Javascript

Javascript

Javascript è, oggi, un linguaggio di programmazione completo e dotato di tutti i più moderni costrutti, da quelli più semplici (funzioni, array…) a quelli più complessi (oggetti, ereditarietà…).

Nell'ambito delle applicazioni web è utilizzabile sia sul backend che sul frontend.

In ambito frontend Javascript è utilizzato, assieme a HTML e CSS, per dare "dinamicità" alle pagine web.

Tramite Javascript è infatti possibile:

- Modificare la struttura della pagina web, aggiungendo/rimuovendo tag, aggiungendo e rimuovendo contenuti, aggiungendo e rimuovendo classi
- Comunicare con il server anche in situazioni in cui questo non è previsto dallo standard HTML (vedi applicazioni Ajax e SPA)

Caratteristiche di Javascript

Alcune caratteristiche di Javascript:

- E' imperativo. Un programma è cioè costituito di un insieme di istruzioni da eseguire
- E' strutturato. Sono disponibili istruzioni di controllo del flusso e delle iterazioni.
- E' procedurale. E' possibile dividere il codice in blocchi richiamabili in un secondo momento.
- E' funzionale. Le funzioni sono un tipo di dato elementare. E' possibile usarle come parametri e come valori di ritorno verso altre funzioni. E' inoltre possibile definire funzioni all'interno di altre funzioni.
- E' orientato agli oggetti. E' cioè possibile definire oggetti ed utilizzare l'ereditarietà, il polimorfismo e l'incapsulamento.
- E' basato sui prototipi. Non esiste il concetto di classe ma solo di oggetti che svolgono il ruoli di "template" da cui ereditare.

Caratteristiche di Javascript

- E' a tipizzazione dinamica. Il controllo dei tipi di dato e la loro conversione (o non conversione) avviene a runtime e non a compile time. Una variabile può cambiare tipo durante il ciclo di vita dell'applicazione.
- E' a tipizzazione debole. Non è necessario dichiarare il tipo delle variabili. Viene dedotto automaticamente. Il tipo di una variabile può inoltre cambiare nel tempo.
- E' interpretato. Il codice sorgente è interpretato a runtime. E' pertanto necessario un interprete.
- Implementa un garbage collector per liberare la memoria quando le variabili non sono più utilizzate.
- E' standardizzato. Esiste uno standard chiamato ECMA-262.

Il browser come interprete javascript

Tutti i browser web contengono un interprete javascript, necessario per eseguire il codice js contenuto nelle pagine html.

I browser mettono inoltre a disposizione del codice javascript due Object Model con i quali interagire per ottenere le funzionalità desiderate:

- BOM – Browser Object Model – Rappresenta le funzionalità e le caratteristiche del browser. Per esempio: la finestra del browser con le sue proprietà, l'url attuale…
- DOM – Document Object Model – Rappresenta la struttura del documento HTML. Per esempio, tag, classi, contenuti…

Usare Javascript in pagine HTML

Sono possibili più modalità per inserire del codice javascript all'interno di una pagina web. I più frequenti sono:

- Inserimento di codice javascript all'interno dei normali tag html, come valore di attributi che rappresentano eventi
- Inserimento di codice javascript all'interno della pagina html dentro il tag <script>
- Import di codice javascript scritto su un file esterno. Anche in questo caso si utilizza il tag <script>

In generale, i tag <script> vanno inseriti prima della chiusura del tag <body>. Ci sono tuttavia eccezioni con alcuni js di terze parti che devono essere inseriti nel tag <head>

Usare Javascript in pagine HTML

- Inserimento di codice javascript all'interno dei normali tag html, come valore di attributi che rappresentano eventi

```
<span onclick="alert('Hello World!');"> Click me </span>
```

L'attributo onclick rappresenta un evento. Può contenere del codice javascript. Un insieme di istruzioni, come nell'esempio o la chiamata ad una funzione definita altrove.

Usare Javascript in pagine HTML

- Inserimento di codice javascript all'interno della pagina html dentro il tag <script>

```
<script type="text/javascript">
      function hw() { alert('Hello World'); }
</script>
```

Il tag script può essere inserito nell'head oppure in un punto qualsiasi del body.

Usare Javascript in pagine HTML

- Import di codice javascript scritto su un file esterno. Anche in questo caso si utilizza il tag <script>

```
<script  type="text/javascript" src="nomefile.js"></script>
```

Node.js come interprete javascript

Node.js è un runtime per l'esecuzione di codice Javascript. Utilizza il motore V8 creato da Google ed utilizzato anche in Chrome.

