This

La parola chiave `this` viene spesso usata nei metodi degli oggetti per riferirsi all'**oggetto stesso** che sta eseguendo quel metodo. In questo modo, il metodo può essere **riutilizzato** da oggetti diversi, adattandosi automaticamente a ciascuno di essi.

Window

L'oggetto window rappresenta la finestra del browser. Usa metodi come `alert`, `confirm`, `prompt`, `setTimeout`....

Var e Let

Utilizzati per la dichiarazione di variabili:
Var ha lo scope di funzione, quindi é visibile all'interno di tutta la funzione in cui é dichiarato
Let ha lo scope di blocco, quindi é visibile solamente all'interno di un blocco operazioni in cui é dichiarato

```
function test() {
  if (true) {
    var x = 10;
    let y = 20;
    console.log(y) //✅ 20 (é visibile all'interno del blocco if)
  }

  console.log(x); // ✅ 10 (var è visibile in tutta la funzione)
  console.log(y); // ❌ Errore: y non è visibile fuori dal blocco
}

test();
```

Closure

Una **closure** è una funzione che **ricorda** le variabili del contesto in cui è stata creata, **anche dopo che quel contesto è finito**.

```
function creaContatore() {
  let count = 0;
  return function () {
    count++;
```

```
    return count;
  };
}

const contatore = creaContatore();
contatore(); // 1
contatore(); // 2
```

