Fondamenti di programmazione

Adriano Venturini
Project Manager
Software Solutions Developer

Presentazione

- Lezione del 29 ottobre 2024
  - *Descrizione del corso*
  - *Obiettivi del corso*
  - *Struttura e contenuti del corso*
  - *Introduzione alla Programmazione e JavaScript (prima parte)*
  - *Esercizi*

Descrizione del Corso

Descrizione del corso

Descrizione del corso

- Questo corso offre un'introduzione esaustiva ai concetti fondamentali della programmazione e alle basi del linguaggio JavaScript.
- Questo corso si propone di fornire una solida base nello sviluppo web front-end.
- È pensato per chi si avvicina per la prima volta alla programmazione e anche per coloro che hanno già delle basi e desiderano approfondire le proprie conoscenze su JavaScript.
- Durante il corso, si apprenderanno i principi fondamentali della programmazione, operando con strutture di controllo, tipi di dati, gestione di input/output, fondamenti di orientamento agli oggetti, il tutto applicato in JavaScript.
- Al termine del corso, gli studenti saranno in grado di creare pagine web strutturate e stilizzate con HTML e CSS, e di aggiungere interazioni di base utilizzando JavaScript.

Presentazione del corso

Obiettivi del corso

Alla fine del corso, i partecipanti saranno in grado di:

- Comprendere i concetti fondamentali della programmazione (algoritmi, flusso di controllo, tipi di dati, funzioni/metodi).
- Applicare i Concetti di Programmazione in JavaScript per Creare Applicazioni Semplici.
- Utilizzare Strutture di Controllo per Implementare Logica Complessa.
- Comprendere e Implementare i Concetti di Base della Programmazione Orientata agli Oggetti (OOP).
- Gestire input e output da console e file.
- Risolvere problemi di programmazione utilizzando un approccio strutturato e modulare.
- Familiarizzare con l'ambiente di sviluppo integrato (IDE) Visual Studio Code.

Presentazione del corso

Struttura e contenuti del corso

- Introduzione alla Programmazione
- Fondamenti di HTML
- Fondamenti di CSS
- Introduzione a JavaScript
- Tipi di Dati e Operatori
- Condizioni e Strutture di Controllo
- Cicli e Iterazioni
- Funzioni e Scope
- Array e Oggetti
- Manipolazione del DOM
- Il Browser Object Model (BOM)
- Eventi e Interattività
- Programmazione Asincrona con JavaScript
- Pianificazione, sviluppo, presentazione e valutazione del Progetto Finale

Introduzione alla programmazione

Introduzione alla programmazione

Obiettivi:

- Familiarizzare con l'ambiente di sviluppo JavaScript e Visual Studio Code.
- Comprendere la sintassi di base JavaScript

Introduzione alla programmazione

Contenuti:

- Cos'è l'Informatica?
- Cos'è la programmazione?
- Introduzione ai linguaggi di programmazione
- Panoramica di JavaScript.
- Installazione e configurazione di Visual Studio Code.
- Installazione e configurazione di Notepad ++.
- Introduzione al Web e al Linguaggio HTML

Introduzione alla programmazione

Cos'è l'informatica?

- L'**informatica** è la scienza che studia il trattamento automatico delle informazioni, concentrandosi su come i dati possono essere rappresentati, elaborati, memorizzati e trasmessi utilizzando sistemi informatici.
  È una disciplina vasta che comprende teorie, metodologie, strumenti e tecnologie per risolvere problemi complessi e migliorare l'efficienza delle attività umane attraverso l'automazione.
- Il termine "informatica" deriva dall'unione delle parole "informazione" e "automatica", e rappresenta appunto la gestione e il trattamento delle informazioni attraverso sistemi automatici (computer).

Introduzione alla programmazione

Aspetti fondamentali dell'informatica:

- Elaborazione dei dati
- Rappresentazione delle informazioni
- Algoritmi e automazione
- Sistemi informatici
- Teoria della computazione
- Interazione uomo-macchina
- Intelligenza artificiale (IA)

Introduzione alla programmazione

Suddivisioni dell'informatica

L'informatica può essere suddivisa in molte aree specializzate. Ecco le principali:

- Informatica teorica:
- Ingegneria del software:
- Sistemi operativi e reti:
- Intelligenza artificiale e machine learning:
- Elaborazione grafica e computer grafica:
- Sicurezza informatica:
- Basi di dati:
- Interfaccia uomo-macchina (HCI):

Introduzione alla programmazione

Storia dell'informatica

L'informatica è una disciplina relativamente giovane, ma le sue radici affondano nell'antichità, quando l'uomo iniziò a sviluppare strumenti per facilitare i calcoli matematici. Nel corso del XX secolo, con l'invenzione dei computer elettronici, l'informatica ha conosciuto uno sviluppo esponenziale, che ha trasformato il mondo in modi senza precedenti.

Precursori storici dell'informatica

Anche se il concetto di computer come lo intendiamo oggi è nato nel XX secolo, esistono diversi **precursori storici** che hanno gettato le basi per la moderna tecnologia computazionale:

- **Abaco (circa 2000 a.C.)**
- **La Macchina Analitica di Charles Babbage (XIX secolo)**
- **La Macchina di Turing (1936)**

Nascita dei computer elettronici

Il vero salto di qualità nell'informatica si è verificato nel XX secolo con l'invenzione dei computer elettronici.

