Informatica I

5 – JavaScript crash course

23-30 Aprile 2013

Corso di Laurea in Matematica e applicazioni

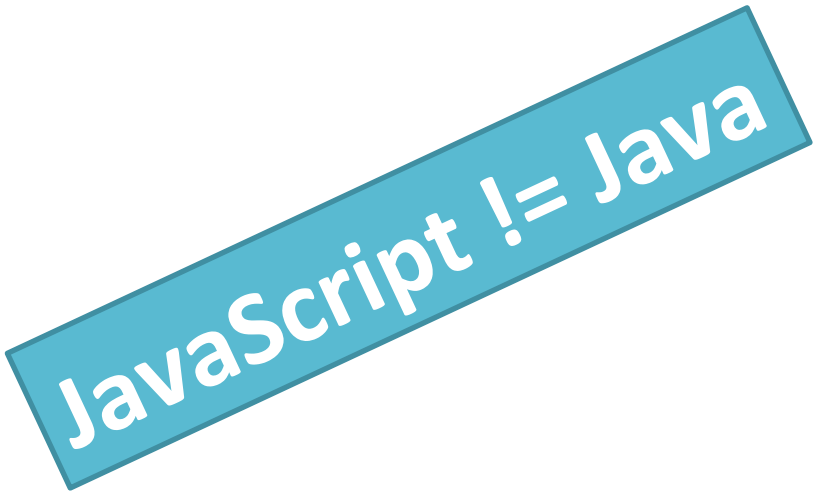
Università di Camerino

A.A. 2012/2013

Il linguaggio JavaScript

JavaScript (JS) è un linguaggio di programmazione con le seguenti caratteristiche:

- **Linguaggio di scripting**
- **Debolmente tipizzato**
- **Object-oriented**
- **Imperativo**
- **Funzionale**



Le slides sono ampiamente basate sulla guida a JS della W3C (World Wide Web Consortium), all'indirizzo http://www.w3schools.com/js/

Linguaggio di scripting

• A differenza dei programmi classici, gli script sono eseguiti in un ambiente software, quindi non vengono compilati o interpretati.

• **JS è il linguaggio di scripting per il web**, e fornisce funzionalità aggiuntive all'HTML.

• Non dev'essere compilato, è direttamente eseguibile da un **web-browser** (Chrome, Firefox, IE, Opera, ...) all'interno di una pagina HTML

Linguaggi weakly typed

JavaScript (JS) è un linguaggio **debolmente tipizzato:** pur essendo definiti, i tipi di dato

- non vengono dichiarati esplicitamente
- vengono convertiti implicitamente.

Weakly typed (JS)

```
var output;
if(input!=null)
  output=3; //output è
  Number
else
  output="invalid
  input";
  //output è String
```

Strongly typed (Java)

```
String output; // il
  tipo di output va
  dichiarato
if(input!=null)
  output="3";
else
  output="invalid
  input";
```

Linguaggi Object-Oriented (OO)

E' un paradigma di programmazione in cui gli elementi principali sono **oggetti**, ovvero strutture dati costituite da

- **proprietà** (gli attributi dell'oggetto) e da
- **metodi** (o **funzioni**, azioni per modificare le proprietà o interagire con altri oggetti).

- In JS i tipi di dato elementari (Number, Boolean, String) sono implementati come oggetti

```
var txt="Hello world!"; // txt è un oggetto String
document.write(txt.toUpperCase());
//chiamo il metodo toUpperCase() definito per gli oggetti
String e il risultato viene dato in input al metodo write
dell'oggetto document (la pagina web)
```

Paradigma imperativo

- *"Prima fai questo, poi fai quello"*

- E' un paradigma di programmazione basato su **statement** (comandi, passi) che modificano lo **stato** del programma

- Alcuni fattori che determinano lo stato del programma
  - Valori delle variabili
  - Prossima istruzione da eseguire
  - Blocco di statement corrente

