Introduzione a JavaScript

Niccolò Maltoni
niccolo.maltoni@diennea.com

Cos’è JavaScript

- JavaScript è un linguaggio di programmazione **interpretato**, **multi-paradigma** e **debolmente tipizzato**.
  - Vedremo cosa significano questi termini nel dettaglio più avanti.

- Sviluppato nel 1995 da Brendan Eich per Netscape come *linguaggio di scripting* per il web.
- Chiamato prima *Mocha* e poi *LiveScript*, viene standardizzato nel 1997 dalla ECMA come ***ECMAScript***.
- È stato rinominato **Java**Script per via della sintassi molto vicina a quella del linguaggio **Java** di Oracle.
  - Come si vedrà però, i due linguaggi sono molto diversi!

- Può essere eseguito non solo nei browser, ma anche in altri ambienti che supportano il motore JavaScript.
- Le capacità di JavaScript dipendono molto dall’ambiente in cui lo si esegue:
  - **Node.js** supporta funzioni che consentono di scrivere/leggere file, eseguire richieste web, etc.
  - Nel **browser**, JavaScript può invece manipolare la pagina, gestire le interazioni con l’utente, etc.

In questo corso ci concentreremo sull’uso di JavaScript nel contesto del web browser.

Ma cos’è il browser?

Il browser è un software che permette di:

- **Acquisire** pagine web (locali o da remoto),
- **Elaborare** le risorse caricate e **mostrarle** all’utente,
- **Navigare** tra le varie risorse.

Il browser **interpreta** il codice HTML, CSS e JavaScript e lo mostra su schermo.

Ce ne sono decine in commercio; alcuni dei più popolari sono:

Mozilla FireFox

Apple Safari

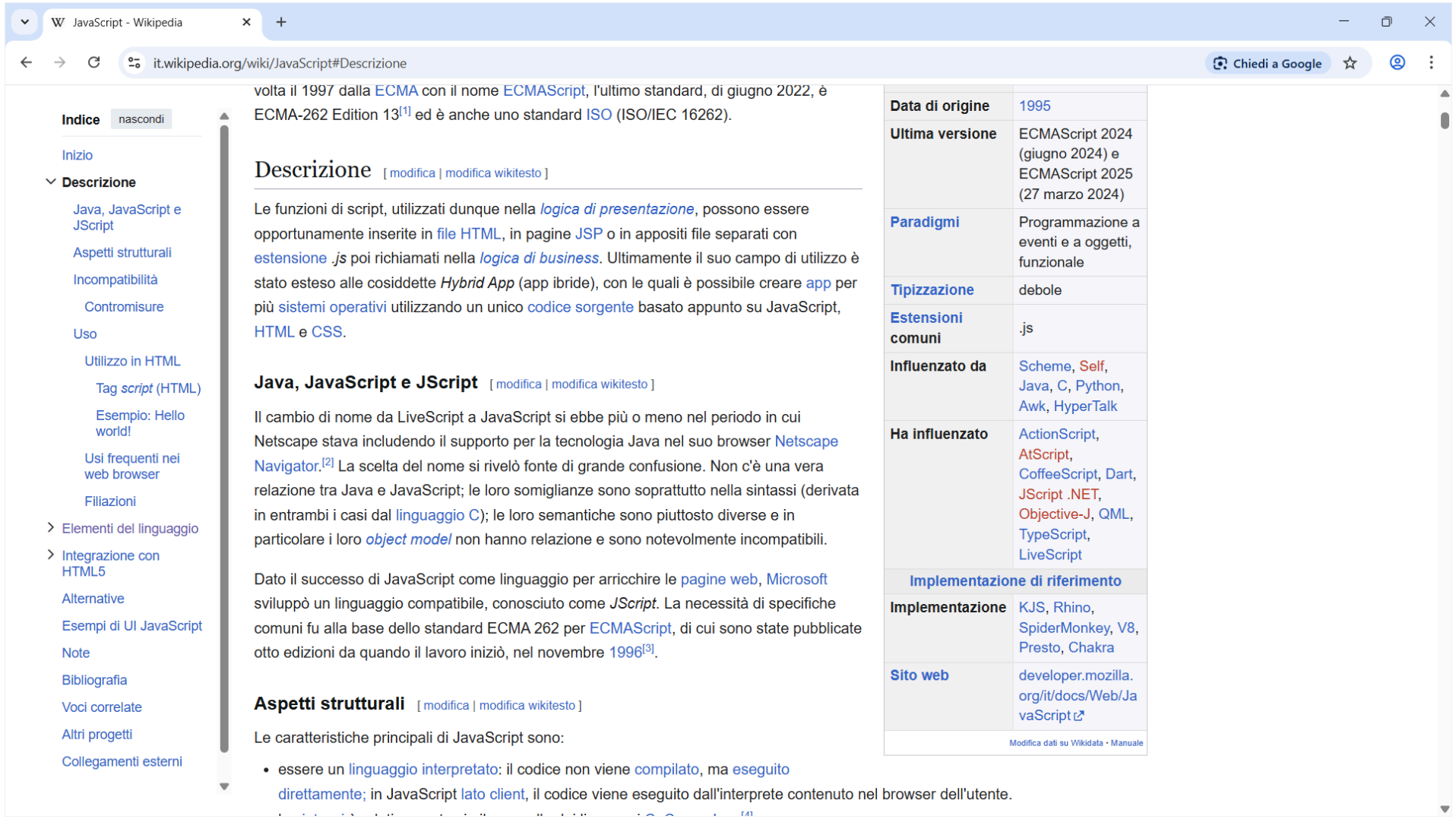
Google Chrome

Microsoft Edge

Opera

Useremo **Google Chrome** per gli esempi, ma i concetti sono validi per tutti i browser moderni.

Esempio di pagina web: Wikipedia



volta il 1997 dalla ECMA con il nome ECMAScript, l'ultimo standard, di giugno 2022, è ECMA-262 Edition 13[1] ed è anche uno standard ISO (ISO/IEC 16262).