- **ENIAC (1945)**
- **Transistor e Circuiti Integrati (1950-1960)**

Introduzione alla programmazione

Storia dell'informatica

Sviluppo del software

Mentre l'hardware dei computer diventava sempre più potente e compatto, il **software** (i programmi che eseguono le operazioni sui computer) ha subito un'evoluzione parallela:

- **Primi linguaggi di programmazione (1950-1960)**
- **Sistema Operativo**.

Internet e comunicazione globale

Uno dei più grandi sviluppi della storia dell'informatica è l'invenzione e la diffusione di **Internet**. Nato come un progetto militare, Internet si è evoluto fino a diventare la più grande rete globale di comunicazione e scambio di informazioni.

- **ARPANET (1960)**
- **World Wide Web (1990)**

Introduzione alla programmazione

L'importanza dell'informatica nella società

L'informatica è diventata una parte integrante della società moderna, influenzando profondamente ogni aspetto della vita quotidiana e trasformando il modo in cui le persone comunicano, lavorano e interagiscono con il mondo.

- **Automazione**: L'informatica ha permesso l'automazione di compiti ripetitivi e complessi in quasi tutti i settori industriali e commerciali. Ciò ha aumentato la produttività e ridotto i costi operativi.
- **Rivoluzione digitale**: Le tecnologie informatiche hanno alimentato la rivoluzione digitale, trasformando industrie come il commercio, i media, la sanità, l'istruzione e l'intrattenimento. Ha aperto nuove possibilità di comunicazione globale e di innovazione tecnologica.
- **Lavoro e carriera**: La programmazione, la gestione dei dati e le tecnologie digitali sono oggi competenze fondamentali richieste in quasi tutti i settori. L'informatica offre una vasta gamma di opportunità lavorative, dalle startup innovative alle grandi aziende tecnologiche.

Introduzione alla programmazione

Dati e Informazioni

- I **dati** e le **informazioni** sono concetti fondamentali nell'informatica, nelle scienze dell'informazione e in molti altri campi.
  Sebbene siano termini spesso usati in modo intercambiabile nel linguaggio comune, hanno significati distinti e specifici quando vengono utilizzati in un contesto tecnico.
  In breve, i dati rappresentano elementi grezzi e non interpretati, mentre le informazioni sono il risultato dell'elaborazione e dell'interpretazione di quei dati.

Introduzione alla programmazione

Cos'è un dato?

I **dati** sono fatti, numeri, misurazioni, osservazioni o descrizioni che non hanno un significato intrinseco finché non vengono elaborati o interpretati.
Sono essenzialmente una rappresentazione grezza della realtà e, da soli, non sono particolarmente utili.

- **Caratteristiche dei dati**:
  - **Grezzi**
  - **Non strutturati o strutturati**
  - **Neutrali**
  - **Formati diversi**

Introduzione alla programmazione

Cos'è un'informazione?

Le **informazioni** sono il risultato dell'elaborazione, organizzazione e interpretazione dei dati.
Quando i dati vengono contestualizzati, organizzati in modo significativo e interpretati in un contesto specifico, diventano informazioni utili e comprensibili.

- **Caratteristiche delle informazioni**:
  - **Significative**
  - **Contestualizzate**
  - **Organizzate**
  - **Utilizzabili per la conoscenza**

Introduzione alla programmazione

Come si relazionano tra loro dati e informazioni?

Il processo di trasformazione dai dati alle informazioni avviene attraverso diverse fasi.
Queste fasi sono legate a operazioni di elaborazione, organizzazione, contestualizzazione e interpretazione.
L'informatica e la scienza dell'informazione si concentrano su come effettuare queste trasformazioni in modo efficiente.

- **Raccolta dei dati**
- **Organizzazione**
- **Elaborazione**
- **Interpretazione**

Introduzione alla programmazione

Il Ciclo Dati-Informazioni-Conoscenza

Una volta che i dati sono stati trasformati in informazioni, queste possono essere utilizzate per generare **conoscenza**. La conoscenza è il livello successivo, che permette di fare previsioni, prendere decisioni informate e risolvere problemi. Questo processo può essere descritto come il ciclo **DIKW** (Dati, Informazioni, Conoscenza, Saggezza):

- **Dati**: Elementi grezzi, senza significato.
- **Informazioni**: Dati elaborati e contestualizzati, che acquisiscono significato.
- **Conoscenza**: Le informazioni, applicate nel contesto giusto, consentono di acquisire comprensione e capacità di previsione.
- **Saggezza**: La capacità di prendere decisioni strategiche basate sulla conoscenza e sull'esperienza accumulata.

Introduzione alla programmazione

Differenza tra informazioni e conoscenza

- **Informazioni**: Sono il risultato dell'elaborazione dei dati.
  Ad esempio, sapere che il negozio ha venduto 200 articoli al giorno in media è un'informazione.
- **Conoscenza**: Rappresenta la comprensione o l'abilità di usare quell'informazione in modo utile.
  Ad esempio, decidere di aumentare il personale nei giorni con maggiori vendite sulla base delle informazioni raccolte è conoscenza.