Paradigma funzionale

JavaScript è basato su funzioni, trattate come ***"first-class citizens",*** ovvero possono essere:

- passate come argomento a altre funzioni
- restituite in output da una funzione
- assegnate a una variabile
- salvate in una struttura dati

```
function add(a, b) {
        return a + b;
}
add(5, 5);
```

```
var add = function(a, b) {
        return a + b;
};
add(5, 5);
```

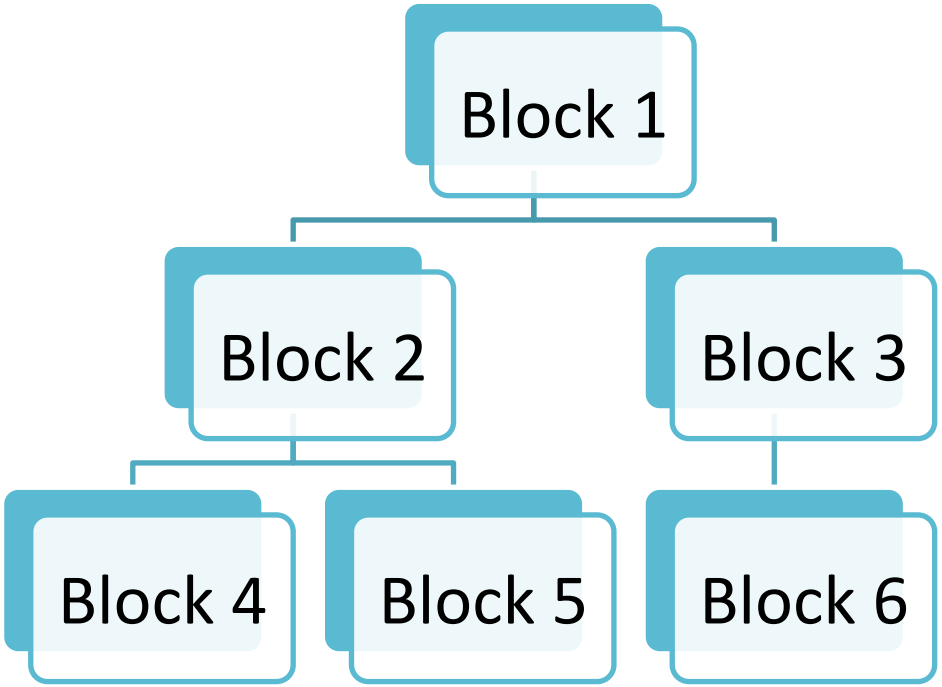
```
function makeAddFunction() {
        return function (a,b) {
                return a + b;
        };
}
```

Visibilità variabili

- Variabili e costanti dichiarate in un blocco di statement (quindi anche nel corpo delle funzioni e delle strutture di controllo) sono **locali** a quel blocco
- Al di fuori non sono riconosciute

```
function add(a,b){
      var sum=a+b;
      return sum;
}
sum=sum-1;
//errore: sum is undefined
```

Visibilità variabili



- Variabili dichiarate in block 1 sono visibili a tutti i sottoblocchi
- Quelle dichiarate in block 2 sono visibili solo da 2, 4 e 5 …
- In generale, se consideriamo gerarchicamente i blocchi di statement, un blocco *i* può vedere solo le variabili dichiarate in *i* e nei blocchi "antenati" di *i*.

