Javascript

Corso Sviluppatore front end

Docente: Ignazio Sammarco

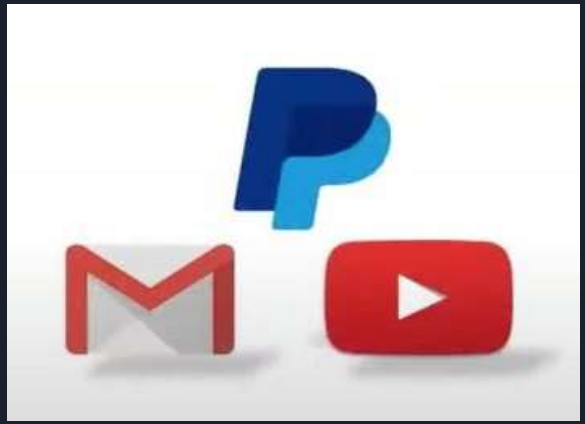
Arces Formazione 2022

Javascript

- ★ Sviluppato nel 1995 in soli 10 giorni da Brendan Eich
- ★ Inizialmente chiamato Mocha, poi LiveScript e infine Javascript
- ★ Creato per aggiungere funzionalità dinamiche alle pagine HTML
- ★ Nessun legame con Java (sintassi diverse)

Javascript

Website & Application



Industry

Javascript


Scripting Language
Client-side Technology
User Validation
Object oriented
JS
Light-weight
Server-side Technology
Platform independent
Case Sensitive

Applicazioni di Javascript

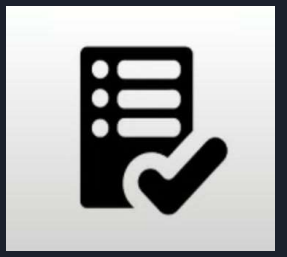
Website Development

Web Application

Game

Smartwatch

Mobile application



Input validation

Applicazioni di Javascript

Javascript è un linguaggio interpretato:

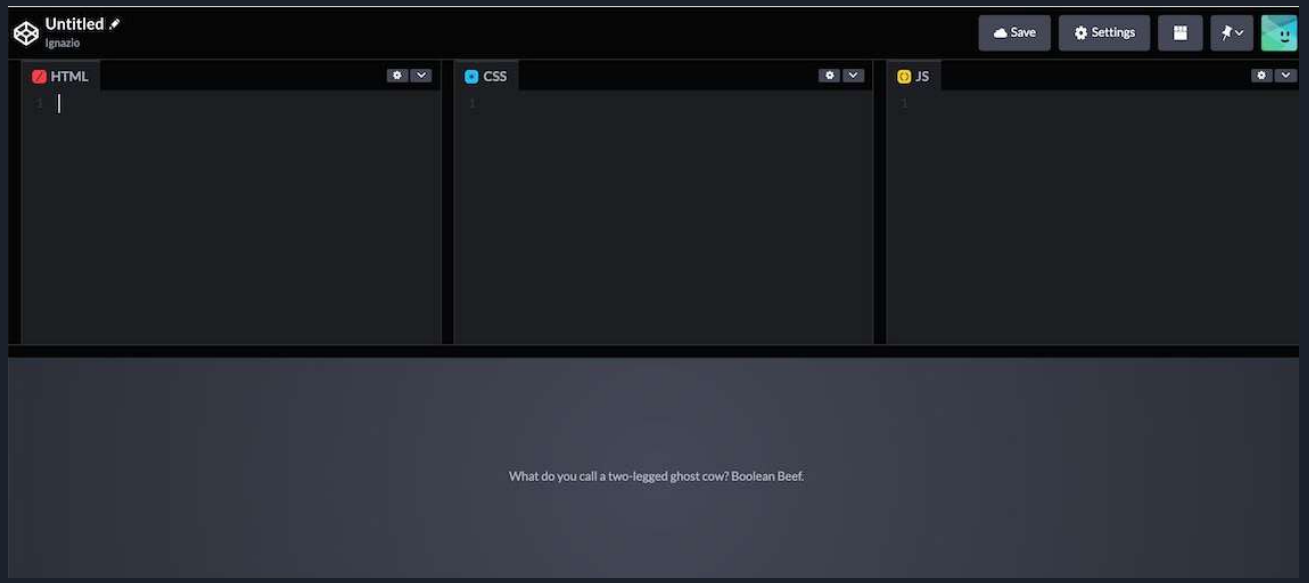
Con questo vogliamo dire che il codice che scriviamo non viene preventivamente compilato (come si fa con Java) e poi eseguito.

Un file javascript si scrive come file sorgente e quindi è eseguito direttamente dal browser.

Il motore javascript (engine) del browser provvederà a interpretarlo in tempo reale e a trasformarlo in linguaggio comprensibile dalla macchina (linguaggio macchina).

Applicazioni di Javascript

Ma facciamo un esempio
di codice javascript.
Per far questo ci affidiamo
ad un sito Codepen.io
dove scriveremo il nostro
primo codice sorgente.