Indice nascondi

- Inizio
- Descrizione
  - Java, JavaScript e JScript
  - Aspetti strutturali
  - Incompatibilità
    - Contromisure
  - Uso
    - Utilizzo in HTML
      - Tag *script* (HTML)
      - Esempio: Hello world!
    - Usi frequenti nei web browser
    - Filiazioni
- Elementi del linguaggio
- Integrazione con HTML5
- Alternative
- Esempi di UI JavaScript
- Note
- Bibliografia
- Voci correlate
- Altri progetti
- Collegamenti esterni

Descrizione [modifica | modifica wikitesto]

Le funzioni di script, utilizzati dunque nella *logica di presentazione*, possono essere opportunamente inserite in file HTML, in pagine JSP o in appositi file separati con estensione *.js* poi richiamati nella *logica di business*. Ultimamente il suo campo di utilizzo è stato esteso alle cosiddette *Hybrid App* (app ibride), con le quali è possibile creare app per più sistemi operativi utilizzando un unico codice sorgente basato appunto su JavaScript, HTML e CSS.

Java, JavaScript e JScript [modifica | modifica wikitesto]

Il cambio di nome da LiveScript a JavaScript si ebbe più o meno nel periodo in cui Netscape stava includendo il supporto per la tecnologia Java nel suo browser Netscape Navigator.[2] La scelta del nome si rivelò fonte di grande confusione. Non c'è una vera relazione tra Java e JavaScript; le loro somiglianze sono soprattutto nella sintassi (derivata in entrambi i casi dal linguaggio C); le loro semantiche sono piuttosto diverse e in particolare i loro *object model* non hanno relazione e sono notevolmente incompatibili.

Dato il successo di JavaScript come linguaggio per arricchire le pagine web, Microsoft sviluppò un linguaggio compatibile, conosciuto come *JScript*. La necessità di specifiche comuni fu alla base dello standard ECMA 262 per ECMAScript, di cui sono state pubblicate otto edizioni da quando il lavoro iniziò, nel novembre 1996[3].

Aspetti strutturali [modifica | modifica wikitesto]

Le caratteristiche principali di JavaScript sono:

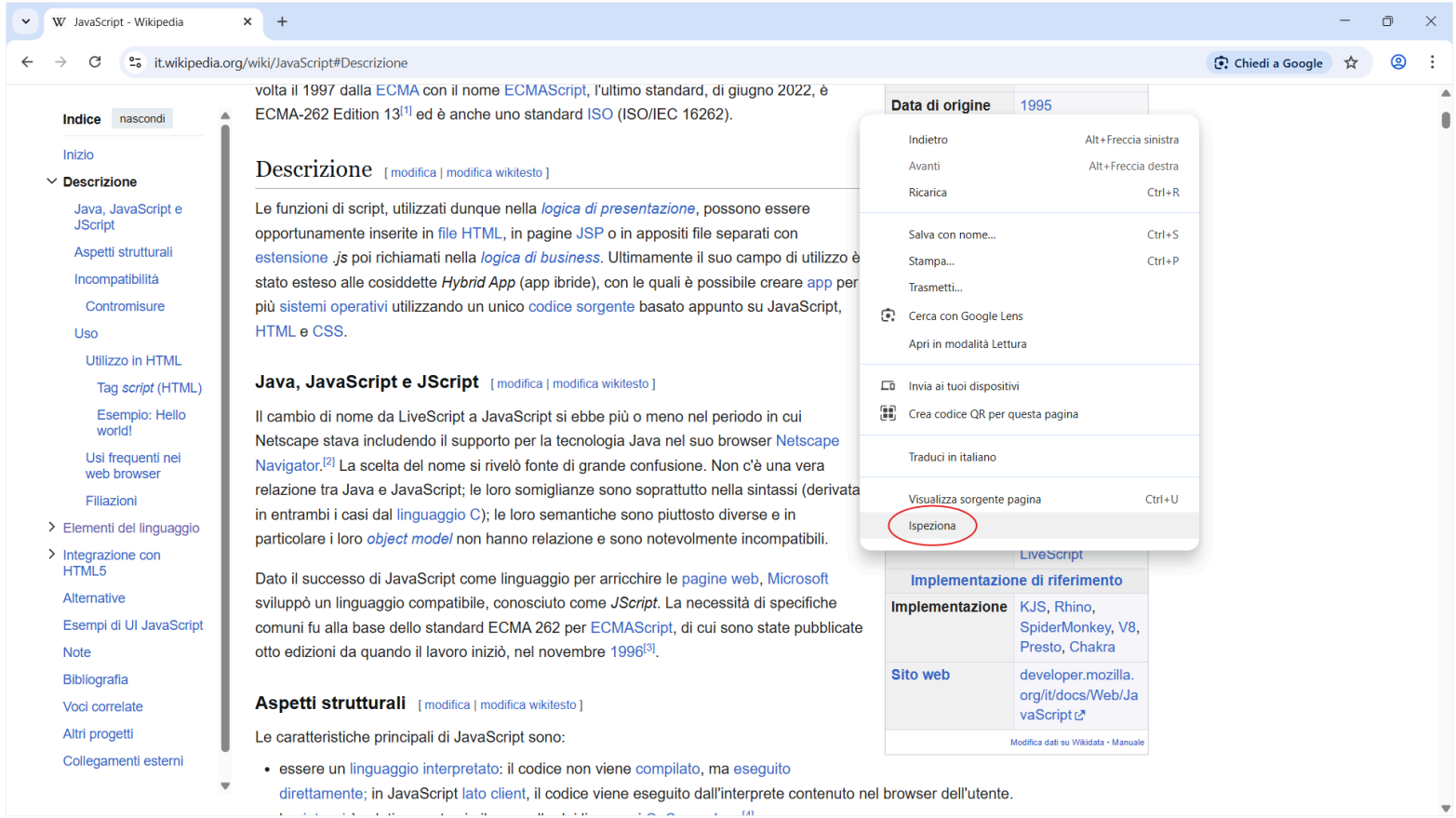
- essere un linguaggio interpretato: il codice non viene compilato, ma eseguito direttamente; in JavaScript lato client, il codice viene eseguito dall'interprete contenuto nel browser dell'utente.

