Scoping

Size	Big topic

Scoping di variabili

Dichiarazione e inizializzazione di una variabile

- *La dichiarazione di una variabile consiste nel definire il suo nome (identificatore)*
- *L'inizializzazione di una variabile consiste nell'assegnarle un valore per la prima volta*

> Una dichiarazione di funzione è un modo diverso (ma equivalente) di scrivere una dichiarazione di variabile, e inizializzarla con un valore di tipo funzione

```
function nome(a1,... an) {
      /* corpo */
}
```

```
var nome = (a1,... an) => {
      /* corpo */
}
```

La dichiarazione può essere fatta sia con che senza inizializzazione:

```
var x
```

```
var x = 5
```

Le variabili non inizializzate hanno valore undefined.

Un nome dichiarato è visibile in certe parti del programma, più o meno ampie a seconda di dove è stato dichiarato e di quale tipo di dichiarazione viene usata. JavaScript prevede tre scope o ambiti di visibilità:

- **Scope globale**
- **Scope di funzione**
- **Scope di blocco**

Scope globale

Le dichiarazioni fatte **fuori** da qualunque funzione avvengono nello **scope globale.**
Queste dichiarazioni sono **visibili in tutto il programma**, sia fuori che dentro le funzioni o i blocchi. Le variabili non dichiarate dal programma, ma usate

senza dichiarazione, si intendono implicitamente dichiarate nello scope globale

```
console.log("benvenuto")
n = 1

function map(f,a) {
  let r=[]
  for (var i=0;i<a.length;i++) {
    r.push(f(n,a[i]))
    n= 1-n
  }
  return r
}

function mul(a,b) { return a*b }

const aa = [4,7,12,3,22,1,6]
var t=map(mul,aa)
console.log(t)
```

Scope di funzione

Ogni funzione definisce il proprio scope di funzione.
I parametri formali si intendono dichiarati nello scope di funzione (ma il nome della funzione no: è nello scope globale!). Le dichiarazioni fatte all'interno di una funzione con var sono visibili nello scope di quella funzione, ma non fuori.

```
console.log("benvenuto")
n = 1

function map(f,a) {
  let r=[]
  for (var i=0;i<a.length;i++) {
    r.push(f(n,a[i]))
    n= 1-n
  }
  return r
}

function mul(a,b) { return a*b }

const aa = [4,7,12,3,22,1,6]
var t=map(mul,aa)
console.log(t)
```

Scope di blocco

Ogni blocco (0 o più comandi racchiusi fra {...}) definisce il proprio scope di blocco.
In particolare, il corpo di una funzione è un blocco. Un blocco può essere contenuto in un altro blocco. Le dichiarazioni all'interno di un for si considerano fatte nel suo blocco1. Le dichiarazioni fatte all'interno di un blocco con let e const sono visibili nello scope di quel blocco, ma non fuori.

```
console.log("benvenuto")
n = 1

function map(f,a) {
  let r=[]
  for (var i=0;i<a.length;i++) {
    r.push(f(n,a[i]))
    n= 1-n
  }
  return r
}

function mul(a,b) { return a*b }

const aa = [4,7,12,3,22,1,6]
var t=map(mul,aa)
console.log(t)
```

Annidamento e shadowing

Notate che gli scope sono contenuti uno nell'altro

- Lo scope globale contiene 0 o più scope di funzione e 0 o più scope di blocco
- Ogni scope di funzione contiene 0 o più scope di funzione, e 1 o più scope di blocco
- Ogni scope di blocco contiene 0 o più scope di funzione e 0 o più scope di blocco

Quando JavaScript incontra una dichiarazione di un identificatore, definisce una nuova variabile con quel nome e inserisce il nome nel blocco corrispondente. Quando JavaScript incontra un riferimento a un identificatore, lo interpreta come un riferimento alla dichiarazione più interna con lo stesso nome, partendo dalla posizione in cui incontra il riferimento.

Sollevamento o hoisting

Le dichiarazioni, ovunque si trovino all'interno di uno scope, sono implicitamente spostate all'inizio del loro scope (hoisting).

> *Attenzione: le inizializzazioni però rimangono dove sono scritte, quindi la variabile prima della riga in cui è dichiarata sarà visibile, ma non inizializzata (undefined nel caos di var)*

> *Eccezione: per le funzioni dichiarate con function, anche l'inizializzazione si considera fatta all'inizio dello scope (vedi esempio precedente con g())*

> *Uso comune: mettiamo da subito le dichiarazioni all'inizio dello scope, così non avremo brutte sorprese...*

Riassumendo:

1. JavaScript ha scope **globale**, di funzione, e di blocco
2. C'è **un solo scope globale**, ma gli **scope di funzione** e di blocco **possono essere annidati** uno nell'altro
3. Le dichiarazioni con `let` e `const` hanno **scope di blocco, o di funzione, o globale** (vale lo scope più interno in cui sono scritte)
4. Le dichiarazioni con `var` hanno **scope di funzione o globale** (vale lo scope più interno in cui sono scritte)
5. Le variabili non dichiarate hanno **scope globale**
6. Le dichiarazioni con `function` si comportano come var, ma si "portano dietro" l'inizializzazione in caso di hoisting