Editor online

https://app.codingrooms.com/

https://app.codingrooms.com/w/pywqyWr5PaYu

https://jsfiddle.net/

https://codesandbox.io/?from-app=1

https://jseditor.io/

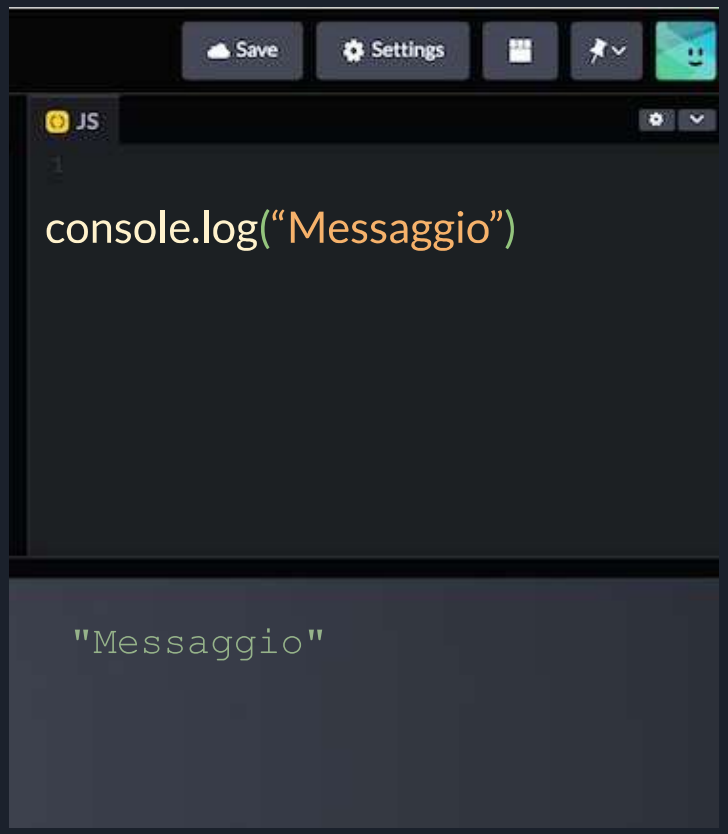
https://stackblitz.com/

Applicazioni di Javascript

Usiamo la funzione

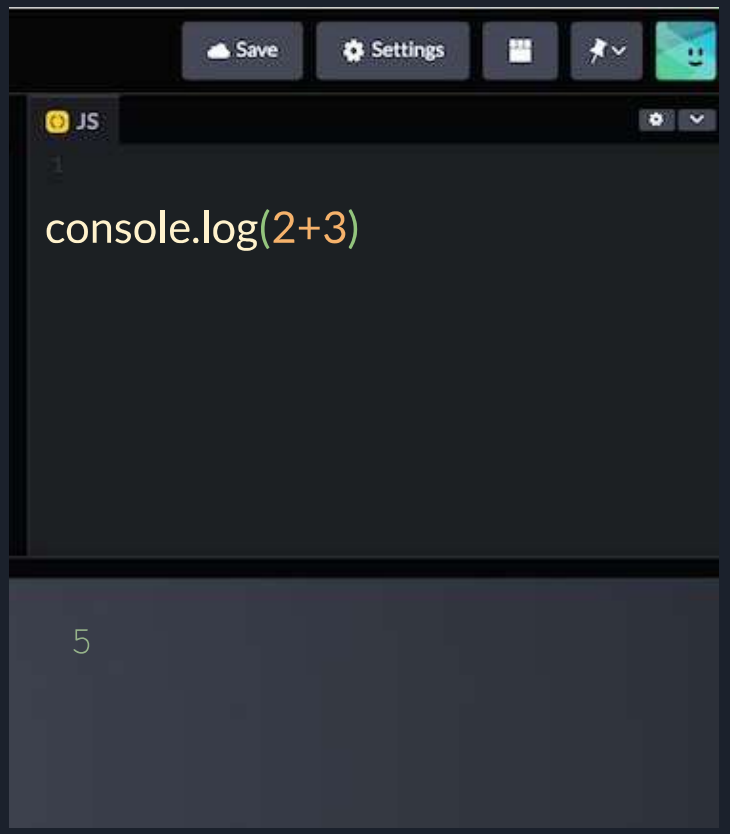
console.log()

Con questa funzione si scrive un messaggio sulla console del browser.



Applicazioni di Javascript

Ma si possono anche inserire espressioni numeriche come somme, moltiplicazioni ecc.



Applicazioni di Javascript

Si possono anche combinare testo ed espressioni numeriche nello stesso messaggio.

```
console.log("la somma è: " + 2 + 3);
```

```
somma = 2 + 3;
console.log("la somma è: " + somma);
```

Javascript

Interazione in Javascript

Javascript

Javascript ci permette di interagire con un utente mentre naviga una pagina con alcune funzioni di base

alert()

prompt()

confirm()

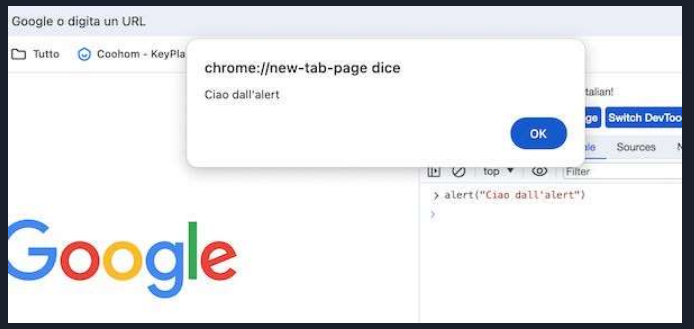
Javascript

alert()

Con questa funzione viene mostrato una finestra con un messaggio e si resta in attesa che l'utente pressi il tasto ok.