Data di origine	1995
Ultima versione	ECMAScript 2024 (giugno 2024) e ECMAScript 2025 (27 marzo 2024)
Paradigmi	Programmazione a eventi e a oggetti, funzionale
Tipizzazione	debole
Estensioni comuni	.js
Influenzato da	Scheme, Self, Java, C, Python, Awk, HyperTalk
Ha influenzato	ActionScript, AtScript, CoffeeScript, Dart, JScript .NET, Objective-J, QML, TypeScript, LiveScript
Implementazione di riferimento	
Implementazione	KJS, Rhino, SpiderMonkey, V8, Presto, Chakra
Sito web	developer.mozilla.org/it/docs/Web/JavaScript

Modifica dati su Wikidata · Manuale

Ispezionare la pagina

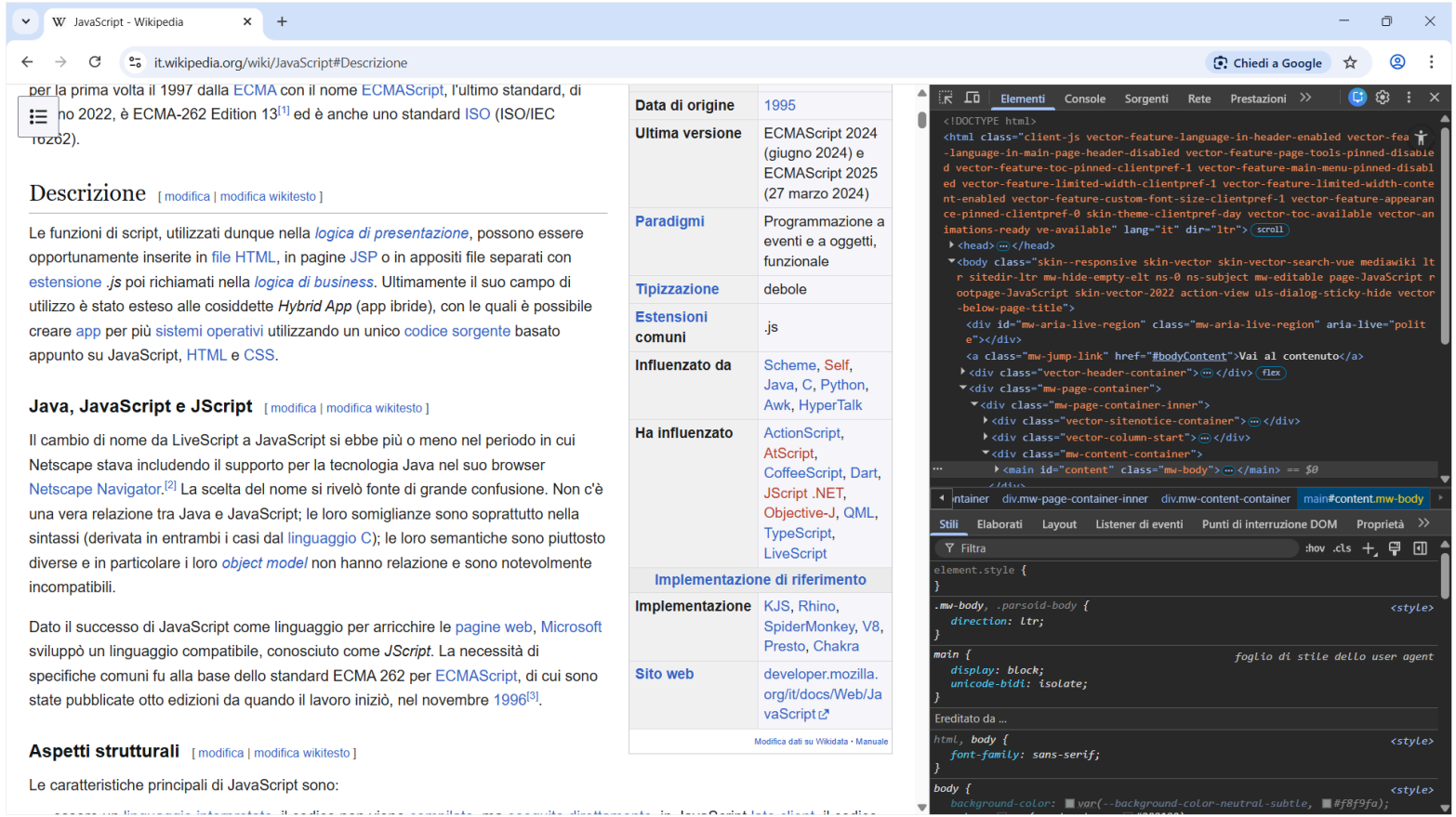
Possiamo ispezionare la pagina web tramite tasto destro su qualsiasi parte della pagina → “Ispeziona”:



Possiamo anche usare la scorciatoia `Ctrl-Shift-I` / `Opt-Cmd-I`.

Ispezionare la pagina: strumenti di sviluppo

In questo modo, possiamo aprire gli strumenti di sviluppo del browser (**DevTools**):



In particolare, i DevTools saranno aperti nello strumento di ispezione.