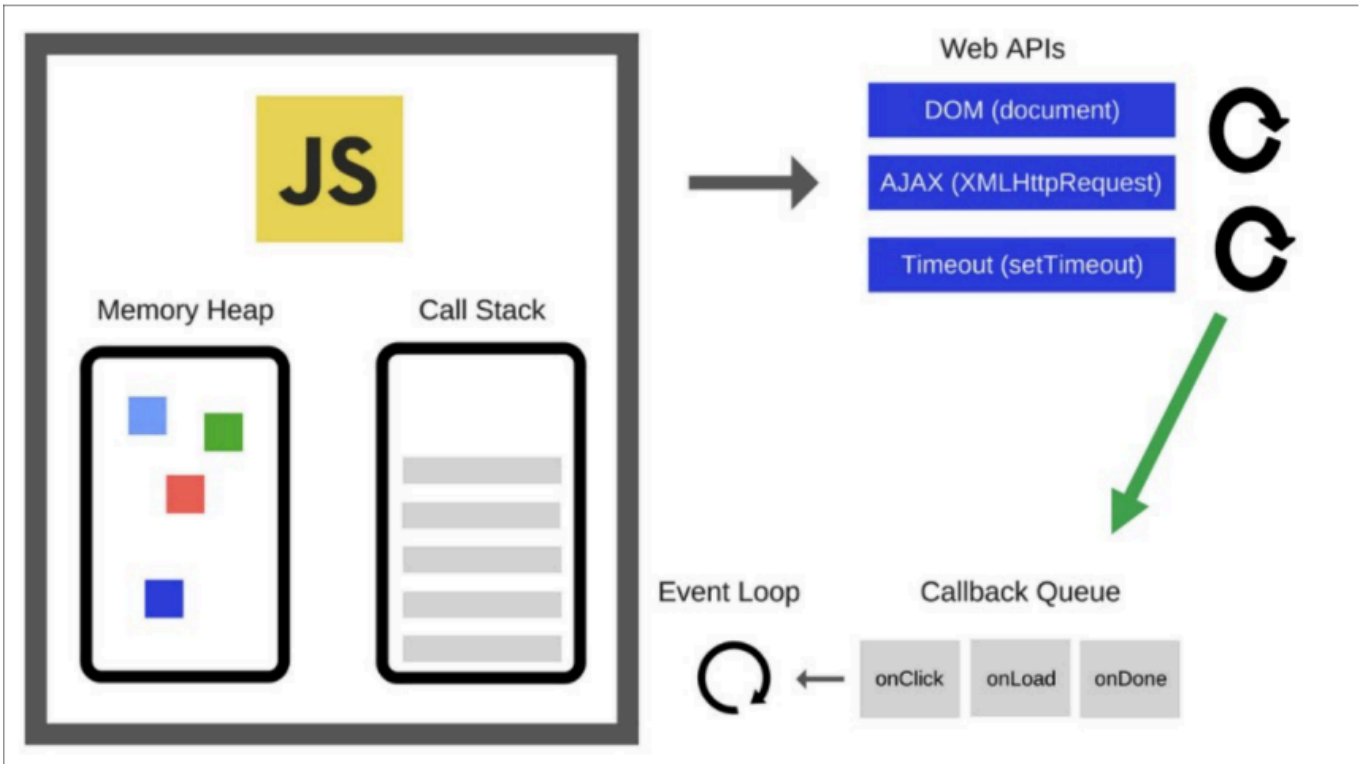
Prototipo

Un **prototipo** in JavaScript è un **oggetto "modello"** da cui altri oggetti **ereditano proprietà e metodi**.
Ogni oggetto ha un collegamento al suo prototipo, formando una **catena di ereditarietà** (prototype chain).

```
let animal = { eats: true };
let rabbit = { jumps: true };
rabbit.__proto__ = animal; // (*)
// ora in rabbit possiamo trovare entrambe le proprietà
alert( rabbit.eats ); // true (**)
alert( rabbit.jumps ); // true
```

Event loop

L'**event loop** è il meccanismo di JavaScript che gestisce **l'esecuzione asincrona**, permettendo di eseguire il codice, gestire eventi e operazioni non bloccanti in modo ordinato.



- Memory Heap: alloca la memoria
- Call Stack: registro delle funzioni attive in esecuzione
- API del Browser
- Callback Queue: contiene le funzioni di callback da eseguire non appena la call stack é vuota

AJAX

Tecnica che permette alle pagine web di comunicare con un server remoto in background senza ricaricare la pagina, usa:

- `XMLHttpRequest` o `fetch`
- Javascript
- HTML DOM per visualizzare o utilizzare dati

XMLHttpRequest

API di JavaScript utilizzata per eseguire richieste HTTP (GET, POST, ecc...) verso un server senza dover ricaricare la pagina. Base di *AJAX* ma sostituita da `fetch`
Procedimento:

- `new XMLHttpRequest()` crea l'oggetto XHR
- `open(method, url, async)` specifica il tipo di richiesta (`method: POST, GET, ecc...`) all' `url` specificato e se la richiesta é asincrona o sincrona

- con `send()` si invia la richiesta al server
- con `responseText` visualizziamo la risposta DOM String
- `xhr.onload` é la funzione da eseguire alla ricezione della risposta

Promise

Una promise é un oggetto che rappresenta il risultato futuro di una operazione asincrona

Fetch

API basate sulle promise per richieste AJAX, sostituiscono `XMLHttpRequest` perché hanno una sinstassi piú semplice

Cors

Meccanismo di sicurezza implementato nei browser.
Serve per controllare se una pagina web puó fare richieste HTTP a un dominio diverso da quello da cui é stata caricata

Quando é permesso?
Permesso solo se il server di destinazione lo consente esplicitamente.

Come?
Inviando una risposta con un header HTTP chiamato:

```
Access-Control-Allow-Origin: *
```

Oppure:

```
Access-Control-Allow-Origin: https://sitoesempio.com
```

NodeJS

NodeJs é un ambiente di esecuzione JavaScript che permette di eseguire codice JS fuori dal browser

MicroTask

Le microtask in javascript sono operazioni asincrone accumulate che vengono eseguite subito dopo la terminazione di un codice sincrono e prima delle macrotask
Questo permette di gestire in modo preciso la sequenza di esecuzione asincrona

Routing

Processo di gestione della navigazione tra diverse pagine in un'applicazione web, senza ricaricare l'intera pagina.
In maniera semplice il routing permette di mostrare contenuti diversi cambiando l'url, ma mantendendo L'SPA (Single Page Application, app web che carica una singola pagina HTML e aggiorna dinamicamente il contenuto)

Middleware

Funzione che si inserisce all'interno del flusso di elaborazione di una richiesta e puó:

- Modificare la richiesta o la risposta
- Bloccare o far proseguire l'esecuzione
- Aggiungere funzionalitá (autenticazione, logging, ecc...)
  In pratica é un filtro tra richiesta e risposta

REST

Representational State Transfer, é un insieme di linee guida che definisce una buona progettazione di una API web.
Con REST si usano le normali richieste HTTP per comunicare con un server e gestire le risorse.
Caratteristiche principali:

- Separazione tra client e server
- Stateless: ogni richiesta HTTP é a se e il server non ricorda nulla delle richieste precedenti
- Risorse accessibili mediante URL
- Risorse rappresentate come tramite JSON o XML

API REST

Comunicano tramite richieste HTTP per eseguire operazioni CRUD(Create, Read, Update, Delete)