Esempio

```
alert("Ciao dall'alert")
```



Javascript

prompt()

La funzione prompt accetta due argomenti

prompt(title, [default]);

mostra una finestra modale con un campo di input e due pulsanti

Esempio

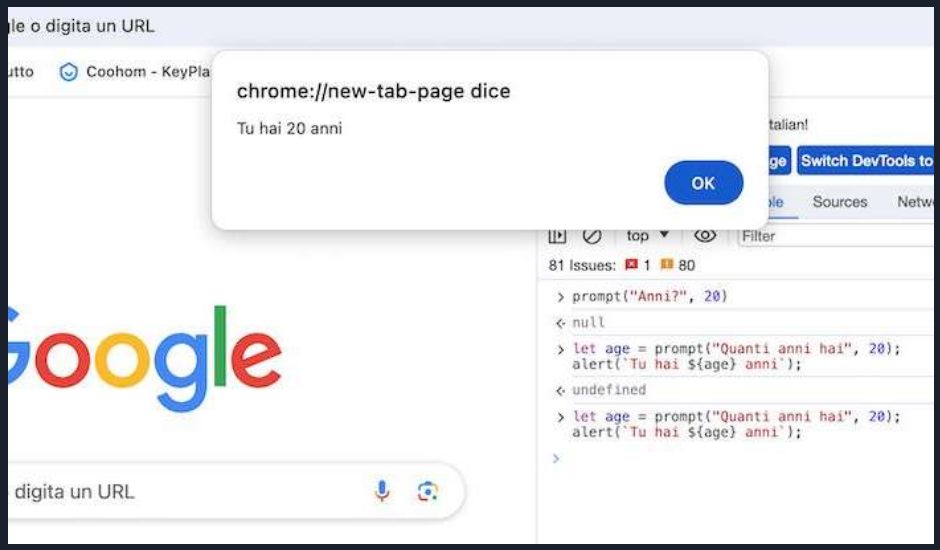
result = prompt("Anni?", 20)

Javascript

prompt()

Il valore immesso nella casella di input del prompt può essere quindi usato da altre funzioni.

```
age = prompt("Quanti anni hai", 20);
alert(`Tu hai ${age} anni`)
```



Javascript

confirm()

Mostra una finestra modale con una domanda e due pulsanti: OK e Cancel

```
result = confirm(question);
```

Se premuto ok result è True altrimenti è False.

Esempio

```
confirm("Sei italiano?")
```

Javascript

Variabili in Javascript

Variabili in Javascript

Possiamo definire una variabile in vari modi

Con la parola chiave var

```
var nominativo;
```

Senza la parola chiave var MA assegnando subito un valore

```
nominativo = "Franco";
```

Variabili in Javascript

Possiamo definire una variabile anche con let

```
let nominativo = "Franco"
```

La visibilità della variabile let è limitata all'interno del blocco if { }
Infatti se facciamo console.log fuori dall'if{} y sarà undefined

```
if (true) {
  let y = 30;
}

console.log(y); // y è undefined
```

Variabili in Javascript

Si può anche assegnare lo stesso valore a più variabili contemporaneamente:

```
nome = cognome = "Andrea"
```

Variabili in Javascript

Una variabile che è stata dichiarata ma a cui non è assegnato nessun valore è undefined

```
var nome;
tipo = typeof(nome)
console.log(tipo)
```

Variabili in Javascript - Scope

Se definiamo una costante dentro un blocco di codice essa non sarà visibile al di fuori del blocco.

Se definiamo una variabile con var essa sarà invece visibile anche fuori.

```
if(true){
    const numero = 10;
    var numero2 = 15;
}
console.log(numero);  //numero not defined
console.log(numero2); //15
```

Variabili in Javascript - Scope

Se definiamo due costanti o variabili dentro due blocchi di codice diversi questi non andranno in conflitto e i loro valori potranno essere mostrati a console

```
if(true){
    const numero = 10;
    console.log(numero);
}

if(false){
    const numero = 5;
    console.log(numero);
}
```

Variabili in Javascript

Le variabili definite al di fuori di una funzione sono globali.

Quindi sono viste **dappertutto**, anche all'interno della funzione.

In questo caso la funzione incrementa() saprà quanto vale la variabile x e aggiungerà a 10 il numero 5.

```
var eta = 10;

function incrementa(valore){
  eta = eta + valore;
}

incrementa(5);

console.log(eta) // eta sarà 15
```

Variabili in Javascript

Ma se definiamo una variabile locale (dentro la funzione) con lo stesso nome di quella globale (in questo caso x), la variabile presente all'interno della funzione avrà priorità su quella all'esterno.

```
var x = 10;

function incrementa(valore) {
    var x = 1;
    x = x + valore;
    console.log(x); // x avrà valore 6
}

incrementa(5);

console.log(x); // x avrà valore 10
```

Variabili in Javascript

Dichiarare una variabile all'interno di un blocco di codice non crea un nuovo *scope* per la variabile! La variabile sarà visibile anche fuori tranne se non si usa let.

```
var x = 10;
var y;
{
 let x = 20;
 y = x + 1;
}
console.log(x); //x sarà 10
console.log(y); //x sarà 21
```

Tipi di dati in Javascript

I tipi di dati in Javascript possono essere:

- string
- number
- boolean
- object
- undefined
- null

Tipi di dati in Javascript

In questo caso la nostra variabile `nome` sarà di tipo string.

Questo perché il tipo sarà inferito al momento dell'assegnazione del valore.

```
nome = "Andrea"
```

Tipi di dati in Javascript

Per conoscere il tipo di una variabile possiamo usare la funzione javascript typeof()

```
tipo = typeof(nome)
console.log(tipo)
```

Console

```
> typeof(a)
< 'string'
```

Tipi di dati in Javascript

Possiamo anche concatenare due o più stringhe.