Commenti

```
// questo è un commento per una sola linea
var pippo;//un cane
/*
questo è un commento per
più linee
...
*/
var pluto; //un cane

// var paperino;
```

Operatori aritmetici

y=5

Operatore	Descrizione	Esempio	y
+	Addizione	y=y + 5	10
–	Sottrazione	y=y – 5	0
*	Moltiplicazione	y=y*5	25
/	Divisione	y=y/5	1
%	Modulo (resto divisione intera)	y=y%5	0
++	Incremento	y++	6
– –	Decremento	y--	4

Tra stringhe, **+** indica la concatenazione. Es.

```
var txt= “Hello” + “ World”;
//txt è “Hello World”
```

Operatori di assegnamento

x=y (x prende il valore di y)

Operatore	Esempio	Equivalente a
+=	y+=5	y=y + 5
-=	y-=5	y=y – 5
***=**	y*=5	y=y*5
/=	y/=5	y=y/5
%=	y%=5	y=y%5

Operatori logici e di confronto

Confronto

- x==y (uguale)
- x!=y (diverso)
- x>=y
- x>y
- x<=y
- x<y

Logici

- x&&y (and)
- x||y (or)
- !x (not)

- Tra quantità numeriche assume il significato ovvio
- Tra stringhe si considera l'ordine lessicografico

```
"pippo"=="pluto" //false
"pippo"!="pluto" //true
"pippo">="pluto" //false
"pippo">"pluto" //false
"pippo"<="pluto" //true
"pippo"<"pluto" //true
```

Operatori logici

Sono valutati con un booleano

- x==y (uguale)
- x!=y (diverso)
- x>=y
- x>y
- x<=y
- x<y

- Tra quantità numeriche assume il significato ovvio

- Tra stringhe si considera l'ordine lessicografico

```
"pippo"=="pluto" //false
"pippo"!="pluto" //true
"pippo">="pluto" //false
"pippo">"pluto" //false
"pippo"<="pluto" //true
"pippo"<"pluto" //true
```

Control flow statement

If

```
if (cond) block1 else block2
```

Switch

```
switch (n){
        case a: block1; break;
        case b: block2; break;
        ...
        default: blockn;
}
```

While

```
while (cond) block
```

Do-While

```
do block while (cond)
```

For

```
for (stmt1; cond; stmt2) block
```

Oggetti in JS

JavaScript è basato su **oggetti**, ovvero strutture dati costituite da

- **proprietà** (valori, attributi associati all'oggetto)
- **metodi** (azioni per modificare le proprietà o interagire con altri oggetti)

Sintassi:

`obj.prop` (accedo alla proprietà `prop` dell'oggetto `obj`)
`obj.meth(...)` (chiamo il metodo `meth` sull'oggetto `obj`)

```
var txt = "Hello World!";
//length è un campo di String
print(txt.length);
//stamperà 12
```

```
var txt = "Hello World!";
/* toUpperCase() è un
metodo di String */
print(txt.toUpperCase());
//stamperà HELLO WORLD!
```

Oggetti in JS

JS permette di creare nuovi tipi di dato (di oggetto) a partire dai tipi di dato elementari

I tipi di dato elementari sono implementati come oggetti e sono:

- **String**
- **Number**
- **Boolean**
- **Array**

Boolean

• L'oggetto **Boolean** è utilizzato per rappresentare valori booleani, e può essere quindi **true** o **false**

```
//se non dichiarato b è false
var b = new Boolean();
var b = new Boolean(true);
//o più semplicemente
var b = true;
```

String

L'oggetto **String** è utilizzato per manipolare stringhe, ovvero sequenze di caratteri

```
var s = new String("Hello World!");
//o più semplicemente
var s = "Hello World!";
```

String – Proprietà e metodi

Proprietà			
length	Lunghezza della stringa	`s.length`	**12**
Metodi			
charAt()	Restituisce il carattere all'indice specificato	`s.charAt(4)`	**"o"**
indexOf()	Restituisce l'indice della prima occorrenza nella stringa del valore dato	`s.indexOf("o")`	**4**
lastIndexOf()	Restituisce l'indice dell'ultima occorrenza nella stringa del valore dato	`s.lastIndexOf("o")`	**7**
replace()	Rimpiazza un valore nella stringa con la stringa data	`s.replace("World", "Pippo")`	**"Hello Pippo!"**
split()	Divide in un array di sottostringhe	`s.split(" ");`	**"Hello", "World!"**
substring()	Restituisce la sottostringa compresa tra due indici dati	`s.substring(3,7)`	**"lo w"**
toUpperCase()	Converte la stringa in maiuscolo	`s.toUpperCase()`	**"HELLO WORLD!"**
toLowerCase()	Converte la stringa in minuscolo	`s.toLowerCase()`	**"hello world!"**

