W2 Notes -

Saturday, May 2, 2020

7:59 PM

I will move these into my online profile:

The material in "JavaScript: Novice to Ninja, 2nd Edition seemed way too simple as I began this week. The code I have written to this point has seemed far more advanced than the material covered. See the following projects and their code.

<https://craigkehl.github.io/lesson11/preston.html> // Inspect the forcast\_api.js

<https://craigkehl.github.io/Salmon_River_Adv/adventures.html> // Inspect trips.js

<https://craigkehl.github.io/Salmon_River_Adv/index.html> // Inspect move\_dom\_elements.js

The most difficult problems in those projects were solved with exhaustive internet searches. I also study Youtube material, LinkedIn Learning courses, and Udemy courses every free moment I get. So, I desiered to pick-up where I left off to apply what I could to this week's team assignment. So I read through the group assignment and formulated ideas on how I would solve the problems.

By Wednesday night when we had our team meeting, I had a jumble of partially working code. My team members patiently listened to what I wanted to do. It was decided to take a more basic approach. However, I was thrilled to learn I could submit additional code later after playing with it more.

Summary of concepts studied this week:

* Function types
  + Notes below:
* Callback functions
  + Needed mainly to respond to an event we can't control the timing of. Usually for a process handed of to the browser.
  + Since functions can be called from within functions, further specialization and reuse of code is achieved
* Bind() method
  + This method seems to scare everyone and was incredibly tough to uncover a relatively simple yet very helpful solution. I had to cross the abyis of "the lost this". I basically needed it to pre-load a parameter not to be exicuted by the first function, but included as the last parameter when passing a callback function as a parameter.
* The Rest parameters through the rest operator
  + Allows you accomidate an unknown amount, not previously specified amount of parameters, delivered as and array providing the methods of Iterables to access and manipulate those parameters.
  + It must be the last parameter.
* Traversing the DOM -
  + This severed knowledge needed for both this and WDD 331
* Iterables vs."Array-Like" objects
  + Strings and node lists are not Iterables, only Array-like and therefor do not have all the methods real Iterables, like a real array, have.
* There are multiple ways of creating array:
  + const array = [];
  + const array = ();
  + Array.of
* push(), pop -
* unshift(), shift()
* splice()
* slice()
* A

Other items I needed to research and better understand why they might be preferred were:

* EventListeners vs. "onclick"
* The timing and perseverience of objected collected from the DOM
  + Don't initialize and populate the value of an input before the desired value is entered from the user. The initial object value will be retained in the variable unless destroyed and reconstructed after the live value has been updated. (Wasted two hours on that one.)
* Why "const" is preferred over "let" when local scope is sufficient and that it can work throughout chances due to the function's destruction and recreation when called again.

My Function Notes:

/\*\* Standard Function declaration \*\*\*\*

 \* Advantage: Can be declared anywhere in your code and

 \* it will be pareced and ready when needed. \*/

function var\_name(pram1, pram2, ...pram) {

  z = a + b;

  return z;

}

/\*\* function expression \*\*\*

 \* Can be more concise \*/

const name = function(pram1, pram2, ...pram) { return z}

/\*\*\* Arrow Functions (old),  Fat Arrow Functions \*\*\*\*/

/\*\* Default syntax:

 \*  Same as "normal" functions, parameters and return   statement are optional. Noteworthy: Semi-colon at end, no function keyword, parentheses around parameters/ arguments.\*/

const add = (a, b) => {

const result = a + b;

return result;

};

/\*\* Shorter syntax, if only one parameter. Noteworthy: Parentheses around parameter list can be omitted \*/

const log = message => {

console.log(message);

return z;

};

/\*\* Empty parameter parentheses if no arguments are receive \*/

const greet = () => {

console.log('Hi there!');

};

/\*\* Short function body, if only one expression is used. An expression result is always returned automatically\*/

const add = (a, b) => a + b;

/\*\* THe function can return an object. Extra parentheses are required around the object, since the curly braces would otherwise be interpreted as the function body delimiters \*/

const loadPerson = pName => ({name: pName });

/\*\* That last case can be confusing: Normally, in JavaScript, curly braces always can have exactly one meaning. \*/

const person = { name: 'Max' }; // Clearly creates an object

if (something) { ... } // Clearly used to mark the if statement block

/\*\* But when using arrow functions, curly braces can have two meanings:

Mark the function body (in default form)

Create an object which you want to return (in shorter function body form)

To "tell" JavaScript what you want to do, wrap the expression (e.g. object creation) in parentheses like shown above. \*/