Function and Scope in JS

Recap on theory of function

Examples of function declaration

Callback function

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete some kind of routine or actions

Scope

Local Scope

Global Scope

Scope is a concept that determines the accessibility of variables

Hoisting

A behavior of Javascript that moves variables and functions declaration are moved to the top of their scope

```
console.log(x)
x = 100
▶ Uncaught ReferenceError: x is not defined
    at <anonymous>:1:13
```

```
console.log(x)
var x = 100
undefined
undefined
```

```
console.log(x)
let x = 100
▶ Uncaught ReferenceError: x is not defined
    at <anonymous>:1:13
```

What is **this**

```
console.log(this)
▶ Window {window: Window, self: Window, document: document, name: "", location: Location, …}
```

```
document.body.addEventListener('click', () => {
    console.log(this)
})
undefined
▶ Window {window: Window, self: Window, document: document, name: "", location: Location, …}
```

```
document.body.addEventListener('click', function(){
    console.log(this)
})
undefined

▶ <body class="feature-filter-bar feature-upvotes tutorials-controller tutorial-single" data-e
facebook-app-id="694818843983011" data-completed-tutorial-id data-tutorial-id="2544" data-js="
```

```
setTimeout(() => {
    console.log(this)
}, 500)
187
▶ Window {window: Window, self: Window, document:
setTimeout(function(){
    console.log(this)
}, 500)
189
▶ Window {window: Window, self: Window, document:
```

```
const harry = {
    name: 'harry',
    age: 15,
    cast: function(){
        console.log(this);
        console.log('lumos!')
    }
}
undefined
harry.cast()
▶ {name: "harry", age: 15, cast: f}
lumos!
```

Arrow function & regular function scope

In classic function expressions, the ***this*** keyword is bound to different values based on the context in which the function is called. Whereas arrow functions use the value of ***this*** in their lexical scope. This leads to very different behaviour.

What's the difference between ***context*** and ***scope***? The ***context*** is (roughly) the object that calls the function. And the ***scope*** is all the variables visible to a function where it is defined. One cares about how it is called, the other cares about how it is defined.

In arrow function, the value of ***this*** comes from the surrounding code block. It doesn't care what calls it, it just cares where it was defined.

Practice exercises

1. Create an index.html, and include a script.js upon pageload
2. Content of the script.js should be wrapped within a self-executed anonymous function to avoid polluting the page global scope
3. Add to the index.html a table element with 3 columns and 2 rows, first row is the header row that contains the table headings (studentNo, name, age), second row contains your own studentNo, name, and age
4. Create a function name executeScript. Function executeScript should add a new row to the table that contains a test user studentNo, name, and age (maybe 123, test user name, 999) when called. Remember to invoke executeScript function
5. Move the script.js injection to the header part of your index.html, is your code still working?
6. Update your code so that executeScript only gets invoked when the DOMContentLoaded event is detected. Is the testUser now getting added to the table? Why?

Practice exercises

1. Copy the following function which will always return an array of 5 objects containing information of 5 test users to your code, add more if you want https://gist.github.com/bch-fullstack/7bdc2288e1a1b5c6da0cf1674477eed3
2. Create a new function addUser that expects a ***single*** object of userInfo as parameter
3. Iterate through the array of 5 users, call addUser on each iteration
4. Add a new function getOldest() that expects an array of userObject as parameter, when getOldest() is called, it iterates through the array of objects, and return the oldest person in the array according to the age, ***do NOT use .find() or filter() method or array methods***
5. Create a new column to the table name isStaff, update your code so that each row in the table will also display an information if the person is a staff or a student. A student always has a non-negative student number, otherwise it is a staff
6. Create a button that will sort the list of users according to their age ***descendingly*** and re-render the table with sorted content upon click. ***Do NOT use .sort() method***

Complete the following calculator app

https://github.com/bch-fullstack/calculator-simple-js

Completed calculator app should be able to perform calculation between 2 numbers (add, subtract, multiply, divide), clearing input fields, and support decimal points.