TeoriaJS

Javascript è un linguaggio sviluppato per rendere interattive le pagine web.
Javascript è:

- **Dinamico**: non è compilato e gira su VM
- **Loosely typed**: non bisogna specificare il tipo di una variabile
- **Case-sensitive**

Transpilers

Sono utilizzati per tradurre linguaggi differenti in una versione specifica di JS. Utili nel caso in cui utilizziamo codice alternativo, ma dobbiamo supportare anche browser più vecchi

Polyfill

Sono librerie JS che aggiunge funzionalità che determinati browser (quelli più vecchi) non supportano, come API o funzioni moderne.

Ad esempio supponiamo di voler utilizzare `Array.prototype.includes` introdotto in ES7 non supportato da Internet Explorer.

```
if (!Array.prototype.includes) {
  Array.prototype.includes = function(searchElement) {
    return this.indexOf(searchElement) !== -1;
  };
}
```

Quindi se il browser non lo supporta, definiamo manualmente il comportamento.

Garbage collector

Algoritmo che capisce gli oggetti non più raggiungibili dallo script e rilascia la memoria.

Metodo

In javascript sono delle funzioni associate ad un oggetto e utilizza i dati dell'oggetto che chiama il metodo.

Strict mode

Si attiva utilizzando `"use strict";` e attiva una modalità più "rigorosa", rendendo errori degli errori che normalmente JavaScript ignora o gestisce male, come ad esempio la dichiarazione di variabili senza

let/var/const.
Utile per avere un codice più sicuro, pulito e debug-friendly.

Scope

Lo scope è la visibilità di una variabile, ovvero la porzione di codice in cui possiamo utilizzare il nome della variabile.

Le variabili definite all'interno di una funzione hanno lo scope relativo al blocco funzionale.
Le variabili definite fuori da ogni funzione, hanno scope globale e sono visibili ad ogni altro js nella pagina.

Parametri mancanti

I parametri mancanti nelle funzioni in javascript sono impostati ad undefined.

```
function somma(a, b) {
  let somma = a + b; //b è undefined
  // int + undefined = NaN (Not a Number)
  return somma;
}

s = somma(3); //il risultato e NaN
```

Possono essere utilizzati valori di default per i parametri per evitare questo comportamento.

Arrow function

È una sintassi compatta per scrivere funzioni.

This

La parola this è spesso utilizzata nei metodi degli oggetti e fa riferimento all'oggetto stesso che utilizza il metodo, in modo da renderlo riutilizzabile per ogni oggetto diverso

Costruttore

Funzione che permette di creare un nuovo oggetto. Viene richiamato con la parola `new`:

- viene creato un nuovo oggetto vuoto e assegnato a `this`
- viene eseguita la funzione costruttore
- viene ritornato `this`

Window

L'oggetto window rappresenta la finestra del browser. Ha i metodi alert, confirm, prompt, setTimeout, setInterval, clearTimeout

Eccezioni

Sono errori che possono verificarsi durante l'esecuzione e bloccare il flusso del codice. Possono essere gestite mediante il blocco

```
try {
  // codice che può generare errore
} catch (errore) {
  // cosa fare se c'è un errore
```

Con il comando `throw` possiamo lanciare un'eccezione personalizzata

```
function dividi(a, b) {
  if (b === 0) {
    throw new Error("Divisione per zero non permessa!");
  }
  return a / b;
}

try {
  dividi(10, 0);
} catch (e) {
  console.log(e.message); // 👉 "Divisione per zero non permessa!"
}
```

Var e Let

Lo scope di Var è il blocco funzionale più vicino, quindi è visibile all'interno della funzione in cui è dichiarato.

