Capire Javascript: le funzioni

Particolarità e utilizzo delle funzioni in Javascript

Paolo Caramanica

Sommario

- Introduzione: i linguaggi procedurali e le funzioni
  - Perché le funzioni?
  - Definire una funzione in Javascript
  - Chiamata, parametri e valore di ritorno
  - Live demo
- Particolarità delle funzioni in Javascript
  - Le function expression
  - Funzioni come valori
  - Le arrow function
  - Live demo
- Lo scope
  - Lo scope
  - Lexical Environment (esempio)
  - Il concetto di closure
  - var vs let
  - Live demo

Introduzione

I linguaggi procedurali e le funzioni

Perché le funzioni?

- Problema:
  - Spesso è necessario effettuare la stessa operazione (o operazioni simili) in diversi punti del codice
  - Questo porta a codice duplicato
- Soluzione:
  - Il paradigma di programmazione procedurale introduce le funzioni (procedure)
  - Le funzioni sono blocchi di codice, identificati da un nome, richiamabili in altri punti del programma
  - Possono ricevere dei parametri in input e produrre un output (valore di ritorno)

Definire una funzione in Javascript

- Per definire una funzione in Javascript:
  - Si usa la keyword function
  - Si sceglie un nome univoco
  - Si inserisce il blocco di codice tra parentesi graffe
- Esempio:

```
function Saluto() {
    alert('Ciao');
}
```

Chiamata, parametri e valore di ritorno

- Chiamata (esecuzione) di una funzione precedentemente definita:

```
Saluto()
```

- Le funzioni possono accettare parametri, o argomenti, cioè valori dati in input alla funzione

```
function Saluto2(nome) { … }
```

- Le funzioni possono restituire un valore (valore di ritorno), cioè fornire un output utilizzabile dal codice che l'ha chiamata

```
function AreaQuadrato(lato) {
    return lato*lato;
}
```

Live demo

Le funzioni in Javascript

Particolarità delle funzioni in Javascript

Le function expression

- In Javascript le funzioni possono essere definite anche tramite le **function expression**
  - Esempio: (function expression)

```
let f = function() { alert('Ciao!'); }
```

è equivalente a (function declaration)

```
function f() { alert('Ciao!'); }
```

Funzioni come valori

- Le funzioni in Javascript sono dei valori:
  - Possono essere assegnate ad una variabile (function espression)
  - Possono essere definite dentro un'altra funzione (nesting)
  - Possono essere passate come parametri ad un'altra funzione (callback)
  - Possono essere restituite da una funzione

Le arrow function

- In Javascript esiste una terza sintassi per definire le funzioni:
  - **Function declaration**:

```
function Somma(a,b) { return a+b; }
```

