
boolean

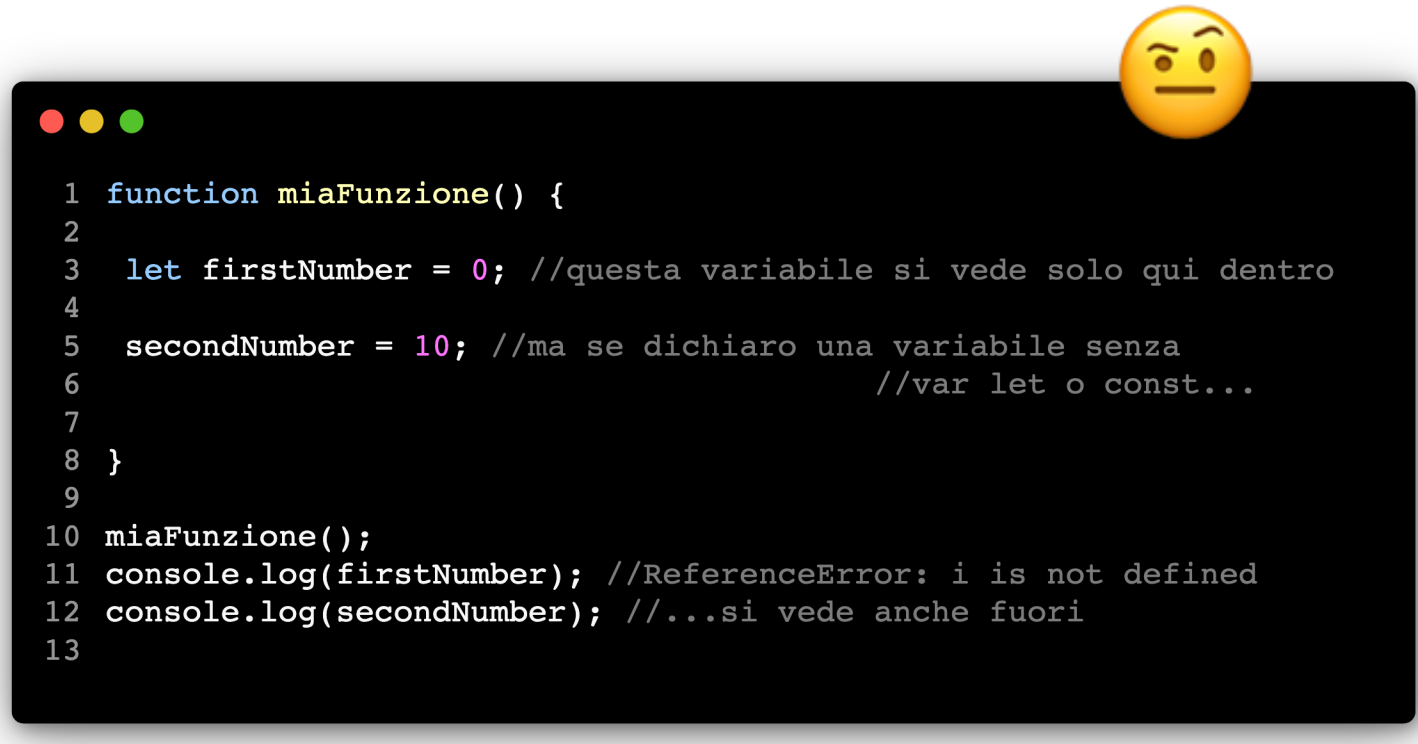
JS - Variable Scope

Variable Scope

O ambito di visibilità

Lo **scope** o *ambito di visibilità* di una variabile è la parte dello script all'interno del quale si può fare riferimento ad essa.

