# JavaScript

Introduzione



## Caratteristiche

- È un linguaggio interpretato che consente di codificare programmi all'interno di pagine html
- È stato sviluppato a partire dal 1995 dalla Netscape per Navigator, ma è supportato anche dagli altri browser
- Il codice JS può essere eseguito sia lato Server sia lato Client
  - sul Client dal Browser, e di questo tratteremo nel corso
  - sul Server dall' ambiente di sviluppo/esecuzione Node.js
- Adotta il paradigma di programmazione ad oggetti

# Funzionalità (lato Client)

Con JS possiamo:

- Controllare le funzioni del browser (apertura di nuove pagine, messaggi di alert, …)
- Modificare aspetto e contenuto delle pagine Web
- Registrare ed usare informazioni sugli utenti (cookies)
- Creare e manipolare elementi grafici nelle pagine
- Manipolare immagini

Con JS non possiamo:

- Accedere (in lettura e/o scrittura) ai file sulla macchina client
- Utilizzare tutti i protocolli Internet, ma solo un sottoinsieme limitato (http, ws, …)
- Operare in multithreading, cioè attivare in modo concorrente diversi processi di elaborazione





# Il Codice JavaScript (1)

- Per includere in codice JS in un documento HTML, si può:
  - Inserirlo direttamente all'interno del tag `<script> … </script>`.
  - Utilizzare lo stesso tag per includere nel documento un file esterno che contiene il codice
- Il codice JS può essere collocato in qualsiasi sezione del documento HTML (`<head>` o `<body>`)
- È possibile inserire anche contenuti da visualizzare solo nei browser che hanno JS disattivato, inserendoli nel tag `<noscript> … </noscript>`

# <script>

## <script>Codice script</script>

**Descrizione:** inserisce nel documento codice codificato in uno script-language

**Tipo:** tag contenitore

**Attributi:** type, src, …

- `type` definisce il linguaggio di script utilizzato. È opzionale: se non specificato il suo valore di default è `text/javascript`
- `src` è l'URL del file che contiene lo script






# JavaScript

Variabili e Tipi di Dato

# Le Variabili

- Identificatore (case sensitive)

  `<varid>::=$|_|<letter>{<letter>|<digit>|$|_}`

- La dichiarazione può essere fatta contestualmente all'assegnazione
  - Dichiarazione con scope globale: `var`
    - `var indice=10, Testo="buongiorno"`
  - Dichiarazione con scope locale: `let`
    - `let indice=10`
  - Dichiarazione di costante: `const`
    - `const indice`
- Una variabile può essere dichiarata subito prima di essere usata





# Binding Variabile-Tipo

- Il Binding variabile-tipo è dinamico

```
let miaVariabile = 10;
    miaVariabile = "Testo";
    miaVariabile = 3.14;
    miaVariabile = true;
```

quindi, in sede di dichiarazione, alla variabile non è associato un tipo di dato

# Tipi di Dato

| Tipo | Operatori |
|---|---|
| Number (Int, Real) | +, -, *, /, %, ++, --, -(opposto), =, +=, -=, *=, /=, %=, ==, !=, >, <, >=, <= |
| BigInt | Gli stessi di Number |
| Boolean | &&, \|\|, ! |
| String | +, += |
| Array | Dipendenti dal tipo degli elementi |
| Symbol | Symbol() |
| Object | new |



# Conversione di Tipo

- Conversione stringa --> numero

```
Y = "3";
Y *= "7";              // Y = 21
```

- Conversione numero --> stringa

```
Y = 3;
Y += "tavoli";  // Y = "3tavoli"
```

# Gli Array

- Sono oggetti, e vengono quindi creati con la sintassi

```
let <nome_variabile> = new Array([<num_elem>])
```

```
let   vettore = new Array(4);
      vettore[0] = "bianco";
      vettore[1] = "nero";
      vettore[2] = 3;
      vettore[3] = 3.14;
```

  - Sintassi alternativa (preferibile)

```
let vettore = ["bianco", "nero", 3, 3.14]
```





# Scope di una Variabile

- In JS le variabili possono essere:
  - Locali, definite all'interno di una funzione o di un blocco di programma ed utilizzabili solo in essi (e nelle eventuali funzioni/blocchi definiti al loro interno)
    - definite tramite il costrutto `let`
  - Globali, definite all'esterno delle funzioni e visibili in ogni parte del programma
    - definite tramite il costrutto `var`
- Lo scope, in JS, è quindi STATICO

# JavaScript

Funzioni ed Oggetti



# Le Funzioni

- Dichiarazione:

```
function <nome_funzione> ([<parametri>]) {<istruzioni>}
```

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

- Attivazione:

```
let c = somma(3,7)
```

- Per chiarezza del codice, preferibilmente:
  - La dichiarazione va inserita nella <head> del documento
  - L'attivazione nel <body>