  - **Function expression**:

```
let Somma = function(a,b) { return a+b; }
```

  - **Arrow function**:

```
let Somma = (a,b) => a+b
```

Live demo

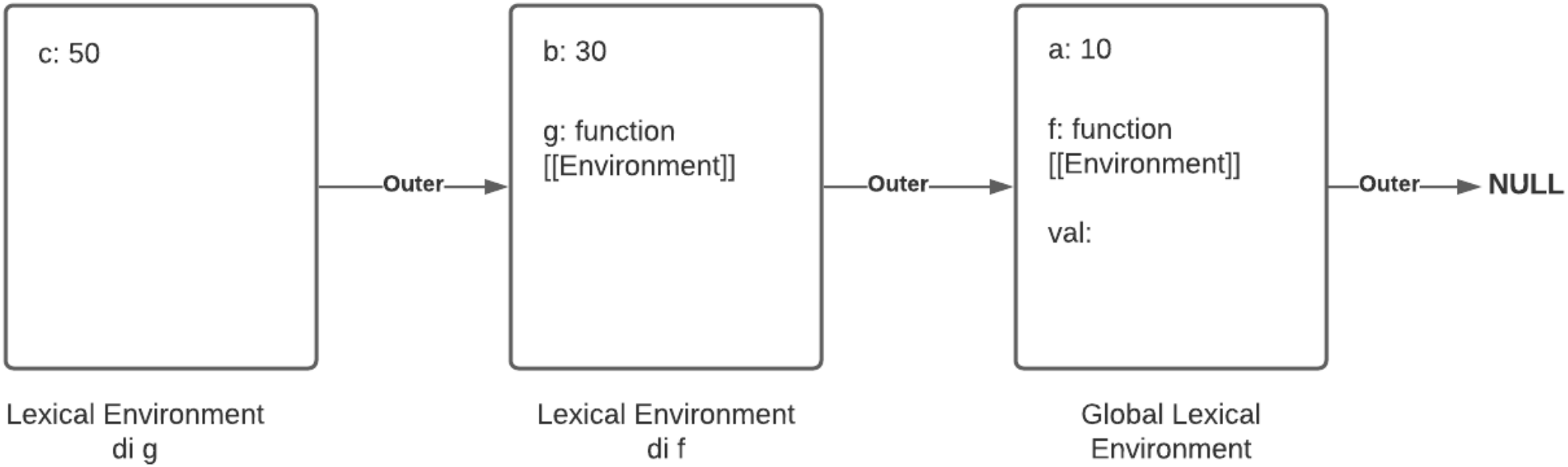
Lo scope

Lo scope e le closure

Lo scope

- Scope: visibilità, quindi possibilità di richiamare una variabile in un determinato punto del codice
- In Javascript:
  - Le variabili definite con let all'interno di un blocco di codice { … } sono visibili solo in tale blocco (**block scope**)
  - Se si fa riferimento ad un nome che non esiste nel blocco { … }, si cerca nel blocco che lo contiene, in modo ricorsivo
- Nelle funzioni innestate:
  - Se si fa riferimento ad una variabile non definita nella funzione stessa, si cerca nella funzione immediatamente superiore, in modo ricorsivo
  - Lo scope viene gestito tramite il **Lexical Environment**

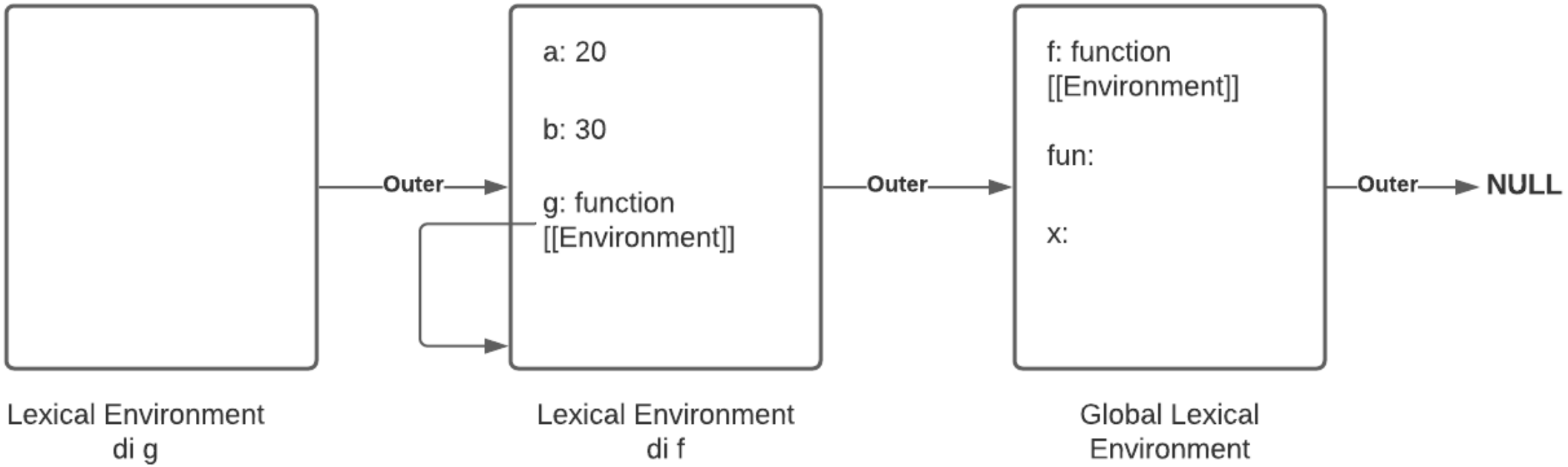
Lexical Environment (esempio)



