Classi, Costruttori, Prototipi

```
class NomeClasse {
  constructor (parametri) {
    // inizializzazione dell'oggetto
  }

  metodo1() {
    // corpo del metodo
  }

  metodo2() {
    // corpo del metodo
  }

  // altre proprietà e metodi
}
```

Costruttori

Un costruttore è una funzione speciale che viene utilizzata per creare e inizializzare oggetti che sono istanze di una classe. Una classe in JavaScript è un tipo di oggetto che definisce un insieme di proprietà e metodi che saranno condivisi da tutte le istanze della classe.

```
constructor (parametri) {
    // inizializzazione dell'oggetto
  }
```

Metodi

È una funzione associata ad un oggetto o ad una classe, che consente di manipolare gli attributi dell'oggetto o di fare operazioni specifiche sulla classe. In altre parole, un metodo definisce il comportamento dell'oggetto o della classe.

Un metodo è simile ad una **funzione**, ma viene definito all'interno di una classe o di un

oggetto e può accedere alle proprietà dell'oggetto o della classe tramite la parola chiave `this`

```
metodo() {
    // corpo del metodo
  }
```

New

L'operatore `new` **crea** un nuovo oggetto vuoto, **chiama** la funzione costruttore passando come this l'oggetto vuoto appena creato, più i parametri indicati, e **restituisce** l'oggetto inizializzato

```
const nuovoOggetto = new FunzioneCostruttrice();
```

Prototipi

Un prototipo è un oggetto dal quale altri oggetti possono ereditare proprietà e metodi. Ogni oggetto in JavaScript ha un prototipo, che può essere definito in modo esplicito o ereditato implicitamente dalla catena dei prototipi.

La catena dei prototipi è il meccanismo che JavaScript utilizza per la gestione dell'ereditarietà tra oggetti. Quando un oggetto cerca una proprietà o un metodo che non è definito al suo interno, JavaScript cerca nella catena dei prototipi dell'oggetto per trovare la proprietà o il metodo richiesto.

La proprietà `prototype` di una funzione è utilizzata per definire il prototipo dell'oggetto creato con l'operatore `new`. Quando si crea un nuovo oggetto con `new`, il prototipo dell'oggetto viene impostato sull'oggetto definito nella proprietà `prototype` della funzione costruttrice.

Quando si vuole leggere il valore di una proprietà di un oggetto, si guarda se l'oggetto ha la chiave cercata.

1. Se la chiave è presente, il valore è quello della chiave nell'oggetto
2. Se la chiave non è presente, e l'oggetto ha un prototipo, si cerca la proprietà nel prototipo
3. Se la chiave non è presente, e l'oggetto non ha un prototipo, il risultato è `undefined`

Se aggiungiamo un metodo a un particolare oggetto (diciamo, *pippo*), il metodo sarà disponibile solo per quell'oggetto.

Se aggiungiamo un metodo al *prototipo* di un oggetto, (diciamo, *Persona*), il metodo sarà disponibile per tutti gli oggetti che hanno lo stesso prototipo.

Tutte le funzioni (e in particolare: le **funzioni costruttore**) hanno una proprietà che si chiama `prototype`, inizializzata automaticamente dal linguaggio quando si dichiara una funzione, che contiene un oggetto pronto per fare da prototipo per gli oggetti inizializzati dalla funzione.

L'operatore `new` assegna automaticamente come prototipo dell'oggetto creato, il `prototype` della sua funzione costruttore.

Ciò crea un vero **legame permanente** fra gli oggetti e i loro costruttori

```
nomeClasse.prototype.nomeMetodo = function() {
 //corpo della funzione
};
```

Classi

La sintassi con la dichiarazione `class` consente di definire funzione-costruttore e metodi di un oggetto di particolare tipo in maniera molto compatta.

All’interno dei metodi (incluso il costruttore), l’oggetto che viene manipolato (o creato) è riferito da `this`. La creazione di nuovi oggetti avviene dunque con `new`

```
class Persona {
  constructor(nome,età) {
    this.nome = nome
    this.età = età
  }
  compleanno() {
    this.età++
  }
}

var pippo = new Persona(“Pippo”,35)
```