Concateniamo la variabile nome con la variabile cognome e una stringa fissa.

```
nome = "Andrea"
cognome = "Belli"
nominat = "Mi chiamo "+nome+" "+cognome


console.log(nominat)
```

Tipi di dati in Javascript

L'operatore += si usa per aggiungere una stringa ad un'altra stringa in modo abbreviato.

```
nominativo = nominativo + " da Roma !"
nominativo += " da Roma !"

console.log(nominativo)
```

Tipi di dati in Javascript

Ma si può usare anche per i numeri.

```
eta = 20
eta += 4

console.log(eta)
```

Tipi di dati in Javascript

Per convertire da un tipo ad un altro (ad es numero->stringa) ci sono diversi modi:

Si possono usare le funzioni String() e toString().

Esiste anche la conversione implicita

```
n = 234;
var s = String(n);
var s = n.toString();

// ma anche conversione implicita
var s = "" + 234
console.log(typeof(s))
```

Tipi di dati in Javascript

Mettendo un segno “+” davanti ad una stringa contenente solo numeri si converte la stringa in number.

```
//conversione da stringa a numero

var s = +"234"
console.log(typeof(s)) // number
```

Condizioni

In Javascript, come in altri linguaggi, esistono gli operatori condizionali.

Il più famoso è l'operatore if.

```
if (age>17) {
 console.log("Maggiorenne");
}else{
 console.log("Minorenne");
}
```

Condizioni

Un operatore simile all'if è l'operatore ternario .

```
age = 20
console.log( age > 17 ? "Maggiorenne" : "Minorenne" )
```

Se l'età è maggiore di 17 la console stamperà la stringa "Maggiorenne " altrimenti stamperà la stringa "Minorenne"

Tipi di dati in Javascript

Array

Array in Javascript

Un array è una variabile che contiene più valori raggruppati sotto un unico nome.

In questo caso l'array è animali e al suo interno contiene tre valori (Cane, Gatto e Cavallo).

Codice

```
const animali = ["Cane", "Gatto", "Cavallo"];

console.log(animali);
```

Output

```
(3) ['Cane', 'Gatto', 'Cavallo']
    0: "Cane"
    1: "Gatto"
    2: "Cavallo"
    length: 3
    [[Prototype]]: Array(0)
```

Array in Javascript

Per accedere ai valori contenuti nell'array basta indicare il nome dell'array seguito dall'indice o posizione in cui si trova quel valore.

Il valore "Cane" in questo caso si trova in posizione o indice 0 dell'array animali.

Codice

```
const animali = ["Cane", "Gatto", "Cavallo"];

console.log(animali[0]);
console.log("Specie: "+animali[1]);
```

Output

```
"Cane"
"Specie: Gatto"
```

Array in Javascript

Array in Javascript

Creare un array

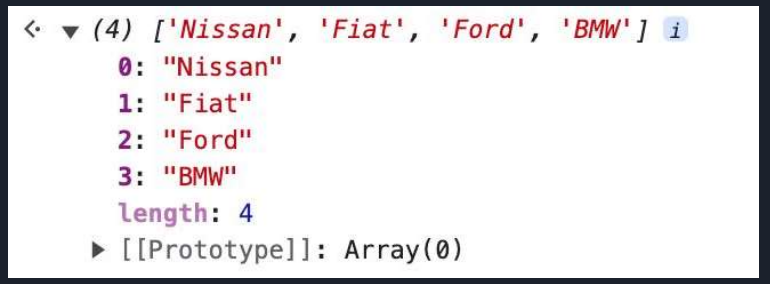
1° MODO

E’ possibile inserire i valori dell’array dentro le parentesi quadre [] separati dalla virgola “,”

Codice

```
let cars = ["Nissan", "Fiat", “Ford”, “BMW”];

console.log(cars);
```

Output



Array in Javascript

Creare un array

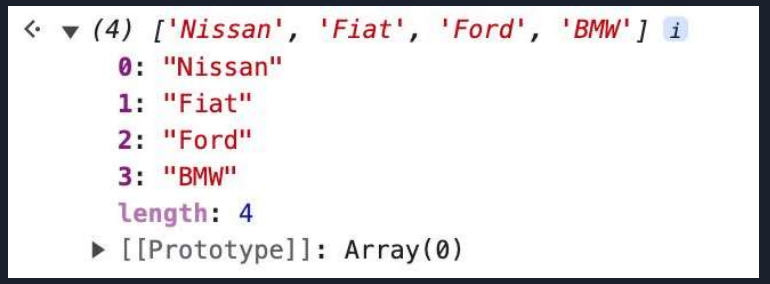
2° MODO

E’ possibile creare un array attraverso un costruttore che istanzia un oggetto di tipo Array.

Codice

```
let cars = new Array("Nissan", "Fiat", "Ford",
"BMW");

console.log(cars);
```

Output



Array in Javascript

Inserire elementi

Una volta creato l'array si possono inserire nuovi elementi con le funzioni:

- push()

Inserisce l'elemento alla FINE

Codice

```
cars.push()("Mercedes");

console.log(cars);
```

Output

```
(3) ['Nissan', 'Fiat', 'Ford', 'BMW', 'Mercedes']
    0: "Nissan"
    1: "Fiat"
    2: "Ford"
    3: "BMW"
    4: "Mercedes"
    length: 5
    [[Prototype]]: Array(0)
```

Array in Javascript

Inserire elementi

Una volta creato l'array si possono inserire nuovi elementi con le funzioni:

- unshift()

