Programming in JavaScript

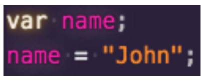
Section 3: ES6 and JavaScript

3.1 Const and Let Variables

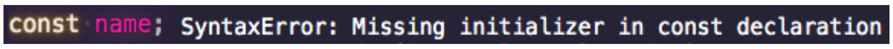
Const and Let variables are a new feature that is introduced into ES6. The syntax is exactly the same as the var variable keyword; however, the behaviour are slightly different and will throw an error depending on the variable type used.

A const variable must be initialised when being declared i.e. we must assign it a value otherwise we would get an error.

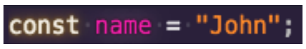
A const or constant variable can store a value just like a variable. However, when declaring a const variable we must assign it a value (unlike var which does not require it to be initialised when declared).



```
var name;
name = "John";
```

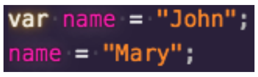


```
const name; SyntaxError: Missing initializer in const declaration
```

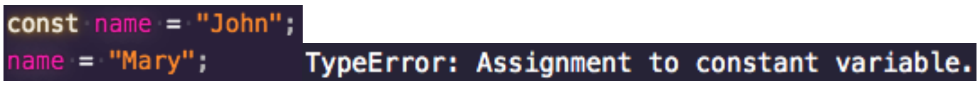


```
const name = "John";
```

The var value can change its value anytime. The const variable on the other hand does not allow you to change the value of the constant variable and will result an error.



```
var name = "John";
name = "Mary";
```

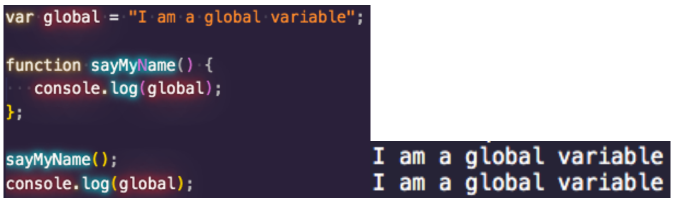


```
const name = "John";
name = "Mary";          TypeError: Assignment to constant variable.
```

Therefore, when declaring a const variable we must assign it a value when it is declared and we cannot re-assign it a new value at any point in the code hence, as the name indicates, it has a constant value.

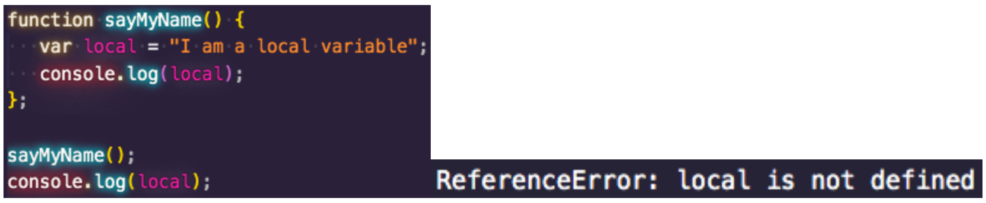
The let variable is almost the same as var but similar to const variables there is a difference between var and let variables. A var variable has two scopes a global and a function scope. The let variable has three scopes a global scope, function scope and a block scope.

When we declare a variable outside a function the variable is said to be declared in the global scope. This means any child functions of the code can access this global variable.



```
var global = "I am a global variable";

function sayMyName() {
  console.log(global);
};

sayMyName();                         I am a global variable
console.log(global);                 I am a global variable
```

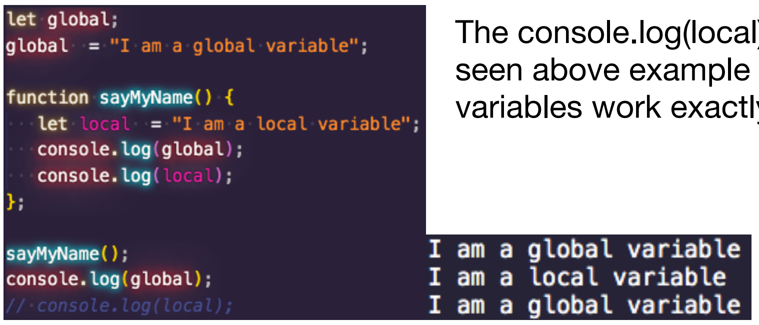
A local scope (or a function scope) is a variable that is only available in the function and is not available outside in the global scope i.e. it is only local to the function it was declared in. The below code will throw a ReferenceError on console.log(local) because the global scope does not have access to the function variable outside the local scope. Therefore, the variable can only be manipulated inside of the function where it was declared.



```
function sayMyName() {
  var local = "I am a local variable";
  console.log(local);
};

sayMyName();
console.log(local);
```

```
ReferenceError: local is not defined
```

The let variable functions exactly the same as the var variable i.e. it can be initialised at a later point in time and has both the global and function scope.



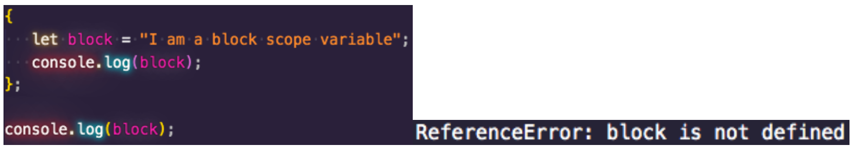
```
let global;
global  = "I am a global variable";

function sayMyName() {
  let local  = "I am a local variable";
  console.log(global);
  console.log(local);
};

sayMyName();
console.log(global);
// console.log(local);
```

```
I am a global variable
I am a local variable
I am a global variable
```

The console.log(local) code will throw a ReferenceError as seen above example for var. As we can see the let and var variables work exactly the same.

The only difference as mentioned above is that the let variable has a third scope which is called the block scope. In JavaScript a block is anything that starts and closes with curly brackets ({ }). The code contained in the curly braces is part of that block of code.

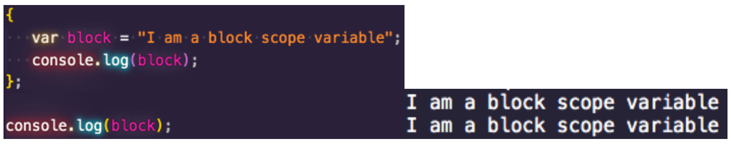
The let variable inside of a block can only be accessed in the block scope and cannot be accessed outside of the block i.e. in the global scope (as seen below this will return a ReferenceError).



```
{
  let block = "I am a block scope variable";
  console.log(block);
};

console.log(block);
```

```
ReferenceError: block is not defined
```

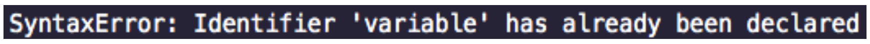
When the block code completes its execution the let variable no longer exists which is why it gives the ReferenceError. This behaviour does not apply to var variables as demonstrated in the below example:



```
{
  var block = "I am a block scope variable";
  console.log(block);
};

console.log(block);
```

```
I am a block scope variable
I am a block scope variable
```

Finally, it is important to note that a let and var variable cannot have the same name (no variables can be declared with the same name) and this will throw a SyntaxError to inform you that the variable name has already been declared. In the below example the variable name is called ‘variable’ and this will change to whatever the name of the variable is that has already been declared.



```
SyntaxError: Identifier 'variable' has already been declared
```