Programmazione Web: JavaScript - Elementi Base

2.1 - Introduzione a JavaScript

JavaScript è nato nei primi anni del World Wide Web, quando divenne chiaro che era necessario aggiungere capacità di elaborazione alle pagine HTML statiche. È stato concepito come un linguaggio di scripting lato client (Client-Side Script), ovvero programmi incorporati nella pagina HTML che vengono interpretati direttamente all'interno del browser.

JavaScript è stato ispirato a Java, ma è molto più semplice e adotta un approccio diverso alla programmazione orientata agli oggetti. Una caratteristica fondamentale di JavaScript è la valutazione dinamica sia delle istruzioni che dei tipi delle variabili. Questo approccio offre vantaggi in termini di velocità e flessibilità di programmazione per progetti di complessità ridotta, ma rende più difficile il debugging in scenari complessi.

2.2 - JavaScript e HTML

Linguaggio con tipizzazione dinamica (non tipizzato) di variabili e istruzioni.

Per incorporare JavaScript in una pagina HTML, si utilizza l'elemento `<script>` :

```
<script type="text/javascript">
// Codice JavaScript
</script>
```

Questo elemento può essere posizionato ovunque nel documento HTML e può essere ripetuto più volte.

Storicamente, l'attributo `language` veniva utilizzato per specificare la versione di JavaScript:

```
<script type="text/javascript" language="javaScript1.5">
// Codice JavaScript
</script>
```

Tuttavia, l'attributo `language` è stato deprecato poiché i suoi valori non sono mai stati standardizzati. È preferibile utilizzare solo `type="text/javascript"` o, più semplicemente, omettere completamente l'attributo `type` poiché JavaScript è il linguaggio di scripting predefinito nei browser moderni.

2.3 - Elementi di Base di JavaScript

Variabili

In JavaScript, le variabili vengono definite utilizzando la parola chiave `var` (il tipo, essendo un linguaggio non tipizzato, viene riconosciuto in automantico):

```
var v = 1;      // numero intero
var n = "Pippo"; // stringa
```

Il tipo di dato viene determinato dinamicamente in base al valore assegnato. Anche se a prima vista potrebbe sembrare che la parola chiave `var` non sia necessaria, essa ha un importante ruolo nel definire il contesto (scope) della variabile, rendendola locale al contesto in cui è definita.

Assegnamento e Operatori

L'operatore di assegnamento in JavaScript è il simbolo `=` (singolo, mentre `==` viene utilizzato per il confronto):

```
A = 2;
B = A * 3;
```

In JavaScript, il punto e virgola (`;`) è un separatore di istruzioni sulla stessa riga. Sebbene non sempre necessario, è una buona pratica di programmazione includerlo per prevenire errori.

Gli operatori aritmetici principali sono:

- `a + b` (somma)
- `a - b` (sottrazione)
- `a * b` (prodotto)
- `a / b` (divisione)

Esistono anche assegnamenti incrementali:

- `a += b` equivalente a `a = a + b`

- `a -= b` equivalente a `a = a - b`
- `a *= b` equivalente a `a = a * b`
- `a /= b` equivalente a `a = a / b`

Gli operatori di incremento/decremento funzionano in due modi:

- Postfisso: `a++` o `a--` (prima valuta, poi incrementa/decrementa)
- Prefisso: `++a` o `--a` (prima incrementa/decrementa, poi valuta)

Istruzioni di Controllo

JavaScript utilizza le stesse strutture di controllo di C, C++ e Java:

- `if`
- `while`
- `do ... while`
- `for`
- Con i comandi `continue` e `break`

Gli operatori di confronto sono:

- `A > 0`
- `A >= 0`
- `A < 0`
- `A <= 0`
- `A == 0`
- `A != 0`

Gli operatori logici sono:

- `&&` (AND)
- `||` (OR)
- `!` (NOT)

Funzioni

Le funzioni in JavaScript seguono questa sintassi:

```
function nome(parametri formali) {
    // codice
    return valore;
}
```

Esempio di una funzione che calcola la somma:

```
function somma(x, y) {
    var r = x + y;
    return r;
}
```

In JavaScript, il tipo dei parametri e del valore restituito non vengono specificati ma sono valutati dinamicamente. Questo permette grande flessibilità, come mostrano i seguenti esempi:

```
var A = 2;
var B = A * 3;
var C = somma(A, B); // C = 8 (somma di numeri)

var A = "Salve ";
var B = "Mondo";
var C = somma(A, B); // C = "Salve Mondo" (concatenazione di stringhe)
```

Funzioni senza Nome (Funzioni Anonime)

Le funzioni possono essere definite senza nome e assegnate a variabili:

```
var fsomma = function(x, y) {
    var r = x + y;
    return r;
}
```

Questo è possibile perché in JavaScript le funzioni sono considerate oggetti. La chiamata di una funzione assegnata a una variabile avviene normalmente:

```
var C = fsomma(A, B);
```

2.4 - Oggetti in JavaScript

In JavaScript, un oggetto è un dato complesso che può avere campi (proprietà) con nomi diversi e metodi. Si posiziona concettualmente a metà tra le strutture del C e gli oggetti di Java. Il modello degli oggetti in JavaScript ha caratteristiche particolari che possono risultare inaspettate per chi inizia a usare il linguaggio.