Introduzione alla programmazione

Il Processo Input → Elaborazione → Output

- Il processo **Input → Elaborazione (Hardware/Software) → Output** è il cuore del funzionamento di un sistema informatico, ed è il meccanismo fondamentale attraverso cui i computer e le macchine automatizzate eseguono compiti. Questo ciclo è utilizzato per spiegare come i sistemi informatici ricevono dati (input), li processano (elaborazione) e restituiscono risultati (output).

Introduzione alla programmazione

Input (Dati in ingresso)

L'**input** è la prima fase del ciclo, in cui il sistema informatico riceve dati o comandi da fonti esterne.
Questi dati possono provenire da diverse tipologie di dispositivi o utenti.

Tipologie di input:

- Dati utente
- Dati da sensori
- File o dati digitali
- Segnali audio o video
- Reti

Introduzione alla programmazione

Elaborazione

L'**elaborazione** è la fase in cui i dati in ingresso vengono manipolati e trasformati in un formato più utile o significativo. Questa fase coinvolge il **hardware** e il **software** del sistema, che lavorano insieme per eseguire calcoli, prendere decisioni, applicare algoritmi e, in generale, processare i dati.

Fasi dell'elaborazione:

- **Interpretazione e decodifica**
- **Calcolo**
- **Trasformazione**
- **Archiviazione**
- **Decisioni logiche**

Introduzione alla programmazione

Coinvolgimento del hardware e del software:

- **Hardware**:
  - Il processore (**CPU**) esegue le istruzioni del software, eseguendo operazioni matematiche e logiche.
  - La memoria (RAM) memorizza temporaneamente i dati che vengono elaborati.
  - I dispositivi di archiviazione (es. hard disk o SSD) mantengono i dati a lungo termine.
- **Software**:
  - Il sistema operativo gestisce le risorse hardware e coordina l'esecuzione dei programmi.
  - Le applicazioni (es. un software per l'elaborazione di testi, un programma di editing video, o un sistema di gestione di database) contengono gli algoritmi e le logiche di elaborazione necessarie per trasformare i dati in informazioni utili.

Introduzione alla programmazione

Output (Risultato dell'elaborazione)

L'**output** è la fase finale del ciclo, in cui i risultati dell'elaborazione vengono restituiti all'utente o ad altri sistemi.
L'output rappresenta l'informazione derivata dai dati elaborati e può assumere diverse forme.

Tipologie di output:

- **Visuale**: Dati presentati su schermi, monitor o display sotto forma di testo, grafici o immagini.
- **Stampato**: Documenti o immagini stampate su carta attraverso una stampante.
- **Audio**: Suoni o parlato generati da altoparlanti o cuffie, come un sistema di navigazione che fornisce istruzioni vocali.
- **Segnale**: Output sotto forma di segnali inviati ad altri dispositivi o sistemi, come il controllo di un robot o l'apertura di una porta automatica.
- **Dati digitali**: Output inviati a file, database o altri sistemi informatici, come i risultati di un'elaborazione inviati a un server.

Introduzione alla programmazione

Relazione tra le fasi del processo Input → Elaborazione → Output

Le tre fasi di **input**, **elaborazione** e **output** sono strettamente collegate tra loro in un ciclo continuo.
Ogni fase dipende dalla precedente per funzionare correttamente, e il risultato finale (l'output) è influenzato dalla qualità e dalla natura dei dati di input e dall'efficacia del processo di elaborazione.

Esempio pratico del ciclo:

Consideriamo un semplice esempio di calcolatrice:

- **Input**: L'utente inserisce i numeri 5 e 3 e l'operazione + (somma) tramite la tastiera della calcolatrice.
- **Elaborazione**: Il processore della calcolatrice esegue l'operazione di somma: 5 + 3
- **Output**: Il risultato, 8, viene visualizzato sul display della calcolatrice.
- Questo ciclo si può ripetere continuamente: l'utente inserisce nuovi input, i dati vengono elaborati e nuovi output vengono visualizzati.

Introduzione alla programmazione

Come hardware e software collaborano nel processo

L'efficienza di un sistema informatico dipende dall'interazione tra hardware e software.

- **Il ruolo dell'hardware**:
  - Il **processore** (CPU) è l'unità che esegue le istruzioni del software. Per esempio, in una CPU, ogni istruzione viene letta, decodificata e eseguita tramite cicli di clock.
  - La **memoria RAM** consente di archiviare temporaneamente i dati su cui il processore sta lavorando. Se la RAM non è sufficiente, il sistema può diventare lento perché il processore deve accedere ai dati sull'hard disk, che è più lento.
  - **Dispositivi di input/output (I/O)**, come tastiera, mouse, monitor, altoparlanti, microfoni, stampanti e scanner, consentono al sistema di interagire con il mondo esterno.

