ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Corso di Laurea in Informatica

Pensare in codice

Una rassegna sistematica della letteratura sui rapporti tra programmazione e abilità cognitive.

Relatore:
Chiar.mo Prof.
Simone Martini

Presentata da:
Chiara Tosadori

III sessione marzo 2025

Anno Accademico 2023/2024

A Zio Gabriele, Nonna Gina e Nonno Bepi,
che vivono nei miei ricordi
e hanno dato forza al mio cammino.

Indice

Elenco delle tabelle

Elenco delle figure

Introduzione

Nel contesto contemporaneo, la programmazione è rapidamente emersa come una competenza fondamentale in ambito professionale. Con l'avanzare della digitalizzazione e l'adozione di tecnologie innovative in una vasta gamma di settori – dall'economia alla sanità, dall'ingegneria all'istruzione – la capacità di scrivere codice non è più una prerogativa esclusiva dei professionisti IT. Questa diffusione capillare e massiccia della programmazione, in particolare nel contesto educativo, ha portato ad interrogarsi sul suo impatto più profondo: si tratta solo di una competenza tecnica fine a sé stessa, o è in grado di apportare cambiamenti significativi alle nostre capacità cognitive? Il cervello di un programmatore è diverso da quello di chi non programma? E, se sì, in che misura le competenze di programmazione possono plasmare il pensiero e influenzare le capacità di problem solving?

Portando all'estremo questa visione, non è difficile immaginare che diversi linguaggi di programmazione possano modellare il pensiero in modi differenti, proprio come le lingue naturali influenzano la percezione e la struttura del ragionamento. Teorie linguistiche fondamentali, come quelle di Chomsky e l'ipotesi di Sapir-Whorf, hanno suggerito che il linguaggio naturale potesse essere uno strumento del pensiero umano. In particolare, soprattutto secondo Whorf, le lingue parlate differiscono in modi psicologicamente significativi, tanto che l'apprendimento della grammatica e del vocabolario di una lingua può influenzare il pensiero in determinate direzioni. Un tempo, questo tema era considerato un tabù in psicologia e linguistica, ma oggi è stato ampiamente rivalutato, con centinaia di studi che hanno evidenziato i legami tra linguaggio, cognizione e percezione Mitchum (2017); Bloom and Keil (2001); Zlatev and Blomberg (2015), suggerendo addirittura la visione del linguaggio come uno strumento per la programmazione cognitiva Lupyan and Bergen (2016).

Analogamente i linguaggi di programmazione potrebbero plasmare in maniera profonda e differente il pensiero e l'immaginazione. Le neuroscienze iniziano a supportare questa ipotesi: studi di fMRI mostrano che il cervello di un programmatore si attiva in modo simile a quando si affrontano problemi logici e matematici, ma anche a quando si impara una lingua straniera. Questo non sorprende, poiché i linguaggi

di programmazione condividono molte caratteristiche strutturali con le lingue naturali, come le regole sintattiche, la composizione gerarchica (Fedorenko et al. (2019)) e persino la presenza di semantica e pragmatica. Inoltre, i linguaggi di programmazione svolgono anche un ruolo conversazionale che si estende sia nel rapporto tra esseri umani, sia in quello tra l'uomo e la macchina Zemanek (1966).

Tuttavia, a differenza delle lingue naturali, i linguaggi di programmazione sono rigidamente formalizzati e privi di ambiguità. Ma forse sono proprio queste caratteristiche, imposte in maniera diversa dai molteplici paradigmi di programmazione, a influenzare il modo in cui le idee vengono espresse e implementate, forzando i programmatori ad adattarsi a queste regole e modellando il loro approccio ai problemi.

Da queste domande è nata una rassegna sistematica della letteratura (SLR) che mira a esaminare la relazione quasi isomorfa tra pensiero, processi cognitivi, coding e linguaggi di programmazione. L'obiettivo di questa ricerca è mappare lo stato attuale della letteratura sull'argomento e fare chiarezza sulle ipotesi che si sono sviluppate nel corso di oltre 50 anni di studi sulla psicologia della programmazione. Verranno analizzate le modalità con cui diversi paradigmi di programmazione – come quello procedurale, orientato agli oggetti e funzionale – possono incidere sui processi cognitivi dei programmatori. Inoltre, saranno esaminati gli effetti di questi linguaggi sulla creatività e sulla capacità di problem solving.