Lo inserisce all' INIZIO

Codice

```
cars.unshift()("FERRARI");

console.log(cars);
```

Output

```
(3) ['FERRARI', 'Nissan', 'Fiat', 'Ford', 'BMW', 'Mercedes']
    0: "FERRARI"
    1: "Nissan"
    2: "Fiat"
    3: "Ford"
    4: "BMW"
    5: "Mercedes"
    length: 6
    [[Prototype]]: Array(0)
```

Array in Javascript

Rimuovere elementi

Da un array si possono rimuovere deglii elementi con le funzioni:

- pop()

Rimuove l'elemento finale

Codice

```
cars.pop();

console.log(cars);
```

Output

```
(3) ['FERRARI', 'Nissan', 'Fiat', 'Ford', 'BMW']
    0: "FERRARI"
    1: "Nissan"
    2: "Fiat"
    3: "Ford"
    4: "BMW"
    length: 5
    [[Prototype]]: Array(0)
```

Array in Javascript

Rimuovere elementi

Da un array si possono rimuovere degli elementi con le funzioni:

- shift()

Rimuove il primo elemento

Codice

```
cars.shift()();

console.log(cars);
```

Output

```
(3) ['Nissan', 'Fiat', 'Ford', 'BMW']
    0: "Nissan"
    1: "Fiat"
    2: "Ford"
    3: "BMW"
    4: "Mercedes"
    length: 5
    [[Prototype]]: Array(0)
```

Array in Javascript

Ma se abbiamo molti valori possiamo accedere anche attraverso un ciclo for.

Il ciclo valuterà tutte le posizioni dell'array ed eseguira per ogni posizione il codice inserito all'interno del blocco di codice for.

Codice

```
const animali = ["Cane", "Gatto"];

for (let index=0; i<animali.length; index++ ) {
  console.log("Specie: "+animali[index]);
}
```

Output

```
"Specie: Cane"
"Specie: Gatto"
```

Array in Javascript

Il ciclo for può essere scritto in altra forma.

Per tutti gli indici presenti nell'array sarà stampato in console il valore.

Codice

```
const animali = ["Cane", "Gatto"];

for (index in animali) {
  console.log("Specie: "+animali[index]);
}
```

Output

```
"Specie: Cane"
"Specie: Gatto"
```

Array in Javascript

E’ possibile anche eseguire la stessa operazione su tutti gli elementi di un array attraverso la funzione map().

Nell’esempio la funzione x=> x*2, passata a map(), raddoppierà ogni elemento dell’array.

Codice

```
const numeri = [1, 4, 9, 16];

const numeri_x2 = numeri.map(x => x * 2);

console.log(numeri_x2);
```

Output

```
Array [2, 8, 18, 32]
```

Array in Javascript

ESERCIZIO

Dato un array di numeri noti

const numeri = [2, 5, 3, 8]

Creare un altro array costituito dai quadrati degli stessi numeri

const quadrati = [4, 25, 9, 64]

N.B. Utilizza la funzione map di js

SOLUZIONE

```
const numeri = [2, 5, 3, 8];
const quadrati = array.map(
    x => x * x
);
console.log(quadrati);
```

Array in Javascript

ESERCIZIO

Costruire la funzione quadratiArray() che prenda i numeri di un primo array e li metta in un secondo array tutti elevati al quadrato.

N.B. Usare un ciclo for

SOLUZIONE

```
function quadratiArray(numeri){
    const numeri_quadrati = [];
    for (let i = 0; i < numeri.length; i++) {
        const element = numeri[i];
        numeri_quadrati.push(element * element);
    }
    return numeri_quadrati;
}

console.log(raddoppia_array(numeri));
```

Tipi di dati in Javascript

Funzioni

Javascript - Funzioni

Una funzione è un insieme di istruzioni racchiuse in un blocco di codice disegnata per eseguire un particolare compito.

- somma di due numeri
- stampare una lista di articoli
- calcolare l'area di un quadrato
- ecc..

Javascript - Funzioni

Le funzioni sono riusabili, sono definite una volta sola e possono essere chiamate all'interno del codice più volte con valori diversi e restituire così risultati diversi.

Javascript - Funzioni

Le funzioni aiutano a dividere un problema molto complesso in operazioni più semplici. E rendono i nostri programmi più facili da capire e da manutenere/modificare.

Javascript - Funzioni

Le funzioni in Javascript hanno una serie di caratteristiche

- può avere un nome
- può accettare argomenti o parametri di ingresso
- può restituire valori
- può essere passata come parametro ad un altra funzione
- e molte altre ...

Javascript - Funzioni

Un esempio di funzione è appunto la funzione somma()

Il nome della funzione è preceduto dalla parola riservata *function*.

All'interno delle parentesi tonde ci sono i parametri (numeri da sommare) e dopo la parola *return* c'è il valore da restituire

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

Javascript - Funzioni

ESERCIZIO

Scrivi una funzione `FormatDate(data)` che accetti una stringa in formato italiano: “GG-MM-AAAA” e lo trasformi in una stringa in formato internazionale: “AAAA/MM/GG”

La stringa ottenuta sarà quella da passare come parametro alla funzione di calcolo dell’età.