DevTools

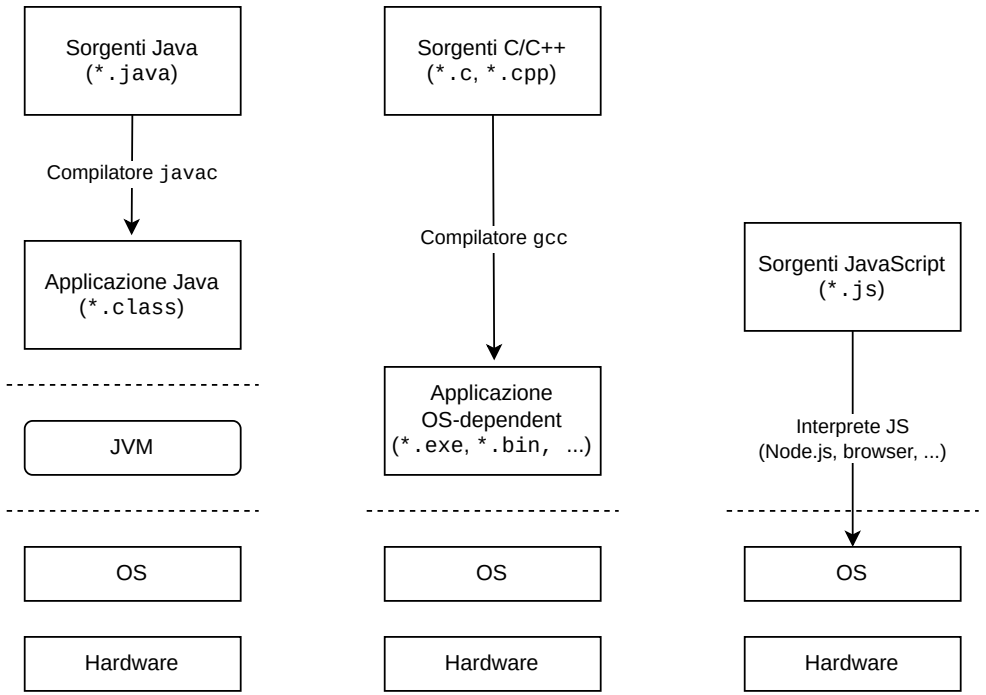
È un insieme di strumenti messi a disposizione dal browser per il debug delle pagine web; ad esempio:

- **Ispezionare** il codice HTML e CSS e **modificarlo** in tempo reale;
- Allacciare un **debugger** al codice JavaScript eseguito dalla pagina;
- Analizzare le **performance** della pagina;
- Monitorare le richieste e le risposte HTTP e tutte le **attività di rete**;
- Aprire una console JavaScript interattiva;
- Emulare il comportamento del sito in **ambienti mobili**.

Vediamo subito la **console**.

Esecuzione di codice

JavaScript è un linguaggio **interpretato**.



- **Compilato**: il sorgente viene tradotto prima in un eseguibile (es. C, Go, Java).
- **Interpretato**: il sorgente è letto ed eseguito "al volo" da un interprete/VM (es. JavaScript, Python).

JavaScript nel browser è **interpretato** dal motore (ad esempio V8 nei browser Chromium).

Lo strumento console

È un **R**ead-**E**val-**P**rint **L**oop (REPL): un ambiente interattivo che legge il codice JavaScript immesso dall'utente, lo valuta nel contesto della pagina caricata, stampa il risultato e torna pronto per un nuovo input

- Utile per stampare **log**, segnalare warning ed errori, controllare variabili e oggetti.
- Esegue il codice **nel contesto della pagina**: possiamo leggere dati e manipolare la pagina.

Sintassi

Un programma JavaScript si compone di **istruzioni**:

```
console.log('Ciao!'); // Stampa in console

/* Anche senza punto e virgola, l'istruzione termina correttamente. */
console.log('Ciao!')
```

Le istruzioni possono essere terminate o meno dal punto e virgola (`;`). Alcune style guide[1] consigliano di **non** usarlo, mentre la maggior parte[2] [3] ne raccomandano l'uso.

```
// Questo è un commento su una riga

/*
  Questo è un commento
  su più righe
*/
```

I **commenti**, resi con `//` o con `/* ... */`, sono righe ignorate dall'interprete, utili per documentare.

1. https://standardjs.com
2. https://google.github.io/styleguide/jsguide.html
3. https://javascript.airbnb.tech

Variabili e costanti

La maggior parte delle volte, le applicazioni JavaScript necessitano di lavorare con informazioni. Una **variabile** è uno spazio di memoria con un nome utilizzato per salvare questi dati.

Si usa `let` per variabili che possono essere riassegnate:

```
let nome = 'Mario';
nome = 'Luigi';
```

Si usa `const` per variabili che possono essere assegnate una volta soltanto:

```
const PI = 3.14;
PI = 3.15; // Errore! Non si può cambiare
```

`let` e `const` sono stati introdotti con ES6; in vecchi script è possibile trovare `var` per dichiarare variabili. La dichiarazione `var` è molto simile a `let` e la maggior parte delle volte sono intercambiabili.

In realtà, internamente sono molto diversi e l'uso di `var` può esporre a problemi che vedremo più avanti. Per questo motivo, nonostante sia pienamente supportato è generalmente sconsigliato in progetti moderni.

Tipi di dato

Un valore in JavaScript ha sempre un tipo specifico.

JavaScript ha diversi **tipi di dato** primitivi:

```
let eta = 25;           // Number (numeri interi e decimali)
let nome = 'Anna';      // String (testo tra apici)
let attivo = true;      // Boolean (true o false)
let vuoto = null;       // Null (valore intenzionalmente vuoto)
let indefinito;         // Undefined (valore non assegnato)
```

Possiamo verificare il tipo con `typeof`:

```
console.log(typeof eta);    // "number"
console.log(typeof nome);   // "string"
console.log(typeof attivo); // "boolean"
```

Una variabile in JavaScript può contenere qualsiasi dato; in particolare, può essere **definita** con un valore di un certo tipo, e poi **assegnata** con un valore di un altro tipo.

Per questo motivo, si dice che JavaScript è un linguaggio **debolmente tipato**.

