LEZIONE 1
JAVASCRIPT: PROGRAMMAZIONE LATO CLIENT

1.0 JAVASCRIPT.

1.1. I LINGUAGGI LATO CLIENT

Le pagine web realizzate fino ad ora, con le conoscenze di html e css acquisite, sono essenzialmente statiche. Il testo, le immagini, i link e gli elementi grafici non permettono un'iterazione con il mondo esterno.

L'utente può leggere, consultare i vari contenuti, spostarsi da una pagina all'altra e al massimo compilare un modulo per inviare dati ad un programma lato server.

Quello che un linguaggio di programmazione lato client, come Javascript, permette di fare è di aggiungere **interattività alle pagine web**. Inoltre può far si che la pagina html interagisca alle diverse sollecitazioni dell'utente. Immaginiamo ad esempio la possibilità di modificare un'immagine al solo clic di un pulsante, oppure visualizzare un messaggio in relazione ad un input in un campo di testo.

Questi esempi, come tanti altri, mostrano le potenzialità che offre un linguaggio lato client che operi su una pagina html inizialmente statica.

Oltre Javascript, esistono vari linguaggi progettati a questo scopo:

- CGI;
- VBScript;
- Java;
- Flash.

Ognuno di questi ha alcuni difetti al cospetto di Javascript. Innanzitutto molti di questi richiedono conoscenze di programmazione di livello superiore rispetto a Javascript, vedi Java. Alcuni influiscono molto sulle prestazioni del server e della rete, vedi CGI; altri invece rischiano di appesantire le pagine html, vedi Flash. Infine altri sono linguaggi proprietari e quindi non integrati in tutti i browser.

1.1.1. LA NASCITA DI JAVASCRIPT.

Javascript fu originariamente sviluppato da **Brendan Eich** della **Netscape Communications** con il nome di **Mocha** e successivamente di **LiveScript**, in seguito è stato rinominato ed è stato formalizzato con una sintassi più vicina a quella del linguaggio Java di Sun Microsystems.

JavaScript è stato standardizzato per la prima volta tra il 1997 e il 1999 dalla ECMA con il nome **ECMAScript**.

L'ultimo standard, del dicembre 1999, è ECMA-262 Edition 3, e corrisponde a JavaScript 1.5.

1.1.2. ASPETTI STRUTTURALI DEL LINGUAGGIO.

Javascript è un **linguaggio interpretato**, il codice non viene prima compilato (come accade con i linguaggi C, Java, etc), bensì all'interno dei browser è presente un **interprete** che ha il compito di leggere il codice e '*eseguirlo al volo*'.

Per questo motivo Javascript è considerato il linguaggio di **scripting** per eccellenza anche in relazione alla semplicità di apprendimento che lo contraddistingue.

Come detto uno degli aspetti significativi di Javascript è che il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che, anche con la presenza di script particolarmente complessi, il server non viene sovraccaricato a causa delle richieste dei clients.

Di contro, nel caso di script che presentino un sorgente particolarmente grande, il tempo per lo scaricamento può diventare abbastanza lungo.

Essendo locato sul client però, **Javascript non può accedere a database** e informazioni che risiedono

sul Server (a meno che non si utilizzi una particolare tecnologia che prende il nome di **AJAX**), questo fa sì che il '*lavoro'* del linguaggio sia orientato esclusivamente sugli elementi e i dati che compongono la pagina web.

Infine, uno degli svantaggi di Javascript è la possibilità che chiunque, visualizzando il sorgente della pagina web, possa leggere il codice sorgente. Questo svantaggio, in termini termini di sicurezza, è divenuto però col passare del tempo un'arma in più allo sviluppo culturale del linguaggio; infatti essendo il web pieno di codici javascript accessibili anche i meno esperti hanno potuto imparare in maniera veloce le basi del linguaggio.

1.1.3. GLI STRUMENTI DI LAVORO.

Per scrivere codice Javascript non sono necessari programmi specifici, compilatori o computer particolari.
Quello di cui si ha bisogno è:

- un programma di scrittura, tipo il **Blocco Note** di Windows;
- un browser web di ultima generazione, tipo **Internet Explorer** o Firefox.