Introduzione alla programmazione

Come hardware e software collaborano nel processo

Il ruolo del software:

- Il **sistema operativo** gestisce tutte le risorse hardware del sistema, consentendo al software applicativo di interagire con il computer senza dover gestire direttamente l'hardware.
- **Software applicativo**: Esegue i compiti specifici per l'utente, come calcoli matematici, elaborazione testi, o manipolazione di immagini.
- **Driver**: Sono componenti software che fungono da intermediari tra il sistema operativo e l'hardware, traducendo i comandi del sistema operativo in operazioni comprensibili dall'hardware.

Introduzione alla programmazione

Cos'è la programmazione?

La **programmazione** è il processo di scrittura di istruzioni che un computer può seguire per eseguire operazioni o risolvere problemi. Queste istruzioni, scritte in un linguaggio di programmazione, costituiscono un programma. I programmatori utilizzano linguaggi formali che possono essere tradotti in **codice macchina**, ossia il linguaggio binario comprensibile dall'hardware.

In sintesi, la programmazione consiste nel:

- Analizzare
- Progettare
- Tradurre
- Testare
- Correggere eventuali errori (debugging).

Introduzione alla programmazione

I **linguaggi di programmazione** sono strumenti utilizzati dai programmatori per scrivere il codice che verrà eseguito dal computer.
Esistono vari tipi di linguaggi di programmazione, classificati in base a come il loro codice sorgente viene trasformato in un formato che il computer può comprendere ed eseguire.

Le tre principali categorie sono:

- **Linguaggi compilati**
- **Linguaggi interpretati**
- **Linguaggi semi-compilati**

Queste differenze influiscono sulle prestazioni, sulla portabilità e sulla velocità di sviluppo di un programma.

Introduzione alla programmazione

Linguaggi Compilati

Un **linguaggio compilato** è un linguaggio il cui codice sorgente viene tradotto interamente in un file eseguibile (codice macchina) prima dell'esecuzione.
Questo processo di traduzione viene fatto da un programma chiamato **compilatore**.

- Il compilatore legge il codice sorgente (scritto nel linguaggio di programmazione) e lo converte in **codice macchina**, che può essere eseguito direttamente dalla CPU del computer.
- Una volta compilato, il programma è indipendente dal linguaggio di programmazione e non necessita più del codice sorgente o del compilatore per essere eseguito.

Introduzione alla programmazione

Linguaggi Compilati

Vantaggi:

- **Prestazioni elevate**: Poiché il codice viene tradotto direttamente in istruzioni di basso livello eseguibili dalla CPU, le prestazioni di un programma compilato sono molto elevate rispetto a quelli interpretati.
- **Esecuzione autonoma**: Una volta compilato, il programma può essere eseguito senza la necessità di un ambiente di sviluppo o un interprete.

Introduzione alla programmazione

Linguaggi Compilati

Svantaggi:

- **Maggiore complessità nel debugging**: Il codice sorgente e il codice compilato non sono uguali, quindi è più difficile individuare errori.
- **Minore portabilità**: Il codice compilato dipende dal sistema operativo e dall'architettura hardware per cui è stato compilato. Se si vuole eseguire il programma su un'altra piattaforma, è necessario ricompilare il codice.
- **Esempi di linguaggi compilati:**
- C, C++, Rust, Fortran

Introduzione alla programmazione

Linguaggi Interpretati

Un **linguaggio interpretato** è un linguaggio il cui codice sorgente viene eseguito direttamente da un programma chiamato **interprete**, senza la necessità di essere compilato in codice macchina.

- L'interprete legge il codice sorgente riga per riga, lo analizza e lo esegue immediatamente.
- Non viene generato un file eseguibile indipendente; ogni volta che il programma viene eseguito, l'interprete deve tradurre e interpretare il codice.

Introduzione alla programmazione

Linguaggi Interpretati

Vantaggi:

- **Maggiore flessibilità durante lo sviluppo**: I linguaggi interpretati consentono modifiche rapide al codice e facilitano il debugging poiché il programmatore può eseguire direttamente il codice senza ricompilarlo.
- **Portabilità**: Il codice sorgente è generalmente portabile tra diverse piattaforme, poiché è l'interprete che si occupa di gestire la compatibilità con il sistema.

Introduzione alla programmazione

Linguaggi Interpretati

Svantaggi:

- **Prestazioni inferiori**: Poiché l'interprete deve tradurre il codice in tempo reale durante l'esecuzione, i programmi interpretati sono generalmente più lenti rispetto ai linguaggi compilati.
- **Necessità dell'interprete**: Per eseguire il codice, è sempre necessario l'interprete installato sul sistema.

Esempi di linguaggi interpretati:

- Python, JavaScript, Ruby, PHP

Introduzione alla programmazione

Linguaggi Semi-Compilati

I **linguaggi semi-compilati** combinano gli aspetti sia dei linguaggi compilati che di quelli interpretati. In questo caso, il codice sorgente viene compilato in un **codice intermedio** (spesso chiamato **bytecode**) che non è eseguibile direttamente dalla CPU del computer, ma che può essere eseguito da un **interprete speciale** o una **macchina virtuale**.

- Questo approccio consente di ottenere prestazioni migliori rispetto ai linguaggi completamente interpretati, mantenendo una buona portabilità.
- Il bytecode è indipendente dalla piattaforma, quindi può essere eseguito su qualsiasi sistema che abbia l'interprete o la macchina virtuale appropriata.