Operatori aritmetici

Per fare calcoli usiamo gli operatori aritmetici:

```
let a = 10;
let b = 3;

console.log(a + b);  // 13 (addizione)
console.log(a - b);  // 7  (sottrazione)
console.log(a * b);  // 30 (moltiplicazione)
console.log(a / b);  // 3.333... (divisione)
console.log(a % b);  // 1  (resto della divisione)
```

Possiamo anche usare operatori di **assegnazione composta**:

```
let x = 5;
x += 3;  // equivale a: x = x + 3  → x diventa 8
x *= 2;  // equivale a: x = x * 2  → x diventa 16
```

Operatori di confronto

Per confrontare valori usiamo operatori che restituiscono `true` o `false`:

```
let a = 10;
let b = 5;

console.log(a === b);  // false (uguaglianza stretta)
console.log(a !== b);  // true  (disuguaglianza stretta)
console.log(a > b);    // true  (maggiore)
console.log(a < b);    // false (minore)
console.log(a >= 10);  // true  (maggiore o uguale)
```

Attenzione: usa sempre `===` invece di `==` (uguaglianza debole che può dare risultati inattesi).

```
console.log(5 === '5');  // false (tipi diversi)
console.log(5 == '5');   // true  (conversione automatica, evitare!)
```

Gli operatori di confronto producono sempre un valore booleano: `true` o `false`.

```
const eta = 17;
const maggiorenne = eta >= 18;

console.log(maggiorenne);  // false

if (maggiorenne) {
  console.log('Accesso consentito');
}
```

Operatori logici: &&, ||, !

Per combinare più condizioni:

&& (**AND**): entrambe true

```
let eta = 20;
let patente = true;

if (eta >= 18 && patente) {
  console.log('Puoi guidare');
}
```

|| (**OR**): almeno una true

```
let pioggia = false;
let neve = true;

if (pioggia || neve) {
  console.log('Porta l\'ombrello o la sciarpa');
}
```

! (**NOT**): inverte il risultato

```
let caldo = false;

if (!caldo) {
  console.log('Non è caldo');
}
```

Stampare in Console: `console.log()` e `alert()`

Per vedere i risultati usiamo:

`console.log()`: stampa nella Console dei DevTools.

```
console.log('Messaggio semplice');
console.log('Il valore è:', 42);

let nome = 'Mario';
console.log('Nome:', nome);
```

`alert()`: mostra una finestra popup nel browser. **Nota**: blocca l'esecuzione fino alla chiusura del popup.

```
alert('Ciao dal browser!');

let eta = 25;
alert('Hai ' + eta + ' anni');
```

Queste funzioni sono utili per test rapidi e debug, ma non sono adatte per applicazioni reali (i DevTools integrano strumenti di debug più avanzati).

Esercizio

Proviamo le basi

Apri la console del browser e prova:

1. Crea una variabile `nome` con il tuo nome e stampala con `console.log()`.
2. Crea due variabili numeriche e stampa la loro somma.
3. Confronta due numeri con `===` e stampa il risultato.
4. Mostra un messaggio con `alert()`.

Esercizio: troviamo il bug

Debug rapido

1. Apriamo questa pagina con il browser e apriamo i DevTools.
2. Impostiamo un breakpoint sulla riga indicata.
3. Osserviamo i valori e i **tipi** delle variabili.
4. Individuiamo il motivo del risultato finale.

```
<!DOCTYPE html>
<html lang="it">
  <head>
    <meta charset="UTF-8">
    <title>Debug rapido</title>
  </head>
  <body>
    <script>
      const base = 10;
      const incremento = '5';
      const moltiplicatore = 2;

      const risultato1 = base + incremento * moltiplicatore;
      const risultato2 = base + incremento + moltiplicatore;

      console.log('Risultato 1: ', risultato1);
      console.log('Risultato 2: ', risultato2);
    </script>
  </body>
</html>
```

Decisioni: `if` e `else`

A volte vogliamo eseguire codice **solo se una condizione è vera**.

```
let eta = 18;

if (eta >= 18) {
  console.log('Sei maggiorenne');
}
```

L'istruzione tra le parentesi graffe si esegue **solo se la condizione è true**.

Possiamo anche eseguire codice **alternativo** con `else if` (per ulteriori condizioni) e `else`:

```
let voto = 6;

if (voto >= 8) {
  console.log('Ottimo!');
} else if (voto >= 6) {
  console.log('Bravo!');
} else {
  console.log('Studia di più');
}
```

Truthy e falsy

In JavaScript, dentro una condizione `if (...)`, i valori vengono convertiti in booleano.

Valori **falsy** principali:

- `false`
- `0`
- `''` (stringa vuota)
- `null`
- `undefined`
- `NaN`

Valori **truthy**: praticamente tutti gli altri

```
if ('ciao') {
  console.log('Stringa non vuota: truthy');
}

if (0) {
  console.log('Non entra mai: 0 e falsy');
}
```

Double bang (`!!`)

Per convertire un valore in booleano, si può usare l'operatore di negazione `!` due volte:

```
const valore = 0;
console.log(!!valore);  // false
```

```
const valore = 'ciao';
console.log(!!valore);  // true
```

Operatore ternario: `condizione ? A : B`

Il ternario è una forma compatta di `if/else`, utile quando dobbiamo scegliere tra due valori.

```
const voto = 7;

const esito = voto >= 6 ? 'Promosso' : 'Bocciato';
console.log(esito);  // 'Promosso'
```

Usiamolo per condizioni semplici. Se la logica è lunga, `if/else` resta più leggibile.

Decisioni multiple: `switch`

Quando dobbiamo gestire molti casi su uno stesso valore, `switch` è spesso più leggibile di una catena `if/else if`.

```
const voto = 8;

switch (voto) {
  case 10:
    console.log('Eccellente');
    break;
  case 8:
    console.log('Molto bene');
    break;
  case 6:
    console.log('Sufficiente');
    break;
  default:
    console.log('Valore non gestito');
}
```

`break` evita di continuare nei casi successivi. `default` gestisce il caso non previsto.