Javascript - Funzioni

ESERCIZIO

```
function calcola_eta(dataNascita){
    let dataCorrente = new Date();
    let annoCorrente = dataCorrente.getFullYear();
    let annoDiNascita = dataNascita.getFullYear();
    let eta = annoCorrente - annoDiNascita;
    return eta;
}

calcola_eta("1980-10-21")
```

Javascript - Funzioni

ESERCIZIO

```
function calcola_eta(dataNascita){ // input AAAA/MM/GG
    let dataCorrente = new Date();
    let annoCorrente = dataCorrente.getFullYear();
    let dataNascitaObj = new Date(dataNascita);
    let annoDiNascita = dataNascitaObj.getFullYear();
    let eta = annoCorrente - annoDiNascita;
    return eta;
 }
```

SOLUZIONE 1

```
function formatDate(data_it){
    giorno = data_it.substring(0,2)
    mese = data_it.substring(3,5)
    anno = data_it.substring(6,12);
    date_en = anno+"-"+mese+"-"+giorno;
    console.log(date_en)
    return date_en;
}

data_en = formatDateEn('22/01/1980')
eta = calcola_eta(data_en);
console.log(eta)
```

SOLUZIONE 2

```
function formatDateEn(data_it) {
    const array = data_it.split('/');
    // const array = data_it.split(/[/-]/);
    const date_en = array[2] + array[1] + array[0];
    return date_en;
}

data_en = formatDateEn('22/01/1980')
eta = calcola_eta(data_en);
console.log(eta)
```

Javascript - Funzioni

Le funzioni in javascript possono anche non avere un nome dopo la parola chiave *function* e in questo caso possono essere assegnate a una variabile o ad una costante.

```
var lancia_dadi = function (colore){
    //Math.random restituisce valori tra 0 e 0,99..
    let risultato = Math.floor(Math.random() * 6) + 1;
    return risultato;
}

console.log(lancia_dadi(colore))
```

Javascript - Definizione delle funzioni

Function definition

Arrow functions

```
function areaQuadrato(lato){
    return lato * lato;
}

let areaQuadrato = function(lato){
    return lato * lato;
}

let areaQuadrato = (lato) => {
    return lato * lato;
}

let areaQuadrato = (lato) => lato * lato;
```

Funzioni in Javascript

Le funzioni sono delle black box.

Prendono qualcosa in input e restituiscono qualcosa in output.

Black perché possiamo anche non sapere come fanno le operazioni.

Funzioni in Javascript

Le funzioni in javascript sono funzioni di prima classe che possiamo trattare come valori.

Possiamo ad esempio assegnarle ad una variabile oppure passarle come parametro ad un'altra funzione.

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

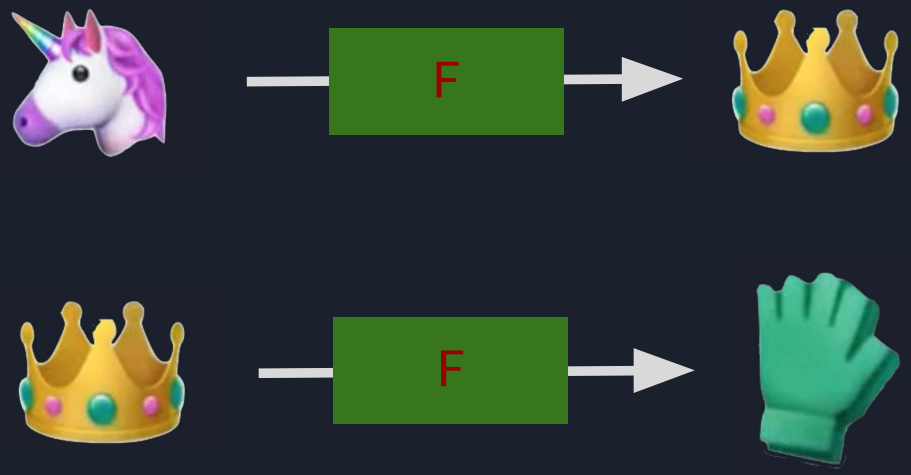
oppure

```
const add = (a,b) => a + b;
```

Funzioni in Javascript

Le funzioni possiamo comporle.

Se l' output di una funzione è compatibile con l'input dell'altra funzione.



Funzioni in Javascript


```
string.charAt(0).toUpperCase()
```

Funzioni in Javascript

Tante piccole funzioni semplici e specializzate possono essere composte a creare componenti complessi.

Il segreto è sapere:

- cosa fa una funzione
- quali input vuole (tipi di dati)   
- cosa restituisce

Array in Javascript

ESERCIZIO

Dato un array di nomi di calciatori

const array = ["totti", "ronaldo", "messi"]

Creare un altro array dove la prima lettera di ogni giocatore è maiuscola

const array 2= ["Totti", "Ronaldo", "Messi"]

SOLUZIONE

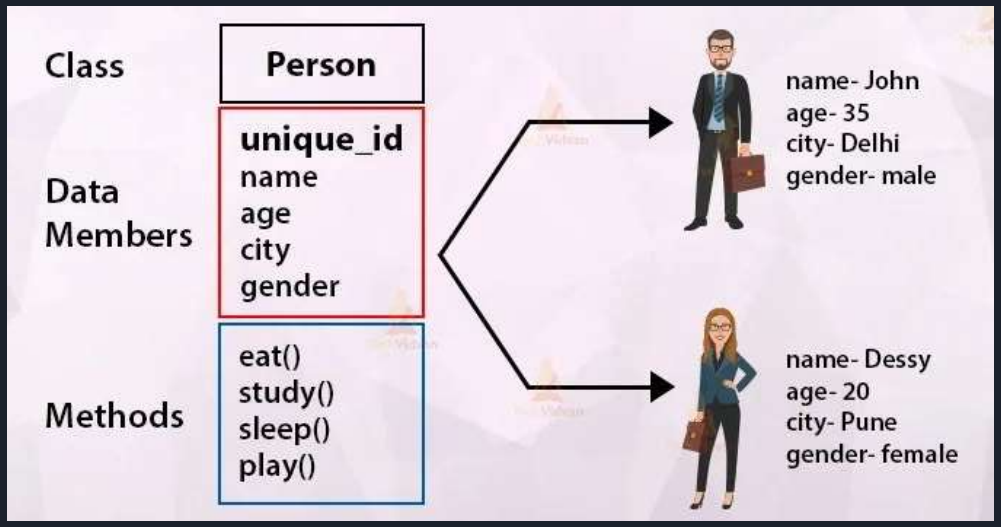
```
const array = ["totti", "ronaldo","messi"];

const array2 = array.map(

   nome => nome[0].toUpperCase()+nome.slice(1)

);

console.log(array2);
```

