Template literals

Manasseh

Template literals

Description: Template literals are string literals allowing embedded expressions . You can use multi-line strings and string interpolation features with them. (They were called string templates in prior editions of ES2015)

Template literals uses the backtick (` -located underneath the esc key) to do different things when formulating strings.

E.g Multiple line capabilities

In es5 \n would be used to make multiple lines:

Let name = “Manasseh works at \n The National Archives”
> “Manasseh works at
The National Archives”

In es6 Template literals uses the backtick to allow for multiple lines to be made :
Let name = `Manasseh works at
The National
Archives`

> “Manasseh works at
The National
Archives”

No more Concatenation...

In es5 you would use the following method of concatenation to add strings and variables together:

```
var firstName = “Manasseh”
var lastName = “Boyd”
console .log (“My name is ” + firstName + “ ” + lastName)
> “My name is Manasseh Boyd”
```

In es6 we can use let instead of var and use place holders which are indicated by a dollar sign $ and curly braces {} instead of the plus sign :

```
let firstName = “Manasseh”
let lastName = “Boyd”
console .log (`My name is ${ firstName } ${ lastName}`) ;
> “My name is Manasseh Boyd”
```

No more concatenation in Objects

Example with Objects in es6:

```
let person = {
firstname: `Manasseh` ,
lastName: `Boyd`,
sayName () {
return `My name is ${this.firstName} ${this.lastName}` ;
}
}

Let name = person.sayName ();

console.log (name);
```

“My name is Manasseh Boyd”

We can now use a shortend method signature. Instead of:

sayName: function ()

There is no need to concatenate all values from a big object which would have looked like this:

“My name is ” + this.firstName + “ ” + this.lastName;

Block scoping

let allows us to block scope our variables:

```
if (true) {
   let name = “Manasseh”;
}

console.log (name);
```

“Running this code will result in an error because the let name is undefined and not available outside of this block of code”

let allows us to protect/keep our variables inside our block of code if we need to, unlike var which can call the values outside of the block of code:

```
if (true) {
var name = “Manasseh”;
}
console.log (name);
> “Manasseh”
```