# Le Funzioni

## Dichiarazione alternativa:

- `let <nome_funzione> = function ([<parametri>]) {<istruzioni>}`

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

- `let <nome_funzione> = ([<parametri>]) => {<istruzioni>}`

```
let somma = (a,b) => {
                let risultato = a + b;
                return risultato;
            }
```





# Classi

- Una Classe è un prototipo che definisce le proprietà ed i metodi degli oggetti che da essa discendono
- In JS un oggetto può essere creato attraverso l'uso della funzione costruttrice (che, di fatto, definisce la classe da cui l'oggetto discende)

```
function Libro (isbn, titolo, autore) {
        this.isbn_libro = isbn;
        this.titolo_libro = titolo;
        this.autore_libro = autore; }
mio_libro = new libro ("111-2222", "Div_comm", "D_Alighieri")
```

classe

oggetto

# Estensione di Oggetti e Classi

- Estensione delle proprietà degli oggetti

```
mio_libro = new Libro ("111-2222", "Div_comm", "D_Alighieri");
mio_libro.prezzo = "12 euro";
```

- Estensione delle proprietà delle classi

```
libro.prototype.prezzo = "12 euro";
```

```
function libro_con_prezzo () {
   this.prezzo = "12 euro";
}
libro_con_prezzo.prototype = new libro();
```



# Metodi di una Classe

- I metodi sono funzioni definite nell'ambito di una classe che si applicano a tutti gli oggetti che da essa discendono
- Una funzione JS diventa metodo di una classe inserendo all'interno della funzione costruttrice di quest'ultima l'istruzione:

```
this.<nome_metodo> = <nome_funzione>
```



# Oggetti 'Singoli'

- In JS è possibile definire oggetti che non discendono da alcuna classe

```
var mio_libro = {
        isbn_libro:"111-2222",
        titolo_libro:"Div_comm",
        autore_libro:"D_Alighieri",
        stampa_prop: function () {
             document.write(this.isbn_libro + ", " +
             this.titolo_libro + ", " + this.autore_libro);
        }
}
```





# JavaScript

Strutture di Controllo Condizionale ed Iterativo

# IF - IF … ELSE

- Sintassi

```
if (<espressione_condizionale>) {
  <istruzioni>;
}
```

```
if (<espressione_condizionale>) {
  <istruzioni>;
}
else {
  <istruzioni>;
}
```





# SWITCH

- Sintassi

```
switch (<espressione>) {
  case <etichetta>: <istruzioni>; break;
  case <etichetta>: <istruzioni>; break;
  …
  default: <istruzioni>;
}
```

# WHILE – DO … WHILE

- Sintassi

```
while (<espressione_condizionale>)   {
   <istruzioni>;
}
```

```
do {
   <istruzioni>;
} while (<espressione_condizionale>);
```

- Le <istruzioni> vengono ripetute fintanto che <espressione_condizionale> è vera





# FOR – FOR … IN

- Sintassi

```
for (<inizializzazione>;<condizione>;<step>) {
   <istruzioni>;
}
```

```
for (<variabile> in <oggetto>)   {
   <istruzioni>;
}
```

- for … in scandisce le componenti di un oggetto, (proprietà e metodi) associandone i nomi ai valori di <variabile> nel corso dell' iterazione



# WITH

- Sintassi

```
with (<oggetto>)   {
   <istruzioni>;
}
```

- Esempio

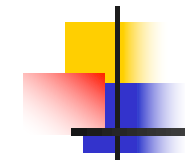
```
mio_libro = new libro();
with (mio_libro) {
  isbn_libro = "11-222-33";
  titolo_libro = "La Divina Commedia";
  autore_libro = "Dante Alighieri”;
}
```





# BREAK, CONTINUE

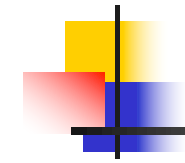
- `break` consente di interrompere un ciclo iterativo prima che la condizione di fine ciclo sia verificata
- `continue` permette di passare direttamente alla successiva iterazione di un ciclo



# JavaScript

## Gestione degli Eventi





# Eventi

- Gli eventi sono condizioni rilevate dal browser e relative ad azioni che:
  - l'utente realizza in fase di visualizzazione di un documento
  - Il browser stesso realizza sul documento
- La maggior parte dei Tag HTML consente di attivare funzioni JS al verificarsi di tali eventi riferiti alle parti di documento da esse definite

# Eventi Principali

| | | |
|---|---|---|
| onBlur | onChange | onClick |
| onDblClick | onFocus | onKeyDown |
| onMouseDown | onMouseMove | onLoad |
| onMouseOver | onMouseUp | onMouseOut |
| onSelect | onSubmit | onReset |
| … | … | … |






