



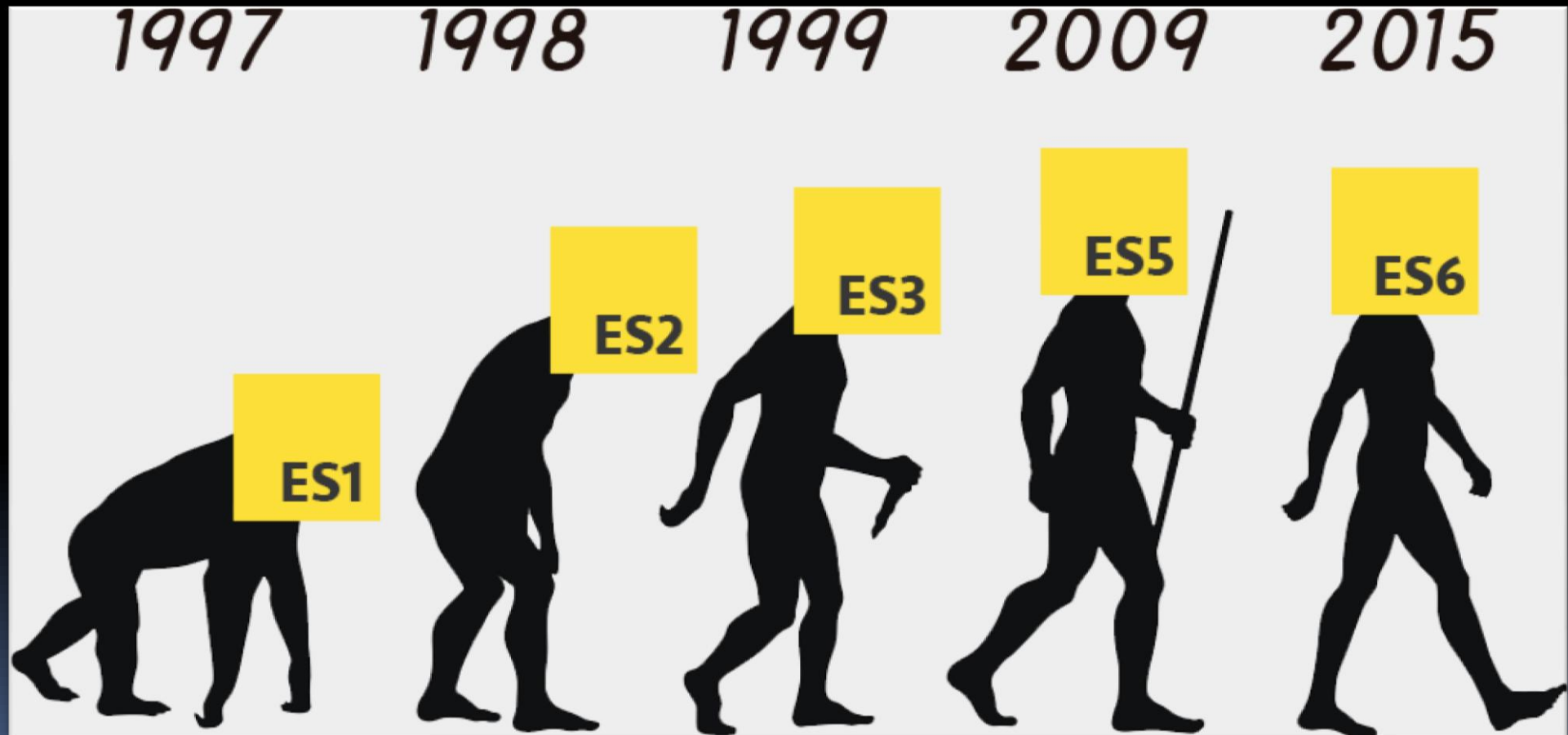
# JAVASCRIPT / TYPESCRIPT



# JavaScript

- Linguaggio di programmazione
  - Appartenente alla famiglia dei linguaggi C/Java
  - Interpretato (o compilato in modalità just-in-time)
  - Nato come linguaggio di scripting
    - Soprattutto per l'aggiunta di dinamicità nelle pagine web
    - Ma oggi utilizzabile in diversi ambienti non-browser
      - Es. NodeJS, Apache CouchDB, Adobe Acrobat
- Prototype-based
- Multi-paradigm
- Single threaded
- Dinamico
- Object oriented
- Debolmente tipizzato
- Imperativo e dichiarativo
- Esteso in numerose librerie e versioni che ne semplificano e potenziano l'utilizzo
  - Es. JQuery



# JavaScript





# JavaScript - ECMAScript - TypeScript

- TypeScript
  - Basato sulle convenzioni di JavaScript
    - Compilato
    - Tipizzazione statica, classi e interfacce
- ECMAScript (ES)
  - Linguaggio di script standardizzato da ECMA International con lo scopo di rendere il JavaScript alla pari degli altri linguaggi di programmazione
  - ECMAScript6 (ES6) è lo sforzo più significativo per consentire la realizzazione di applicazioni complicate in JavaScript
    - Oggi la maggior parte dei browser interpreta correttamente ES6
- È possibile convertire i programmi scritti in uno di questi standard attraverso specifici programmi chiamati transpilers

# JavaScript – ECMAScript – TypeScript

 	
JAVASCRIPT	TYPESCRIPT
<ul style="list-style-type: none"><li>1 It does not support optional parameters.</li><li>2 It is interpreted language that is why it highlights the errors at runtime.</li><li>3 JavaScript does not support modules.</li><li>4 In this, number, string are the objects.</li><li>5 JavaScript does not support generics.</li></ul>	<ul style="list-style-type: none"><li>1 It supports optional parameters.</li><li>2 It compiles the code and highlights errors during the development time.</li><li>3 TypeScript gives support for modules.</li><li>4 In this, number, string are the interfaces.</li><li>5 TypeScript supports generics.</li></ul>
<a href="http://www.cynoteck.com">www.cynoteck.com</a>	

 	
ECMASCRIPT	TYPESCRIPT
<ul style="list-style-type: none"><li>1 ES6 does not support all data types.</li><li>2 ES6 does not support these features.</li><li>3 ES6 has two scopes: Global Scope Local Scope</li><li>4 Typescript and ES6 both are having same loops: Definite Indefinite</li><li>5 Typescript and ES6 both are having same loops: Definite Indefinite</li></ul>	<ul style="list-style-type: none"><li>1 All primitive data types are supported by TypeScript.</li><li>2 Generics and type annotations, as well as Inference, Enums, and Interfaces, are all included in TypeScript.</li><li>3 Typescript has three scopes: Global Scope Class Scope Local Scope</li><li>4 In this, number, string are the interfaces.</li><li>5 TypeScript Modules are of two types: Internal External modules</li></ul>
<a href="http://www.cynoteck.com">www.cynoteck.com</a>	



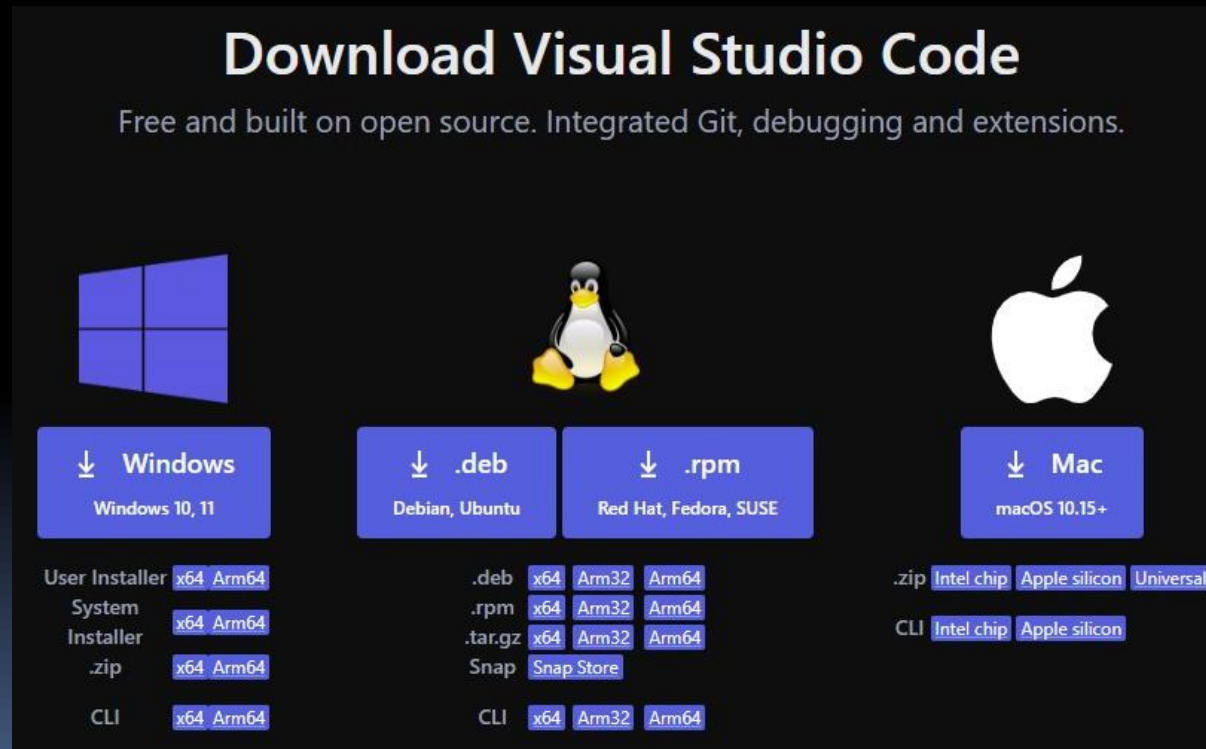
# IDE

- Integrated Development Environment
  - Applicazione che consente di gestire le diverse fasi di sviluppo senza la necessità di ulteriori tools esterni ad essa
  - In genere prevede la possibilità di installare appositi plugin che configurano operazioni complesse
- Esistono diversi IDE consigliabili per la programmazione in JavaScript/React
  - Useremo Visual Studio Code
    - Un IDE snello, altamente configurabile, multiplatforma

# VSCode

## ■ Installazione

- <https://code.visualstudio.com/Download>



The screenshot shows the Visual Studio Code download page. At the top, it says "Download Visual Studio Code" and "Free and built on open source. Integrated Git, debugging and extensions." Below this, there are three main sections for different operating systems: Windows, Linux, and Mac. Each section has a download button and a list of available download options.

**Download Visual Studio Code**  
Free and built on open source. Integrated Git, debugging and extensions.

**Windows**  
Windows 10, 11

User Installer: x64, Arm64  
System Installer: x64, Arm64  
.zip: x64, Arm64  
CLI: x64, Arm64