Le implicazioni di questa analisi sono molteplici e rilevanti: tali ricerche potrebbero favorire l'adozione di linguaggi più in sintonia con il profilo cognitivo degli sviluppatori, promuovendo così creatività e produttività. Inoltre, potrebbero fornire spunti per la progettazione di nuovi linguaggi meglio rispondenti alle sfide cognitive della programmazione e capaci di facilitare un apprendimento più intuitivo e personalizzato. Ma soprattutto, questa analisi potrebbe contribuire a chiarire il ruolo della programmazione nell'insegnamento, approfondendo sia il perché e in quali contesti essa debba essere insegnata, sia il come, evidenziando le strategie e i linguaggi più efficaci. In questo modo, la programmazione potrebbe diventare il catalizzatore cognitivo che si sospetta possa essere.

Struttura della tesi

Il lavoro è organizzato come segue:

- Nel **Capitolo 1**, verrà descritta la metodologia adottata per la selezione degli studi, comprendendo le domande di ricerca, i database consultati, i criteri di inclusione ed esclusione.

- Nel **Capitolo 2**, verranno presentati i risultati della rassegna, analizzando come la programmazione influenzi le abilità cognitive, il rapporto tra coding e linguaggio naturale, il modo in cui i diversi linguaggi di programmazione modellano il pensiero e il legame tra programmazione e creatività.
- Nel **Capitolo 3**, verranno analizzate le conclusioni della rassegna , discutendo anche dei limiti della ricerca, del ruolo dell'insegnamento nella programmazione e delle possibili applicazioni concrete dei risultati emersi.

Capitolo 1

Metodologia e selezione degli studi

Questo capitolo descrive la metodologia adottata per condurre una rassegna sistematica della letteratura, finalizzata a esplorare i contributi accademici che trattano il rapporto tra processi cognitivi, coding e linguaggi di programmazione, con un focus sulle abilità creative.

È stata scelta la rassegna sistematica (SLR: Systematic Literature Review) perché permette di analizzare in modo approfondito lo stato dell'arte in questo specifico ambito scientifico, offrendo nuove prospettive di ricerca. Le SLR sono, infatti, essenziali per aggregare e sintetizzare i risultati degli studi precedenti, fornendo così una visione d'insieme del tema e mettendo in evidenza le diverse metodologie adottate.

Questa rassegna si ispira alle linee guida di Kitchenham (2004) e si basa sul protocollo PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses), entrambi riconosciuti e utilizzati a livello internazionale per la conduzione di rassegne sistematiche della letteratura.

In particolare, Kitchenham ha sviluppato un framework specifico per le revisioni sistematiche in ambito informatico, adattando i principi delle rassegne utilizzate nelle scienze mediche e sociali alle esigenze dell'ingegneria del software e delle discipline computazionali. Questo metodo è ampiamente riconosciuto per la sua capacità di sintetizzare in modo oggettivo lo stato dell'arte su un determinato argomento, facilitando l'identificazione di tendenze e lacune nella letteratura esistente.

A complemento di questo approccio, la checklist PRISMA consente di strutturare il processo di selezione degli studi in fasi ben definite (identificazione, screening, eleggibilità e inclusione), assicurando la trasparenza nella documentazione dei criteri di ricerca e delle esclusioni. Questo metodo risulta particolarmente utile in contesti

multidisciplinari, come quello di questa rassegna, in cui convergono informatica, neuroscienze cognitive e linguistica computazionale.

1.1 Domande di ricerca

Il primo passo di questa rassegna sistematica è consistito nel definire le domande di ricerca che hanno guidato lo studio. Queste domande dovevano essere specifiche, mirate e pertinenti all'argomento analizzato. Come indicato nell'introduzione, l'obiettivo è comprendere l'influenza dei linguaggi di programmazione sui processi cognitivi e sul pensiero umano.