http://www.w3schools.com/jsref/jsref_obj_string.asp

Number

L'oggetto **Number** è utilizzato per rappresentare valori numerici, ma di per se non fornisce metodi interessanti per manipolarlo

```
var n = new Number(5);
//o più semplicemente
var n = 5;
```

- L'oggetto **Math** è utilizzato per eseguire operazioni matematiche su numeri.
- **Math** non può essere creato (tipo `var m = new Math();`), ma si possono accere comunque a metodi e proprietà

Math - Proprietà

Sintassi: `Math.nomeProprietà`

- E (ritorna il numero di Eulero)
- LN2 (ritorna il logaritmo naturale di 2)
- LN10 (ritorna il logaritmo naturale di 10)
- LOG2E (ritorna il logaritmo in base 2 di E)
- LOG10E (ritorna il logaritmo in base 10 di E)
- PI (ritorna π)
- SQRT1_2 (ritorna la radice quadrata di 1/2)
- SQRT2 (ritorna la radice quadrata di 2)

http://www.w3schools.com/jsref/jsref_obj_math.asp

Math - Metodi

Sintassi: `Math.nomeMetodo(...)`

- abs(x) (ritorna il valore assoluto di x)
- acos(x) (ritorna l'arccos di x)
- asin(x) (ritorna l'arcsin di x)
- atan2(y,x) (ritorna l'arctan di y/x)
- ceil(x) (ritorna la parte intera superiore di x)
- cos(x) (ritorna cosx, con x in radianti)
- exp(x) (ritorna E^x)
- floor(x) (ritorna la parte intera inferiore di x)
- log(x) (ritorna il logaritmo naturale di x)
- max(x,y,z,...,n) (ritorna il massimo tra gli argomenti)
- min(x,y,z,...,n) (ritorna il minimo tra gli argomenti)
- pow(x,y) (ritorna x^y)
- random() (ritorna un numero random tra 0 e 1)
- round(x) (arrotonda all'intero più vicino)
- sin(x) (ritorna sinx , con x in radianti)
- sqrt(x) (ritorna la radice quadrata di x)
- tan(x) (ritorna tanx , con x in radianti)

http://www.w3schools.com/jsref/jsref_obj_math.asp

Esercizio 1

Scrivere la funzione `cut(s,n)`, che presa una stringa `s` ritorna una stringa `r` di lunghezza <=`n` t.c.

- **`r`=`s`, se la lunghezza di `s` è <= di `n`**
- **`r` = s_{0j}+"..." se la lunghezza di `s` è > di `n`. s_{0j} è un prefisso di `s`.**

Esempio

```
cut("Ciao", 5); // = "Ciao"
cut("Paperino", 7); // = "Pape..."
```

```
function cut(s,n){
   if(s.length<=n)
      return s;
   else
      return s.substring(0,n-3)+"...";
}
```

Null e Undefined

- **undefined**: un'espressione che non ha un valore definito

Variabili senza valore:

```
var myVariable;
print(myVariable);
// il valore stampato è: undefined
```

Funzioni senza valore di ritorno:

```
function stampaQualcosa(){print("Hello World");}
var result=stampaQualcosa();
print(result);
/* stampa "Hello World" (chiamata alla funzione)
e undefined (il valore della variabile result)*/
```

- **null**: un'espressione che ha un valore definito, ma tale valore è nullo (non ha un valore concreto)

```
var myVar1=null;
var myVar2;
print(myVar1==myVar2);
```