Una volta scritto un codice, che prenderà il nome di **script**, questo va testato semplicemente all'interno del browser. A supporto di ciò la maggior parte dei browser moderni fornisce appositi messaggi di errore qualora il codice non sia funzionante, aiutando di conseguenza lo sviluppatore a correggere le imperfezioni.

2.0. LE BASI DEL LINGUAGGIO.

2.0.1. IL TAG SCRIPT.

Come per il CSS anche il Javascript ha bisogno di alcuni tag contenitori. Il tag in questione è **<script>** e la sintassi di base è la seguente:

<script type='text/javascript'>
...
</script>

Nel momento in cui l'interprete del browser incontra questo tag, ne legge il type per identificare il linguaggio utilizzato (nel nostro caso javascript) ed effettua l'esecuzione del codice.
A differenza del CSS, le cui regole vanno specificate nell'head della pagina, il codice Javascript può essere inserito in qualsiasi punto della pagina interno al body. E' buona norma però inserire il tag <script> all'interno dell'head così da non avere porzioni di codice sparso nelle pagine.
Un aspetto fondamentale da tenere sempre in considerazione è che tutto ciò che è racchiuso all'interno di questi tag è visto dal browser come codice Javascript, quindi è assolutamente essenziale non inserire codice html, css o comunque elementi estranei al linguaggio che provocherebbero, di conseguenza, errori in fase di visualizzazione della pagina.

2.0.2. JAVASCRIPT INTERNO ED ESTERNO.

Il codice Javascript può essere, come detto, scritto all'interno del file html (come mostrato nell'immagine successiva):

```
<html>
<head>
        <title>Codice incorporato</title>
        <script type='text/javascript'>
        ...
        </script>
</head>
<body>
...
</body>
</html>
```

In alternativa è possibile realizzare dei file esterni con estensione **.js** che conterranno esclusivamente codice Javascript.
In questo caso, proprio come si è fatto con i fogli di stile, il file .js va richiamato in maniera opportuna all'interno dell'head della pagina html.
Supponendo di aver realizzato un file chiamato script.js:

```
<html>
<head>
        <title>Codice richiamato</title>
        <script type="text/javascript" src="script.js"></script>
</head>
<body>
...
</body>
</html>
```

Infine il file Javascript esterno non dovrà contenere il tag <script> di apertura e il tag </script> di chiusura.
E' consigliabile utilizzare sempre file esterni, soprattutto quando il codice è soggetto a frequenti richiami da parte di pagine diverse. In questo caso avere il codice esterno, permette modularità e semplicità in una eventuale fase di modifica.
Un ultimo fattore da considerare, nel momento in cui si dispone il codice Javascript in una pagina html, è che questo viene interpretato dall'alto verso il basso. Quindi se ad esempio si inserisce del codice subito dopo il tag <body> e un'altra porzione prima della chiusura dello stesso, il browser eseguirà il codice man mano che lo incontra, sovrascrivendo e riutilizzando eventuali variabili definite prima.

2.0.3. GESTIRE I BROWSER NON COMPATIBILI.

Quando Javascript a metà degli anni '90 intraprese il suo sviluppo, molti dei browser dell'epoca non contemplavano al loro interno un interprete per il linguaggio e di conseguenza era necessario gestire questa eccezione.
Oggi, nonostante tutti i browser in circolazione sappiano leggere il Javascript è possibile che un utente disabiliti, per scelta, l'esecuzione del codice.
Il tag **<noscript>** ci permette di specificare un'azione differente da intraprendere qualora le condizioni sopraelencate diventino vere:

```
<noscript>
        <div>
        Attenzione: per visualizzare correttamente questa pagina è necessario avere
        attivo Javascript.
        </div>
</noscript>
```

Nell'esempio, il browser visualizzerà un div con un messaggio qualora Javascript risulti non attivo. In alternativa si sarebbe potuto intraprendere un reindirizzamento ad altra pagina oppure presentare un layout differente.