Tramite node è possibile utilizzare javascript anche per la realizzazione di applicazioni backend, in quanto non si è più vincolati alla presenza di un browser.

Node ha una architettura molto performante ed adatta a situazioni in cui sono necessarie altre performance, come nei sistemi real time. Node è inoltre orientato ai messaggi e asincrono.

L'esecuzione di codice js in node è molto semplice. Da riga di comando: `node nomefile.js`

Javascript – Sintassi di base

Javascript – sintassi di base

- Javascript è case sensitive
- Le istruzioni vanno terminate con il ; solo se scritte sulla stessa riga.
- I blocchi di istruzioni vanno identificati con le parentesi graffe { }
- L'indentazione non è necessaria per l'esecuzione del codice (rimane fondamentale per la comprensione).
- I commenti si creano con i caratteri // per i commenti a singola linea e con i caratteri /* */ per i commenti a linea multipla
- Gli operatori matematici sono = + - * / % ++ -- += -= > >= < <= !=
- Gli operatori logici sono == !(not) &&(and) || (or)

Javascript – variabili

Le variabili si istanziano con la parola chiave var, senza specificarne il tipo.

Il tipo verrà dedotto dal valore assegnato alla variabile. In caso di mancata assegnazione la variabile varrà undefined.

```
var x = 10;
var y = '10';
var z = true;
var w = null;
var j; //undefined
```

- Qualsiasi numero (intero, decimale) è di tipo number
- I valori booleani si esprimono con true e false.
- null identifica una variabile inizializzata con valore non noto
- undefined identifica una variabile non inizializzata

null e undefined sono sia valori che tipi di dato.

Javascript – visibilità delle variabili

Le variabili si differenziano il locali, se definite all'interno di una funzione, o globali, se definite all'esterno di tutte le funzioni.

Le variabili locali sono "visibili" all'interno della funzione in cui sono state definite e all'interno di tutte le funzioni eventualmente definite dentro di essa.

Le variabili globali sono invece "visibili" in ogni punto del codice.

Le variabili javascript, definite con var, godono della proprietà di hoisting, ovvero sono visibili in ogni parte della funzione (o del blocco <script>) in cui sono definite, anche prima della definizione stessa (ovviamente con valore undefined).

Javascript – visibilità delle variabili

La visibilità di una variabile definita con var è molto ampia è può portare ad errori, soprattutto legati a due aspetti:

- La stessa variabile può essere definita più volte senza generare errori.
- Una variabile è visibile anche esternamente al blocco che l'ha definita.

```
var temp="valore temporaneo";
if( … )
{
        var temp = "nuovo valore";
}
console.log(temp);
```

Javascript – let e const

Per ovviare a questi problemi la versione 6 dello standard del linguaggio (ES6) introduce altre due parole chiave:

- **let:** permette di definire una variabile che può essere assegnata più volte ma inizializzata una volta sola
- **const:** permette di definire una costante che può essere inizializzata ed assegnata una sola volta.

L'utilizzo di let e const inibisce la proprietà di hoisting

```
let temp="valore temporaneo";
if( … ){
        let temp = "nuovo valore"; //ERRORE
        var temp = "nuovo valore"; //ERRORE
        temp = "nuovo valore"; //OK
}
```

Javascript – stringhe

Le stringhe in javascript possono essere contenute, indifferentemente, tra una coppia di apici singoli oppure tra una una coppia di doppi apici oppure tra una coppia di backtick `

I carattere di apertura e di chiusura della stringa devono coincidere e all'interno di una stringa aperta con un carattere è ammesso l'utilizzo degli altri tipi di carattere.

Questa tripla possibilità permette javascript di essere inserito in più contesti, adattandosi di conseguenza.

```
var a = '10';

var b = "10";

var c = '"Hello World"';

var w = "Reggio nell'Emilia";
```

Javascript – stringhe

L'utilizzo di backtick ` (alt+96 su windows e option+9 su mac) permette di abilitare due caratteristiche non disponibili con gli altri delimitatori:

- stringhe multiline:

```
var s = `Hello
World`;
```

- Interpolazione:

```
var n = 10;
var s1 = `Il numero inserito è ${n}`;
var s2 = `Il doppio del numero inserito è ${n*2}`;
```

Javascript – stringhe

Le stringhe sono oggetti e come tali hanno proprietà e metodi. Tra le proprietà troviamo:

- **lenght** – restituisce il numero di caratteri di cui la stringa è composta

Tra i metodi troviamo invece:

- **slice( ), substring( ), substr( )** – estraggono una porzione di una stringa creandone una nuova. I tre metodi sono identici nell'obiettivo ma differenti nei parametri utilizzati.
- **replace ( )** – sostituisce una parte di una stringa con un'altra
- **toUpperCase( ), toLowerCase( )** – converta la stringa, rispettivamente in tutti caratteri maiuscoli o minuscoli.
- **concat( )** – concatena due stringhe (in alternativa utilizzare +)
- **charAt( )** – estrae un carattere specificandone l'indice (in alternativa utilizzare [ ])
- **repeat( )** – genera una nuova stringa ripetendo n volte quella iniziale

Javascript – stringhe

Tra i metodi troviamo invece:

- **split( )** – converte una stringa in un array di stringhe specificando un separatore
- **indexOf( ), search( )** – cercano una stringa all'interno di un'altra e restituisce l'indice di partenza della prima occorrenza. Hanno lo stesso scopo ma parametri differenti
- **lastIndexOf( )** – cerca una stringa all'interno di un'altra e restituisce l'indice di partenza dell'ultima occorrenza
- **includes( )** – cerca una stringa all'interno di un'altra e restituisce true/false in base all'esito della ricerca
- **startsWith( ), endsWidth( )** – Determina se una stringa inizia/finisce con un'altra stringa passata come parametro

Javascript – valori booleani

In Javascript esistono i valori true e false ma il concetto di vero/falso è più ampio:

- In js sono considerati come **FALSO** i seguenti valori:
  - false
  - undefined
  - null
  - 0
  - ''
  - NaN
- In js è considerato vero tutto ciò che non è falso

`false || 'Hello'` è perfettamente valido e restituisce 'Hello'

Javascript == vs ===

In Javascript esiste anche l'operatore di ugualianza ===. Ci sono differenze rispetto al classico operatore ==:

- == converte i tipi dei due operandi prima di effettuare il confronto
- === non converte i tipi dei due operandi

```
const a = "test"
const b = "test"
console.log(a == b) //true
console.log(a === b) //true
```

```
const a = 1234
const b = '1234'
console.log(a == b) //true
console.log(a === b) //false
```

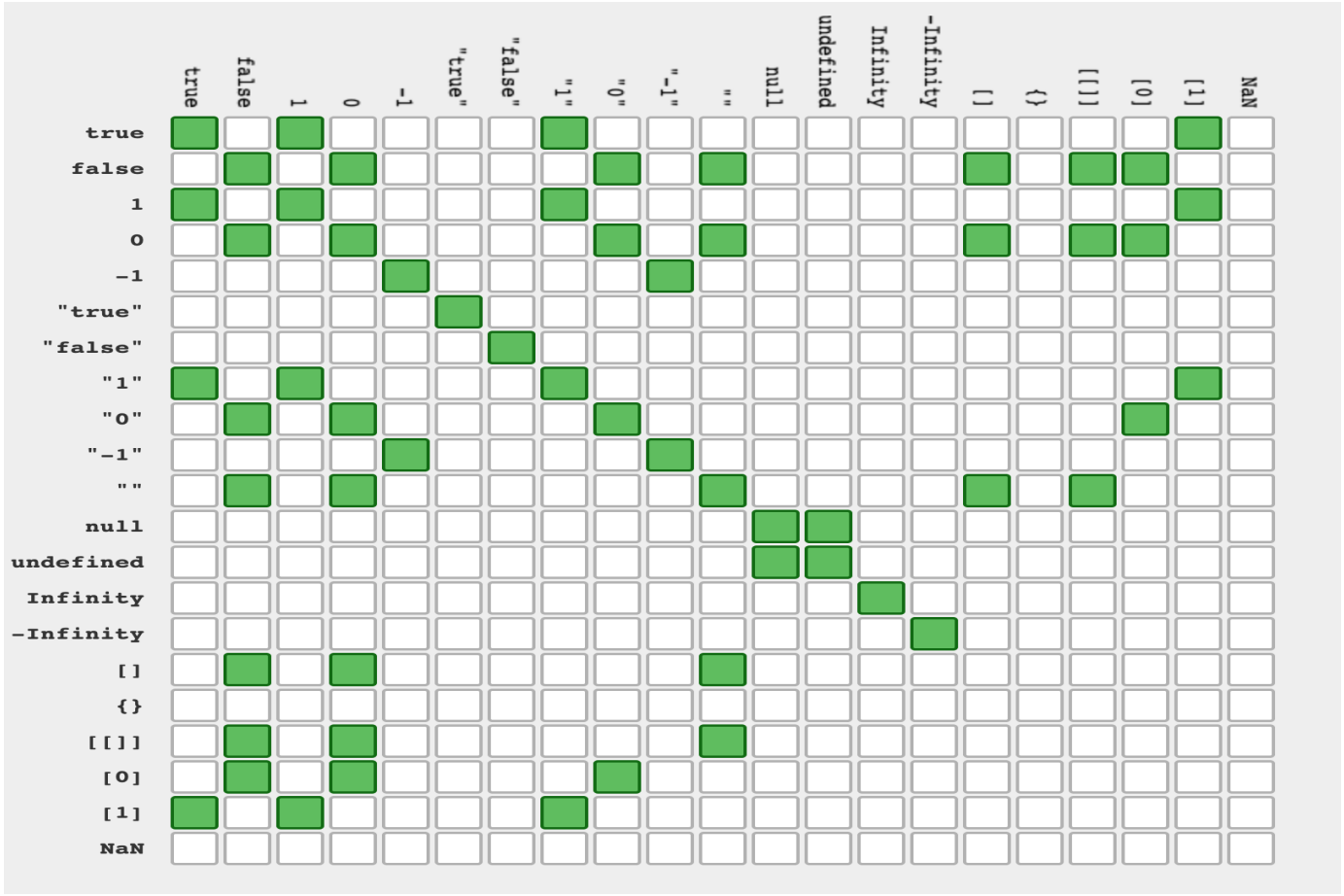
```
console.log(0 == false) //true
console.log(0 === false) //false
```

```
console.log("" == false) //true
console.log("" === false) //false
```