Stampa `false` perchè `null!=undefined`

Array

- L'array è la struttura dati più semplice in JS
- Rappresenta una **sequenza indicizzata** di elementi, ed è utilizzata per memorizzare valori multipli in una singola variabile

- Ogni elemento di un array di lunghezza *N* è in una precisa posizione a cui è associato un indice numerico da *0* a *N-1*
- Paragonabile a un vettore

a =

a[0]	a[1]	...	a[N-2]	a[N-1]
0	1	...	N-2	N-1

`a[i]` è l'i-esimo elemento dell'array `a`

Array - Esempio

Problema: Memorizzare la lista dei nomi delle proprie macchine

Soluzione 1: una variabile per elemento

```
var car1 = “Saab”;
var car2 = “Volvo”;
var car3 = “BMW”;
```

Soluzione 2: Array

```
var myCars=new Array();
myCars[0]="Saab";
myCars[1]="Volvo“;
myCars[2]="BMW";
```

myCars =

“Saab”	“Volvo”	“BMW”
0	1	2

Array - Dichiarazione

```
var myCars=new Array();//dimensione non specificata
var yourCars=new Array(3); //dimensione specificata
myCars[0]="Saab";
myCars[1]="Volvo“;
myCars[2]="BMW";
yourCars[4] = “Fiat”;
```

myCars

yourCars

undefined	undefined	undefined
0	1	2

myCars.length = 0

yourCars.length = 3

Array - Dichiarazione

```
var myCars=new Array();//dimensione non specificata
var yourCars=new Array(3); //dimensione specificata
myCars[0]="Saab";
myCars[1]="Volvo“;
myCars[2]="BMW";
yourCars[4] = “Fiat”;
```

myCars

“Saab”	“Volvo”	“BMW”
0	1	2

myCars.length = 3

yourCars

undefined	undefined	undefined
0	1	2

yourCars.length = 3

Array - Dichiarazione

```
var myCars=new Array();//dimensione non specificata
var yourCars=new Array(3); //dimensione specificata
myCars[0]="Saab";
myCars[1]="Volvo“;
myCars[2]="BMW";
yourCars[4] = “Fiat”;
```

myCars

“Saab”	“Volvo”	“BMW”
0	1	2

yourCars

undefined	...	undefined	“Fiat”
0	...	3	4

`myCars.length = 3`

`yourCars.length = 5`

Sintassi alternative

```
var myCars = new
Array("Saab","Volvo","BMW");
```

```
var myCars =
["Saab","Volvo","BMW"];
```

Array - Accesso

Per accedere (in lettura o scrittura) un elemento dell'array **a** all'indice **i** → **a[i]**

```
var a = new Array();
var b = new Array("Fiat","Kia");
//accessi in scrittura
a[0]="Saab";
a[1]="Volvo";
a[2]="BMW";
//accesso in lettura
var mysecondCar = a[1];
```

Array – Tipi d'oggetto

Nell'array è possibile mettere oggetti di diverso tipo

```
var a = new Array();

a[0]="Saab";
a[1]="Volvo“;
a[2]=4;
a[3]=function(…){…};
```

Array – Proprietà e metodi (1/2)

array a e b dell'esempio precedente

Proprietà			
length	Lunghezza dell'array	`a.length`	**3**
Metodi			
concat()	Unisce due o più array e restuisce una copia degli array uniti	`r=a.concat(b)`	**r: ["Saab", "Volvo", "BMW", "Fiat", "Kia"]**
indexOf()	Cerca un elemento nell'array e ne restituisce la posizione	`r=a.indexOf("BMW")`	**r: 2**
lastIndexOf()	Restituisce l'indice dell'ultima occorrenza di un elemento nell'array	`r=a.lastIndexOf("BMW")`	**r: 2**
pop()	Rimuove l'ultimo elemento di un array, e lo restituisce	`r=a.pop()`	**a: ["Saab", "Volvo"]** **r: "BMW"**
push()	Aggiunge un elemento alla fine dell'array, restituisce la nuova lunghezza	`r=a.push("Kia")`	**a: ["Saab", "Volvo", "BMW", "Kia"]** **r: 4**

http://www.w3schools.com/jsref/jsref_obj_array.asp

Array – Proprietà e metodi (2/2)