Tipi di dati in Javascript

Oggetti

Oggetti in Javascript

Gli oggetti in JavaScript rappresentano entità complesse che raggruppano dati (proprietà) e comportamenti correlati (metodi)



Oggetti in Javascript

In Javascript qualsiasi elemento può essere considerato un oggetto, anche quegli elementi che noi abbiamo definito come tipi primitivi:

```
let s = "Hello"
s.toUppercase()
```

Pur essendo una string (tipo primitivo) al suo interno possiamo trovare dei metodi, come se fosse un oggetto.

Oggetti in Javascript

Un primo modo per definire un oggetto in Javascript è attraverso la sintassi:

literal

Oggetti in Javascript

Gli oggetti *literal* sono collezioni di coppie chiave-valore, dove le chiavi sono i nomi delle proprietà dell'oggetto e i valori possono essere di qualsiasi tipo, inclusi altri oggetti, array e funzioni.

Oggetti in Javascript

Gli oggetti literal sono racchiusi all'interno di due parentesi graffe e le coppie chiave valore sono separate da due punti.

Codice

```
const persona = {
      nome: "Marco",
      età: 25
}
console.log(persona);
```

Output

```
{nome: "Marco", età: 25}
 1.     età: 25
 2.     nome: "Marco"
 3.     __proto__: Object
```

Oggetti in Javascript

Per accedere ad un determinato valore bisogna indicare il nome dell'oggetto seguito da un punto e la chiave.

oggetto.chiave

Codice

```
var nome = persona.nome;
var età = persona.età;

console.log("Nome: " + nome);
console.log("Età: " + età);
```

Output

```
"Nome: Marco"
"Età: 25"
```

Oggetti in Javascript

Gli oggetti in JavaScript sono entità dinamiche, la loro struttura può essere modificata anche dopo averla definita.

Codice

```
var persona = {};
persona.nome = "Mario";
persona.cognome = "Rossi";
persona.indirizzo = {
 via: "Via Garibaldi",
 numero: 15,
 CAP: "00100",
 citta: "Roma"
};
persona.eta = 32;
console.log(persona)
```

Output

```
// [object Object]
{
  "nome": "Mario",
  "cognome": "Rossi",
  "indirizzo": {
    "via": "Via Garibaldi",
    "numero": 15,
    "CAP": "00100",
    "citta": "Roma"
  },
  "eta": 32
}
```

Oggetti in Javascript

Un oggetto può contenere anche delle funzioni (metodi) al suo interno, oltre le proprietà.

In questo caso abbiamo un metodo calcolaEta()

Codice

```
const persona = {
 nome: 'Marco',
 anno_nasc: 1998,
 calcolaEta: function () {
   const eta = new Date().getFullYear() -
this.anno_nasc;
   return `L'età di ${this.nome} è ${eta} anni`;
 },
};
console.log(persona.calcolaEta());
```

Oggetti in Javascript

Ecco un altro esempio di metodo che restituisce il nominativo completo a partire dalle informazioni separate.

Codice

```
persona = {
 nome: "Mario",
 cognome: "Rossi",
 getNominativo: function() {
     return this.nome+this.cognome;
 }
}
console.log(persona.getNominativo());
```

Oggetti in Javascript

ESERCIZIO

Crea un oggetto `person` di tipo Literal con tre chiavi: nome, età e paese e inserisci all'interno dell'oggetto il metodo showPerson(){} che faccia vedere una stringa di presentazione con i tre valori di nome, età e paese concatenati tra loro e separati da una virgola.

Suggerimento

Utilizza il template *literals* (quello con il simbolo tilde ` e il simbolo ${}) per mostrare il messaggio di presentazione.

Oggetti in Javascript

SOLUZIONE

```
var persona = {
    nome: "Marco",
    età: 25,
    paese: "Italia",
    showPerson: function() {
        return `${persona.nome} ha ${persona.età} anni e vive in ${persona.paese}`;
    }
}
```

```
console.log( persona.showPerson());
```

Oggetti in Javascript

```
// ************************************************************
// Il problema con gli oggetti literal è la ripetitività del codice
// se voglio creare più di un oggetto devo copiare e ripetere la struttura
// e il metodo getNominativo()
// ************************************************************
```

Oggetti in Javascript

Se volessimo creare più oggetti con la notazione Literal (parentesi graffe) dobbiamo ripetere il metodo getNominativo() più volte.

```
persona1 = {
 nome: "Mario",
 cognome: "Rossi",
 getNominativo: function() {
     return this.nome+this.cognome;
 }
}

persona2 = {
 nome: "Marco",
 cognome: "Bianchi",
 getNominativo: function() {
     return this.nome+this.cognome;
 }
}
```

Oggetti in Javascript

Un oggetto può essere definito attraverso una funzione COSTRUTTORE.

E poi essere istanziato con la keyword new

Codice

```
//Constructor
function Book() {
 console.log('Book initialized');
}
const book1 = new Book();
const book2 = new Book();
```

Console

```
Book initialized
Book initialized
```

Oggetti in Javascript

All'oggetto Book possiamo passare dei parametri, ad esempio: title, author, year.

