you can access a variable not using `var` or using `window.foo` is a global variable

Callbacks

Functional Utilities

Advanced Scope: Closure

Summary



passes: 13 failures: 0 duration: 0.01s 100%

Scope Exercises

- ✓ a function has access to its own local scope variables
- ✓ inputs to a function are treated as local scope variables
- ✓ block scope can be created with let
- ✓ a function has access to the variables contained within the same scope that function was created in
- ✓ a function's local scope variables are not available anywhere outside that function
- ✓ a function's local scope variables are not available anywhere outside that function, regardless of the context it's called in
- ✓ if an inner and an outer variable share the same name, and the name is referenced in the inner scope, the inner scope variable masks the variable from the outer scope with the same name. This renders the outer scope variables inaccessible from anywhere within the inner function block
- ✓ if an inner and an outer variable share the same name, and the name is referenced in the outer scope, the outer value binding will be used
- ✓ a new variable scope is created for every call to a function, as exemplified with a counter
- ✓ a new variable scope is created for each call to a function, as exemplified with uninitialized string variables
- ✓ an inner function can access both its local scope variables and variables in its containing scope, provided the variables have different names
- ✓ between calls to an inner function, that inner function retains access to a variable in an outer scope. Modifying those variables has a lasting effect between calls to the inner function.
- ✓ the rule about retaining access to variables from an outer scope still applies, even after the outer function call (that created the outer scope) has returned

Figure 1: Scope Rules