Lo scope di Let è il blocco di operazioni in cui si trova, all'esterno di questo non è visibile

```
function test() {
  if (true) {
    var x = 10;
    let y = 20;
  }
  console.log(x); // ✅ 10
  console.log(y); // ❌ ReferenceError: y is not defined
}
test();
```

Closure

È una funzione che ricorda le variabili dello scope esterno, anche dopo la sua terminazione. Quindi la closure racchiude dentro di se i valori delle variabili dell'ambiente in cui è stata creata.

```
function saluta(nome) {
  return function() {
    console.log("Ciao, " + nome);
  };
}

const salutoLuca = saluta("Luca");
salutoLuca(); // 👉 "Ciao, Luca"
```

`saluta("Luca")` ritorna una funzione interna che ricorda la variabile `nome`, anche se `saluta` è terminata.

Prototipi

Ogni oggetto ha una speciale proprietà detta prototype, che può essere null o contenere un riferimento ad un altro oggetto, ovvero il Prototipo, che permette all'oggetto di utilizzare i metodi e le proprietà del prototipo anche se non le possiede direttamente.

```
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
```

DOM (Document Object Model)

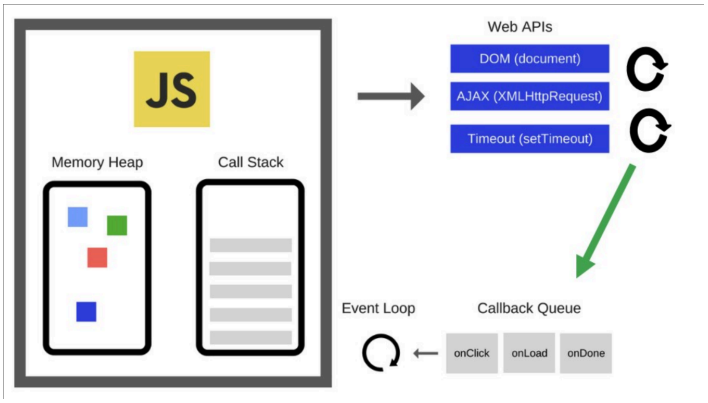
Interfaccia di programmazione per HTML, fornisce una mappa strutturata del documento html in cui ogni elemento è un nodo e la radice è document
Document ha vari metodi:
`getElementById(mioDiv)` ritorna il nodo associato al div con id `mioDiv`
`getElementByTagName(p)` ritorna una node list degli elementi `p`
`getElementByClassName(myclass)` ritorna una lista di elementi con classe `myclass`

Codice sincrono e asincrono

Il codice sincrono è eseguito aspettando la fine dell'esecuzione della linea precedente. Questo può causare blocchi per l'esecuzione di operazioni lunghe.

Il codice asincrono non blocca l'esecuzione. Alcune operazioni sono eseguite in parallelo, senza bloccare l'esecuzione del codice.

Event loop



Il motore è costituito dal:

- memory heap: in cui si alloca la memoria
- call stack: è il registro delle funzioni attive in esecuzione
- API del browser
- callback queue (macrotask queue) contiene le funzioni di callback da eseguire non appena la call stack è vuota

AJAX

È una tecnica che permette ad una pagina web di comunicare con un server in background, senza ricaricare la pagina.
Utilizza:

- un oggetto `XMLHttpRequest` (oppure `fetch`)
- Javascript
- HTML DOM per visualizzare o utilizzare i dati
  È utilizzato per:
- aggiornare delle parti della pagina
- caricare dati da un API
- inviare dati al server mediante form
- rendere le pagine interattive

XMLHttpRequest

`new XMLHttpRequest()` crea un nuovo oggetto XHR
Il metodo `open(method, url, async)` specifica il tipo di richiesta (`method` POST, GET, ecc) all'`url` specificato e se la richiesta è asincrona o meno.
`send()` invia la richiesta al server
`responseText` è la risposta DOM String
`xhr.onload` è la funzione da eseguire alla ricezione della risposta

Promise

Una promise è un oggetto che rappresenta il risultato futuro (positivo o negativo) di un'operazione asincrona.

```
const promessa = new Promise((resolve, reject) => {
  // Operazione asincrona
  const successo = true;

  if (successo) {
    resolve("Tutto ok!");
  } else {
    reject("Errore!");
  }
});
```

Fetch

Sono API basate sulle promise per richieste AJAX, sostituiscono XMLHttpRequest perché hanno una sintassi più semplice.