Codice

```
//Constructor
function Book(title, author, year) {
  this.title = title;
  this.author = author;
  this.year = year;
}
const book1 = new Book("Nome Rosa", "Eco U.", "1992");
console.log(book1)
```

Oggetti in Javascript

A questo punto possiamo scrivere il metodo getSummary() una SOLA VOLTA e poi usare la stessa funzione più volte

Codice

```
//Constructor
function Book(title, author, year) {
  this.title = title;
  this.author = author;
  this.year = year;
  this.getSummary = function () {
    return 'Hi,  ' + this.title + ' book was written by'
           +this.author;
   };
}
const book1 = new Book("Nome Rosa", "Eco U.", "1992");
const book2 = new Book("Odissea", "Omero", "720 a.C");
```

Oggetti in Javascript

Se guardiamo nella console vediamo che la funzione getSummary(), anche se la sua definizione è unica, si trova in ogni oggetto istanziato.

SI USA MOLTA MEMORIA !!

```
console.log(book1)
console.log(book2)
```

Console

```
Book {title: 'Nome Rosa', author: 'Eco U.', year: '1992', getSummary: ƒ}
    author: "Eco U."
    getSummary: ƒ ()
    title: "Nome Rosa"
    year: "1992"
    [[Prototype]]: Object

Book {title: 'Odissea', author: 'Omero', year: '780 a.C', getSummary: ƒ}
    author: "Eco U."
    getSummary: ƒ ()
    title: "Nome Rosa"
    year: "1992"
    [[Prototype]]: Object
```

Oggetti in Javascript

La soluzione è mettere il metodo getSummary() all'interno del prototype.

```
//Constructor
function Book(title, author, year) {
  this.title = title;
  this.author = author;
  this.year = year;
}

Book.prototype.getSummary = function () {
    return 'Hi,  ' + this.title + ' scritto da
'+this.author;
};
```

```
const book1 = new Book("Nome Rosa", "Eco U.", "1992");
```

Oggetti in Javascript

Se facciamo un console.log() dell'oggetto a questo punto vedremo la situazione a destra.

Console

```
Book {title: 'Nome Rosa', author: 'Eco U.', year: '1992'}
    author: "Eco U."
    title: "Nome Rosa"
    year: "1992"
    [[Prototype]]: Object
        getSummary: ƒ ()
        constructor: ƒ Book(title, author, year)
        [[Prototype]]: Object
```

Prototype Javascript

Abbiamo detto che gli oggetti JS sono dinamici

Quindi, se creo prima una classe Person che modella una persona

```
function Person(firstName, lastName){
 this.firstName = firstName;
 this.lastName = lastName;
}
```

Prototype Javascript

In un secondo tempo posso aggiungere un metodo nuovo, che ad esempio mi da il nome completo della persona

```
function Person(firstName, lastName){
 this.firstName = firstName;
 this.lastName = lastName;
}

function(){
 return this.firstName + " " + this.lastName;
}
```

Prototype Javascript

Questa funzione però la devo assegnare ad ogni oggetto creato sullo stampo della classe Person

```
const person1 = new Person("Bruce", "Wayne");
person1.getFullName = function(){
 return this.firstName + " " + this.lastName;
}
```

```
const person2 = new Person("Clark", "Kent");
person2.getFullName = function(){
 return this.firstName + " " + this.lastName;
}
```

Prototype Javascript

E qui ci viene in aiuto il concetto di Prototype

```
Person.prototype.getFullName = function(){
  return this.firstName + " " + this.lastName;
}
```

```
console.log(person1.getFullName());     -> Bruce Wayne
console.log(person2.getFullName());     -> Clark Kent
```

Prototype Javascript

il Prototype viene assegnato una volta sola e può essere usato da più oggetti

```
Person.prototype.getFullName = function(){
  return this.firstName + " " + this.lastName;
}
```

```
console.log(person1.getFullName());     -> Bruce Wayne
console.log(person2.getFullName());     -> Clark Kent
```

Prototype Javascript

Se creo due persone con l'oggetto Person e aggiungo una proprietà al prototype vedremo quella proprietà in tutti gli oggetti creati.

```
function Person(first, last, age, eye) {
 this.firstName = first;
 this.lastName = last;
}
Person.prototype.nationality = "English";

const myFather = new Person("John", "Doe");
console.log(myFather.nationality)

const myMom = new Person("Lisa", "Dan");
console.log(myMom.nationality)
```

Oggetti in Javascript

Un metodo va messo in `prototype` quando si desidera condividere lo stesso comportamento tra diverse istanze dell'oggetto, risparmiando memoria e semplificando la manutenzione del codice.

Tuttavia, se il metodo ha uno scopo specifico legato a una singola istanza dell'oggetto, può essere più appropriato definirlo direttamente nel costruttore.

Oggetti in Javascript

ESERCIZIO

Dato un array di oggetti che rappresentano gli articoli in un carrello della spesa, **scrivi la funzione**:

sommaQuantitaArticolo (carrello, articolo)

che accetti i parametri carrello (l'array degli oggetti) e articolo (l'articolo specifico) e che restituisca la somma delle quantità di quell'articolo.

```
var carrello = [
   { articolo: 'Mele', quantita: 3 },
   { articolo: 'Banane', quantita: 2 },
   { articolo: 'Mele', quantita: 5 },
   { articolo: 'Arance', quantita: 1 },
   { articolo: 'Mele', quantita: 2 },
   { articolo: 'Arance', quantita: 5 }
];
```