Esercizio

Guess the output

Apriamo la console e, per ogni snippet, seguiamo questo flusso:

1. prediciamo l'output
2. eseguiamo in console
3. confrontiamo il risultato

- `5 === '5'`
- `5 == '5'`
- `Boolean('0')`
- `Boolean(0)`
- `Boolean('')`
- `Boolean([])`
- `17 >= 18 ? 'OK' : 'NO'`
- `6 >= 6 ? 'Promosso' : 'Bocciato'`
- `switch (6) { case 6: console.log('Sufficiente'); break; default: console.log('Altro'); }`
- `switch (9) { case 10: console.log('Eccellente'); break; case 9: console.log('Ottimo'); break; default: console.log('Altro'); }`
- `Boolean({})`
- `Boolean(null)`
- `Boolean(undefined)`

Esercizio

Correggi il caso

Questo snippet non si comporta come previsto. Correggiamolo.

```
const ruolo = 'admin';

switch (ruolo) {
  case 'admin':
    console.log('Accesso admin');
  case 'user':
    console.log('Accesso utente');
  default:
    console.log('Accesso guest');
}
```

Obiettivo: con `ruolo = 'admin'` deve stampare solo `Accesso admin`.

Cicli: `for`

Spesso vogliamo ripetere un'azione **più volte**.

La forma più comune è il ciclo `for`:

```
for (let i = 0; i < 5; i++) {
  console.log('Numero:', i);
}
// Stampa: 0, 1, 2, 3, 4
```

Il ciclo `for` ha tre parti:

- **`let i = 0`**: inizializza il contatore a 0
- **`i < 5`**: continua finché i è minore di 5
- **`i++`**: incrementa i di 1 dopo ogni ripetizione (equivalente a `i = i + 1`)

Cicli: `while`

Possiamo ripetere finché una condizione è vera con `while`:

```
let numero = 1;

while (numero <= 5) {
  console.log('Numero:', numero);
  numero = numero + 1;  // Incrementa manualmente
}
// Stampa: 1, 2, 3, 4, 5
```

Attenzione: se la condizione non diventa mai falsa, il ciclo **non termina mai** (infinito!).

```
let x = 1;
while (x > 0) {
  console.log('Infinito!');
  // x non cambia mai... ciclo infinito!
}
```

Generalmente `for` è più sicuro per cicli controllati, `while` per cicli “finché accade una cosa”.

Break: uscire da un ciclo

Possiamo uscire da un ciclo anticipatamente con `break`:

```
for (let i = 0; i < 10; i++) {
  if (i === 5) {
    break;  // Esce dal ciclo quando i = 5
  }
  console.log(i);
}
// Stampa: 0, 1, 2, 3, 4
```

Utile quando cerchiamo qualcosa:

```
let numeri = [2, 4, 7, 10];
let trovato = false;

for (let i = 0; i < numeri.length; i++) {
  if (numeri[i] === 7) {
    console.log('Trovato 7 alla posizione', i);
    trovato = true;
    break;
  }
}
```

I tre pilastri dello sviluppo web

- **HTML (Hyper Text Markup Language):** definisce la **struttura** della pagina web.
- **CSS (Cascading Style Sheets):** definisce l'**aspetto visivo** e la presentazione (colori, dimensioni, animazioni).
- **JavaScript:** aggiunge l'**interattività** e il **comportamento** alla pagina.

Ci sono altri "first-class citizens" del web, come ad esempio **WebAssembly**, che non approfondiremo.

HTML

- Non è un linguaggio di programmazione, ma un **linguaggio di markup**.
- La sua funzione è descrivere la **struttura di base** e il **significato** degli elementi.
- È composto da **tag** (come `<h1>`, `<p>`, `<table>`) che istruiscono il browser sul tipo di contenuto da mostrare.

CSS

- La sua funzione è descrivere l'**aspetto visivo** (presentazione) degli elementi.
- È composto da regole che istruiscono il browser su come mostrare ciascun contenuto.
- **CSS è opzionale**: una pagina senza CSS è funzionale al 100%, ma risulterà un semplice insieme di blocchi.

JavaScript

- È un **linguaggio di scripting** utilizzato per aggiungere **interattività** alla pagina.
- Viene eseguito lato **client**, ovvero dal browser dell'utente.
- Permette di manipolare la pagina tramite funzioni eseguite quando si verificano **determinati eventi**.

Struttura di un documento HTML

```
1  <!DOCTYPE html>                                  ①
2  <html lang="it">                                 ②
3    <head>                                         ③
4      <title>Titolo della pagina</title>
5    </head>
6    <body>                                         ④
7      <h1>Questo è un Titolo</h1>
8      <p>Questo è un paragrafo.</p>
9    </body>
10 </html>
```

① **`<!DOCTYPE html>`**: dichiara che il file è HTML5.

② **`<html>`**: la radice del documento.

③ **`<head>`**: contiene i metadati (titolo, link ai CSS). **Non è visibile** sulla pagina.

④ **`<body>`**: contiene tutto ciò che l'utente **vede** (testi, immagini, video).

Struttura di un documento HTML: DOCTYPE

<!DOCTYPE> rappresenta il tipo di documento. Aiuta il browser a capire come interpretare il codice della pagina.

Viene inserito una sola volta (all'inizio della pagina) e può indicare diversi tipi di documento:

- HTML5:

```
<!DOCTYPE html>
```

- HTML4 STRICT:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
```

- XHTML 1.1:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

Lavoreremo con HTML5.

Sintassi HTML: Elementi

Un **elemento** è definito da un **tag di inizio** e un **tag di fine**. Il contenuto di questi tag rappresenta il singolo elemento.

```
<h1>Questo è un Titolo</h1>
```

Nota: Gli elementi devono sempre essere **chiusi** tramite tag di fine.

Tutti gli elementi possono essere **nidificati** uno dentro l'altro:

```
<body>
  <h1>Questo è un Titolo</h1>
  <p>Questo è un paragrafo.</p>
</body>
```

Esistono anche alcuni elementi particolari, chiamati **elementi vuoti**. Questi elementi *non* devono essere chiusi.