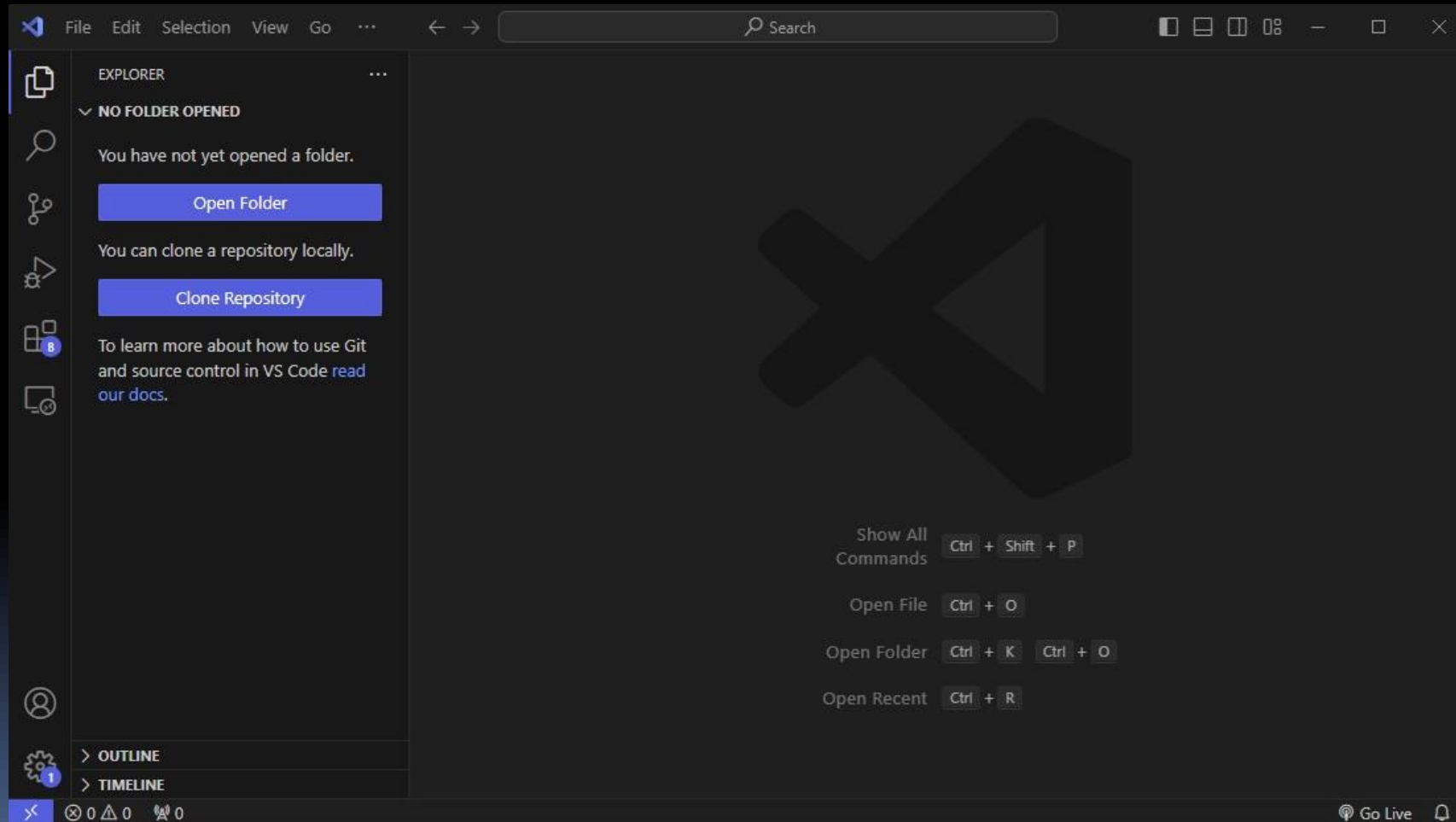
**Linux**  
Debian, Ubuntu: .deb  
Red Hat, Fedora, SUSE: .rpm

.deb: x64, Arm32, Arm64  
.rpm: x64, Arm32, Arm64  
.tar.gz: x64, Arm32, Arm64  
Snap: Snap Store  
CLI: x64, Arm32, Arm64

**Mac**  
macOS 10.15+

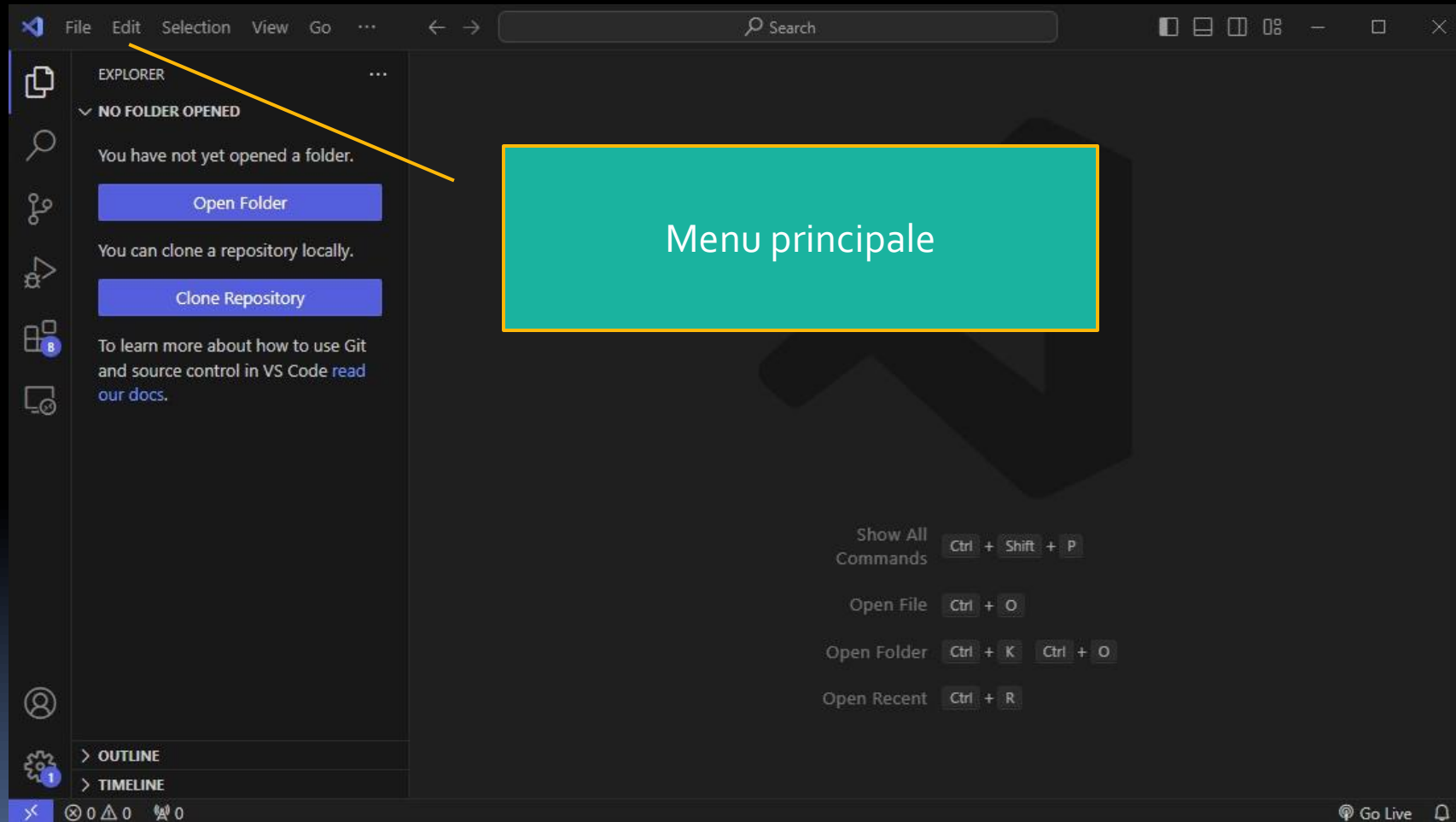
.zip: Intel chip, Apple silicon, Universal  
CLI: Intel chip, Apple silicon