CORS

Acronimo di Cross Origin Resource Sharing, impedisce ad una web application con dominio X di richiedere dati ad un dominio Y tramite AJAX se non ha abilitato il CORS.

Il CORS viene implementato dal server inviando degli headers nelle risposte, come ad esempio l'header `Access-Control-Allow-Origin`

Le richieste possono essere:

- semplici, ovvero utilizza solo metodi HTTP `GET, POST, HEAD` ed header come `accept, accept-language, content-language, content-type`
- in preflight, sono richieste più avanzate in cui il browser invia prima una richiesta di `OPTIONS` per chiedere il permesso al server. Il preflight scatta nel momento in cui si fanno richieste con metodi HTTP `PUT, DELETE`

Metodi HTTP

In HTTP i metodi più comuni sono GET e POST. Il metodo GET permette di ottenere il contenuto di una risorsa indicata nell'endpoint. Il metodo POST viene utilizzato per inviare informazioni al server, come ad esempio i dati di un form.

NodeJS

È un'ambiente di esecuzione JavaScript lato server, ovvero permette di eseguire codice JS al di fuori del browser.
Utilizza un event loop per gestire le operazioni asincrone: invece di bloccare il thread principale durante operazioni di I/O (lettura di un file, invio richieste di rete) NodeJS delega queste operazioni a dei thread che le gestiscono in background. Una volta che l'operazione è completata, la callback corrispondente viene messa nella coda dei messaggi e l'event loop la esegue non appena lo stack pool è vuoto.

Microtask

Parte fondamentale dell'esecuzione asincrona in JS. Sono attività che vengono eseguite dopo che l'event loop ha completato il ciclo in corso, ma prima che riprenda ad elaborare la message queue (Macro-task). Sono utilizzate per operazioni che vanno eseguite rapidamente ed hanno una maggiore priorità rispetto alle macro-task.

Esempio di microtask: callback di `.then` e `.catch`

Quando una promise viene risolta, il suo callback è inserito nella coda delle microtask. Una volta che il call stack è vuoto, l'event loop esegue tutte le microtask in coda, per poi passare alle macrotask.

```
console.log("A");

setTimeout(() => {
  console.log("B"); // macrotask
}, 0);

Promise.resolve().then(() => {
  console.log("C"); // microtask
});

console.log("D");
```

- A e D vengono stampati in ordine in quanto operazioni sincrone
- Poi viene stampato C in quanto microtask
- infine B in quanto macrotask

Routing

Il routing determina quale azione intraprendere in risposta ad una richiesta di un URL specifico. Lato server è il meccanismo che associa gli URL alle funzioni del server. Quando un server riceve una richiesta, il router analizza la URL e decide quale codice eseguire per la risposta.

Middleware

Un middleware è una funzione che ha accesso all'oggetto richiesta `req`, all'oggetto risposta `res` e ad una funzione callback denominata `next`, utilizzata per passare il controllo al middleware successivo.

REST

Representational State Transfer è un insieme di linee guida che definisce come progettare API web. Le caratteristiche principali sono:

- separazione client e server
- stateless: ogni richiesta HTTP è indipendente, ovvero il server non ricorda nulla di richieste precedenti
- risorse accessibili mediante URL
- risorse rappresentate come dati tramite JSON o XML

API REST

Comunicano tramite richieste HTTP per eseguire operazioni CRUD (Create, Read, Update, Delete). È necessario:

- definire le risorse ed i relativi endpoint
- utilizzare i metodi HTTP standard per gli endpoint (GET, POST, PUT, DELETE)
- gestire le richieste utilizzando i codici di stato HTTP corretti
- assicurare la statelessness, ovvero ogni richiesta HTTP deve contenere tutte le informazioni necessarie per essere compresa e gestita
- implementare l'autenticazione per richieste sensibili (opzionale)