Metodi			
reverse()	Inverte l'ordine degli elementi	`a.reverse()`	**a: ["BMW", "Volvo", Saab"]**
shift()	Rimuove il primo elemento di un array, e lo restituisce	`r=a.shift()`	**a: ["Volvo", "BMW"]** **r: "Saab"**
slice()	Restituisce un sottoarray compreso tra due indici	`r=a.slice(1,2)`	**r: ["Volvo"]**
splice()	Rimuove n elementi a partire dall'indice i, e aggiunge eventuali nuovi elementi. Restituisce l'array modificato	`r=a.slice(1,1,"Alfa")`	**r: ["BMW","Alfa", "Saab"]**
sort()	Ordina gli elementi dell'array	`a.sort()`	**a: ["BMW", "Saab", "Volvo"]**
unshift()	Aggiunge nuovi elementi all'inizio dell'array e restituisce la nuova lunghezza	`r=a.unshift("Fiat","Lancia")`	**a: ["Fiat", "Lancia", "Saab", "Volvo", "BMW"]** **r: 5**

http://www.w3schools.com/jsref/jsref_obj_array.asp

Array e strutture di controllo

Per scorrere gli elementi di un array si utilizzano le strutture di controllo cicliche (`while` o `for`)

Es. Stampa degli elementi di `a`

While

```
var i = 0;
while(i<a.length){
   print(a[i]);
   i++;
}
```

For

```
for(var i=0; i<a.length; i++)
   print(a[i]);
```

Codice più compatto

For...in

```
var car;
for(car in a)
   print(car);
```

Array - Ordinamento

- Di default, il metodo `sort()` ordina gli elementi dell'array secondo l'ordine lessicografico ascendente
- Questo vale anche per i numeri → es. 101 < 22

Per definire altri tipi di ordinamento bisogna passare al metodo `sort()` una funzione di comparazione che prende in input due elementi `a` e `b` e restituisce

- Un positivo se `a>b`
- Un negativo se `a<b`
- Zero se `a=b`

Ordine numerico ascendente

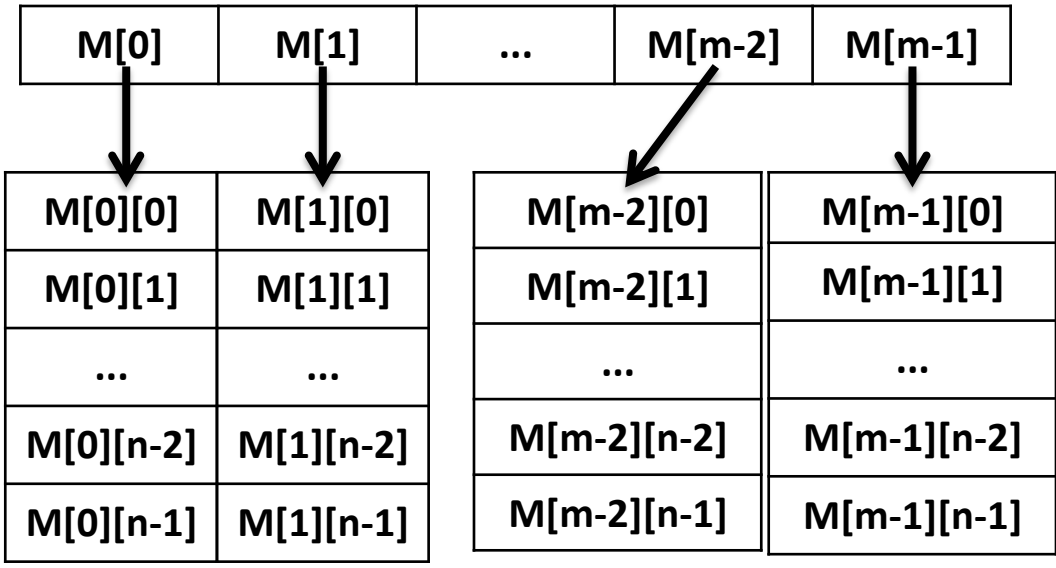
```
var points = [40,100,1,5,25,10];
function comparePoints(a,b){
   return a - b;
}
points.sort(comparePoints);
```