2.0.4. I COMMENTI.

I commenti sono le parti del programma che non vengono lette dall'interprete e che, quindi, servono a spiegare e a chiarire.
Questi sono **racchiusi tra barre e asterischi** come nell'esempio sotto riportato:

```
/* Commento per codice Javascript*/
```

Il commento può essere posto su più righe o su una riga singola, mentre non è accettato dall'interprete il commento annidato.
Un altro tipo di commento è la **doppia barra**, ma è valida solo per commenti posti su una singola riga, anche se non la occupano per intero:

```
// Commento per codice Javascript
```

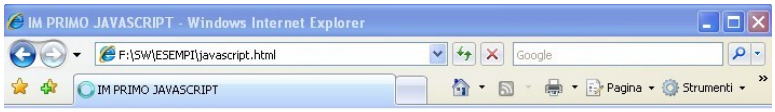
I commenti Javascript non possono essere inseriti al di fuori dei tag che individuano lo script, altrimenti HTML li considererà come parte del testo, e viceversa non si possono utilizzare i tag di commenti HTML all'interno dello script.

2.0.5. IL PRIMO SCRIPT.

Siamo pronti per entrare nel vivo della programmazione Javascript. Il primo esempio che analizzeremo sarà il classico 'hello world' che ci permetterà inoltre di introdurre il concetto di variabile e alcune regole di sintassi.

```
<script type='text/javascript'>
        // Inizializziamo una variabile con una stringa di saluto
        var stringa = 'Benvenuti in Javascript!';
        // Visualizziamo a video la stringa
        alert(stringa);
</script>
```

Per testare il codice, questo va posto all'interno dell'head di una pagina html con body vuoto. Apriamo quindi il browser e richiamiamo la pagina:



All'apertura della pagina il browser visualizzerà una finestra al centro del monitor con il testo definito all'interno del codice.
Il codice Javascript si compone essenzialmente di due istruzioni:

- definizione di una variabile;
- stampa a video della variabile tramite **alert.**

La parola chiave **var** serve a introdurre una variabile per la prima volta, nel momento in cui questa viene definita. Ulteriori accessi, modifiche o stampe della variabile non richiederanno l'uso di questa parola chiave.

Dopo il var troviamo il nome della variabile, nel nostro caso stringa, quindi il segno di uguale (**=**) che sta ad indicare un'operazione di assegnazione.
Possiamo così tradurre la prima riga di codice:
Crea una variabile chiamata *stringa* e impostala a *Benvenuti in Javascript*
La riga di codice si conclude con un punto e vergola (**;**), così come qualsiasi altra istruzione in Javascript ed indica appunto la conclusione dell'istruzione.
A questo punto la variabile stringa è pronta per la stampa.
A tale scopo si è utilizzata una funzione di base del linguaggio, l'**alert box** (finestra di avviso).
La funzione, una volta invocata, visualizza a video una **piccola finestra** contenente il testo passato come parametro alla funzione.
E' necessaria a questo punto una breve parentesi per chiarire in maniera semplice il concetto di funzione:
"una funzione è una sequenza finita di istruzioni atte ad eseguire un'operazione o a risolvere un determinato problema".
Una funzione, una volta definita, può essere **chiamata** o **invocata** all'interno del programma che la definisce e può richiedere eventualmente parametri in ingresso.
L'**alert**, così come altre, sono **funzioni native** del linguaggio Javascript.
Tornando infine all'analisi del codice: viene invocata la funzione alert, passando come parametro la variabile stringa definita in precedenza, e chiuso il tag <script> ad indicare la conclusione dello script.