Javascript == vs ===

Tabella di riepilogo dell'uguaglianza effettuata con ==.

(sorgente https://dorey.github.io/JavaScript-Equality-Table/)



Javascript – Controllo del flusso

Javascript – if

L'operatore if ha una forma abbastanza standard:

```
if (condizione) { }
else if(condizione) { }
else { }
```

- La condizione deve essere sempre contenuta in parentesi tonde
- Le parentesi graffe non sono obbligatorie se il blocco di codice è composto da una sola istruzione
- Il blocco else if è facoltativo e ne possono esistere più di uno
- Il blocco else è facoltativo. Se presente va sempre dopo l'ultimo blocco else if

Javascript – operatore ternario

Per semplici espressioni l'operatore if può anche essere scritto in una sola riga:

```
condizione ? valoreSeVero : valoreSeFalso
```

La condizione viene valutata e viene restituito valoreSeVero o valoreSeFalso in base al risultato, rispettivamente, vero o falso.

L'operatore ternario è molto comodo per valorizzare variabili in modo condizionale:

```
var a=10;

var b=20;

var c= a<b ? a : b;
```

Javascript – switch case

Tramite l'operatore switch case è possibile valutare una espressione ed eseguire blocchi di codice distinti in base al valore.

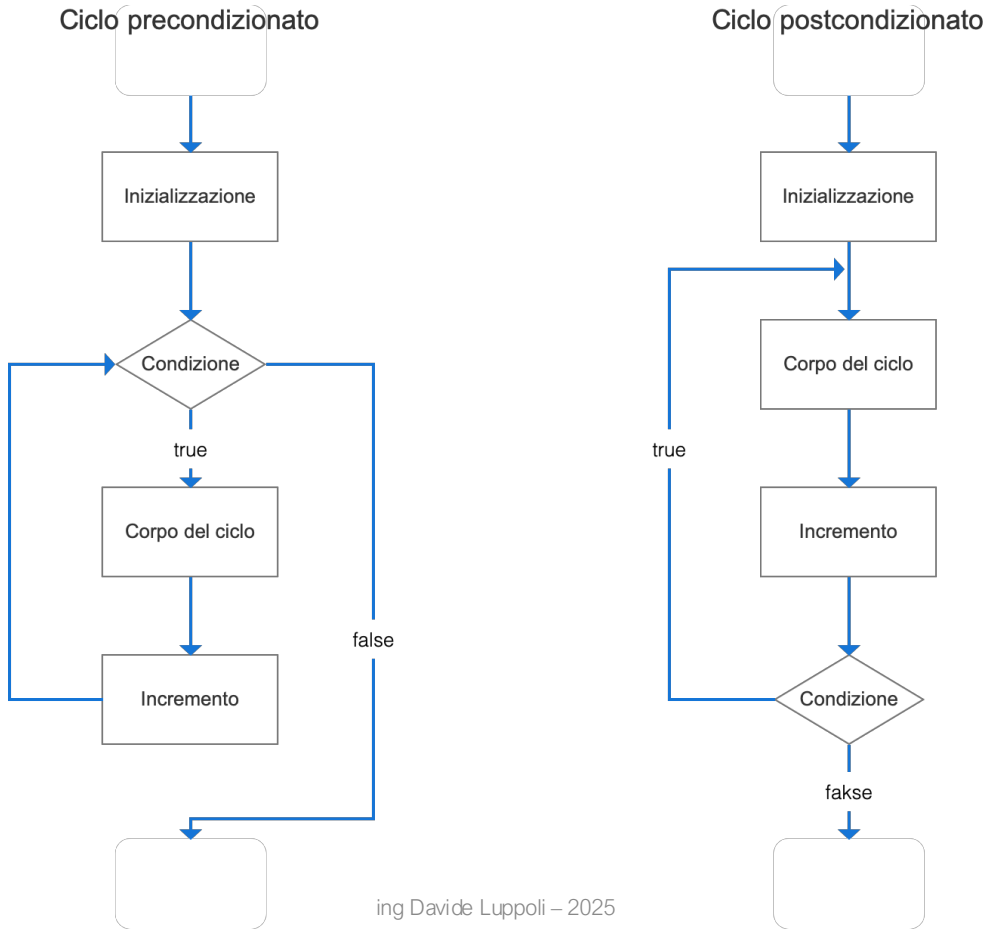
```
switch (espressione)
{
          case valore1:
                    istruzioni;
                    break;
          case valore2:
          case valore3:
                    istruzioni;
                    break;
          default:
                    istruzioni;
}
```