Introduzione alla programmazione

Linguaggi Semi-Compilati

Vantaggi:

- **Buon compromesso tra prestazioni e portabilità**: Poiché il codice sorgente viene compilato in bytecode, le prestazioni sono migliori rispetto ai linguaggi completamente interpretati. Inoltre, essendo il bytecode **indipendente dalla piattaforma**, il programma può essere eseguito su diverse piattaforme senza ricompilazione.
- **Portabilità**: La macchina virtuale o l'interprete necessario per eseguire il bytecode può essere disponibile su diverse piattaforme, rendendo i programmi semi-compilati molto portabili.

Introduzione alla programmazione

Linguaggi Semi-Compilati

Svantaggi:

- **Performance non ottimale**: Anche se il bytecode migliora le prestazioni rispetto ai linguaggi interpretati, rimane comunque più lento rispetto ai linguaggi compilati direttamente in codice macchina.
- **Necessità di un ambiente runtime**: Per eseguire il bytecode, è necessario che sia presente un interprete specifico o una macchina virtuale (es. la JVM per Java o il CLR per C#).

Introduzione alla programmazione

Esempi di linguaggi semi-compilati:

- **Java**: Il codice sorgente Java viene compilato in bytecode, che viene poi eseguito dalla **Java Virtual Machine (JVM)**.
- **C#**: Il codice sorgente C# viene compilato in un formato intermedio chiamato **Microsoft Intermediate Language (MSIL)**, eseguito dal **Common Language Runtime (CLR)**.

Introduzione alla programmazione

Il ruolo di JavaScript come linguaggio interpretato.

- JavaScript è definito come un **linguaggio interpretato**, il che significa che il suo codice viene letto ed eseguito riga per riga da un interprete, piuttosto che essere pre-compilato. Questo aspetto ha importanti implicazioni sul funzionamento e sull'utilizzo di JavaScript nel contesto del web, dove viene eseguito direttamente dal browser dell'utente.

Introduzione alla programmazione

Il Ruolo di JavaScript come Linguaggio Interpretato nel Browser

- JavaScript viene comunemente eseguito nel contesto del browser web (come Chrome, Firefox, Safari), che contiene un motore JavaScript in grado di interpretare ed eseguire il codice JavaScript. Ogni browser moderno possiede un motore JavaScript dedicato, ad esempio:
  - **V8** in Google Chrome;
  - **SpiderMonkey** in Firefox;
  - **JavaScriptCore** (o Nitro) in Safari.
- Questi motori leggono il codice JavaScript direttamente all'interno del browser e lo interpretano per produrre comportamenti dinamici sulle pagine web.

Introduzione alla programmazione

Ciclo di vita di un programma JavaScript

- Il ciclo di vita di un programma JavaScript può essere suddiviso in una serie di fasi che spiegano come il codice JavaScript viene caricato, interpretato, eseguito e completato all'interno di un browser o di un ambiente runtime. Comprendere queste fasi è fondamentale per sapere come il browser gestisce il codice JavaScript e quali eventi avvengono "dietro le quinte".

Introduzione alla programmazione

Riepilogo del Ciclo di Vita di un Programma JavaScript

- **Caricamento e Parsing del codice HTML**: Il browser legge il file HTML, costruendo il DOM e individuando gli script JavaScript.
- **Scaricamento ed Esecuzione del codice JavaScript**: Gli script vengono scaricati e interpretati in base alla loro posizione e attributi (async, defer).
- **Parsing e Compilazione del codice JavaScript**: Il motore JavaScript interpreta e trasforma il codice in una forma che può essere eseguita.
- **Esecuzione**: Il codice JavaScript viene eseguito riga per riga, gestendo variabili, funzioni e la logica definita.
- **Manipolazione del DOM**: Il codice può modificare la struttura e il contenuto della pagina, rendendo il sito interattivo.
- **Garbage Collection**: Il motore JavaScript rimuove automaticamente dalla memoria le risorse non utilizzate.
- **Conclusione**: Al termine dell'esecuzione, la pagina attende ulteriori interazioni o eventi fino a quando non viene chiusa o ricaricata.

Introduzione alla programmazione

Linguaggio di Programmazione

- Un **linguaggio di programmazione** è un sistema formale che fornisce un insieme di regole per descrivere computazioni e manipolare dati.
  I programmatori usano i linguaggi di programmazione per dare istruzioni precise ai computer su come eseguire compiti specifici.
  Ogni linguaggio di programmazione ha una **sintassi** (insieme di regole grammaticali) e una **semantica** (significato delle istruzioni) che determinano come le istruzioni vengono interpretate ed eseguite.