2.0.6. VARIABILI E TIPI DI DATO.

Nel paragrafo precedente abbiamo analizzato la sintassi necessaria ad istanziare una variabile:
var nome_variabile = valore;
La scelta del nome della variabile non ha particolari vincoli, è consigliabile utilizzare sempre nomi che ne indichino il contenuto, il tipo o la funzione che ricoprono. E' importante però sapere che il nome delle variabile è **sensibile alle maiuscole**, quindi settare la variabile **my_variabile** non è uguale ad utilizzare poi la variabile **My_Variabile**.
Inoltre non sono consentiti spazi.
Nel codice visto in precedenza abbiamo istanziato una variabile di **tipo string,** una sequenza di caratteri alfanumerici racchiusi tra apici singoli o doppi**:**

```
var stringa = "Questa è una stringa!";

var altra_stringa = 'Questa è una seconda stringa';
```

Una variabile può assumere anche **valore numerico** (intero o decimale), in questo caso scriveremo:

```
var numero = 10;

Var altro_numero = 14.5;
```

Qualora invece dovessimo scrivere un valore numerico tra apici:

```
var non numero = "10";
```

questo verrà visto dall'interprete Javascript come una stringa.
Infine una variabile può assumere **valore booleano**, cioè true o false. In questo caso la sintassi sarà:

```
var booleano = true;

var altro_booleano = false;
```

2.0.7. OPERAZIONI CON LE VARIABILI.

Le variabili possono combinarsi tra loro e dare vita a nuove variabili. Le operazioni ammissibili sono le classiche operazioni aritmetiche (somma, sottrazione, moltiplicazione e divisione). Con le stringhe invece è possibile effettuare concatenazioni, ricerche, tagli e operazioni di formattazione.
Analizziamo la seguente porzione di codice Javascript:

```
var num_1 = 10;
var num_2 = 4;
var somma = num_1 + num_2;
```

Istanziate due variabili, **num_1** e **num_2**, queste vengono sommate tra loro e il risultato posizionato in una variabile denominata **somma**.
Diversamente nel seguente codice:

```
var str_1 = "Ciao ";
var str_2 = "studente!";
var stringa = str_1 + str_2;
```

essendo le due variabili, **str_1** e **str_2**, di tipo string, l'operazione di somma effettua una concatenazione che come risultato ci darà la variabile **stringa="Ciao studente!"**.

2.0.8. LEGGERE E SCRIVERE VARIABILI A VIDEO

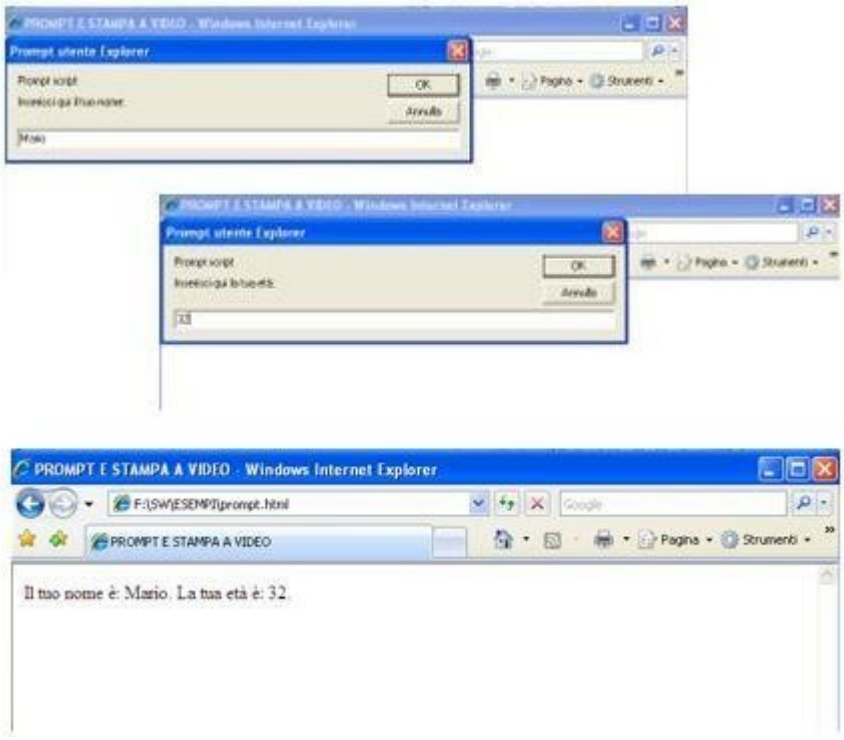
Dopo aver visto la funzione **alert,** un'altra utile funzione è denominata **prompt**.
Tale funzione permette all'utente di immettere dati o informazioni all'interno di un codice Javascript.
La sintassi è la seguente:
var nome_variabile = prompt('Messaggio utente' , 'valore di default');
Analizziamo il seguente codice:

```
<script type='text/javascript'>

        // Acquisiamo due variabili dall'utente

        var nome = prompt('Inserisci qui il tuo nome:','');

        var eta = prompt('Inserisci qui la tua età:','');

        // Stampiamo a video una stringa concatenata con le variabili immesse

        window.document.write('Il tuo nome è: ' + nome + '. ');

        window.document.write('La tua età è: ' + eta + '.');

</script>
```