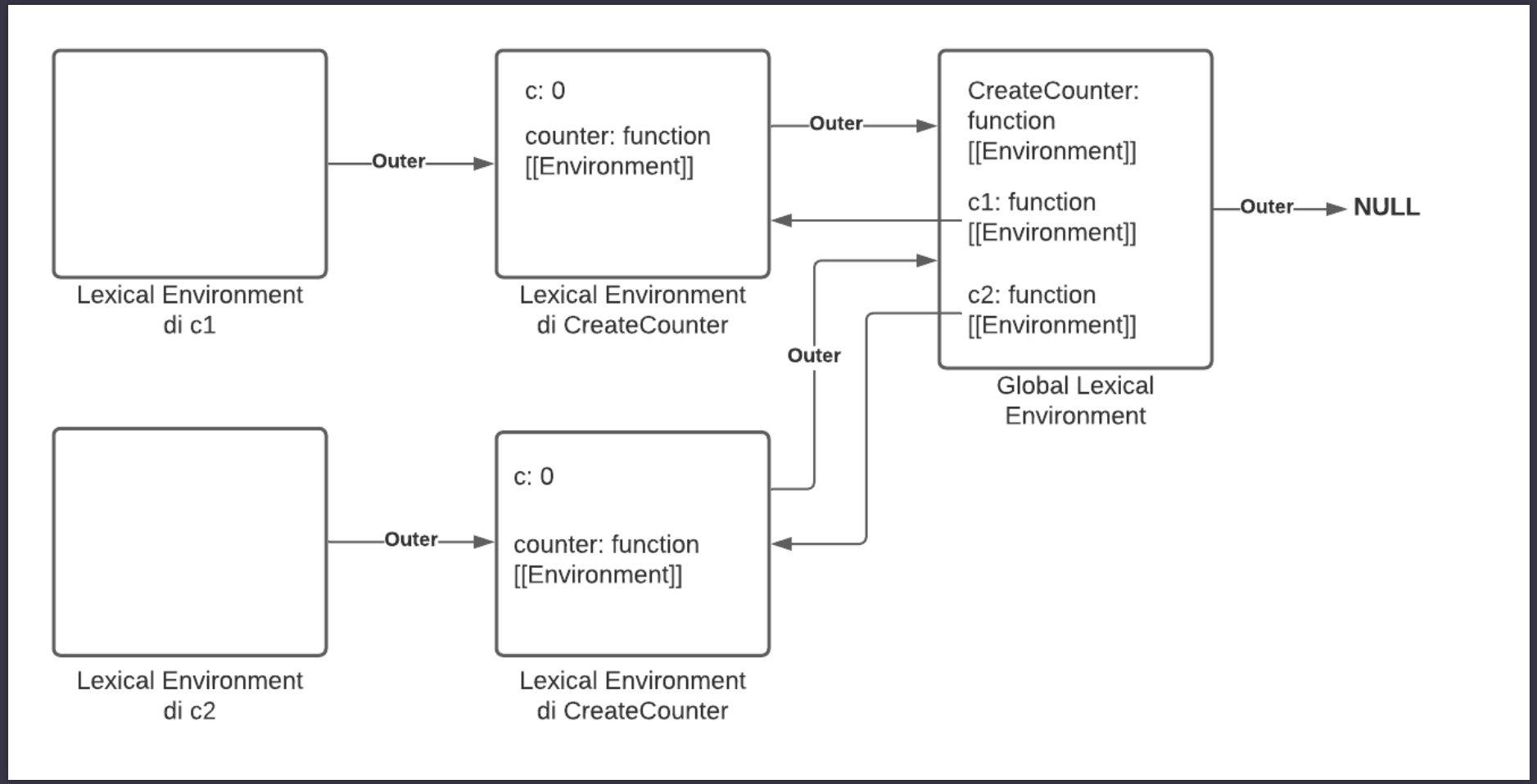
Il concetto di closure

- Nell'esempip precedente, il lexical environment di una funzione:
  - Viene creato quando inizia l'esecuzione
  - Viene eliminato quando termina l'esecuzione
- Supponiamo che una funzione b sia definita dentro una funzione a:
  - Che succede se b continua ad eseguire dopo che a è terminata?
- In Javascript, una funzione «ricorda» l'environment in cui viene creata e le relative variabili, cioè è una **closure**.

Lexical environment (esempio 2)



Lexical environment (esempio 3)



var vs let

- Le variabili dichiarate con var:
  - Possono essere utilizzate prima della dichiarazione
  - Possono essere ri-dichiarate
  - Non hanno block scope (sono visibili globalmente o all'interno della funzione in cui sono definite)

Live demo

Contatti

Email: paolocaramanica@gmail.com

Linkedin: https://it.linkedin.com/in/paolo-caramanica-436942a/it-it

Facebook: https://it-it.facebook.com/paolo.caramanica

Q&A