# VSCode

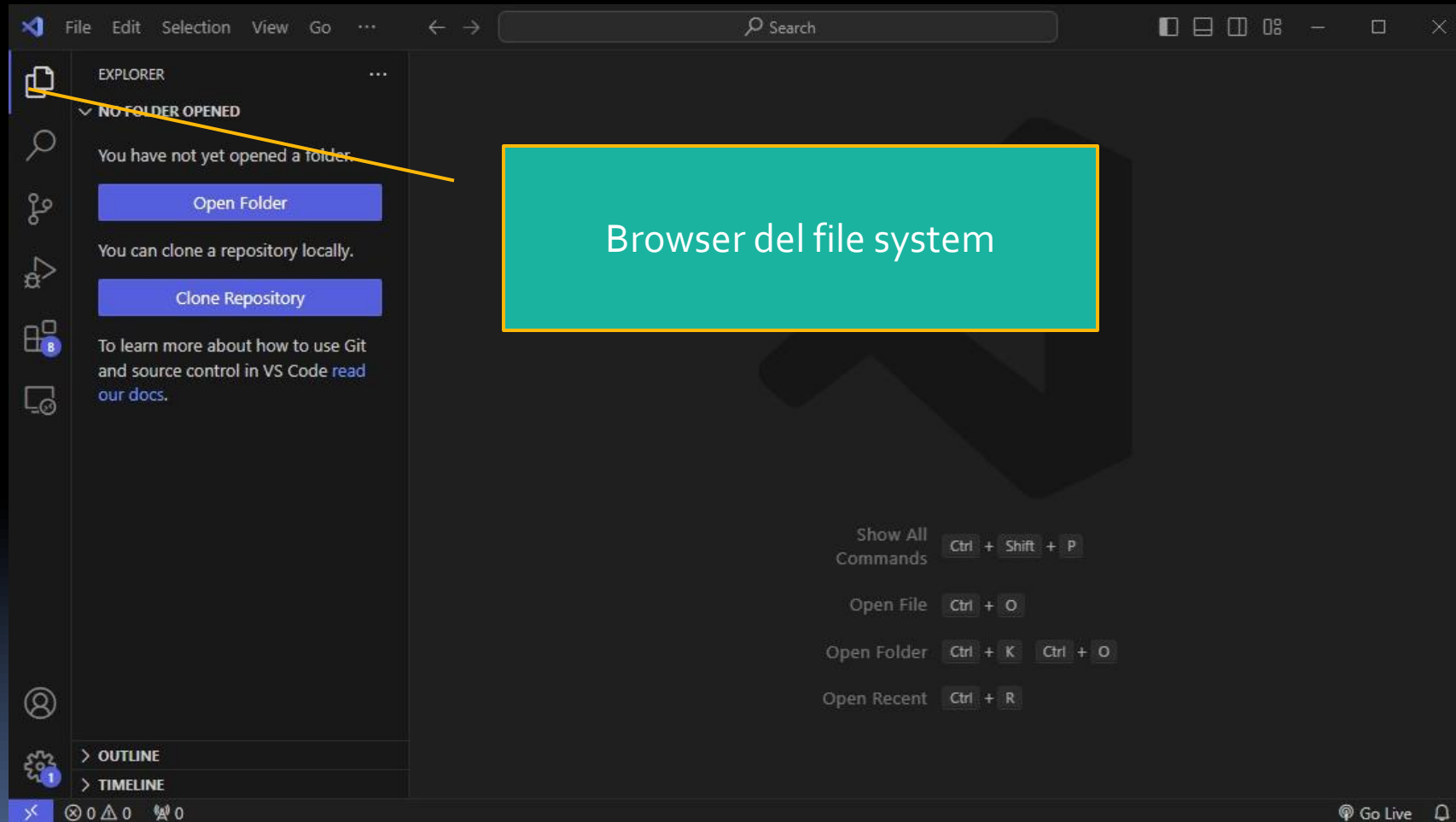




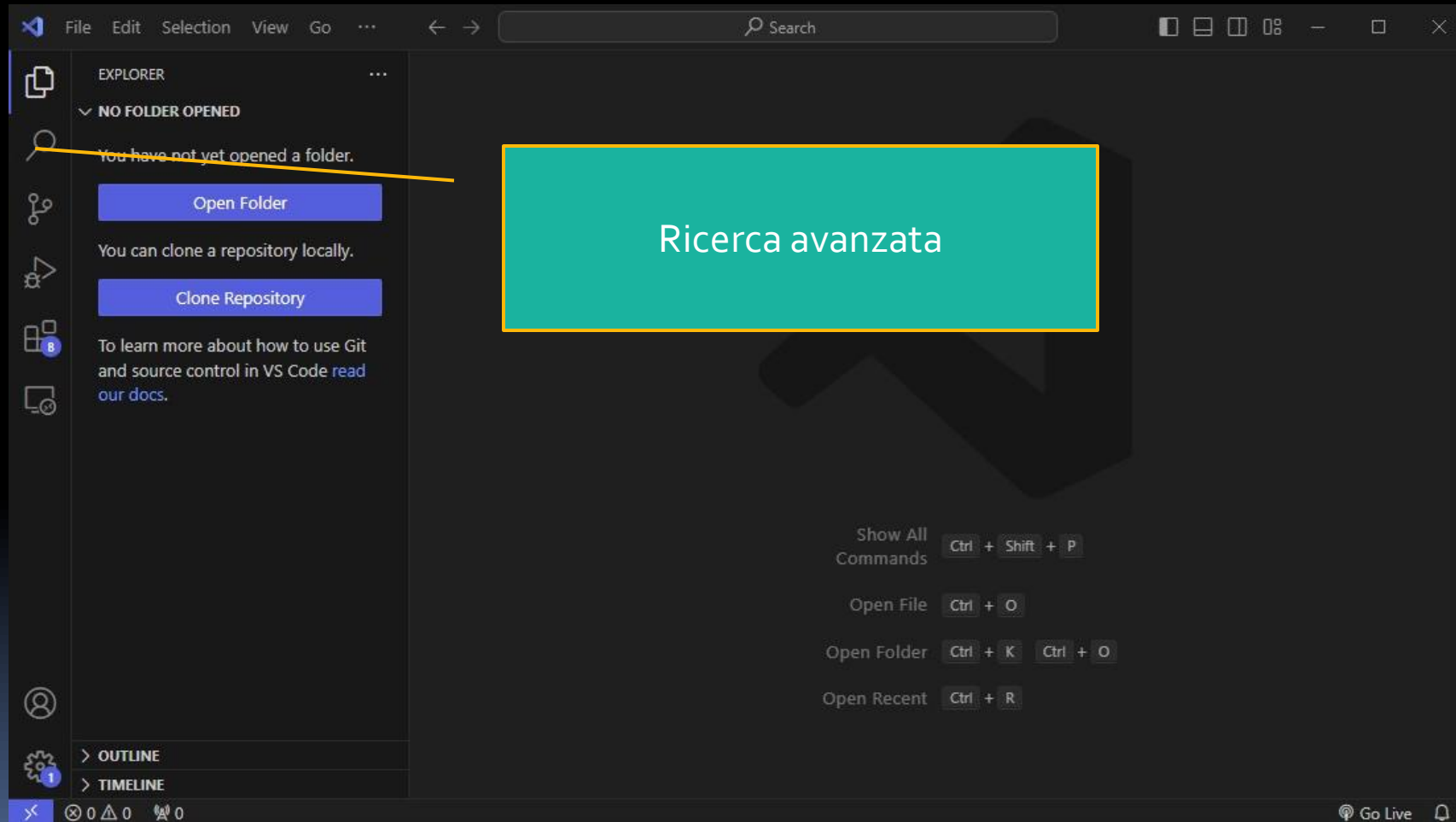
# VSCode



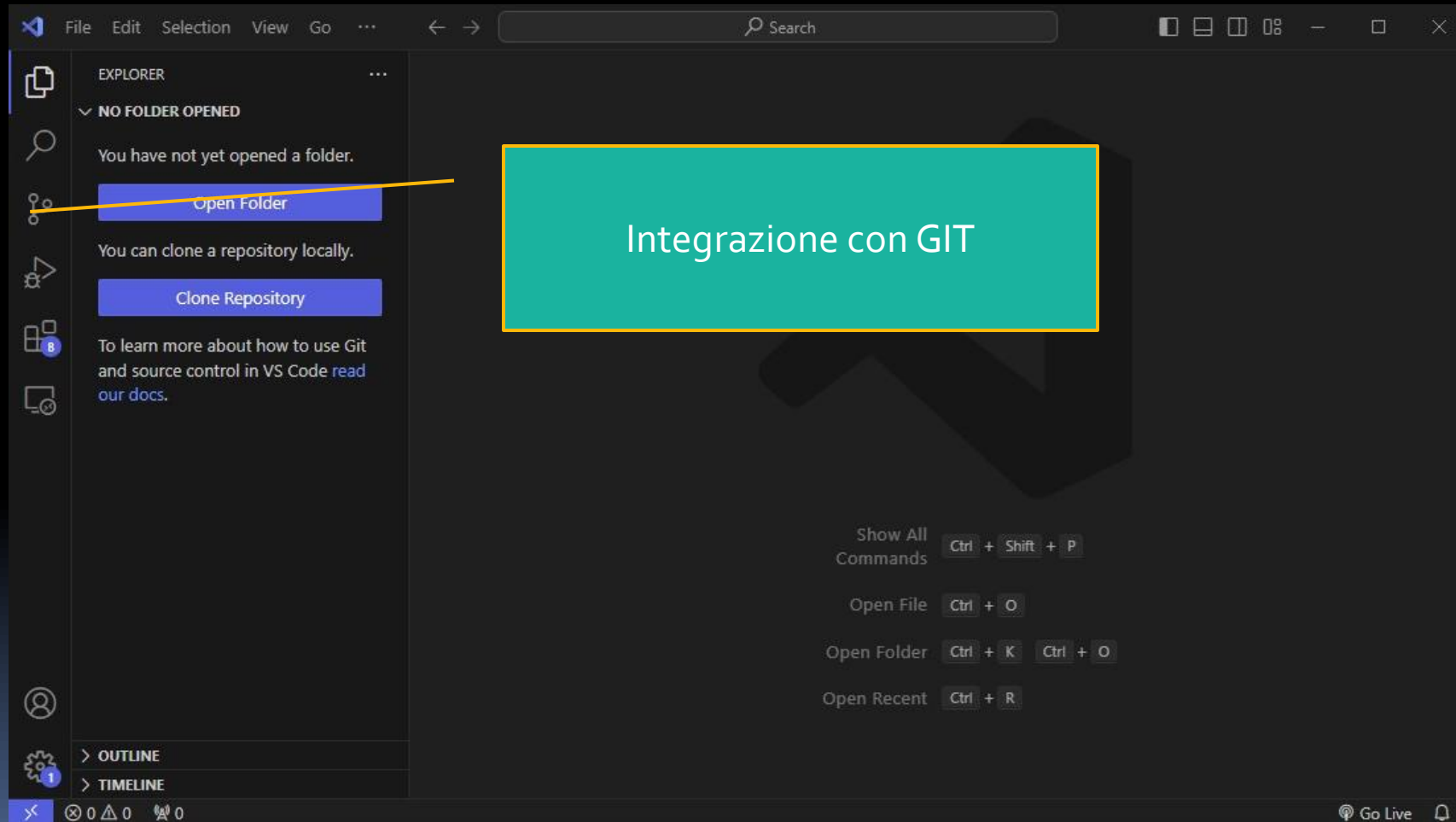
# VSCode



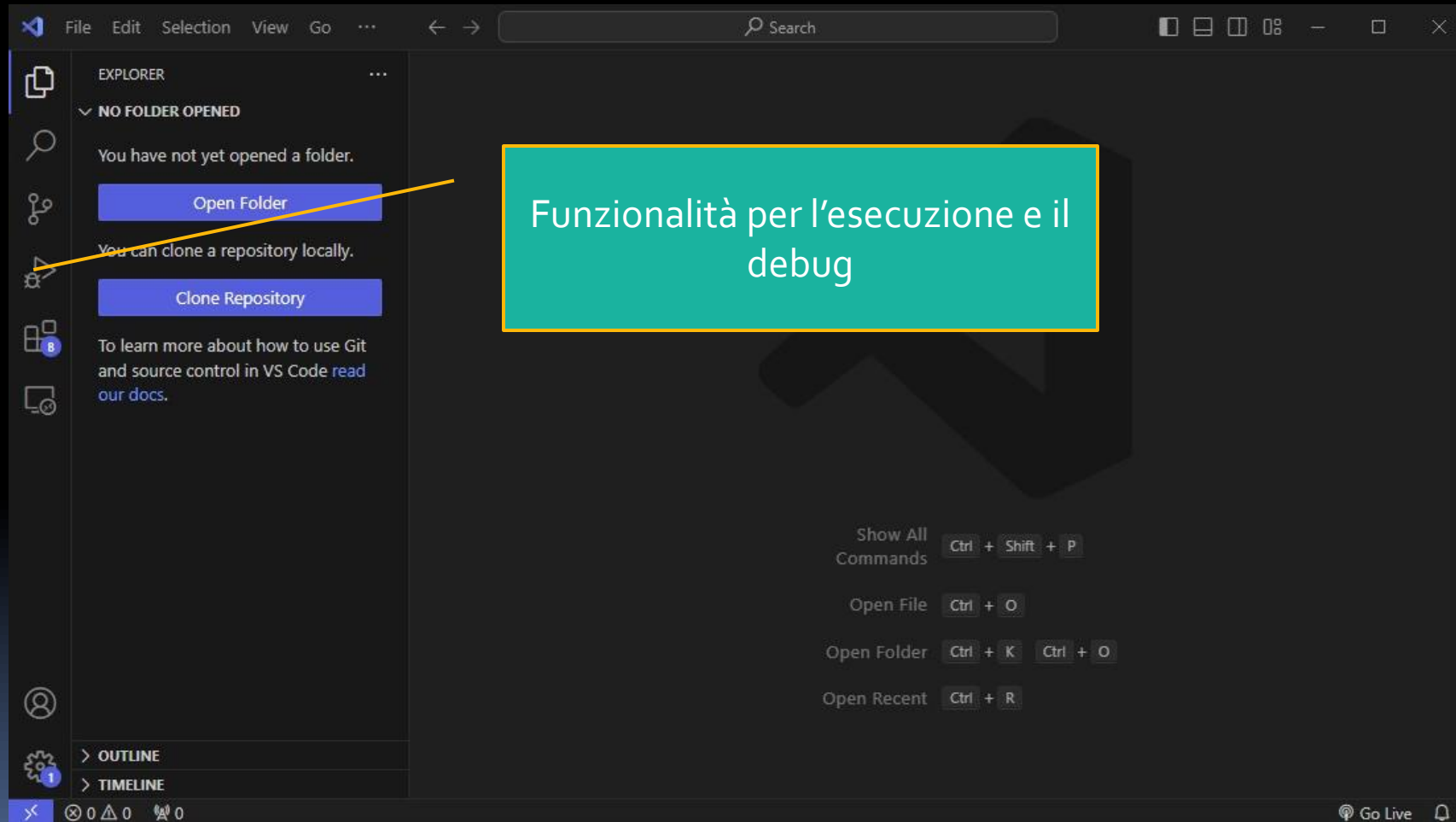
# VSCode



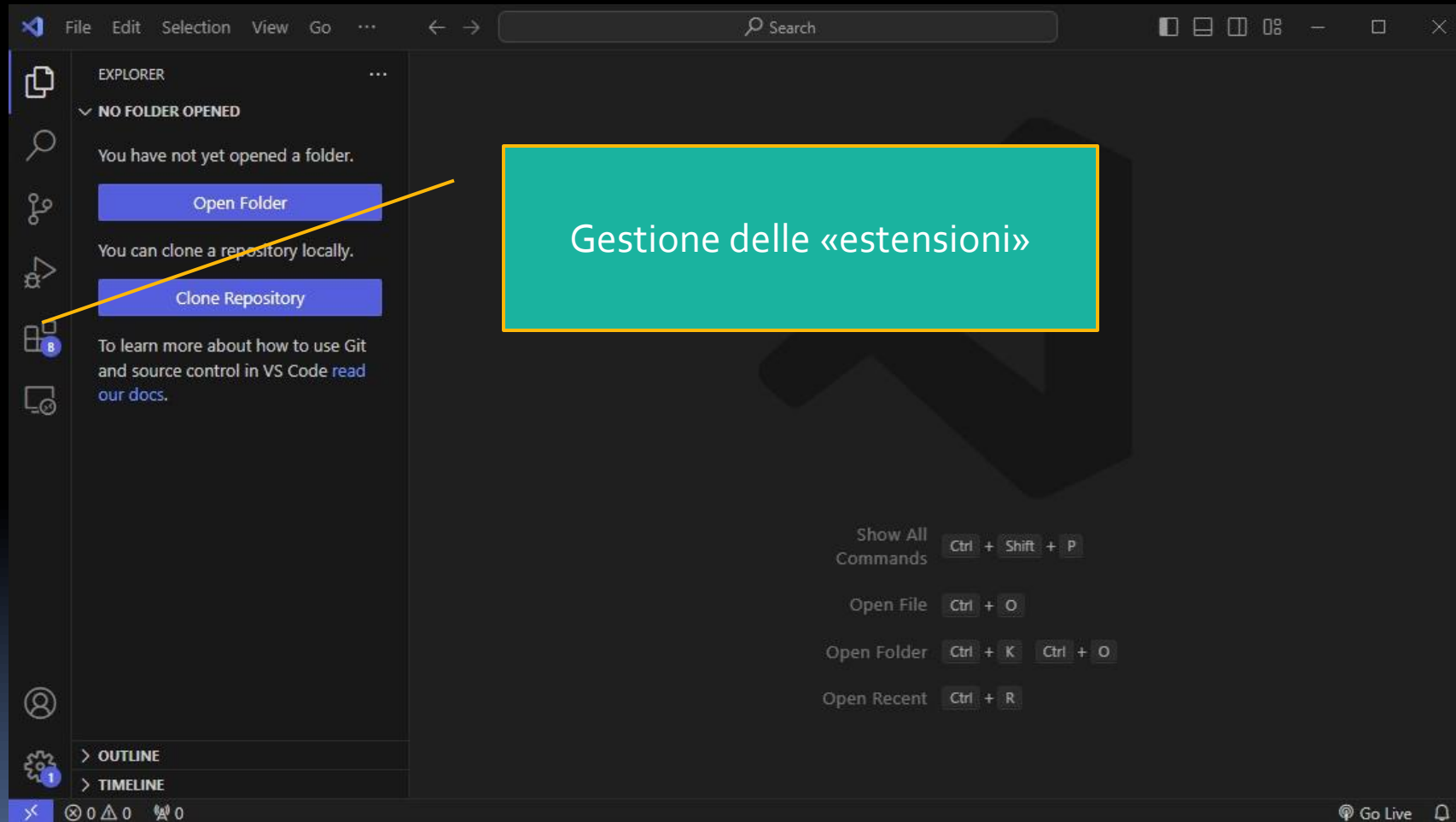
# VSCode



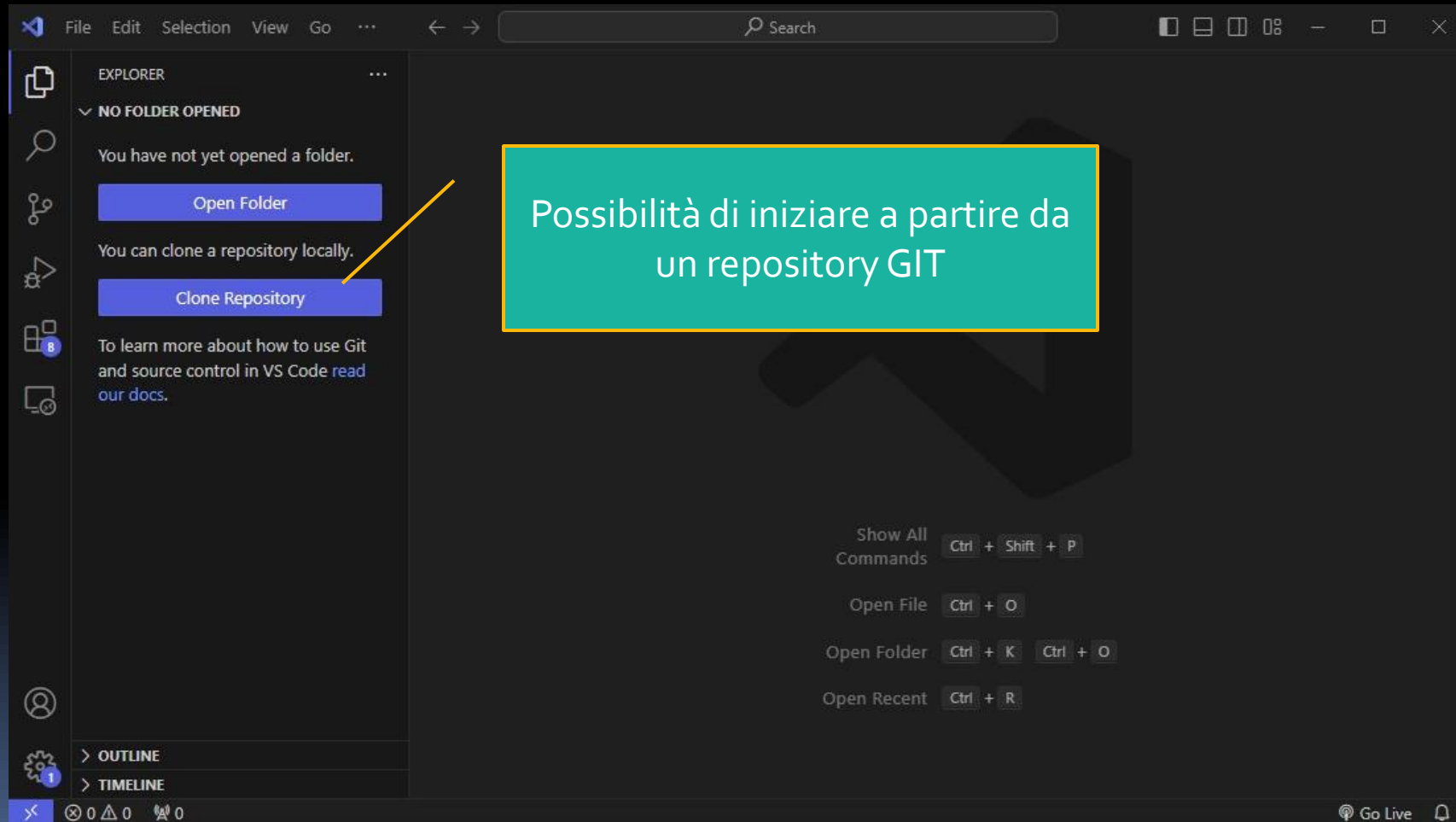
# VSCode



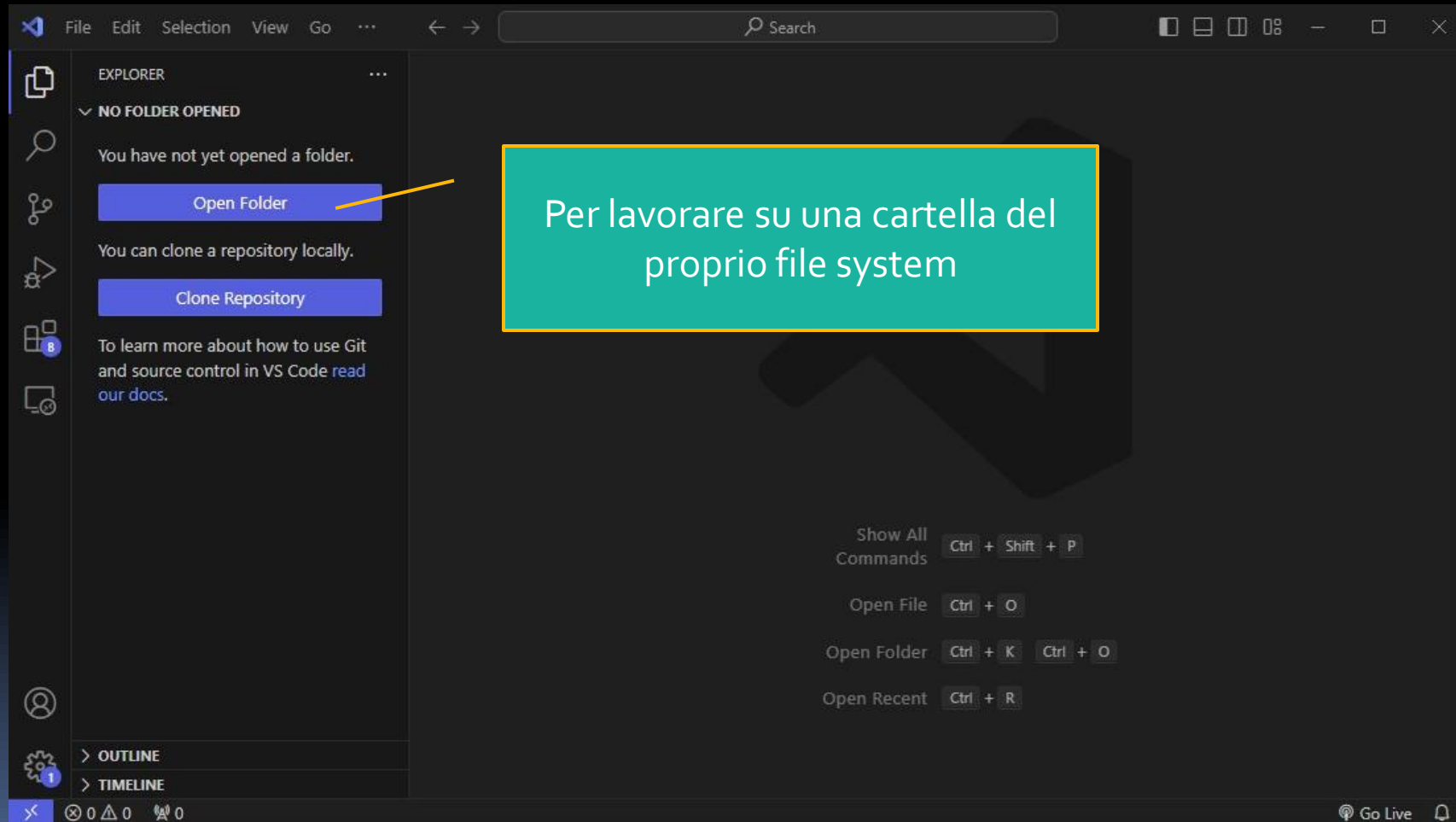
# VSCode



# VSCode

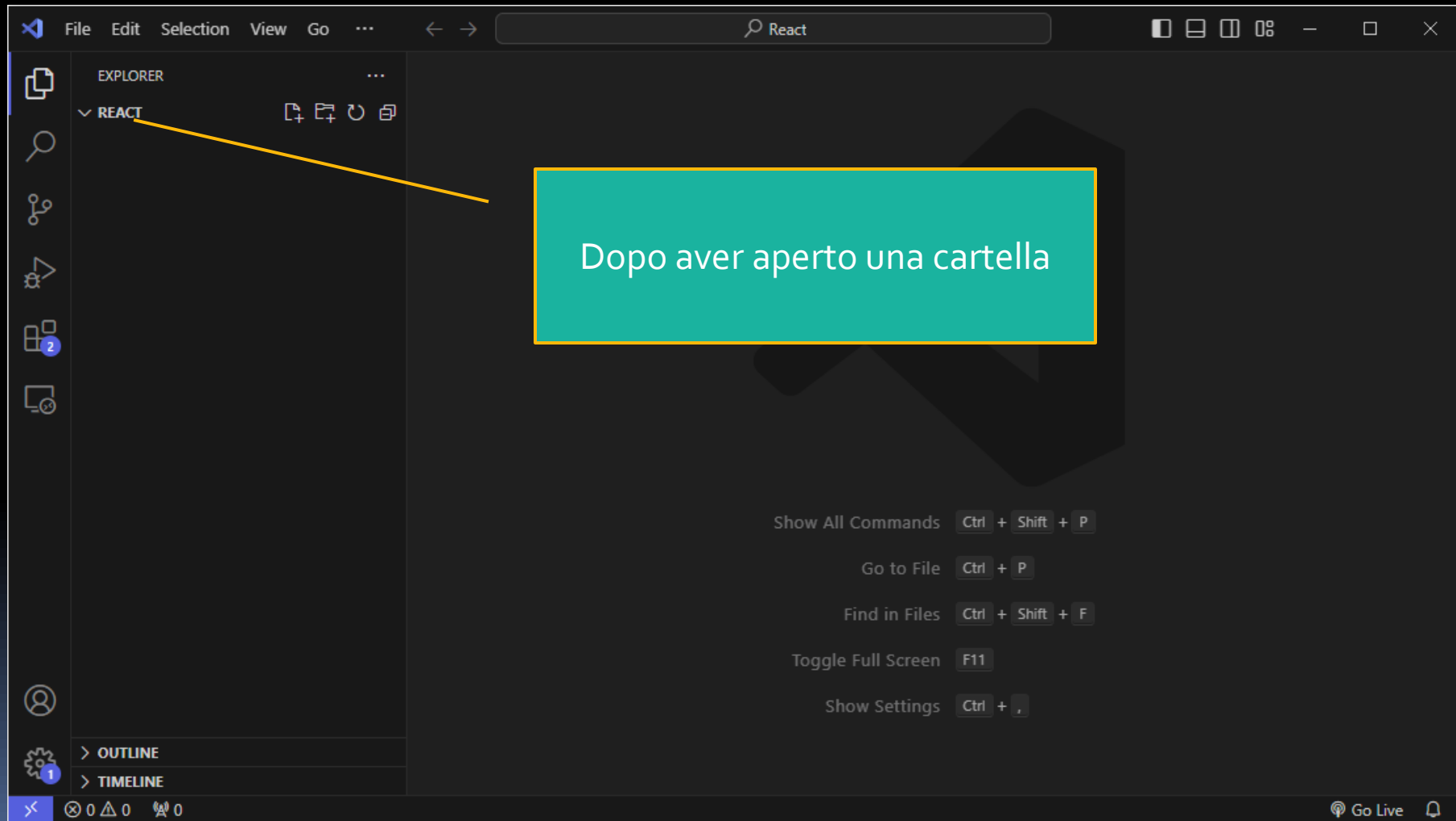


# VSCode

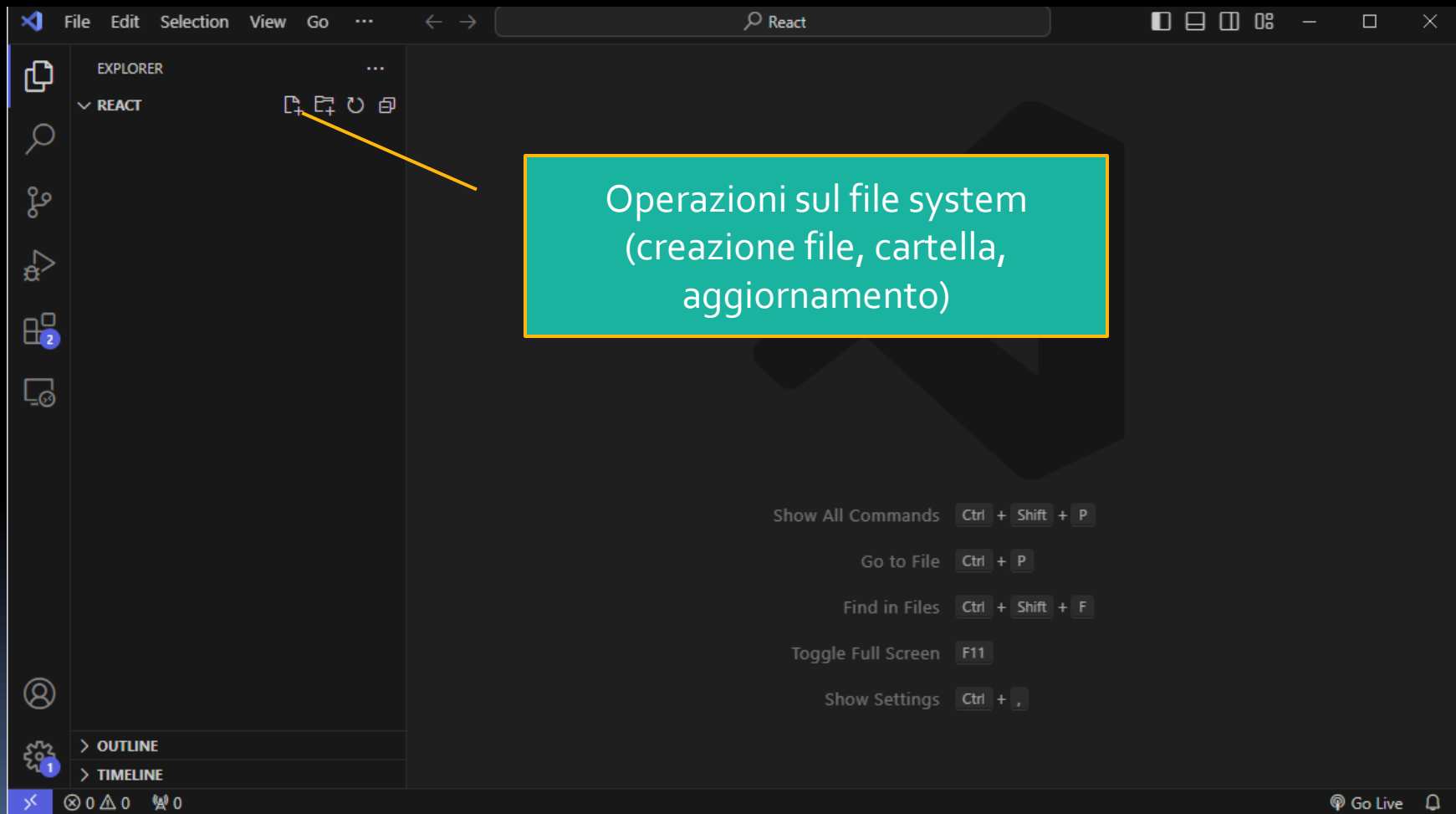




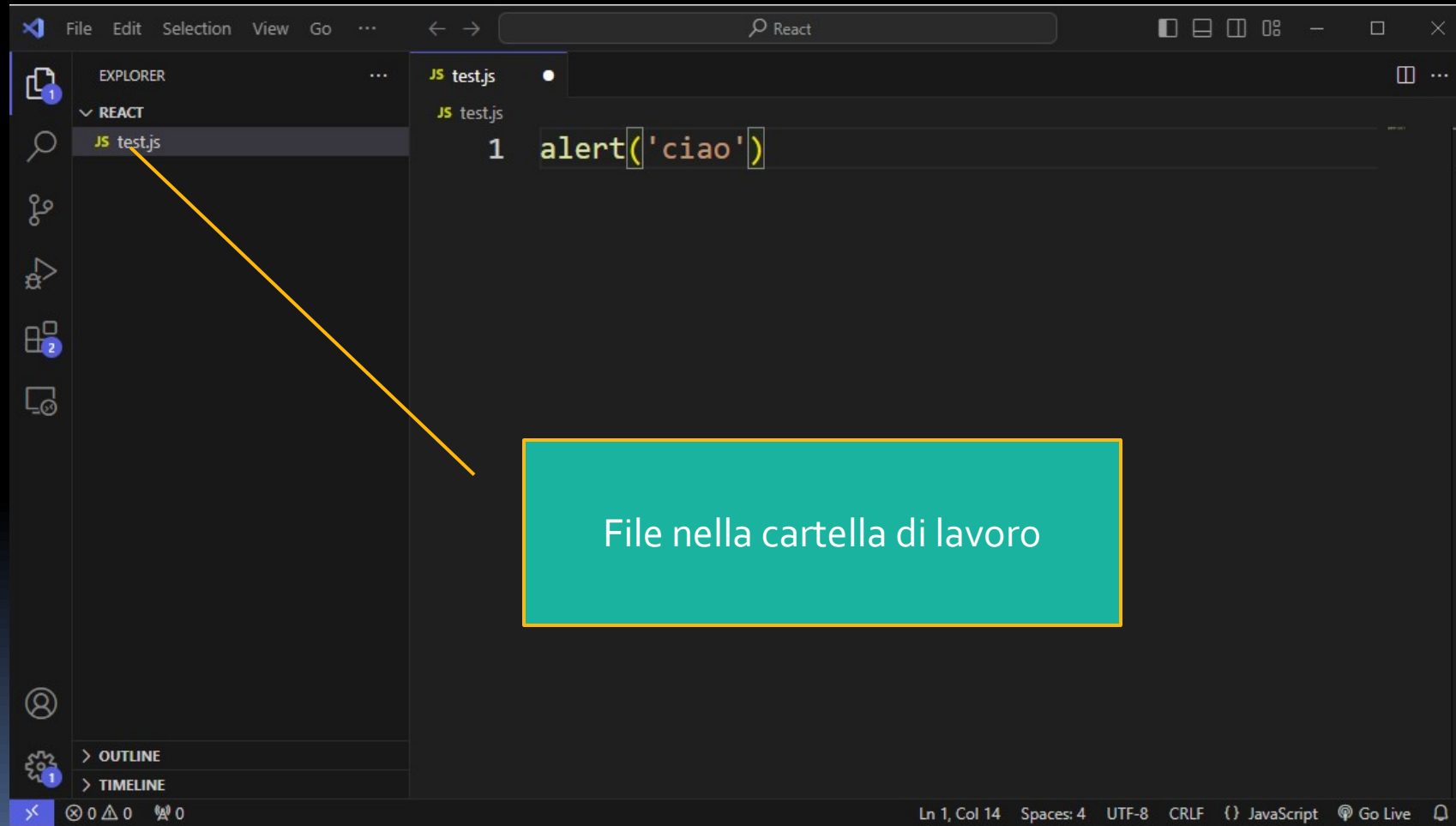
# VSCode



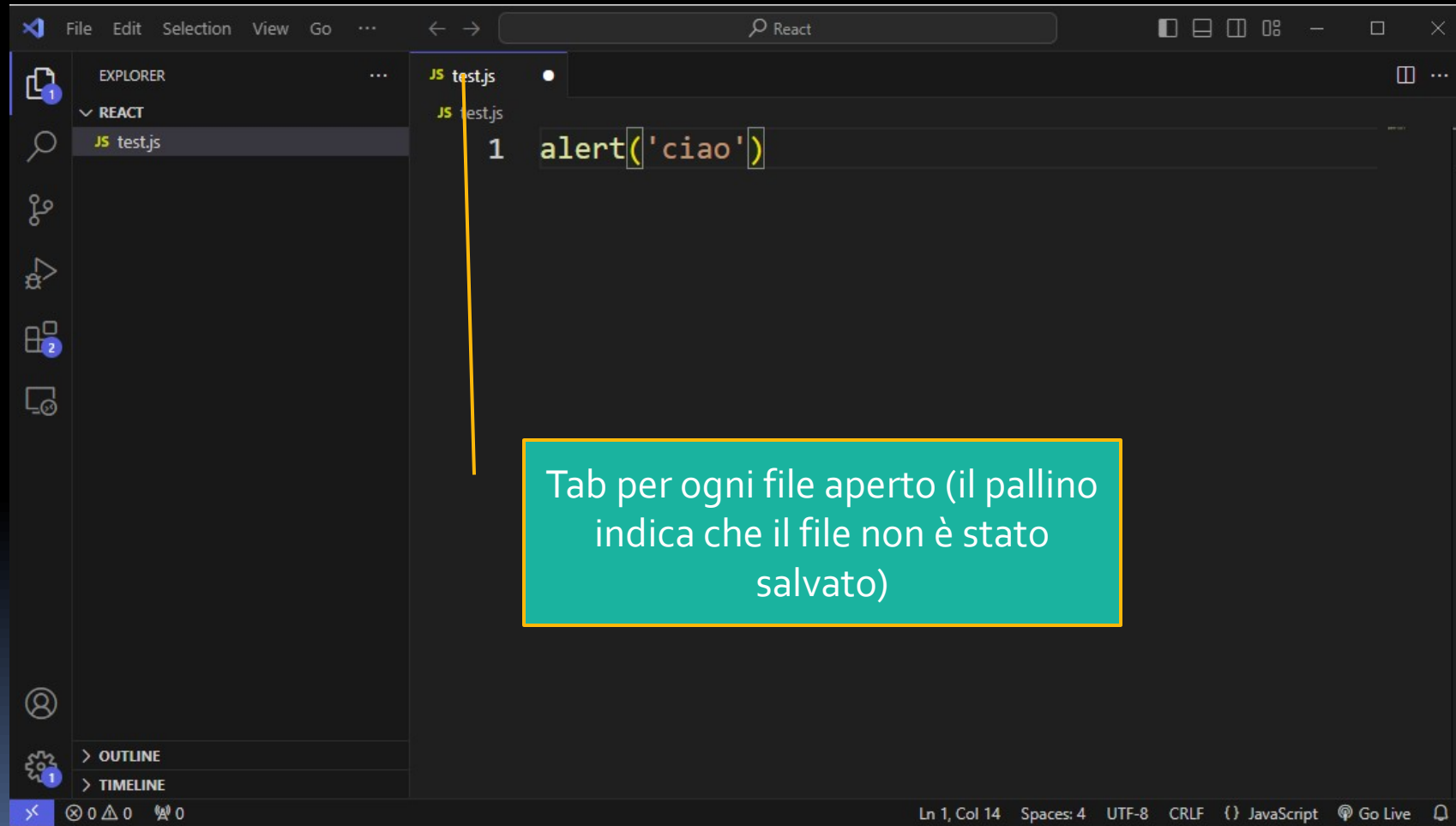
# VSCode



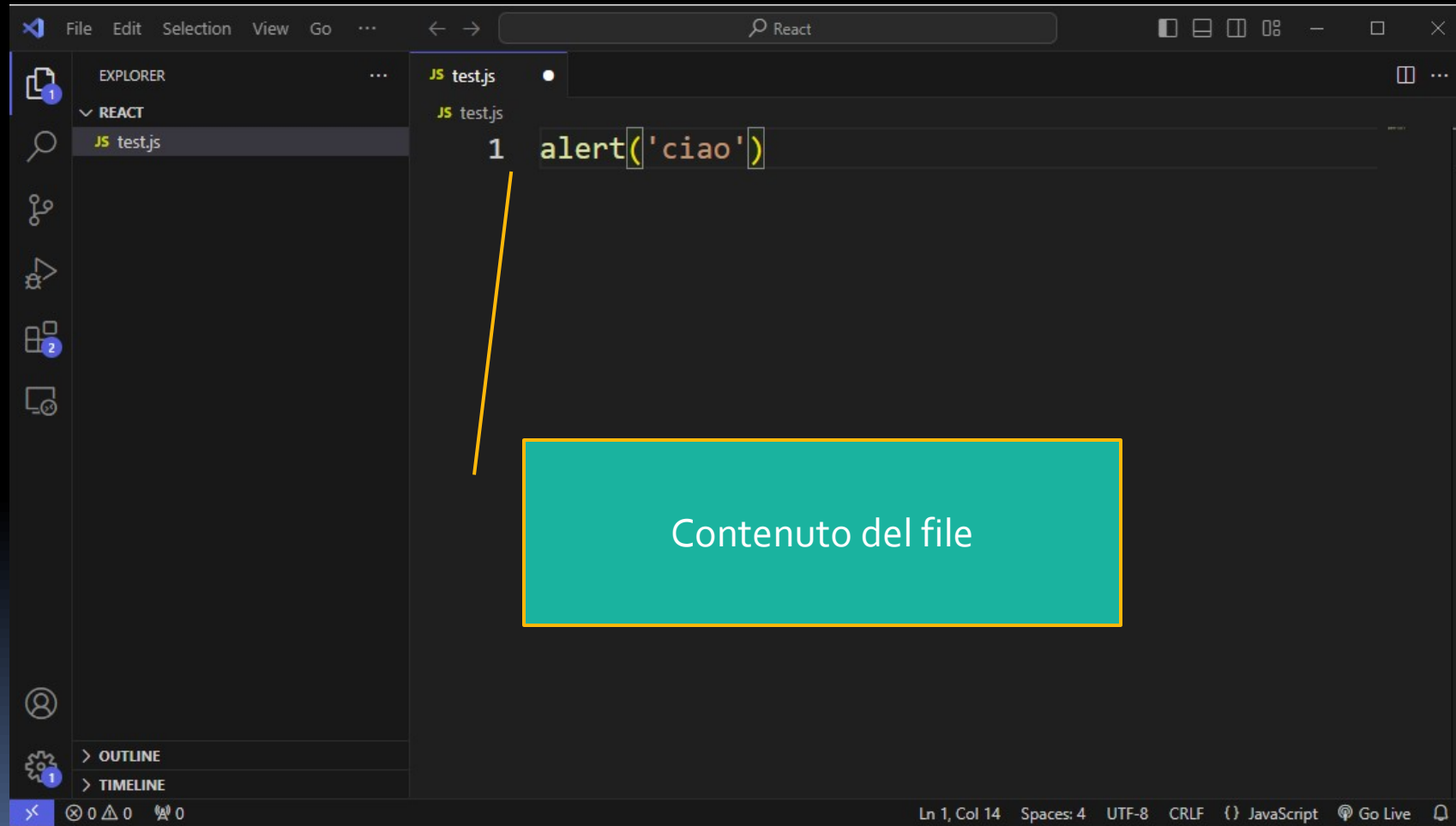
# VSCode



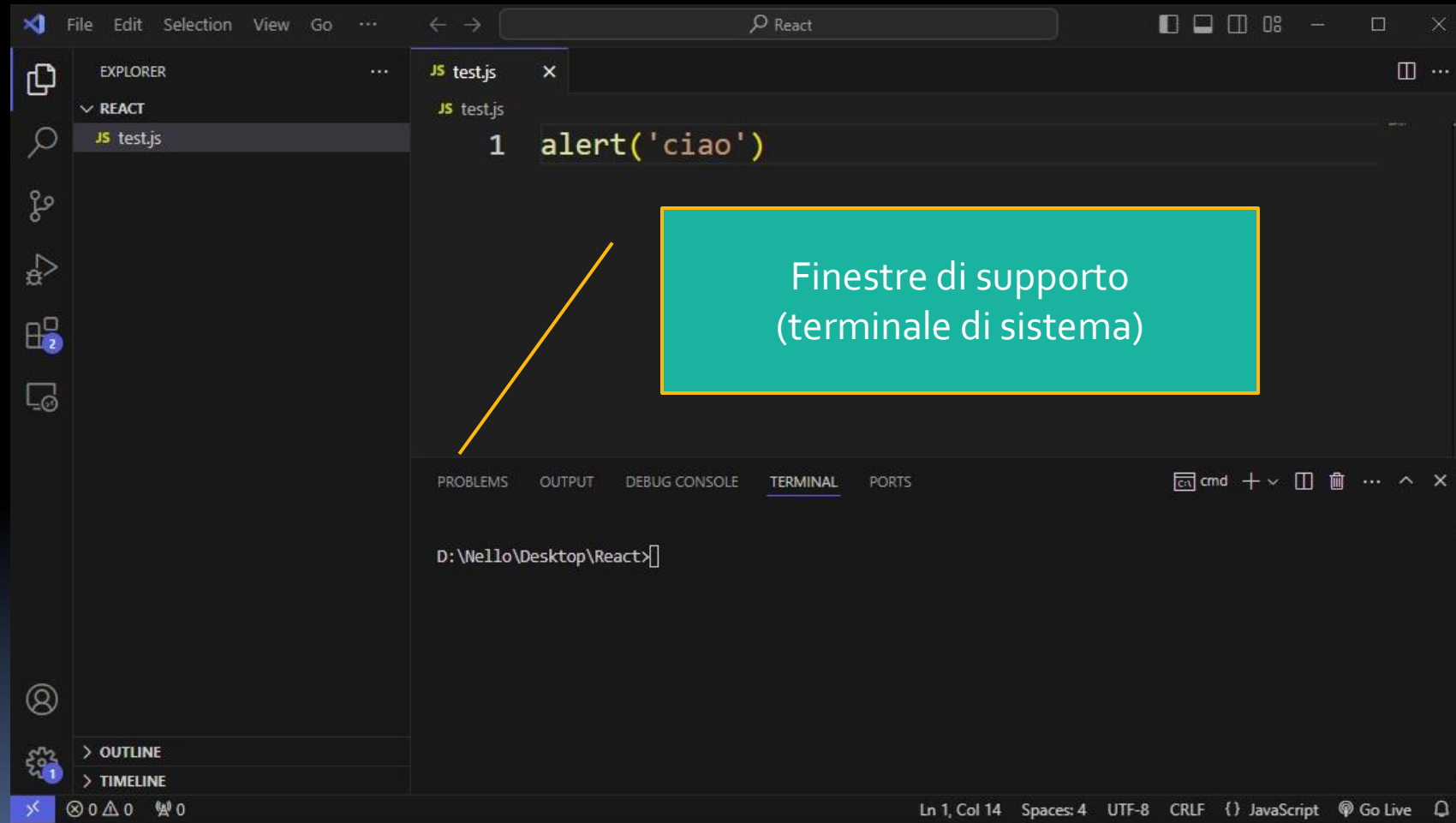
# VSCode



# VSCode



# VSCode





# Introduzione al Linguaggio





# Variabili e Tipi di Dato

- Ogni applicazione ha a disposizione un'area di memoria RAM nella quale manipolare le informazioni
- Ogni informazione è caratterizzata da un dominio di valori che può contenere