In pratica rappresenta un insieme di regole per cui la variabile è accessibile al nostro script.



```
1  function miaFunzione() {
2
3   let firstNumber = 0; //questa variabile si vede solo qui dentro
4
5   secondNumber = 10; //ma se dichiaro una variabile senza
6                                          //var let o const...
7
8  }
9
10 miaFunzione();
11 console.log(firstNumber); //ReferenceError: i is not defined
12 console.log(secondNumber); //...si vede anche fuori
13
```

Variable Scope

Global Variables

Una **variabile** visibile e utilizzabile ovunque nel nostro codice è detta **globale**.

```
1 const nome = 'Marta';
2
3
4 function stampaNome() {
5  console.log(nome);
6 }
7
8 stampaNome();
9 // Marta
```

Variable Scope

Global variables are Evil

- **Principio di località:**
  Da dove arriva quel valore?
- **Rompe il concetto di funzione:**
  cioè una piccola funzionalità auto-contenuta (self-contained)
- **Difficile da controllare:**
  se la funzione ritornasse “buongiorno” al posto di “ciao”, in quale delle 10K righe di codice c’è un errore?

```
1 const saluto = 'Ciao';
2 saluta();
3
4 //10 k LOCs dopo…
5
6 function saluta() {
7  alert(saluto);
8 }
```

```
1 const saluto = 'Ciao';
2 saluta(saluto);
3
4 //…
5
6 function saluta(mess) {
7  alert(mess);
8 }
```

Variable Scope

Local Scope

Esistono alcuni modi per **creare nuovi scope** (chiamati anche local scope o nested scope).

Function Scope

Ogni volta che definiamo una funzione viene creato un nuovo scope e tutte le variabili dichiarate al suo interno non sono accessibili al di fuori della funzione. Questa regola è valida con qualunque keyword venga usata per la dichiarazione (var, const, let).

Quando proviamo a richiamare esternamente una variabile dichiarata all'interno di una function otteniamo un errore.

```
1  function myFunction(params) {
2   var variable = 'pippo';
3   let secondVariable = 10;
4   const constantVariable = 'constance';
5  }
6
7  console.log(variable);
8  // Uncaught ReferenceError: variable is not
   defined
9  console.log(secondVariable);
10 // Uncaught ReferenceError: secondVariable is
   not defined
11 console.log(constantVariable);
12 // Uncaught ReferenceError: constantVariable
   is not defined
```

Block scope

Differenze tra var, let e const

Con ES6 è stato introdotto **anche** il

Block scope.

In che consiste?

Tutti i blocchi di codice tra parentesi graffe creano un nuovo scope.

Abbiamo quindi:

1. Scope delle condizioni
2. Scope dei cicli

```
1  if (condition) {
2   //scope
3  }
4
5  for (condition) {
6   //scope
7  }
8
9  function name(params) {
10  //Scope
11 }
```

Il nuovo block scope

Come si comportano i vari tipi di variabili?

Ma se dichiariamo le variabili nel **block scope** di una **condizione o ciclo** le cose cambiano!

La **var** può essere richiamata anche al di fuori del block scope.

Let e const invece **rimangono confinate tra le parentesi graffe.**

```
1  if(true) {
2   var variable = 'pippo';
3   let secondVariable = 10;
4   const constantVariable = 'constance';
5  }
6
7  console.log(variable);
8  //'pippo'
9
10
11
12
13 console.log(secondVariable);
14 // Uncaught ReferenceError: secondVariable is
   not defined
15 console.log(constantVariable);
16 // Uncaught ReferenceError: constantVariable
   is not defined
```

Il nuovo block scope

Come si comportano i vari tipi di variabili?

Cosa ci aspettiamo che succeda in questo caso?

```
1
2 for (let index = 0; index < 10; index++) {
3  console.log(index);
4 }
5
6 if(index === 10) {
7  console.log(index);
8 }
```

Ritornando alle nostre nuove variabili.

Perché sono così utili?