Array multidimensionali

- Finora abbiamo visto array monodimensionali, ovvero vettori.
- Si possono creare **array multidimensionali** ad esempio per creare **matrici** (array bidimensionali) o strutture più complesse

Es. Creazione di una matrice m x n inizializzata a 0 (ovvero Mij = 0, ∀i,j)

```
var a = new Array(m);
for(var i=0; i<a.length; i++){
   a[i]=new Array(n);
   for(var j=0; j<a[i].length;
j++)
      a[i][j]=0;
}
```



Esercizio 2

Ordinare un array di stringhe a secondo il numero di caratteri che compongono le stringhe contenute.

Es. "Pippo" (5 caratteri) < "Paperino" (8 caratteri)

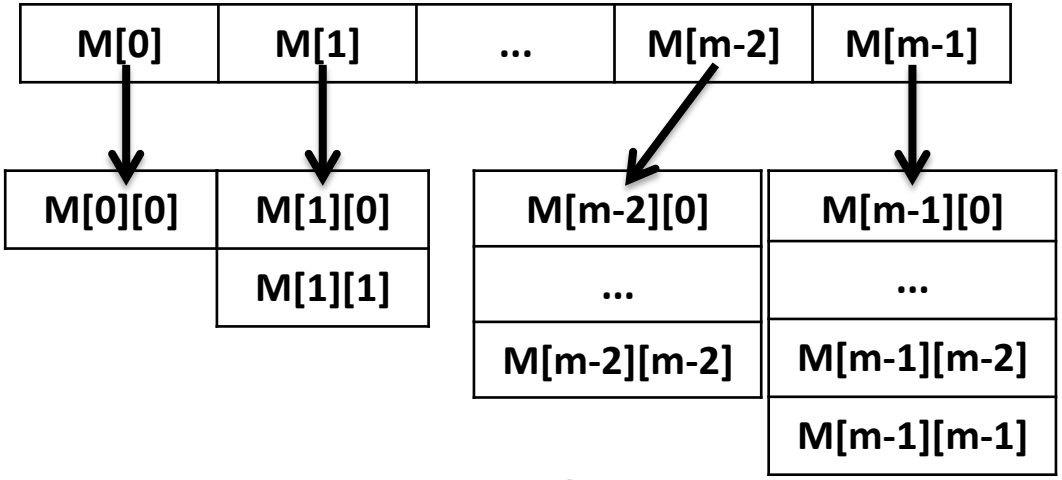
```
function myCompare(a,b){
   return a.length - b.length;
}
a.sort(myCompare);
```

oppure

```
a.sort(function(a,b){
      return a.length - b.length;
   }
);
```

Esercizio 3

Creare un array multidimensionale con la seguente struttura



e che contenga un numero random tra 0 e 100

```
var a = new Array(m);
for(var i=0; i<a.length; i++){
   a[i]=new Array(i+1);
   for(var j=0; j<a[i].length; j++)
      a[i][j]=Math.random()*100;
}
```

Array associativi

- Finora abbiamo visto array indicizzati con indici numerici
- JS supporta i cosidetti **array associativi** ovvero collezioni di coppie *(chiave, valore)*

```
var persona = new Array();
persona["nome"] = "Alessandro";
//chiave è "nome"; valore è "Alessandro"
persona["cognome"] = "Del Piero";
//chiave è "cognome"; valore è "Del Piero"
persona["nome"] = undefined;
//ora il valore dell'elemento con chiave "nome" è indefinito
```

ATTENZIONE: `persona["nome"]` è come dire `persona.nome`, ovvero crea una proprietà chiamata `nome` all'interno dell'oggetto `persona` di tipo `Array`.