  - Un nome è una sequenza di caratteri alfanumerici a dimensione variabile
  - L'età di una persona può essere gestita come un numero intero in un range, ad esempio, 0-100
- Quando si intende manipolare un'informazione occorre informare la macchina dello spazio da mettere a disposizione





# Variabili e Tipi di Dato

- Prima di poter gestire un'informazione occorre dichiarare il proprio intento di occupare un'area di una determinata dimensione
- Una volta stabilito il dominio, è possibile richiedere alla macchina la gestione dell'area necessaria, apponendo ad essa un'etichetta
  - Che si chiamerà “nome di variabile” e servirà per leggere e scrivere in tale area
  - Ad essa andrà associata l'informazione relativa al tipo di dato gestito in tale area



# Variabili e Tipo di Dato

- JavaScript è un linguaggio a “tipizzazione dinamica”
  - Cioè è in grado di gestire in un’area di memoria un dato appartenente ad un qualsiasi dominio
    - Quindi una variabile può fare riferimento ad un qualsiasi tipo di informazione in un determinato momento
- TypeScript è, invece, (debolmente) “tipizzato”
  - Quindi è possibile stabilire preventivamente il tipo di dato gestito da una variabile (e questo non potrà variare in futuro)

# Variabili e Tipi di Dato

- Dichiarazione di variabile:
  - `let nome_variabile: tipo_di_dato`
- Inizializzazione
  - `let titolo: string = 'corso JavaScript'`
  - L'operatore `=` è detto operatore di assegnazione ed ha un compito fondamentale:
    - Valuta un'espressione che si trova alla sua destra (right value)
    - Scrive il risultato dell'espressione nella variabile che si trova alla sua sinistra (left value)
- Se il valore gestito in un'area di memoria non cambia nel tempo, si può utilizzare l'istruzione `const`