```
<p>This is a <br> paragraph with a line break.</p>
```

Alcuni esempi notevoli sono l'interruzione di riga `<br>` e l'immagine `<img>`.

Sintassi HTML: Attributi

Tutti gli elementi HTML possono avere degli **attributi**.

Gli attributi forniscono informazioni **addizionali** agli elementi e sono specificati sempre nel **tag di inizio**.

Gli attributi sono nella forma `nome="valore"`:

```
<html lang="en">

<a href="https://www.w3schools.com">Visit W3Schools</a>

<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">
```

Elementi fondamentali della pagina

1. Abbiamo già detto che `<!DOCTYPE>` deve essere sempre presente all'inizio del documento ed è obbligatorio.
2. Subito dopo troviamo il tag radice, con cui iniziano e finiscono tutti i file HTML: `<HTML>`.
    - L'attributo `lang` indica la lingua della pagina ed è fortemente consigliato.
3. Segue il tag `<HEAD>`:
    - Non viene renderizzato dai browser;
    - Contiene tutte le informazioni necessarie per visualizzare correttamente la pagina, detti **metadati**.
4. Infine, il tag `<BODY>`, che contiene tutto quello che viene visualizzato dal browser.

Vediamo di seguito alcuni elementi notevoli della pagina.

Elementi notevoli: titoli e paragrafi

Gli elementi **header** (`<h1>` … `<h6>`) danno una struttura al contenuto; **`<h1>`** dovrebbe essere il titolo principale, unico per pagina.

L'elemento **`<p>`** è usato per paragrafi di testo.

Per dare enfasi o significato specifico:

- **`<strong>`** o **`<b>`**: testo **grassetto**, indica importanza forte.
- **`<em>`** o **`<i>`**: testo *corsivo*, indica enfasi.
- **`<mark>`**: evidenzia il testo (come un evidenziatore giallo).
- **`<code>`**: frammento di codice (monospazio).

```
<h1>Titolo principale</h1>

<p>
  Questo è <strong>importante</strong>
  e questo <em>enfatizzato</em>.
</p>
```

Elementi notevoli: liste, link e immagini

Liste raggruppano elementi correlati:

- `<ul>` + `<li>`: lista non ordinata (pallini).
- `<ol>` + `<li>`: lista ordinata (numeri).

Link (elemento `<a>`) collegano pagine:

```
<a href="https://example.com" target="_blank">Vai a Example</a>
```

Immagini (`<img>`): `src` (percorso) e `alt` (testo alternativo) sono obbligatori.