Creare nuovi tipi di dato

Il paradigma Object Oriented permette di creare nuovi tipi di dato (ovvero, nuovi tipi di oggetto) a partire dai più semplici.

SINTASSI

- **Constructor:** è la funzione usata per creare nuovi oggetti.

All'interno del costruttore si dichiarano le **proprietà** dell'oggetto con la sintassi: `this.proprietà=...`

- **Metodi:** zero o più metodi (funzioni di gestione e manipolazione dell'oggetto)

La parola chiave `this` indica l'oggetto che si sta definendo

Esempio: creare un oggetto Frigorifero che ha come proprietà l'elenco degli alimenti che contiene e la quantità presente per ogni elemento; e che contenga un metodo per aggiungere una certa quantità di un elemento o per toglierla

Esempio - Frigorifero

```
function Frigorifero(){
   this.alimenti = new Array();
}
```

COSTRUTTORE

INIZIALIZZAZIONE PROPRIETA'

```
Frigorifero.prototype.aggiungi = function(alimento, quantità){
   if(this.alimenti[alimento]==undefined)
      this.alimenti[alimento]=quantità;
   else this.alimenti[alimento]+=quantità;
}
Frigorifero.prototype.rimuovi= function(alimento, quantità){
   var daTogliere=0;
   if(this.alimenti[alimento]!=undefined){
      daTogliere=min(this.alimenti[alimento], quantità);
      this.alimenti[alimento]-=daTogliere;
   }
   return daTogliere;
}
```

METODI

```
var f=new Frigorifero();
f.aggiungi("Prosciutto", 200);
f.rimuovi("Prosciutto", 150);//ritorna 150
f.rimuovi("Parmigiano", 50);//ritorna 0
```

Prototype

L'oggetto `prototype` rappresenta il modello (la classe) del tipo di oggetto che creiamo. Facilita l'estensione (**ereditarietà**) di tipi d'oggetto esistenti.

```
function FrigoConFreezer(){
}
//FrigoConFreezer eredita metodi e proprietà di Frigorifero
FrigoConFreezer.prototype = new Frigorifero();

FrigoConFreezer.prototype.rimuoviTutto= function(){
   //posso accedere alle proprietà di Frigorifero
   this.alimenti = new Array();
}
```

Esercizio - Cerchio

Esempio: creare un oggetto `Cerchio` *che ha come proprietà il solo raggio (da passare nella funzione costruttore) e tre metodi:*

- *calcolaArea(): ritorna l'area del cerchio*
- *calcolaCirconferenza(): ritorna la circonferenza del cerchio*
- *calcolaDiametro(): ritorna il diametro del cerchio*

```
function Cerchio(raggio){
   this.raggio = raggio || 1; //significa che se l'input è
undefined prende 1
}
Cerchio.prototype.calcolaArea = function(){
   return Math.pow(this.raggio,2)*Math.PI;
}
Cerchio.prototype.calcolaCirconferenza = function(){
   return 2*Math.PI*this.raggio;
}
Cerchio.prototype.calcolaDiametro = function(){
   return 2*this.raggio;
}
```

Dichiarazione metodi

Sintassi alternativa (non equivalente). I metodi si dichiarano nel costruttore

```
function Cerchio(raggio){
   this.raggio = raggio || 1;
   this.calcolaArea = function(){
   return Math.pow(this.raggio,2)*Math.PI;
   }
   this.calcolaCirconferenza = function(){
   return 2*Math.PI*this.raggio;
   }
   this.calcolaDiametro = function(){
   return 2*this.raggio;
   }
}
```