Javascript – switch case

- I blocchi di istruzioni associati ad ogni case non devono essere racchiusi in parentesi graffe.
- Il case di default, eseguito se tutti i case precedenti non sono verificati, non è obbligatorio
- La parola chiave break, posta alla fine di ogni case, interrompe il flusso di esecuzione facendolo uscire dallo switch. Non è obbligatoria ed in caso di omissione l'esecuzione continua verso i case successivi.
- Il case di default, essendo l'ultimo non necessita mai di break

Controllo del flusso – cicli

Tramite i cicli è possibile eseguire un blocco di codice più volte. Esistono due tipologie di cicli, precondizionati e postcondizionati, con i seguenti diagrammi di flusso:



Controllo del flusso – cicli

In Javascript i principali costrutti che permettono di realizzare cicli sono:

- While (precondizionato)
- do … while (postcondizionato)
- For (precondizionato)
- For of (senza condizioni, lo vedremo in seguito)
- For in (senza condizioni, lo vedremo in seguito)

Javascript – while

L'operatore while permette di realizzare cicli precondizionati, eseguendo un blocco di codice fintanto che una determinata condizione è vera:

```
while (condizione)
{ … }
```

- La condizione deve essere sempre contenuta in parentesi tonde
- Le parentesi graffe non sono obbligatorie se il blocco di codice è composto da una sola istruzione
- Se la condizione rimane sempre vera si genera un loop infinito

Javascript – do while

L'operatore do while permette di realizzare cicli postcondizionati, eseguendo un blocco di codice fintanto che una determinata condizione è vera. L'esecuzione avviene comunque almeno una volta.

```
do{ … }
while (condizione)
```

- La condizione deve essere sempre contenuta in parentesi tonde
- Le parentesi graffe non sono obbligatorie se il blocco di codice è composto da una sola istruzione
- Se la condizione rimane sempre vera si genera un loop infinito

Javascript – for

L'operatore for permette di realizzare cicli precondizionati e articolati caratterizzati da espressioni di inizializzazione, condizione di uscita e espressioni di iterazione.

```
for(espressioneIniziale; condizione; espressioneIterazione)
{ … }
```

Per prima cosa viene eseguita l'espressioneIniziale. Viene poi valutata la condizione, se è vera si esegue il corpo contenuto tra le graffe, se è falsa l'esecuzione passa oltre.

Una volta terminata l'esecuzione del corpo viene eseguita l'espressione Iterazione e si torna a valutare la condizione.

Il ciclo va avanti fino a quando la condizione non diventa falsa.

Javascript – break e continue

All'interno degli operatori di iterazione (while, do while, for e for in) è possibile utilizzare le istruzioni break e continue con le quali modificare il flusso di esecuzione:

- break: interrompe completamente il ciclo. L'esecuzione riprende dall'istruzione successiva al ciclo.
- continue: interrompe la singola iterazione del ciclo. L'esecuzione passa all'iterazione successiva valutando le condizioni nel modo previsto dal particolare tipo di ciclo utilizzato.