A photograph of a man and a woman lying in bed, looking thoughtful or confused. The woman is on the left, propped up on her elbow, looking towards the right. The man is on the right, also propped up, looking towards the left. They are both wearing casual clothing. The background is a simple bed with white pillows and a light-colored blanket.

**A COSA STARÀ PENSANDO?**

**IN JAVASCRIPT "100"/"10"  
FA 10. MA "10"+"10" FA "1010"...**

# Operatori

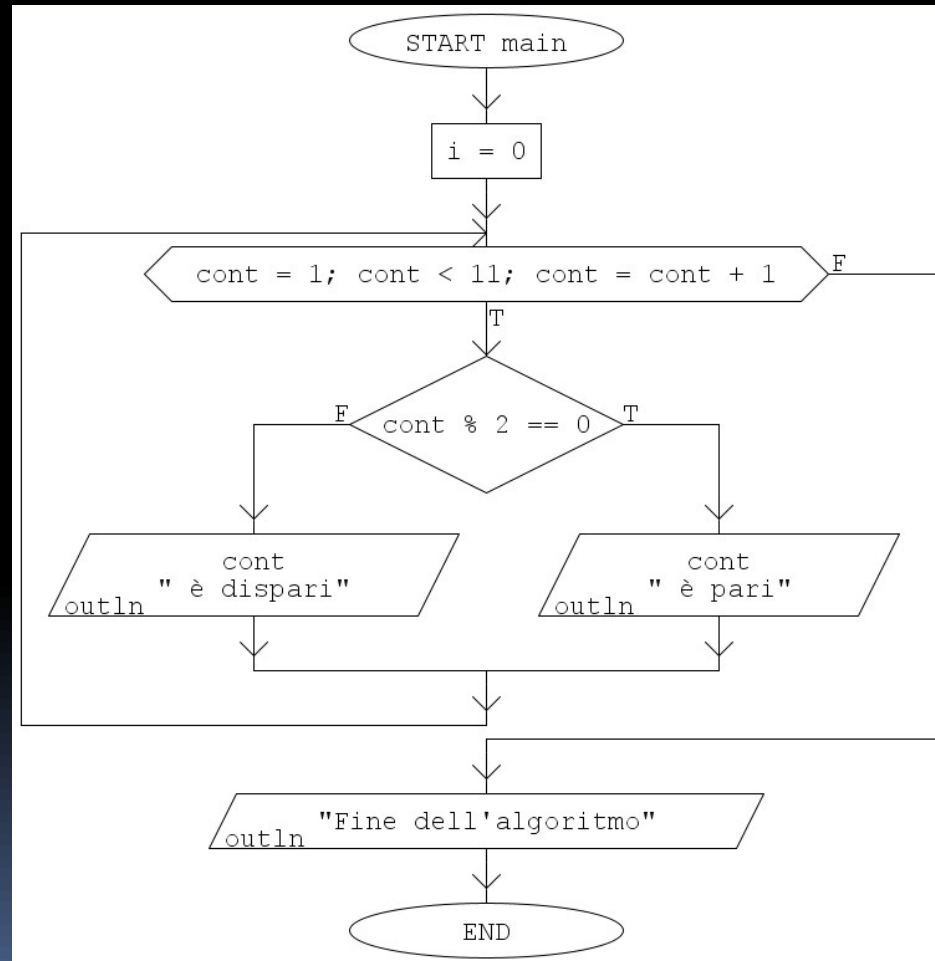
- Espressione: singola entità di codice che produce un risultato
  - Può essere un “letterale” o il risultato di un’operazione
- Operazione: unit of work che produce un’espressione
- Operatore: entità di codice che elabora uno o più operandi producendo un risultato
  - Operatori stringa:
    - Concatenazione: `let n = 'Mario' + ' Rossi'`
      - Dopo l’operazione n conterrà la stringa ‘Mario Rossi’
  - Operatori aritmetici: `+` `-` `*` `/` `%`
    - Esempio: `let result = 10 + 20 - 30 * 40 / 50 + 5 % 3`