Il codice inizia definendo due variabili, **nome** e **eta**, avvalorate tramite richiesta all'utente con la

funzione prompt.

L'istruzione **window.document.write** permette di stampare all'interno della pagina html una stringa di caratteri.
Evitando di addentrarci nella sintassi di questa dichiarazione, è importante tenerla in considerazione ogni qualvolta si desideri scrivere su una pagina html inizialmente **vuota**.
E' possibile difatti inserire come argomento anche codice html, per formattare i caratteri e strutturare la pagina:
window.document.write('<h1>Titolo della pagina</h1>');
In definitiva tramite questa istruzione è possibile realizzare **a volo una pagina html**, semplicemente combinando insieme una sequenza di write.



2.1. CONDIZIONI E ITERAZIONI.

2.1.1. ISTRUZIONI CONDIZIONALI.

Ora che si è visto come scrivere su una pagina html (window.document.write), o in una finestra di allarme (alert), è necessario capire come dire a Javascript di scrivere cose diverse in relazione ad una determinata **condizione**. L'istruzione condizionale per eccellenza è l'IF (se), la cui sintassi è la seguente: **if(condizione){**

…

}else{

…

}

Se la condizione viene verificata e il risultato è **vero** allora viene eseguito il codice compreso nella prima coppia di parentesi graffe, in caso contrario verrà eseguito il blocco di istruzioni presente dopo l'**else** (altrimenti). Vediamo in pratica un esempio di condizione:

```
<script type='text/javascript'>
        // Definiamo tre variabili
        var a = 10;
        var b = 7;
        var max = 0;
        // Controlliamo se a è maggiore di b
        if(a>b){
                // Se è vero allora a è massimo
                max = a;
        }else{
                // Altrimenti b è il massimo
                max = b;
        }
</script>
```

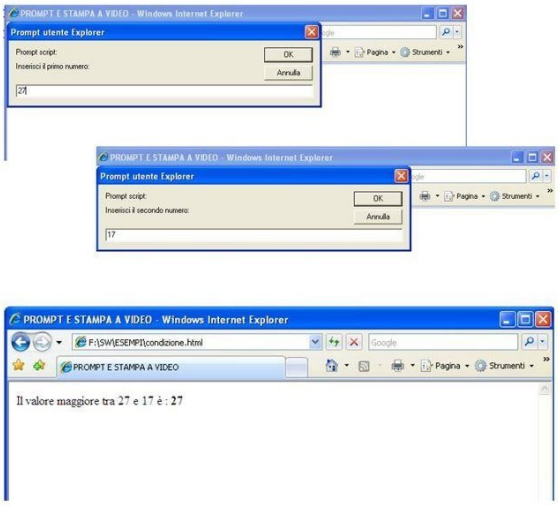
In questo caso la condizione risulterà sempre vera, fin quando non verranno modificati i valori di a e b. All'interno dell'if possiamo inserire diverse operazioni di confronto, tra cui:

- uguaglianza: **if(a==b){…**
- minore: **if(a<b){…**
- maggiore: **if(a>b){…**
- maggiore-uguale: **if(a>=b){ …**
- minore-uguale: **if(a<=b){ …**
- diverso: **if(a!=b){ …**

