

Ecma script 6 Course Summary v2 – 31-03-2019

Notes from Pluralsight course = S.White

let & const

- **let**: has a true block scoping
- **var**: function scoping – avoid it
- **const**: create a read only variable.
Has a true block scoping. Now
throws an error if you try to assign
it. Was not the case before ES6
even if const existed!

Destructuring

- **Array:**

```
var doWork = function(){  
    return [1, 3, 2];  
};  
let [x, y, z] = doWork();
```

- **Swap variables:**

```
let x=2;  
let y=2;  
[x, y] = [y, x];
```

Destructuring

- Object:

```
let doWork = function() {  
    return { firstName: "Scott",  
            lastName: "Allen",  
            handles: { twitter: "OdeToCode" }  
    };  
};
```

```
let { firstName: a, handles:{twitter : b}} = doWork();
```

- Create two variables a & b set with object members

Destructuring

- **Function parameters:**

```
let doWork = function(url, {data, cache} ) {  
    return data;  
};
```

```
let result = doWork(  
    "api/test",  
    { data: "test", cache: false }  
);  
expect(result).toBe("test");
```

- Create two variables data & cache set with object members passed as parameter

Destructuring

- **Object - shortcut:**

```
let doWork = function() {  
    return { firstName: "Scott",  
            lastName: "Allen",  
            handles: { twitter: "OdeToCode" }  
    };  
};
```

```
let {    firstName:  
    handles:{twitter}  
} = doWork();
```

- Create two variables firstName & twitter set with object members

Default parameters

- **Standard parameters:**

```
let doWork = function(a = 1, b = 2, c = 3){  
    return [a,b,c];  
};
```

```
let [a,b,c] = doWork(5, undefined);  
expect(a).toBe(5);  
expect(b).toBe(2);  
expect(c).toBe(3);
```

Default parameters

- **Destructuring parameters:**

```
let doWork = function(url, {data = "Scott", cache = true} ) {  
  return data;  
};
```

```
let result = doWork( "api/test", { cache: false } );  
expect(result).toBe("Scott");
```


REST parameters

- **REST parameters:**

```
let doWork = function(name, ...numbers){  
  let result = 0;  
  numbers.forEach(function(n){  
    result += n;  
  });  
  return result;  
};
```

```
let result = doWork("Scott", 1, 2, 3);  
expect(result).toBe(6);
```

- Numbers is a true Array object. REST numbers must be the last parameter

Spread Operator

- **As parameter:**

```
let doWork = function(x, y, z) {  
    return x + y + z;  
}  
  
var result = doWork(...[1, 2, 3]);  
expect(result).toBe(6);
```

- **Build an array:**

```
var a = [4, 5, 6];  
var b = [1, 2, 3, ...a, 7, 8, 9];  
expect(b).toEqual([1,2,3,4,5,6,7,8,9]);
```

Template literal

```
let doWork = function(name){  
    return `Hello, ${name}`;  
};
```

```
let result = doWork("Scott");  
expect(result).toBe("Hello, Scott");
```

- Note the template literal starts with a back tick!
- We didn't summarize template literal tag

Object: definition

```
class Employee {
```

```
    doWork() {  
        return "complete!";
```

```
    }
```

```
    getName() {  
        return "Scott";
```

```
    }
```

```
}
```

```
let e = new Employee();
```

```
expect(e.doWork()).toBe("complete!");
```

```
expect(e.getName()).toBe("Scott");
```

Object: constructor

```
class Employee {  
  constructor(name) {  
    this._name = name;  
  }  
  doWork() {  
    return "complete!";  
  }  
  getName() {  
    return this._name;  
  }  
}  
  
let e1 = new Employee("Scott");  
let e2 = new Employee("Alex");  
expect(e1.getName()).toBe("Scott");  
expect(e2.getName()).toBe("Alex");
```