# Funzioni

- La gestione dell'informazione viene affidata a sequenze di istruzioni che determinano il flusso di operazioni da effettuarsi sul dato
- Queste operazioni vengono incluse in contesti “denominati” che prendono il nome di funzione
- `function funcName(params) {`
  - ... operazioni sui dati
- `}`
- La funzione denomina il blocco di codice racchiuso tra le parentesi graffe e consente la sua esecuzione semplicemente scrivendo un'istruzione di chiamata:
  - `funcName()`

# Il Controllo del Flusso





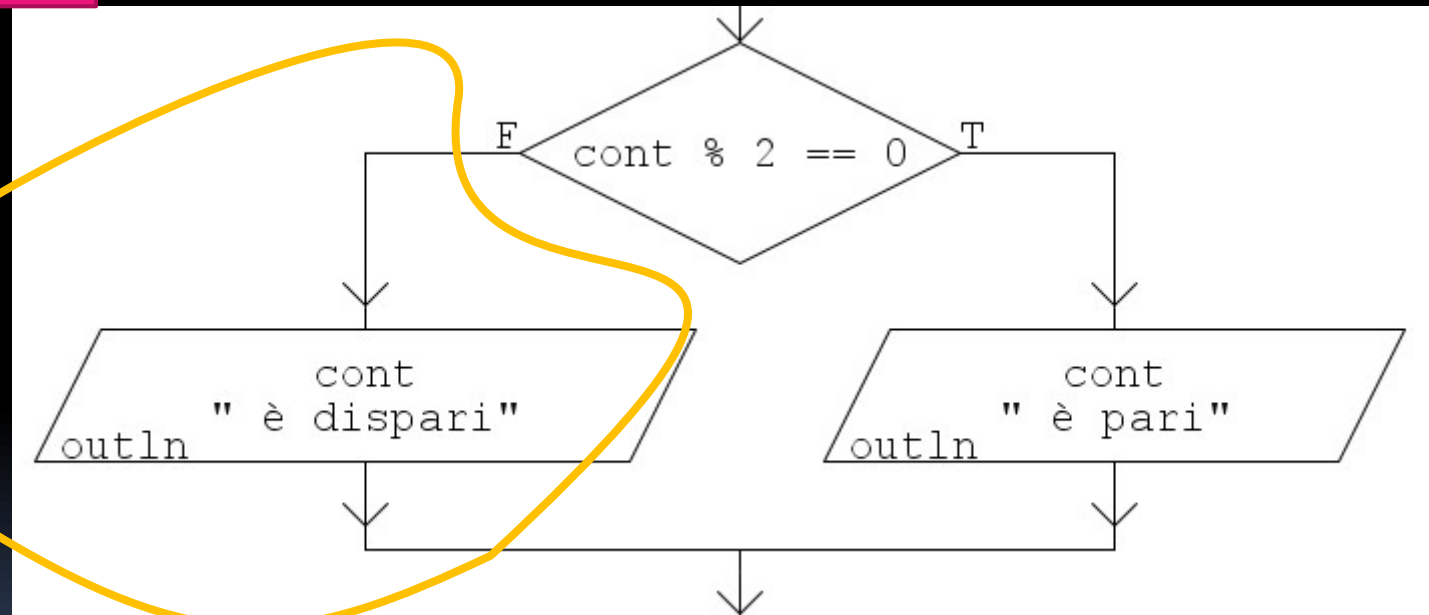
# Il Controllo del Flusso

- Il flusso di esecuzione è controllato attraverso la possibilità di testare il valore di espressioni
- Sulla base del test il flusso può essere instradato su un “ramo” piuttosto che un altro
- Alla base del test c'è l'istruzione “if”
- È possibile effettuare “salti” condizionati in maniera da eseguire ciclicamente delle istruzioni sulla base del test su una condizione
- In questo caso si usa l'istruzione “while”

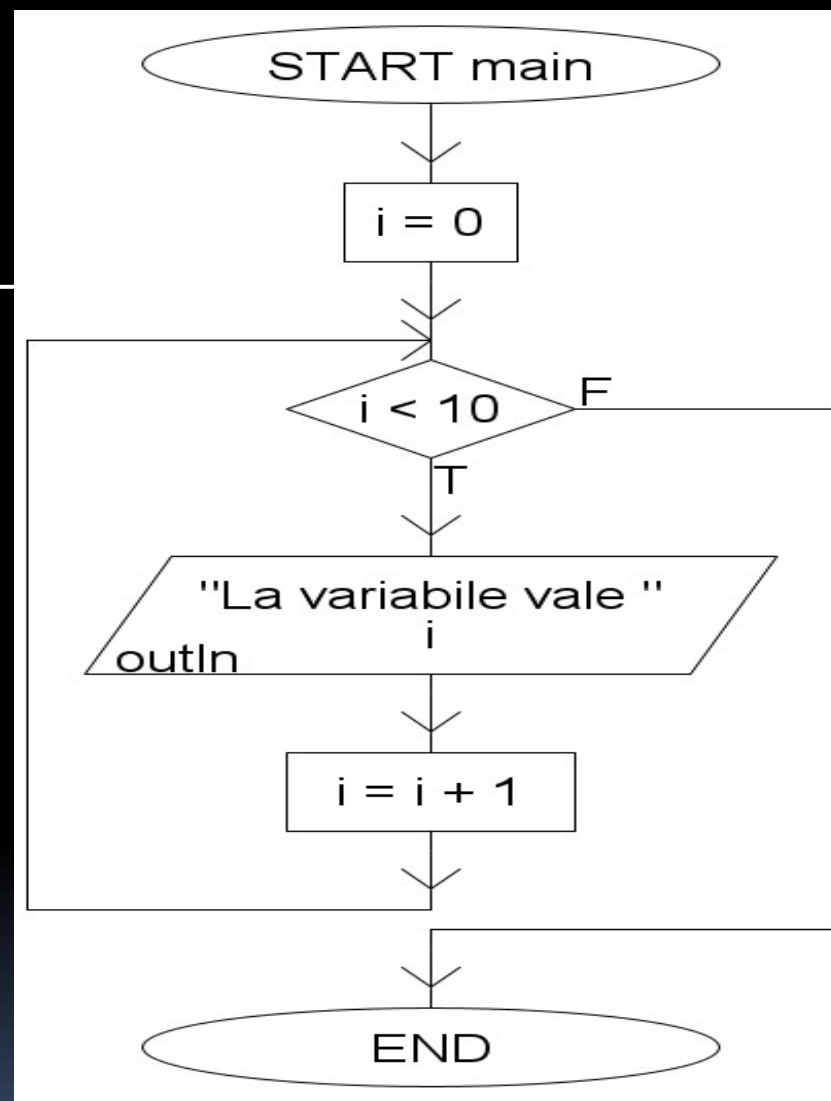


# Istruzione if

Ramo facoltativo



## Istruzione while



# Controllo del Flusso

- Set di istruzioni:
  - if – if/else – if/else if/else
  - while – do/while
    - Funzionano allo stesso modo, cambia solo il momento in cui viene valutata la condizione di permanenza nel ciclo (all’inizio o alla fine)
      - Conseguenza è che il do/while viene eseguito almeno una volta
    - L’istruzione prevede che all’interno del ciclo venga modificata la condizione di permanenza (uscita) al fine di prevenire cicli infiniti
  - for
    - Sostanzialmente è un’abbreviazione di while:
      - Prevede tre sezioni inserite in parentesi tonda dopo l’istruzione: inizializzazione, condizione di permanenza, modifica della condizione



# Arrow Functions

- Concetto di funzione anonima
  - In Javascript una funzione può trovarsi dappertutto, anche in un'altra funzione
  - Lifecycle di una variabile
- Importanza delle funzioni high-order
- Sintassi delle arrow function
  - Concetto di “cattura” di variabili



# Concetto di Algoritmo

- Sequenza di passaggi elementari (noti) per lo svolgimento completo e ripetibile di un compito
  - Esempi:
    - Cottura della pasta
    - Lancio di un razzo verso la luna
    - Calcolo del codice fiscale di una persona



# TypeScript

- Linguaggio di programmazione open source sviluppato da Microsoft
  - Super-set di JavaScript che basa le sue caratteristiche su ECMAScript 6
- Estende la sintassi di JavaScript in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica
  - Progettato per lo sviluppo di grandi applicazioni
  - Destinato a essere compilato in JavaScript



# TypeScript

- Tipi di dato
  - string
    - `let nome:string = 'fabio'`
  - number
    - `let index: number = 1.0`
  - boolean
    - `let go = true`
  - unknown
    - `let case: unknown = 23`
  - void
    - Indica che una funzione non restituisce alcun risultato
  - never
    - Indica che una funzione non prevede uscita



# TypeScript - Stringhe

- Sequenze di caratteri gestiti come un dato alfanumerico
- Si tratta di un elemento complesso sul quale si applicano funzioni che ne gestiscono il contenuto restituendo un nuovo dato:
  - **concat:** concatena più stringhe
  - **length:** ritorna la lunghezza di una stringa
  - **trim:** rimuove gli spazi a sx e dx di una stringa
  - **replace:** sostituisce una stringa o un carattere
  - **slice:** estrae una sezione di una stringa e restituisce una nuova stringa
  - **toLowerCase:** trasforma una stringa da maiuscolo in minuscolo
  - **toUpperCase:** trasforma una stringa da minuscolo a maiuscolo



# TypeScript – Operazioni Matematiche

- `abs`
  - ritorna il valore assoluto passato come parametro
- `floor`
  - forza il decimale al numero più piccolo
- `round`
  - arrotonda l'argomento al valore successivo se la parte frazionaria è  $>$  di 5
- `pow`
  - esponenziale del primo numero passato in ingresso
- `random`
  - genera numeri casuali
- `ceil`
  - forza il decimale al numero più grande
- `sqrt`
  - radice quadrata del numero passato come parametro

# TypeScript – Date e Orari

- “Oggetto” Date!
  - Si tratta di una struttura complessa (della quale parleremo più avanti)
    - Al momento è possibile pensare ad un oggetto come ad una funzione al cui interno ci siano altre funzioni
- Creazione di Date
  - `new Date()`
    - Crea una nuova data con l’ora corrente
  - `new Date(value)`
    - Crea una data a partire da un valore specifico (timestamp, stringa, ecc.)