Analizziamo ora il seguente codice Javascript

```
<script type='text/javascript'>
        // Acquisiamo due numeri dall'utente
        var a = prompt('Inserisci il primo numero:','');
        var b = prompt('Inserisci il secondo numero:','');
        // Converto i valori in numerici
var a = parseInt(a);
var b = parseInt(b);
        // Verifichiamo quale sia il maggiore tra a e b
        // Stampiamo a video il maggiore con formattazione html
        if(a>b){
                window.document.write('<p>Il valore maggiore tra '+a+' e '+b+' è
        : ' + '<strong>' + a + '</strong></p>');
        }else{
                window.document.write('<p>Il valore maggiore tra '+a+' e '+b+' è
        : ' + '<strong>' + b + '</strong></p>');
}
</script>
```

In questo esempio, acquisiti i due valori tramite prompt, questi vengono innanzitutto convertiti in valori numerici grazie alla funzione **parseInt**.
Successivamente viene verificato quale dei due numeri sia maggiore e stampato a video un paragrafo (p) con una stringa contenente il risultato.
La conversione è necessaria perché i valori da prompt sono di default considerate come stringa. E' possibile **evitare la conversione con il parseInt**, in questo caso sono lasciate a voi le considerazione sul comportamento del codice.



2.1.2. COSTRUTTI ITERATIVI: FOR.

Il più classico tra i costrutti per le iterazioni è il **for** che esegue ciclicamente le istruzioni al suo interno finché non viene raggiunto il limite indicato da una condizione.
La condizione di fine ciclo viene verificata tipicamente su una variabile utilizzata come contatore.
Questa variabile è la protagonista della definizione del ciclo in cui si compiono tre operazioni:

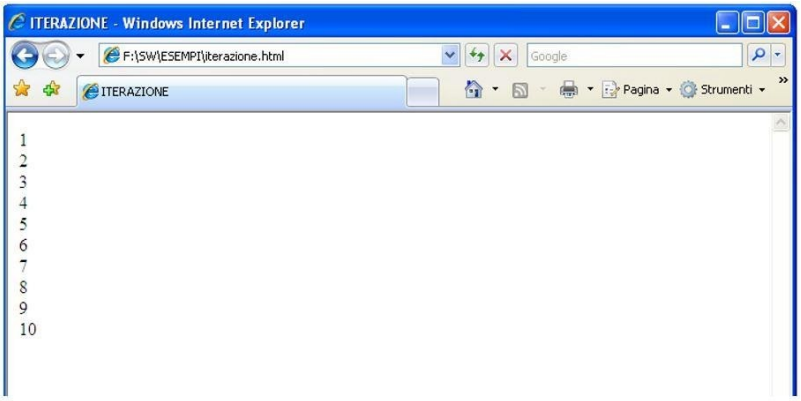
- l'inizializzazione della variabile, in cui viene definito il valore di partenza;
- la verifica della condizione: se vera fa rimanere nel ciclo, se falsa lo interrompe;
- l'incremento (o comunque la modifica) della variabile.

Vediamo la sintassi:

for(inizializzazione, condizione,incremento){
...
}

Per fare un esempio, scriviamo a video i numeri da 1 a 10 nel seguente modo:

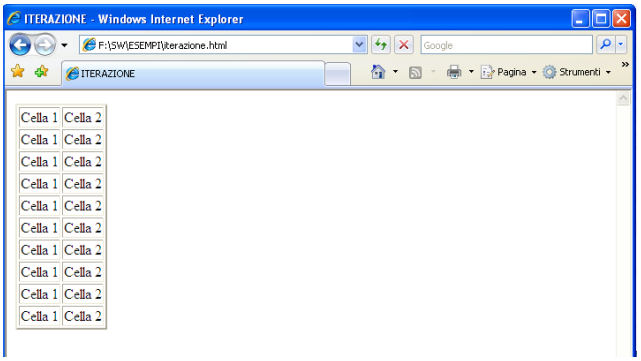
```
<script type='text/javascript'>
        for(var i = 1; i<11; i++){
                window.document.write(i + '<br/>');
        }
</script>
```



