Intro To JavaScript – Day 6

The `this` Keyword in JavaScript

Let's see if we can tackle how this works in JavaScript, it is NOT the same a C++/C#/Java!

Definition: The `this` keyword refers to the *owner* of the function being executed. `this` is a hidden parameter passed to all functions.

`this` is always defined to something!

Let's look at some examples...

Example 1 : Default Owner

(run this using Firefox/Chrome)

```
function myFunction() {
    console.log(this);
}
myFunction();
```

- When run, we see Window is show in the console.
- Click on Window and see that it is the web page, look to find `myFunction` as being part of the Window object.

Example 2 : Object Owner

```
let person = {
    name: {
        last: 'Doe',
        first: 'John'
    },
    age: 26,

    getName: function() {
        return this.name.first + ' ' + this.name.last;
    }
};
console.log(person.getName());
```

- This looks a little more like what you are used to.
- When calling `getName`, it is bound to the `person` object because of the way it is called.

Example 3 : Another Owner

```
let factory = {
    name: {
        last: 'Generator',
        first: 'Function',
    },

    getName: function() {
        return function() {
            return this.name.first + ' ' + this.name.last;
        }
    }
};
```

```
let me = {
    name: {
        last: 'Mathias',
        first: 'Dean'
    },

    getName: factory.getName()
};
console.log(me.getName());
```

- This time, we define an object named `factory` that has a `getName` function that returns a function!
- The returned function refers to `this`, but the `this` is rebound when it gets assigned to another object.
- The `me` object uses the `factory` object to obtain a `getName` function.
- When `me.getName()` is called, the function returned by `factory` is called and `this` is bound to `me`, instead of `factory`.

Duck Typing

"If it walks like a duck and quacks like a duck, then it is a duck!"

Remember that JavaScript is dynamically typed. The type of a parameter to a function is not known until the function is actually called. This means that different types that have the same (sub)set of properties can be passed into a function expecting to only use those common properties.

```
function reportPosition(source) {
    console.log('lat: ' + source.lat + ', lon: ' + source.lon);
}
let airplane = {
    lat: 12345,
    lon: 23456,
    speed: 400,
    altitude: 35000
};
let city = {
    name: 'Logan',
    population: 50000,
    lat: 54321,
    lon: 65432
};
reportPosition(airplane);
reportPosition(city);
```