Introduzione alla programmazione

Tipi di linguaggi di programmazione

I linguaggi di programmazione possono essere classificati in diversi modi. Una delle distinzioni principali è tra:

- **Linguaggi di basso livello**: Questi linguaggi sono più vicini all'hardware del computer e consentono al programmatore di controllare direttamente l'architettura della macchina, come la gestione della memoria. Gli esempi includono il linguaggio assembly e il linguaggio macchina. Sono più difficili da usare, ma offrono un alto grado di controllo sulle operazioni del sistema.
- **Linguaggi di alto livello**: Sono più astratti rispetto al linguaggio macchina, offrendo sintassi e costrutti più simili al linguaggio umano. Questi linguaggi sono progettati per essere più facili da leggere, scrivere e mantenere. Gli esempi includono Python, Java, C#, C++, Ruby e molti altri.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

Ogni linguaggio di programmazione possiede determinati componenti essenziali:

a. **Sintassi**
   La sintassi di un linguaggio di programmazione definisce il modo in cui le istruzioni devono essere scritte.
   Ad esempio, in un linguaggio come Python, l'indentazione è essenziale per definire blocchi di codice, mentre in C o Java i blocchi sono definiti da parentesi graffe {}.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

b. Tipi di dati

I linguaggi di programmazione offrono vari tipi di dati, come:

- **Tipi primitivi**: Questi includono interi (int), numeri in virgola mobile (float o double), caratteri (char), e booleani (bool), che rappresentano i dati di base con cui un programma lavora.
- **Tipi complessi**: Molti linguaggi supportano strutture più complesse come **array**, **stringhe**, **liste**, **set**, **mappe** o **dizionari**.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

c. Operatori

Gli operatori sono simboli o parole riservate che eseguono operazioni sui dati. Possono essere classificati come:

- **Operatori aritmetici** (+, -, *, /) per effettuare calcoli matematici.
- **Operatori di confronto** (==, !=, >, <) per confrontare valori.
- **Operatori logici** (&&, ||, !) per combinare condizioni logiche.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

d. Controllo di flusso

Le istruzioni di controllo di flusso decidono l'ordine in cui le istruzioni vengono eseguite.

Questi includono:

- **Condizionali**: Le istruzioni if, else, switch permettono al programma di prendere decisioni in base a determinate condizioni.
- **Cicli**: Le istruzioni for, while, do-while permettono di ripetere un blocco di codice fino a che una condizione specifica è soddisfatta.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

e. Funzioni o metodi

Le **funzioni** (o **metodi** in linguaggi orientati agli oggetti) sono blocchi di codice riutilizzabili che eseguono operazioni specifiche.

Una funzione può ricevere input sotto forma di **parametri**, eseguire operazioni e restituire un risultato.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

f. Strutture dati

Le strutture dati permettono di organizzare e gestire grandi quantità di dati in modo efficiente.
Esempi comuni includono:

- **Array**: Una collezione di elementi dello stesso tipo, accessibili tramite indici.
- **Liste**: Una collezione dinamica di elementi che possono essere di tipi diversi e possono crescere in modo dinamico.
- **HashMap** o **dizionari**: Collezioni di coppie chiave-valore, che permettono un accesso efficiente ai dati.

Introduzione alla programmazione

Componenti di un linguaggio di programmazione

g. Eccezioni e gestione degli errori

I linguaggi di programmazione offrono meccanismi per gestire gli errori che si verificano durante l'esecuzione del programma, come errori di divisione per zero o accessi a indici di array non validi.
La gestione delle eccezioni permette di catturare questi errori ed evitare che il programma si arresti in modo anomalo.

Introduzione alla programmazione

Paradigmi di programmazione

I linguaggi di programmazione seguono diversi **paradigmi** (stili o modelli) di programmazione:

- **Programmazione procedurale**: Si basa sull'uso di procedure o funzioni. Il programma è una sequenza di istruzioni che vengono eseguite in ordine. Linguaggi come C sono principalmente procedurali.
- **Programmazione orientata agli oggetti (OOP)**: Si basa sull'uso di oggetti, che sono istanze di classi. Le classi incapsulano dati (attributi) e comportamenti (metodi), e favoriscono il riutilizzo del codice e la modularità. C#, Java, e Python supportano l'OOP.

Introduzione alla programmazione

Paradigmi di programmazione

- **Programmazione funzionale**: è un paradigma di programmazione che si concentra sull'uso di **funzioni matematiche** per risolvere problemi.
  In questo approccio, il programma viene costruito combinando funzioni, senza fare uso esplicito di variabili mutabili o di stati che cambiano.
- **Programmazione logica**: Si basa su regole logiche e deduzioni. Il linguaggio più noto per la programmazione logica è Prolog.

Introduzione alla programmazione

Ecosistema e librerie

Ogni linguaggio di programmazione ha un **ecosistema** di librerie e framework che estendono le sue capacità.

- Le **librerie** sono collezioni di funzioni e strumenti che i programmatori possono riutilizzare per risolvere problemi comuni (come la gestione dei file, la rete, o la creazione di interfacce utente).
- I **framework**, invece, forniscono strutture predefinite per costruire applicazioni complesse, come applicazioni web o mobili.

Introduzione alla programmazione

JavaScript: un linguaggio moderno e versatile

- JavaScript è un linguaggio di programmazione **moderno** e **versatile**, progettato inizialmente per aggiungere interattività alle pagine web, ma che è diventato una delle tecnologie più potenti e flessibili nel mondo dello sviluppo software. La sua versatilità e le continue evoluzioni ne hanno fatto uno dei linguaggi più utilizzati e richiesti, sia per lo sviluppo frontend (lato client) che backend (lato server).