# TypeScript – Date e Orari

- Recupero di informazioni da una data:
  - `getDate()`
    - Restituisce il giorno del mese (1-31)
  - `getMonth()`
    - Restituisce il mese (0-11)
  - `getFullYear()`
    - Restituisce l'anno (es. 2024)
  - `getHours()`
    - Restituisce l'ora (0-23)
  - `getMinutes()`
    - Restituisce i minuti (0-59)
  - `getSeconds()`
    - Restituisce i secondi (0-59)
  - `getMilliseconds()`
    - Restituisce i millisecondi (0-999)
  - `getTime()`
    - Restituisce il timestamp (millisecondi dal 1 gennaio 1970)

# TypeScript – Date e Orari

- Impostazione delle informazioni in una data:
  - setDate(day)
    - Imposta il giorno del mese
  - setMonth(month)
    - Imposta il mese
  - setFullYear(year)
    - Imposta l'anno
  - setHours(hours)
    - Imposta l'ora
  - setMinutes(minutes)
    - Imposta i minuti
  - setSeconds(seconds)
    - Imposta i secondi
  - setMilliseconds(milliseconds)
    - Imposta i millisecondi
  - setTime(milliseconds)
    - Imposta la data e l'ora in base al timestamp (espresso in millisecondi)

# TypeScript – Date e Orari

- Formattazione:
  - `toString()`
    - Restituisce la data in formato leggibile (es. "Wed Oct 02 2024")
  - `toLocaleTimeString()`
    - Restituisce l'ora in formato leggibile (es. "15:45:00 GMT+0200")
  - `toLocaleDateString()`
    - Restituisce la data in formato locale
  - `toLocaleTimeString()`
    - Restituisce l'ora in formato locale
  - `toISOString()`
    - Restituisce la data in formato ISO (es. "2024-10-02T13:45:00.000Z")

# Enumerazioni

- Le enumerazioni sono particolari strutture che consentono di gestire l'autodocumentazione del codice
- `enum Months { January = 1, February, March, }`
  - Il valore January è inizializzato con 1, quindi i seguenti membri vengono incrementati automaticamente da quel momento in poi
- Le enumerazioni possono essere eterogenee
  - `Enum LikeHeterogeneousEnum { No = 0, Yes = "YES", }`



# Arrays

- Un array è una raccolta (omogenea) di valori dello stesso tipo di dati
  - In un array è possibile aggiungere, rimuovere, ordinare gli elementi attraverso metodi come
    - push – pop – sort – shift – unshift – reverse – length – indexOf
- Si tratta dell'unica struttura che funziona da “contenitore”
  - Gli elementi sono
    - principalmente accessibili tramite un indice numerico
    - probabilmente gestiti in memoria come elementi contigui per favorire l'accesso veloce (casuale) ai singoli elementi contenuti

# Funzioni in TypeScript

```
function add(x: number, y: number, z?: number): number {  
    return x + y + z ?? 0  
}
```

```
console.log(add(1,3))
```

- Il simbolo ? Indica che il parametro è opzionale





# Classi

- Concetto di “Classe”?
  - Classificazione completa di un’informazione
    - Contiene tutte le “proprietà” veicolate da un’informazione e tutte le operazioni per la loro gestione
- Fondamentale per la riutilizzabilità e l’organizzazione del codice
  - Rappresentano dei nuovi tipi di dato complessi, composti da un’aggregazione di tipi elementari

# Classi

```
class Persona {  
  nome: string;  
  età: number;  
  
  constructor(nome: string, età: number) {  
    this.nome = nome;  
    this.età = età;  
  }  
  
  saluta() {  
    console.log(`Ciao, mi chiamo ${this.nome} e ho ${this.età} anni.`);  
  }  
}
```

```
const persona1 = new Persona('Mario', 30);
```

```
persona1.saluta(); // Output: Ciao, mi chiamo Mario e ho 30 anni.
```



# Classi

- Concetti fondamentali:
  - **Proprietà**
    - Variabili associate a una classe.
      - Esempio: nome, età.
  - **Metodi**
    - Funzioni associate a una classe.
      - Esempio: saluta()
- **this**
  - Particolare “pseudo-variabile” che consente di accedere a proprietà e metodi nel codice interno alla classe

# Ereditarietà

```
class Studente extends Persona {  
  matricola: number;  
  
  constructor(nome: string, età: number, matricola: number) {  
    super(nome, età);  
    this.matricola = matricola;  
  }  
  
  mostraMatricola() {  
    console.log(`La mia matricola è ${this.matricola}.`);  
  }  
}  
  
const studente1 = new Studente('Luigi', 22, 12345);  
studente1.saluta(); // Output: Ciao, mi chiamo Luigi e ho 22 anni.  
studente1.mostraMatricola(); // Output: La mia matricola è 12345.
```



# Ereditarietà

- Gli elementi interni ad una classe sono “accessibili” attraverso regole di controllo dell’accesso, applicabili a tutti gli elementi interni ad una classe attraverso i seguenti “modificatori”:
  - Public
    - Indica membri accessibili da qualsiasi parte del programma
  - Private
    - Indica membri accessibili solo all’interno della classe
  - Protected
    - Indica membri accessibili all’interno della classe e delle sue sottoclassi

# Compatibilità Ereditaria

- Una sottoclasse è compatibile, a runtime, con una variabile che si riferisce ad una superclasse:
  - Es: `let p: Persona = new Studente(...)`
- Questo consente di scrivere degli algoritmi di gestione di oggetti di una classe che potranno ricevere in input oggetti diversi appartenenti ad una qualsiasi sottoclasse necessaria in un particolare punto dell'applicazione
- Astrazione vs Implementazione?



# Accesso alla Classe Base

- La parola chiave `super` viene utilizzata nel paradigma dell'ereditarietà quando nella sottoclasse si desidera accedere a proprietà o metodi di una superclasse



# Classi Astratte

- Una classe astratta è una classe che prevede una dotazione di metodi alcuni dei quali non possono essere scritti al momento della scrittura della classe
  - Viene definita con la keyword `abstract`
  - I metodi che non è possibile scrivere prevedono solo la definizione ma non l'implementazione e sono preceduti anch'essi dalla keyword `abstract`
- Una classe astratta è che non può essere istanziata (attraverso una `new`)
  - ma può comunque essere base per una classe non astratta
  - e può avere caratteristiche non astratte
  - inoltre una variabile può essere definita come appartenente ad un tipo astratto e sfruttare la compatibilità ereditaria per valorizzarla





# Interfacce

- Una classe che contiene solo metodi astratti può essere definita anche come interfaccia tramite la keyword “interface”
  - L'utilità delle interfacce è evidente nella gestione delle regole di ereditarietà
    - Infatti una classe può derivare da un'unica superclasse ma può implementare più interfacce



# Programmazione Generica

- I generics permettono di definire algoritmi che sono in grado di lavorare con diversi tipi di dato garantendo un controllo del tipo a tempo di compilazione

# Tuple

- Una tupla è un tipo TypeScript che rappresenta diverse informazioni (di tipo diverso) valorizzandole in un contesto unico
  - Si tratta di un elemento immutabile
    - Spesso viene indicata come “record”
      - `let user: [number, string] = [1, "Steve"]`
  - È possibile avere array di tuple
    - `let users: [number, string][]`
    - `users = [[1, "Steve"], [2, "Bill"], [3, "Jeff"]]`
    - `users.push([4, "John"])`

# Metodi Statici

- Sono metodi che effettuano delle elaborazioni non influenzano lo “stato” di un oggetto

```
class Calcolatrice {  
    static somma(a: number, b: number): number {  
        return a + b;  
    }  
}  
  
console.log(Calcolatrice.somma(5, 3)); // Output: 8
```



# Moduli

- Un modulo è un file TypeScript che contiene codice che può essere importato ed esportato
- Ogni modulo ha il proprio scope privato
  - le variabili e le funzioni definite all'interno di un modulo non sono accessibili al di fuori di esso, a meno che non vengano esplicitamente esportate
- Vantaggi:
- **Organizzazione:** Mantiene il codice organizzato e modulare
- **Riutilizzabilità:** Facilita il riutilizzo del codice in diversi progetti
- **Manutenibilità:** Rende il codice più facile da mantenere e aggiornare



# Moduli

- **Esportazione:**

- Utilizzando la parola chiave `export`, funzioni classi interfacce o variabili possono essere rese disponibili per l'uso in altri moduli

- **Importazione:**

- Utilizzando la parola chiave `import`, è possibile includere e utilizzare il codice esportato da altri moduli



# Esportazione

- **Esportazione Nominata:**
  - Esporta più elementi con nomi specifici.
    - `export function funzione1() { ... }`
    - `export function funzione2() { ... }`
- **Esportazione di Default**
  - Esporta un singolo elemento come predefinito.
    - `export default function funzionePrincipale() { ... }`



# Importazione

- **Importazione Nominata**

- Importa elementi specifici da un modulo
  - `import { funzione1, funzione2 } from './modulo'`

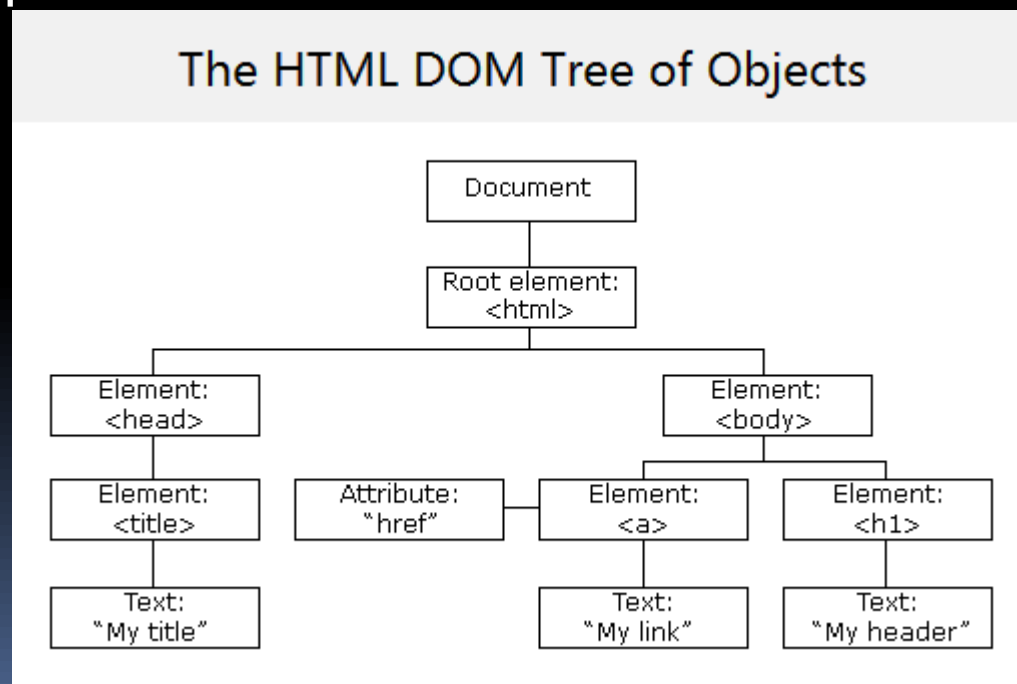
- **Importazione di Default**

- Importa l'elemento esportato di default.
  - `import funzionePrincipale from './modulo'`



# Document Object Model

- Il browser gestisce la pagina in memoria come una gerarchia di oggetti (struttura ad albero)
  - Tramite i quali ne controlla tutte le caratteristiche





# Document Object Model

- Il DOM permette di accedere e manipolare gli elementi
  - L'oggetto principale è il document che consente l'accesso all'albero
    - Esso mette a disposizione metodi per l'accesso ai diversi elementi
      - getElementById()
      - getElementsByTagName()
      - getElementsByClassName()
      - querySelector()
    - Su ogni elemento è possibile accedere agli attributi (in lettura e scrittura)
      - element.setAttribute(name, value)
      - let a = element.getAttribute(name)
    - O gestire gli "eventi"
      - element.addEventListener(name, function())



# Introduzione a React

- Libreria JavaScript per creare interfacce utente
- Sviluppata da Facebook
- Basata su componenti



# Vantaggi di React con TypeScript

- **Componenti Tipizzati**
  - Migliora la manutenibilità e la sicurezza del codice
- **Autocompletamento e IntelliSense**
  - Migliora l'esperienza di sviluppo
- **Errori di Compilazione**
  - Rileva errori prima dell'esecuzione

# Componenti

- Blocchi fondamentali di un'app React
  - Possono essere funzionali o di classe
  - Riutilizzabili e modulari

```
import React from 'react';

type SalutoProps = {
  nome: string;
};

const Saluto: React.FC<SalutoProps> = ({ nome }) => {
  return <h1>Ciao, {nome}!</h1>;
};
```

# JSX

- **JSX (JavaScript XML)**
  - Sintassi simile a HTML per descrivere l'interfaccia utente

```
const elemento: JSX.Element = <h1>Ciao, mondo!</h1>;
```

# Props

- Servono per passare dati ad un componente

```
type SalutoProps = {  
  nome: string;  
};  
  
const Saluto: React.FC<SalutoProps> = ({ nome }) => {  
  return <h1>Ciao, {nome}!</h1>;  
};
```

# State

- Gestione di componenti il cui stato cambia nel tempo
- Essi dipendono da “hook”

```
import React, { useState } from 'react';

const Contatore: React.FC = () => {
  const [conteggio, setConteggio] = useState<number>(0);

  return (
    <div>
      <p>Conteggio: {conteggio}</p>
      <button onClick={() => setConteggio(conteggio + 1)}>Incrementa</button>
    </div>
  );
};
```



# Eventi

- React gestisce eventi come click, submit, ecc.

```
const Bottone: React.FC = () => {  
  const handleClick = (event: React.MouseEvent<HTMLButtonElement>) => {  
    alert('Bottone cliccato!');  
  };  
  
  return <button onClick={handleClick}>Cliccami</button>;  
};
```