```
<img src="logo.png" alt="Il nostro logo" width="200" height="150">
```

Altri elementi e attributi HTML

HTML5 è un *living standard*, ovvero viene continuamente aggiornato dal WHATWG (*Web Hypertext Application Technology Working Group*) senza versioni fisse, bensì attraverso revisioni incrementali per adattarsi alle evoluzioni del web.

Di conseguenza, negli anni sono stati sia aggiunti che deprecati numerosi elementi HTML, a seconda delle esigenze che l'ecosistema richiede. Al momento, HTML5 conta più di 100 elementi supportati.

Questo modulo vedrà largo uso di HTML, ma non è incentrato su di esso, e non avrebbe senso approfondirli tutti. Qualora fosse necessario, potremo fare riferimento a queste risorse:

MDN Web Docs

WHATWG HTML spec

W3Schools

L’editor di testo

Per scrivere codice si utilizza un editor di testo; ne esistono di semplici, come il blocco note, e di più avanzati.

I vantaggi di **IDE** (*ambienti di sviluppo integrato*) ed editor di testo avanzati rispetto agli editor di testo semplici sono:

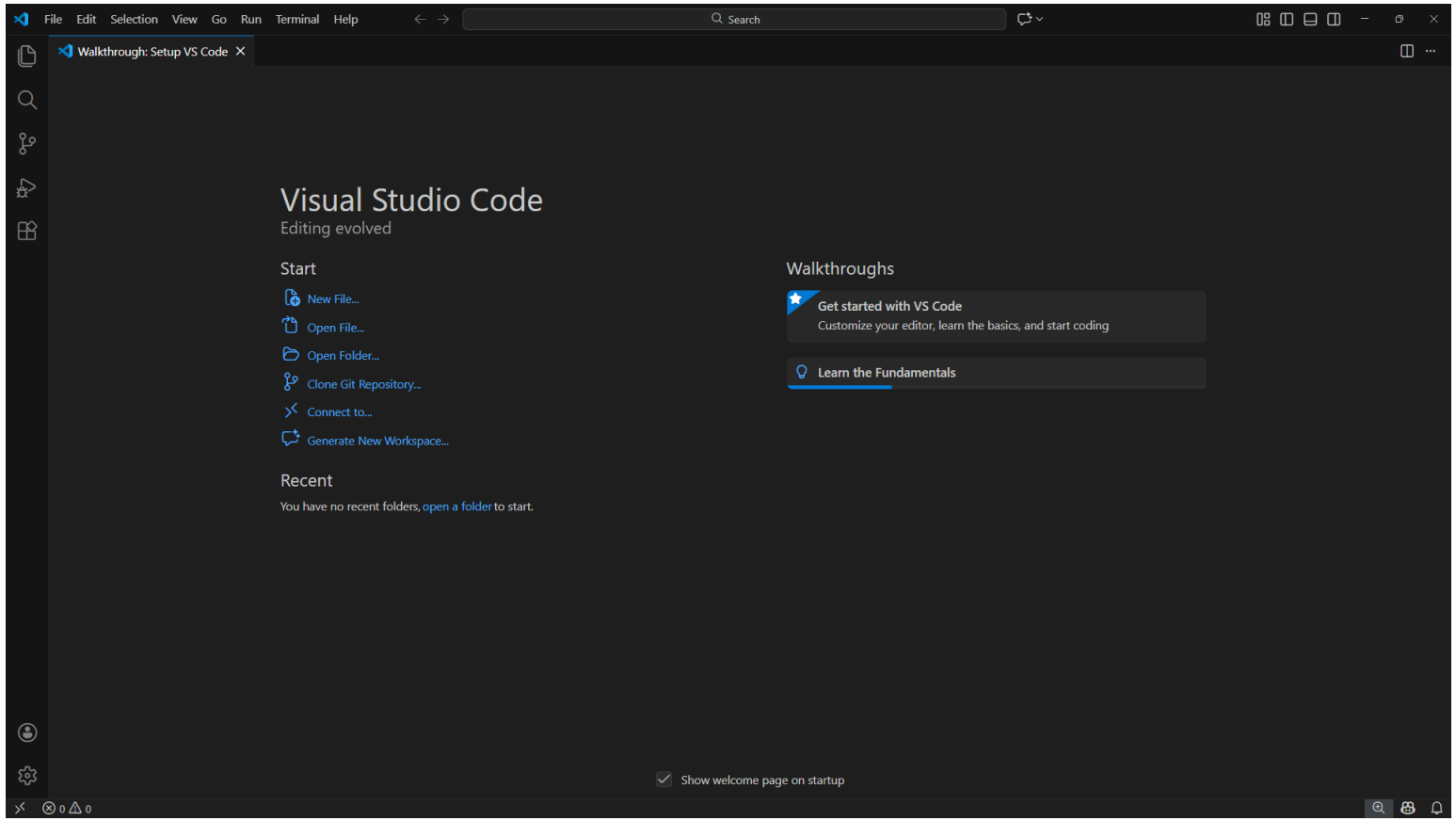
- Evidenziazione della sintassi
- Completamento automatico
- Integrazione con strumenti di sviluppo (linting, debug, etc.)
- Integrazione con il controllo del codice sorgente (`git`) e i servizi di hosting (e.g. GitHub)
- Terminale integrato per eseguire comandi direttamente dall’editor

Noi useremo **Microsoft Visual Studio Code**, uno degli editor più usati nel contesto dello sviluppo web.

Lo possiamo scaricare e installare direttamente dal sito ufficiale: https://code.visualstudio.com/

Visual Studio Code (VSCode)

- Visual Studio Code è un editor di codice sorgente leggero, versatile e multipiattaforma.
- Estendibile attraverso un ecosistema di estensioni per vari linguaggi di programmazione e strumenti.



Utilizzo di VSCode: shortcut

Alcune scorciatoie da tastiera:

- `Ctrl-Shift-P`: command palette (*l'unico shortcut che veramente serve ricordare*)
- `Ctrl-Space`: intellisense (autocompletamento)
- `Ctrl-S`: salva file corrente
- `Ctrl-PgUp` / `Ctrl-PgDn`: tab precedente/successivo
- `Ctrl-W`: chiudi la tab corrente
- `Ctrl-F`: cerca nel file corrente
- `Ctrl-H`: sostituisci nel file corrente
- `Ctrl-Shift-F`: cerca in tutti i file del workspace
- `Ctrl-Shift-H`: sostituisci in tutti i file del workspace

Esercizio: una prima pagina HTML

Il ricettario

L'obiettivo è realizzare una semplice pagina web che elenca una ricetta.

La pagina deve comporsi di un **titolo** con il nome del piatto, una breve **descrizione**, una lista di **ingredienti** e una lista di **passaggi** per la preparazione.

Step suggeriti:

1. Creiamo una cartella per l'esercizio
2. Con VS Code, apriamo come workspace la cartella creata
3. Creiamo un file e salviamolo come `index.html`; sarà vuoto
4. Aggiungiamo la struttura base (`html`, `head`, `body`)
5. Inseriamo un `<h1>` con il nome di un piatto.
6. Aggiungiamo un paragrafo `<p>` con una breve descrizione.
7. Aggiungiamo una lista puntata `<ul>` per gli ingredienti.
8. Concludiamo inserendo una lista numerata `<ol>` per i passaggi della preparazione.

Esercizio: Risultato atteso

Il ricettario

Risultato atteso:

Spaghetti alla Carbonara

Un classico della cucina romana, amato in tutto il mondo per la sua cremosità e il sapore deciso.

Ingredienti

- 320g di spaghetti
- 150g di guanciale
- 6 tuorli d'uovo
- 50g di pecorino romano
- Pepe nero q.b.

Preparazione

1. Mettere a bollire l'acqua per la pasta.
2. Tagliare il guanciale a listarelle e rosolarlo in padella finché non diventa croccante.
3. In una ciotola, sbattere i tuorli con il pecorino e abbondante pepe nero.
4. Scolare la pasta al dente, conservando un po' di acqua di cottura.
5. Versare la pasta nella padella col guanciale (fuori dal fuoco) e aggiungere la crema di uova.
6. Mescolare velocemente aggiungendo acqua di cottura se necessario per creare una crema liscia.

Inserire JavaScript nella pagina

Per utilizzare JavaScript in una pagina HTML, si usa il tag **`<script>`**.

Possiamo puntare un riferimento a un file `.js` esterno, utilizzando l'attributo `src`:

```
<body>
  <h1>Ciao da HTML!</h1>

  <script src="script.js"></script>
</body>
```

Oppure, possiamo inserirlo inline:

```
<script>
  console.log('Codice JS direttamente nell\'HTML');
</script>
```

Generalmente, è consigliabile includere file esterni, principalmente per ragioni di manutenibilità e automazione.

Attenzione: la posizione in cui viene inserito il blocco è rilevante! Ad esempio, è necessario inserirlo **alla fine del `body`** per permettere il caricamento della pagina prima dell'esecuzione.

Esercizio: JS in una pagina HTML

Una prima pagina con JavaScript

1. Crea una cartella `primo-progetto`
2. Apri la cartella in VSCode (File → Open Folder)
3. Crea un file `index.html` con questa struttura:

```html
<!DOCTYPE html>
<html lang="it">
  <head>
    <meta charset="UTF-8">
    <title>Prima Pagina JS</title>
  </head>
  <body>
    <h1>Benvenuto!</h1>
    <p>Questa è la mia prima pagina con JavaScript.</p>

    <script src="script.js"></script>
  </body>
</html>
```

4. Crea un file `script.js` nella stessa cartella; esso deve essere un breve programma che:
   - definisce una variabile `nome` (stringa) e una variabile `eta` (numero)
   - calcola un anno di nascita usando l'anno corrente e `eta`
   - calcola il resto della divisione di `eta` per 3
   - verifica se l'utente è maggiorenne con un confronto (`>= 18`)
   - stampa in console tutte le informazioni calcolate
   - mostra un `alert` con un saluto personalizzato che usa `nome`