Esplorare questa relazione potrebbe fornire spunti per lo sviluppo di nuovi linguaggi di programmazione più intuitivi, produttivi e accessibili, con potenziali impatti sull'apprendimento del coding e sulla produttività aziendale. Inoltre, una maggiore comprensione degli effetti cognitivi della programmazione potrebbe contribuire a migliorare le metodologie didattiche, favorendo l'insegnamento del coding sin dalle scuole primarie e secondarie. Questo approccio non solo rafforzerebbe le competenze digitali nel settore informatico, ma potrebbe anche aprire nuove prospettive in ambiti umanistici e creativi, ampliando il ruolo della programmazione come strumento di espressione e innovazione.

A questo scopo, sono state formulate due domande di ricerca principali (Research Questions, RQ) e, per la prima di esse, tre sotto-domande (Sub-Research Questions, SRQ). Questa suddivisione consente di analizzare nel dettaglio i diversi aspetti dell'influenza della programmazione sulle capacità cognitive e sulla creatività.

Le domande di ricerca sono le seguenti:

- **RQ1:** I linguaggi di programmazione influenzano i processi cognitivi? Se sì, in che modo?
  - **SRQ1.1**: Quali sono i benefici generali della programmazione sul pensiero?
  - **SRQ1.2:** In che modo i linguaggi di programmazione si relazionano con il linguaggio naturale?
  - **SRQ1.3:** I diversi linguaggi di programmazione modellano il pensiero in modi differenti?
- **RQ2:** Qual è il rapporto tra programmazione e creatività, e in che modo i linguaggi di programmazione lo influenzano?

Queste domande sono state scelte perché, in modo chiaro e sistematico, permettono di individuare schemi ricorrenti nella letteratura esistente, evidenziando le tendenze di ricerca più significative e le eventuali lacune che richiedono ulteriori approfondimenti.

1.2 Selezione degli studi

1.2.1 Database

Le banche dati consultate in questa ricerca sono tra le più autorevoli e diffuse a livello globale:

- **IEEE Xplore:** Biblioteca digitale di riferimento per l'ingegneria e la tecnologia, offre accesso a riviste, atti di conferenze e standard tecnici.
- **ResearchGate:** Social network accademico che consente di condividere pubblicazioni, collaborare con colleghi e richiedere direttamente articoli agli autori.
- **Google Scholar:** Motore di ricerca gratuito che indicizza articoli, tesi e libri accademici in numerose discipline.
- **ACM Digital Library:** Archivio dell'Association for Computing Machinery, rinomato per la qualità delle pubblicazioni in informatica.
- **ScienceDirect:** Piattaforma editoriale che raccoglie articoli peer-reviewed in ambito scientifico, tecnologico e ingegneristico.
- **Scopus:** Database bibliografico multidisciplinare che indicizza articoli scientifici, atti di conferenze e brevetti.
- **Springer Nature:** Editore scientifico di primo piano con un vasto catalogo di riviste e libri accademici.
- **Siti web universitari:** Fonti di tesi, dissertazioni e articoli prodotti da docenti e ricercatori.
- **Siti web generici:** Utilizzati per contestualizzare l'argomentazione nella sezione introduttiva della SLR.

1.2.2 Query di ricerca

Sono state elaborate query di ricerca per le diverse domande di ricerca (QR), integrate con parole chiave e frasi, al fine di recuperare sistematicamente informazioni pertinenti dalle banche dati bibliografiche. La costruzione delle query ha seguito un

Concept 1	**Concept 2**
Programming	Cognitive processes
Coding	Problem-solving
Programming languages	Creativity
Computer programming	Learning
Software programming	Mental models
Programming task	Cognitive benefits
	Human Thinking
	Thought
	Computational thinking
	Cognitive impact

Tabella 1.1: Termini inclusi nella ricerca.

approccio metodico, utilizzando i termini riportati nella **Tabella 1.1** e operatori booleani.