# JavaScript

## Il Browser Object Model (BOM)

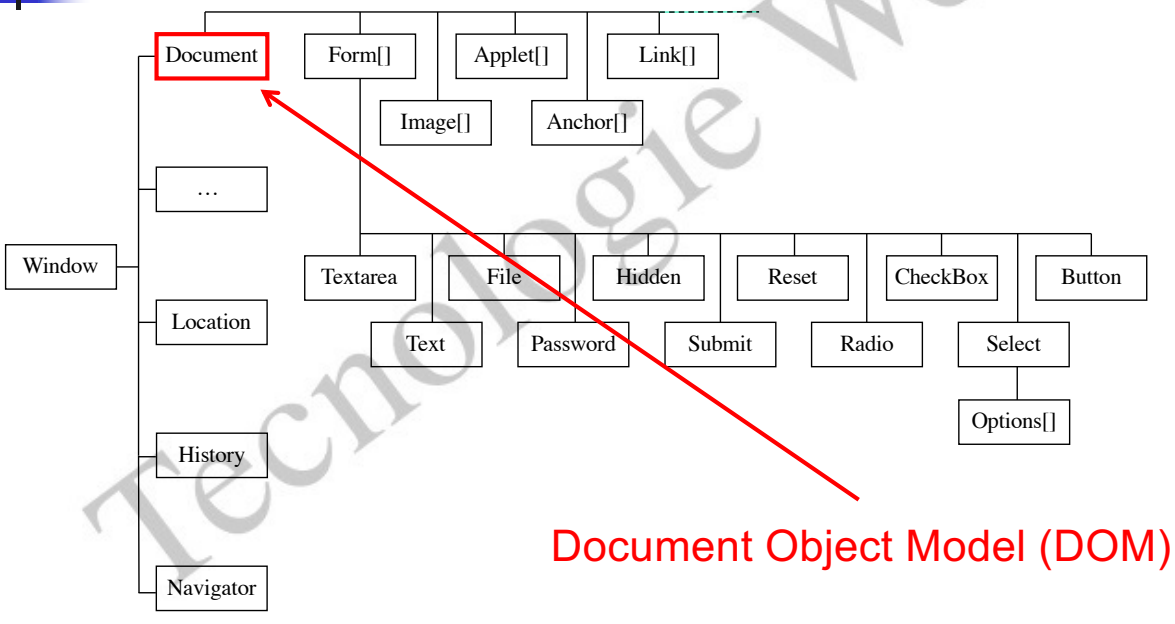
# Oggetti JS

- Il Browser, al termine del caricamento di un documento, associa una serie di oggetti JS ai contenuti della pagina
  - Document
  - Location
  - History
  - Navigator
  - …
- A questi oggetti si può accedere tramite script JS per manipolare il contenuto della pagina visualizzata



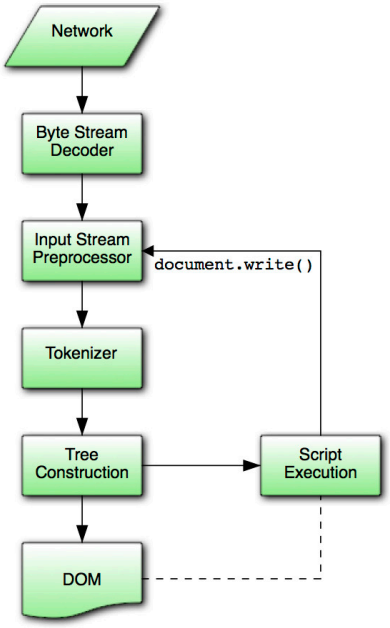


# Gerarchia Oggetti JS




Document Object Model (DOM)

# HTML Parsing - Creazione DOM




- Byte Stream Decoder
  Individuazione della codifica usata nel documento per i caratteri (UTF-8, ISO-8859-5, …)
- Input Stream Preprocessor
  Conversione dei codici in caratteri UNICODE
- Tokenizer
  Accorpamento dei caratteri in termini (parole)
- Tree Construction
  Creazione della gerarchia (albero) del DOM



# WINDOW

| Proprietà | Metodi |
|---|---|
| name | alert() |
| parent | close() |
| self | confirm() |
| top | focus() |
| … | open() |
| | prompt() |
| | … |



JS6

# DOCUMENT

| Proprietà | Metodi |
|---|---|
| lastModified | write() |
| title | writeln() |
| URL | createElement() |
| forms | getElementById() |
| images | getElementsByName() |
| … | querySelector() |
| | … |

**JS7/8/9/10**





# FORM

| Proprietà | Metodi |
|---|---|
| action | reset() |
| method | submit() |
| name | |
| elements[] | |
| length | |
| … | |

**JS11**

# MATH

- È un oggetto predefinito JS al quale sono associate (come metodi) le funzioni matematiche di JS

| Proprietà | Metodi | | |
|---|---|---|---|
| E | abs() | exp() | random() |
| LN10 | acos() | floor() | round() |
| LN2 | asin() | log() | sin() |
| PI | atan() | max() | sqrt() |
| SQRT2 | ceil() | min() | tan() |
| … | cos() | pow() | … |