Introduzione a JavaScript

Corso di Laboratorio di Informatica

Prof. Dei Rossi, Leonardo Essam

Cenni storici

JavaScript è un linguaggio di programmazione che nasce nel 1995 per rendere dinamiche e interattive le pagine web.

JavaScript nasce nel periodo del Web 1.0, un periodo dove le pagine Web erano puro codice HTML (se andava bene c'era anche un po' di CSS...) ed erano di sola consultazione (*read-only*).

Dal '95 in poi la questa tendenza inizia piano-piano a cambiare, passando da un uso di Internet solo *read-only* ad un uso **read-write**.

Funzionamento

JavaScript è un linguaggio di programmazione multi-paradigma orientato agli **eventi**. Viene utilizzato sia lato *client* (esecuzione nel browser) sia lato server (Node.js, non oggetto di questo corso).

JavaScript è un linguaggio interpretato: il codice non viene compilato, ma eseguito direttamente; lato client, il codice viene eseguito dall'interprete contenuto nel browser.

Ha una sintassi *C-like* ed è un linguaggio **debolmente tipizzato**.

Dichiarazione e assegnazione di variabili

Esistono tre modi per assegnare variabili in JavaScript:

```
1  var name = "Alice";
2  let age = 25;
3  const country = "Italy";
```

Osserviamo che `const` si usa per le costanti (valori immutabili)...
...ma `var` e `let`?

Dichiarazione di variabili: let vs. var (1)

In JavaScript, `let` e `var` sono due modi per assegnare variabili.

La differenza sta nell'ambito (*scope*) della variabile:

```
1  if (true) {
2      var x = 10;
3      let y = 20;
4  }
5
6  console.log(x);     // 10 (var esce dal blocco)
7  console.log(y);     // ReferenceError (let e' solo nel blocco)
```

Dichiarazione di variabili: let vs. var (2)

La scelta di uno dei due metodi di dichiarazione modifica anche il comportamento di JavaScript in caso di ri-dichiarazione (*redeclaration*):

```
1  var a = 5;
2  var a = 6;    // OK - operazione permessa
3
4  let b = 5;
5  let b = 6;    // ERRORE - gia' dichiarata
6
7  b = 6;        // OK - riassegnazione permessa
```

Dichiarazione di variabili: let vs. var (3)

Un'altra differenza tra `let` e `var` si trova nell'*hoisting*[1].

In JavaScript, l'hoisting consente di utilizzare funzioni e variabili prima che siano dichiarate.

```
1  console.log(foo);
2  var foo = 'foo';
```

Cosa stampa `console.log(foo);` ?

[1] *What is Hoisting in JavaScript?*, freeCodeCamp (link)

Dichiarazione di variabili: let vs. var (4)

```
> console.log(foo);
  var foo = 'foo';
  undefined                                    VM55:1
```

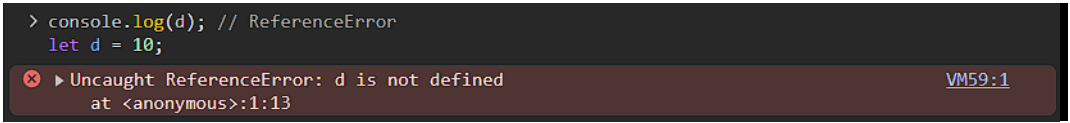
Questo perché l'interprete JavaScript (ricordiamo: nel browser) **divide dichiarazione e assegnazione di funzioni e variabili:** "innalza" le dichiarazioni in cima all'ambito in cui sono contenute prima dell'esecuzione.

Questo processo è chiamato **hoisting** (dall'inglese *to hoist*: sollevare, issare) e consente di utilizzare `foo` prima che sia dichiarata.

Dichiarazione di variabili: let vs. var (5)

La differenza tra `let` e `var` (nell'hoisting) sta nel fatto che:

- `var` viene inizializzata con `undefined` subito, quindi si può usare prima della sua reale dichiarazione;
- `let` è in una "Temporal Dead Zone" (TDZ) fino alla riga della dichiarazione.



Conversioni tra tipi di dato (1)

JavaScript è un linguaggio dinamicamente tipizzato, quindi il tipo delle variabili cambia in base al valore che viene assegnato (o riassegnato) durante l'esecuzione del programma.

In JavaScript si può scrivere:

```
1  var a = 10;
2  console.log(typeof a);  // number
3  a = "Ciao";
4  console.log(typeof a);  // string
```

Conversioni tra tipi di dato (2)

In Java invece questo non è possibile[2]:

```
1  Integer a = 10;
2  System.out.println(a.getClass().getName()); // java.lang.Integer
3
4  a = "Ciao"; // ERRORE DI COMPILAZIONE (incompatible types)!
5  // java.lang.String cannot be converted to java.lang.Integer
```

[2] Perché Java è un linguaggio staticamente tipizzato.

Conversioni tra tipi di dato (3)

Per essere sicuri di avere i tipi di dato corretti (ad esempio, quando si prende in input un valore dall'utente), JavaScript mette a disposizione dei metodi per il *parsing* di una data variabile e convertirla nel tipo desiderato (*casting*).

```
1  var a = parseInt("1");
2  var b = parseFloat("3.14");
3  var c = parseInt("CIAO");
4
5  console.log(typeof a);      // number
6  console.log(typeof b);      // number (??)
7  console.log(typeof c);      // number (???)
```

Conversioni tra tipi di dato (4)

È facilmente intuibile che JavaScript raggruppi tutti i valori numerici che solitamente rappresentiamo come `int`, `float` e `dobule` dentro al tipo di dato `number`.

Allora perché il valore di ritorno di `parseInt("CIAO");` è comunque un tipo `number` se effettivamente non è un numero?

Conversioni tra tipi di dato (5)

Analizziamo la funzione `parseInt`, non avendo accesso al codice sorgente possiamo fare solo un'ipotesi:

```
1  int parseInt(string a) {
2      [...]
3  }
```

Il tipo di ritorno sarà sempre `int`, anche se il valore non è convertibile in un numero (in questo caso: un numero intero)!

Cosa restituisce quindi? Provate!

Conversioni tra tipi di dato (6)

Aprite il browser e andate su:

https://tinyurl.com/cs300-lab-02

In fondo alla pagina troverete una voce "Try yourself".

Buonarroti