Esistono tre modalità principali per definire oggetti in JavaScript:

Modalità 1: Estensione di Object

```
pers = new Object();
pers.name = "Giuseppe";
pers.height = "1.80m";
```

Si parte dall'oggetto base `Object` (vuoto o quasi) e si aggiungono campi dopo la creazione. I campi funzionano come variabili contenute all'interno dell'oggetto e **possono anche essere funzioni** (metodi):

```
pers.run = function() {
    this.state = "running";
    this.speed = "4m/s";
}
```

La parola chiave `this` accede all'oggetto che contiene il metodo. La chiamata del metodo avviene con la notazione puntata:

```
pers.run(); // Dopo questa chiamata, l'oggetto pers avrà due campi in più: state e speed
```

Modalità 2: Quasi una Classe

Questa modalità permette di creare una struttura comune da replicare facilmente, simile a una classe nei linguaggi tipizzati. JavaScript procede per clonazione, utilizzando un "costruttore" di oggetti:

```
function Person(pname, pheight) {
    this.name = pname;
    this.height = pheight;
    this.run = function() {
        this.state = "running";
        this.speed = "4m/s";
    }
}
```

La clonazione avviene tramite l'operatore `new` :

```
var pers = new Person("Giuseppe", "1.80m");
pers.run();
var pers2 = new Person("Paola", "1.60m");
pers2.run();
```

La chiamata alla funzione run aggiunge alle persone anche gli attributi `state` e `speed` , con i valori definiti durante la definizione della funzione Person (come se fosse una classe).

In questa modalità, la funzione clonata struttura tutti gli oggetti derivati nello stesso modo, funzionando quasi come una classe. Dopo la clonazione, la struttura dell'oggetto può essere ulteriormente estesa.

Modalità 3: Oggetti come Literals

Questa modalità permette di definire oggetti "al volo" all'interno di parentesi graffe:

```
pers = {
    name: "Giuseppe",
    height: "1.80m",
    run: function() {
        this.state = "running";
        this.speed = "4m/s";
    }
}
```

È pratica per oggetti semplici o unici, ma non agevola il riuso. Anche gli oggetti creati con questa modalità possono essere estesi successivamente aggiungendo nuovi campi. Sono particolarmente utili per predisporre oggetti di configurazione richiesti dai framework.

È importante notare che JavaScript non supporta l'information hiding: tutti i campi di un oggetto sono visibili e accessibili dall'esterno.

2.5 - Oggetti Standard

JavaScript fornisce due tipi principali di oggetti standard: Array e String.

Array

Gli array in JavaScript sono oggetti:

```
var a = new Array(10); // Crea un array con 10 elementi iniziali
```

Gli elementi dell'array non sono tipizzati e possono contenere valori di tipo diverso:

```
a[0] = "Pippo";
a[1] = 2;
```

Proprietà principali:

- `length` : numero di elementi nell'array

Metodi principali:

- `shift()` : restituisce e rimuove il primo valore
- `unshift(v)` : inserisce il valore v all'inizio
- `push(v)` : aggiunge v alla fine dell'array
- `pop()` : restituisce e rimuove l'ultimo valore
- `indexOf(v)` : restituisce la posizione di v (-1 se non presente)

Stringhe

Le stringhe in JavaScript sono oggetti di tipo `String` . Le costanti stringa definite con `"testo"` o `'testo'` vengono automaticamente convertite nell'oggetto corrispondente.

Proprietà principali:

- `length` : numero di caratteri nella stringa

Metodi principali:

- Metodi per formattazione HTML: `anchor()`, `big()`, `blink()`, `bold()`, `fixed()`, `fontcolor()`, `fontsize()`, `italics()`, `link()`, `small()`, `strike()`, `sub()`, `sup()`
- Metodi per manipolazione del testo: `toLowerCase()`, `toUpperCase()`, `charAt(x)` (restituisce il carattere in posizione x)

Esempio di utilizzo:

```
var message = "Welcome to our site!";
var format = message.toUpperCase();
var size = 1;
for (i = 0; i < format.length; i++) {
    document.write(format.charAt(i).fontsize(size).bold());
    if (size < 7) size++; else size = 1;
}
```

2.6 - Dichiarazione di Variabili: const, var, let

JavaScript offre tre modi per dichiarare variabili:

- `const`: dichiara una variabile il cui valore non può essere modificato (costante)

  ```
  const d = 1;
  ```

- `var`: contestualizza la variabile nella funzione o nell'oggetto di contesto

  ```
  var d = 1;
  d = 2; // Valore modificabile
  ```

- `let`: contestualizza la variabile nel blocco di codice che contiene la definizione

  ```
  var d = 1;
  {
      let d = 2; // Questa è una variabile diversa, visibile solo all'interno delle graffe
  }
  // Qui d vale ancora 1
  ```

La scelta tra queste tre modalità di dichiarazione influisce sullo scope (visibilità) della variabile e sulla possibilità di modificarne il valore.