1.2.3 Criteri di inclusione

Gli studi sono stati selezionati in base ai seguenti criteri:

- **Rilevanza:** Gli articoli devono esaminare la relazione tra processi cognitivi e programmazione informatica/linguaggi/paradigmi di programmazione. Sono stati considerati anche studi che indagano la rilevanza delle teorie del pensiero linguistico e le analogie tra il linguaggio naturale e i linguaggi di programmazione, al fine di fornire una solida base teorica per la formulazione di ipotesi volte a colmare le lacune presenti nella letteratura esistente.
- **Tipologia di pubblicazione:** Sono stati considerati solo articoli pubblicati in riviste peer-reviewed, atti di conferenze, tesi e fonti affidabili provenienti da siti web. Tuttavia, queste ultime sono state utilizzate esclusivamente nella stesura dell'introduzione e della conclusione, al fine di mantenere il rigore scientifico nella risposta alle domande di ricerca.
- **Lingua:** Sono stati inclusi solo studi pubblicati in inglese o in italiano.
- **Intervallo temporale:** Le pubblicazioni esaminate coprono approssimativamente il periodo dal 1970 al 2024, privilegiando gli studi più recenti per cogliere gli sviluppi e le tendenze attuali ,sebbene, per prassi, le rassegne sistematiche della letteratura si concentrino generalmente sugli ultimi 10-15 anni. Questo approccio è stato influenzato da un'analisi storica della psicologia della programmazione condotta da Blackwell Blackwell et al. (2019) che ha evidenziato

come, tra gli anni '70 e '90, la ricerca si sia focalizzata principalmente sugli impatti cognitivi della programmazione, per poi spostarsi progressivamente su altri temi nel corso del tempo. Per questo motivo, si è deciso di includere anche studi che, pur trattando linguaggi ormai obsoleti o tecniche in disuso, risultano ancora estremamente rilevanti per comprendere la relazione cognitiva tra i linguaggi di programmazione e i programmatori.

1.2.4 Criteri di esclusione

Gli studi sono stati esclusi in base ai seguenti criteri:

- **Irrilevanza**: Sono stati esclusi gli articoli che trattano la programmazione senza riferirsi alla scrittura, alla comprensione, alla modifica o alle tecniche del codice. In particolare, sono stati esclusi quelli incentrati su diagrammi UML, diagrammi di flusso o altre rappresentazioni di programmi che non analizzano il codice sorgente, così come quelli che non esaminavano il rapporto tra le abilità cognitive e l'apprendimento della programmazione.
- **Mancanza di peer-reviewed**: Per mantenere il rigore scientifico, sono stati esclusi gli articoli che non sono stati sottoposti a rassegna tra pari, così come i manoscritti ancora in fase di rassegna.
- **Duplicati**: Se più pubblicazioni riportavano lo stesso studio, è stata mantenuta solo la versione più completa o recente.
- **Articoli con senza testo o fonti insufficienti**: Sono stati esclusi gli articoli che presentano solo un abstract senza il relativo testo integrale o quelli per cui non erano disponibili sufficienti fonti per completare la bibliografia.

1.2.5 Riepilogo della selezione degli studi

Dai 126 articoli inizialmente trovati attraverso le query di ricerca, dopo un accurato processo di selezione illustrato nel diagramma in **Figura 1.1**, sono stati scelti 41 articoli, tra paper scientifici e atti di conferenze, per rispondere alle domande della Systematic Literature Review.

Ogni articolo è stato esaminato attentamente e sottoposto a una rassegna rigorosa seguendo le metodologie tipiche delle SLR. In particolare, per ciascun studio sono stati analizzati e documentati i seguenti aspetti: contesto, obiettivi, metodologia, risultati, punti di forza, limiti e pertinenza con la SLR.

Tra questi 41 studi, sono inclusi anche quelli che trattano esclusivamente il linguaggio naturale, i quali verranno utilizzati per formulare ipotesi atte a colmare l'enorme gap nella letteratura. Oltre agli articoli selezionati, sono state consultate pagine web per

la stesura dell'introduzione e altri articoli per la definizione tecnica dei paradigmi e dei linguaggi di programmazione. Sebbene questi ultimi non rientrino tra gli studi analizzati nella SLR, saranno comunque riportati nella bibliografia.