Introduzione alla programmazione

Caratteristiche Moderne di JavaScript

JavaScript è evoluto enormemente sin dalla sua creazione nel 1995. Alcune delle sue caratteristiche più moderne includono:

- **Sintassi ECMAScript 6+ (ES6 e versioni successive)**: Le specifiche ES6 (introdotte nel 2015) hanno portato grandi miglioramenti, come le **arrow functions**, **let** e **const** per dichiarare variabili con ambito di blocco, **template literals** per una gestione più comoda delle stringhe, **destructuring** per facilitare la manipolazione di oggetti e array, e molto altro. Questi aggiornamenti rendono JavaScript più leggibile e facile da mantenere.
- **Moduli (Modules)**: Con il supporto per i moduli JavaScript (import ed export), gli sviluppatori possono organizzare il codice in file distinti e gestire le dipendenze in modo più efficiente. Questo migliora la modularità, semplifica la manutenzione e permette una suddivisione naturale dei progetti.

Introduzione alla programmazione

Caratteristiche Moderne di JavaScript

- **Promise, async/await**: JavaScript ha evoluto il suo modello asincrono introducendo le Promise, che permettono una gestione semplificata delle operazioni asincrone, e il costrutto async/await, che rende più facile e leggibile la scrittura di codice asincrono, simile al codice sincrono. Questo è particolarmente utile per operazioni come chiamate API o manipolazioni di dati esterni.
- **Programmazione Orientata agli Oggetti (OOP)**: Sebbene JavaScript sia un linguaggio orientato agli oggetti basato su prototipi e non su classi come Java o C++, ha integrato il supporto alle classi con la sintassi class a partire da ES6, permettendo una struttura più simile a quella della programmazione OOP tradizionale. Questo consente agli sviluppatori di implementare concetti OOP come **ereditarietà**, **incapsulamento** e **polimorfismo** in modo intuitivo.

Introduzione alla programmazione

Versatilità di JavaScript: Frontend, Backend e Oltre

JavaScript è un linguaggio **multiparadigma** e si adatta a diversi stili di programmazione, rendendolo estremamente versatile:

- **Frontend Development**: JavaScript è il linguaggio principale per lo sviluppo frontend. Con l'uso di framework e librerie come **React**, **Vue.js** e **Angular**, gli sviluppatori possono creare interfacce utente dinamiche, reattive e complesse. Questi strumenti permettono di gestire il DOM in modo efficiente e offrono componenti riutilizzabili che semplificano il lavoro su progetti di grandi dimensioni.
- **Backend Development**: Grazie a **Node.js**, JavaScript può essere eseguito anche lato server. Node.js ha aperto nuove possibilità per lo sviluppo backend, permettendo di gestire operazioni di rete, database, file system e molto altro. Con strumenti come **Express** e **NestJS**, JavaScript viene utilizzato per creare server scalabili e performanti, con il vantaggio di poter utilizzare lo stesso linguaggio per l'intera applicazione (frontend e backend).

Introduzione alla programmazione

Versatilità di JavaScript: Frontend, Backend e Oltre

- **Full-Stack Development**: La possibilità di utilizzare JavaScript sia lato client che lato server ha reso popolare il concetto di **full-stack development**. Gli sviluppatori full-stack utilizzano JavaScript per costruire applicazioni complete e gestire ogni aspetto del ciclo di vita del software, con strumenti e stack come **MEAN** (MongoDB, Express, Angular, Node.js) e **MERN** (MongoDB, Express, React, Node.js).
- **Mobile Development**: Con l'avvento di **React Native** e **Ionic**, JavaScript è utilizzato per creare applicazioni mobile native. React Native, ad esempio, consente di scrivere codice JavaScript che viene tradotto in codice nativo per iOS e Android, consentendo agli sviluppatori di creare app mobile con una singola codebase.

Introduzione alla programmazione

Versatilità di JavaScript: Frontend, Backend e Oltre

- **Desktop Application Development**: Grazie a strumenti come **Electron**, è possibile sviluppare applicazioni desktop multipiattaforma (Windows, MacOS e Linux) utilizzando JavaScript, HTML e CSS.
  Applicazioni popolari come Visual Studio Code e Slack sono state sviluppate con Electron, dimostrando l'efficacia di JavaScript anche in questo contesto.
- **Internet of Things (IoT)**: JavaScript sta guadagnando terreno anche nel mondo dell'IoT, grazie a librerie e framework come **Johnny-Five** e **Espruino** che consentono di programmare dispositivi hardware.
  Questo rende JavaScript uno strumento potente per la programmazione di dispositivi connessi.

Introduzione alla programmazione

Installazione di Visual Studio Code

Visual Studio Code è un editor di codice gratuito e open-source sviluppato da Microsoft, molto apprezzato per la sua leggerezza e le sue potenti funzionalità. Per installarlo:

- **Download di Visual Studio Code**:
  - Vai sul sito ufficiale di Visual Studio Code: https://code.visualstudio.com/.
  - Seleziona la versione per il tuo sistema operativo (Windows, macOS, o Linux).
  - Scarica il file di installazione e avvialo.
- **Installazione**:
  - Su **Windows**: Esegui il file .exe e segui le istruzioni. È consigliabile selezionare l'opzione per aggiungere Visual Studio Code al "PATH" per facilitare l'avvio da riga di comando.
  - Su **macOS**: Trascina l'app nella cartella Applicazioni.
  - Su **Linux**: Segui le istruzioni sul sito per installarlo tramite il pacchetto appropriato o da terminale.
- **Avvio di Visual Studio Code**:
  - Una volta installato, avvia Visual Studio Code. Se stai usando la riga di comando, puoi digitare code (se l'hai aggiunto al PATH) per avviarlo direttamente.