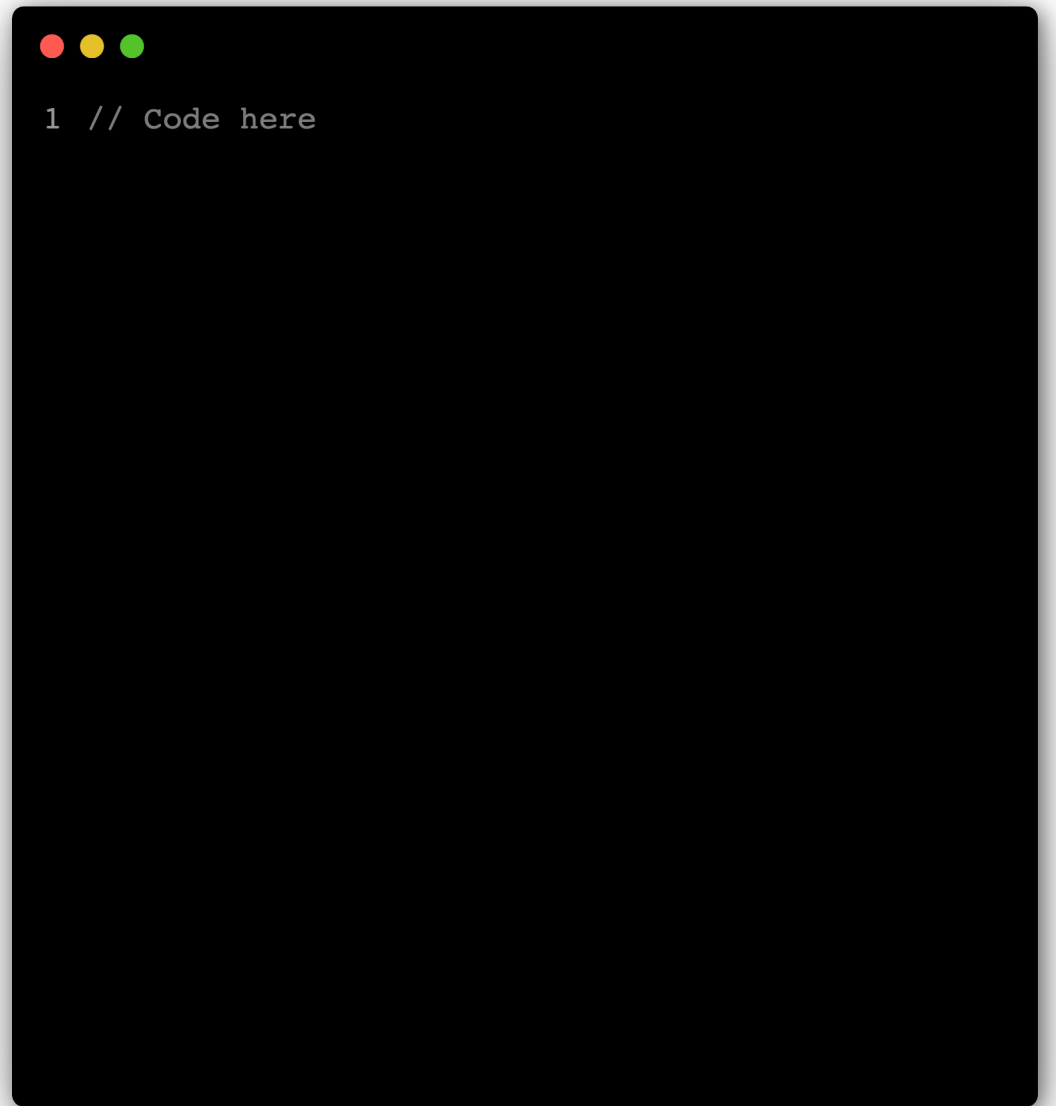
Let e const

Perché?

- **Let** ci aiuta a definire delle variabili con uno scope limitato, in questo modo sarà più difficile avere conflitti con i nomi.

- **Const** ci aiuta a creare delle costanti nelle quali inserire dei dati che sappiamo necessari per il funzionamento del nostro script e che non devono assolutamente essere cancellati per errore.

- Sia **Let** che **Const** inoltre ci consentono di ricevere un errore se proviamo a richiamare le variabili prima della loro effettiva dichiarazione.



Variable Scope

E ora?

```
a = 0;
b = "buongiorno"
var c = "ciao";

saluta();

console.log("a is " + a);
console.log("b is " + b);

function saluta() {
 var d = "asd";
 console.log("a is " + a);
 var b = a + 1;
 console.log("b is " + b);
 console.log(c);
}

```

LIVE CODING

ESERCITAZIONE