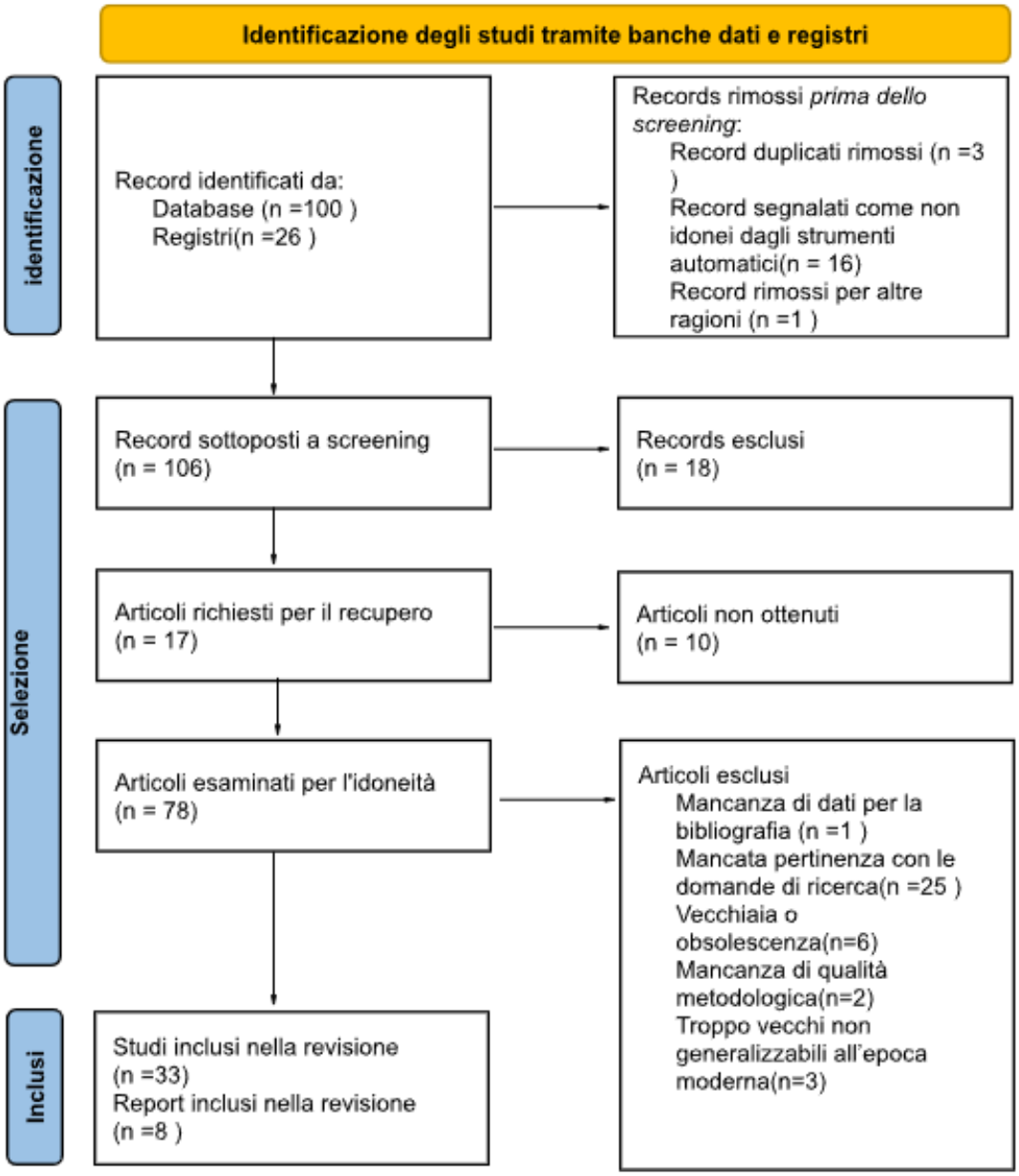

Figura 1.1: Diagramma di flusso PRISMA

1.2.6 Gap nella letteratura

Durante l'analisi è emerso in modo estremamente evidente un gap nella letteratura. La rassegna dei lavori selezionati ha evidenziato una chiara mancanza di studi che esaminino direttamente come i linguaggi di programmazione influenzano il pensiero. Gran parte delle ricerche si concentra su aspetti pratici e tecnici, mentre l'impatto cognitivo delle diverse strutture sintattiche, paradigmi e modelli di programmazione rimane poco esplorato, nonostante oltre 50 anni di studi sull'argomento. Come mostrato nella **Figura 1.2**, la letteratura attuale affronta solo marginalmente la relazione tra linguaggi di programmazione e processi cognitivi. È possibile che questo gap derivi da diversi fattori. Innanzitutto, la forte interdisciplinarità dell'argomento, che coinvolge ambiti distinti come informatica, neuroscienze e psicologia cognitiva, comporta l'uso di metodologie e linguaggi teorici differenti, rendendo difficile integrare e confrontare i risultati. A ciò si aggiungono le difficoltà metodologiche; misurare l'impatto della programmazione sui processi cognitivi richiede strumenti sperimentali complessi (come fMRI o test standardizzati) e protocolli che spesso presentano limitazioni, come campioni ridotti o la mancanza di studi longitudinali. Inoltre, i linguaggi di programmazione e gli ambienti di sviluppo evolvono rapidamente, per cui molti studi possono diventare velocemente obsoleti o non confrontabili a causa delle diverse tecnologie adottate. Questi fattori, insieme