Introduzione alla programmazione

Installazione di Notepad++

- **Scarica Notepad++**:
  - Vai sul sito ufficiale di Notepad++: https://notepad-plus-plus.org/.
  - Clicca su **Download** e seleziona la versione più recente compatibile con il tuo sistema operativo (Windows).
- **Installa Notepad++**:
  - Apri il file scaricato (file .exe) e segui le istruzioni di installazione.
  - Puoi scegliere di installare Notepad++ nella directory predefinita o in un'altra posizione a tua scelta.
  - Durante l'installazione, puoi selezionare opzioni aggiuntive come creare un'icona sul desktop.
- **Avvia Notepad++**:
  - Dopo aver completato l'installazione, avvia Notepad++ dal menu Start o dall'icona sul desktop.

Introduzione alla programmazione

Configurazione di Notepad++ per lo Sviluppo in JavaScript

Notepad++ supporta nativamente la sintassi JavaScript, ma puoi migliorare l'esperienza con alcuni plugin e configurazioni.

A. Impostazione del Linguaggio JavaScript

- Notepad++ riconosce la sintassi JavaScript automaticamente per i file con estensione .js. Per abilitare la sintassi JavaScript su un file:
  - Vai su Language (Linguaggio) nel menu in alto.
  - Seleziona J e poi **JavaScript**.
  - In alternativa, puoi semplicemente salvare il file con l'estensione .js (ad esempio, script.js), e Notepad++ applicherà automaticamente la colorazione sintattica.

Introduzione alla programmazione

Configurazione di Notepad++ per lo Sviluppo in JavaScript

B. Attivazione dell'Evidenziazione della Sintassi

- L'evidenziazione della sintassi è utile per migliorare la leggibilità del codice. Notepad++ applica una colorazione predefinita, ma puoi personalizzarla:
  - Vai su Settings (Impostazioni) > Style Configurator… (Configuratore di Stile).
  - Nel menu a sinistra, seleziona il linguaggio **JavaScript**.
  - Qui puoi cambiare i colori per parole chiave, commenti, stringhe e altri elementi di sintassi JavaScript. Personalizza come preferisci per rendere il codice più leggibile.

Introduzione alla programmazione

Salvataggio e Test su Browser (per codice HTML con JavaScript)

Codice JavaScript direttamente in un file HTML

- Con Notepad ++, crea un nuovo file HTML:
  - Vedi esempio HTML TestLog.html
- Salva il file con estensione .html e aprilo in un browser per visualizzare i risultati.
- Puoi vedere i messaggi di console.log aprendo gli **Strumenti per sviluppatori** (F12) nel browser, e andando alla scheda **Console**.

Introduzione alla programmazione

Conclusione

- Ora Notepad++ è configurato per lo sviluppo di base in JavaScript. Nonostante sia un editor semplice, con queste impostazioni puoi creare un ambiente produttivo, perfetto per chi cerca un editor leggero senza le complessità di un IDE.
- Notepad++ può essere utilizzato con Node.js per eseguire JavaScript o con un browser per testare i risultati direttamente nelle pagine web.

Introduzione alla programmazione

Cos'è il Web

- Il World Wide Web (WWW) è una collezione di documenti e risorse collegati tra loro tramite link e URL. I documenti web sono accessibili tramite internet, attraverso i browser web come Google Chrome, Firefox o Safari. Ogni pagina web è scritta in linguaggi specifici come HTML, CSS, e JavaScript.

Cos'è HTML

- HTML (HyperText Markup Language) è il linguaggio di markup utilizzato per strutturare e presentare contenuti su pagine web. È un linguaggio basato su tag che consente di creare testi, immagini, link e form, fornendo la struttura della pagina web. HTML non è un linguaggio di programmazione, ma un linguaggio di markup che definisce la struttura dei documenti.

Un **documento HTML** è composto principalmente da:

- **Tag**: Elementi che identificano il tipo di contenuto o la struttura, ad esempio <h1>, <p>, <img>.
- **Attributi**: Forniscono informazioni aggiuntive sui tag, come src per un'immagine o href per un link.
- **Contenuto**: Il testo, le immagini o altri elementi che appaiono nella pagina.

Introduzione alla programmazione

1. **Struttura di Base di una Pagina HTML**
   - Vedi file: Struttura_Base_HTML.pdf
2. **Tag HTML di Base**
   - Vedi file: Tag_HTML.pdf
3. **Esempio Completo di Pagina HTML**
   - Vedi file: Esempio_HTML.pdf
4. **Aggiunta di CSS per lo Stile**
   - Vedi file: pagina_con_css.pdf

Grazie per l'attenzione

Adriano Venturini