La variabile **i** viene inizializzata ad 1 e quindi **incrementata ad ogni ciclo**, fino a quando la condizione **i<11** smette di essere vera.
Quando **i** assume valore 11, il ciclo si interrompe.
Un ciclo for è utile quando è necessario aggiungere una serie di elementi ad una pagina html. Un classico esempio potrebbe essere un ciclo che ad ogni iterazione aggiunge una riga ad una tabella:

```
<script type='text/javascript'>
        window.document.write("<table border='2'>");
        for(var i = 1; i<11; i++){
                window.document.write("<tr><td>Cella 1</td><td>Cella
        2</td></tr>");
        }
        window.document.write("</table>");
</script>
```

Nonostante l'esempio sia a chiarire l'uso di un ciclo, è evidente notare le potenzialità di Javascript nella realizzazione **dinamica di codice html**. Nelle lezioni successive verrà trattato con maggior cura il cosiddetto **DHTML** (Dynamic HTML).



2.1.3. COSTRUTTI ITERATIVI: WHILE.

Un altro controllo di flusso si può avere con l'istruzione **while** (finché), la cui sintassi è la seguente:
while(condizione){
…

}
Riprendendo l'esempio iniziale del for, otteniamo:

```
<script type='text/javascript'>
        var i = 1;
        while(i<11){
                window.document.write(i + '<br/>');
                i++;
        }
</script>
```

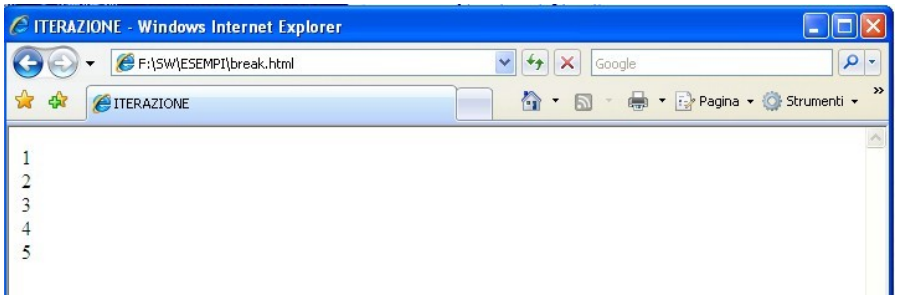
All'interno del while abbiamo definito il valore booleano true e incrementato un contatore che non è soggetto a verifica. Quindi il ciclo stamperà i valori da **1 a infinito**.
Logicamente questo esempio è poco utile ed inoltre provarlo potrebbe causare un **blocco del browser**. Nonostante ciò è utile conoscere questa possibilità in vista di codici più complessi.

2.1.4. BREAK E CONTINUE

I comandi **break** e **continue** servono ad ottimizzare i cicli for e while oltre che all'operatore condizionale if.
Il comando **break**, infatti, interrompe un blocco di istruzioni saltando alla prima istruzione seguente il blocco contenente il **break**. L'utilizzo appropriato è quello di evitare la formazione di loop senza uscita:

```
<script type='text/javascript'>
        var i = 1;
        while(i<11){
                window.document.write(i + '<br/>');
                if(i==5){
                        break;
                }
                i++;
        }
</script>
```

Nell'esempio, simile a quello visto in precedenza, il ciclo si interromperà quando **i** è uguale a **5**, nonostante la condizione **i<11** risulti ancora vera.



Il comando ***continue***, invece, indica di continuare il blocco ma interrompendo l'iterazione in quel punto e riprendendo dall'inizio del blocco.

```
<script type='text/javascript'>
        var i = 1;
        while(i<11){
                if(i==5){
                        i++;
                        continue;
                }
                 window.document.write(i + '<br/>');
                i++;
        }
</script>
```

In questo caso il valore **5** non verrà stampato a video grazie all'istruzione continue che farà riprendere il ciclo tralasciando l'istruzione while.