all'interesse mutato dei ricercatori verso gli ambiti innovativi e applicativi della programmazione Blackwell et al. (2019), contribuiscono a creare un vuoto nella letteratura, evidenziando la necessità di studi integrati e metodologicamente rigorosi che possano colmare queste lacune. Questo gap, insieme ad altre limitazioni della SLR, verrà discusso più approfonditamente nelle conclusioni.

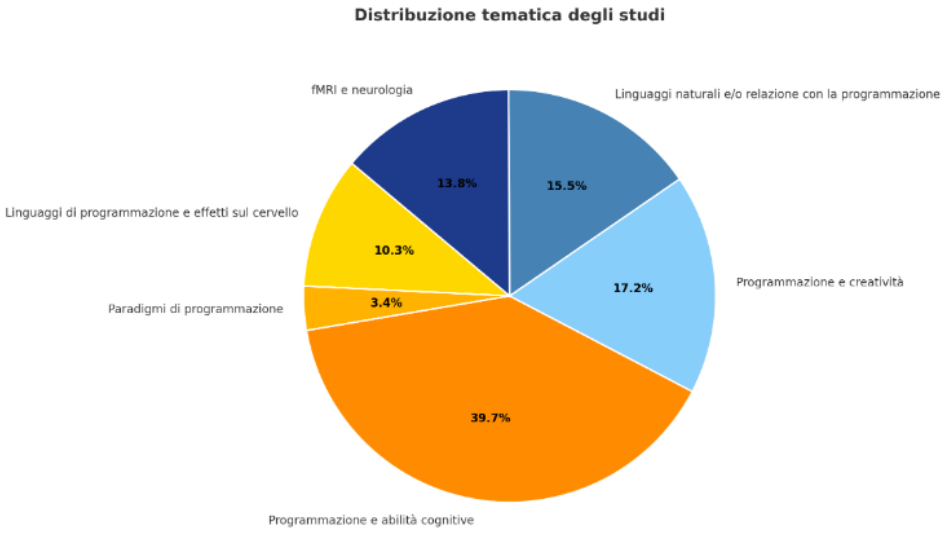

Figura 1.2: Distribuzione tematica degli articoli
(alcuni articoli trattano più di un tema, per cui sono inclusi in tutte le categorie tematiche che affrontano).

Capitolo 2

Risultati della rassegna

2.1 Effetti della programmazione sulle abilità cognitive

Fin dalla nascita della programmazione, è emerso un forte interesse nel comprenderne le implicazioni cognitive. Quasi parallelamente, infatti, si sono sviluppate due correnti di pensiero. La prima, di stampo "tecno-romantico", attribuiva alla programmazione la capacità di migliorare le abilità cognitive, come il pensiero astratto, le euristiche di risoluzione dei problemi e l'abilità di pianificazione. Questa visione associava la programmazione ad altre discipline considerate benefiche, come la matematica, il latino e il gioco degli scacchi, in virtù della loro comune struttura rigorosa e dell'uso di simboli e regole. L'altra visione, invece, considerava l'informatica, e in particolare la programmazione, un'attività sterile, asettica e fine a sé stessa, che coinvolgeva solo nozioni tecniche e non riutilizzabili (Pea and Kurland (1984)).

Negli anni '60, Seymour Papert iniziò a esplorare le potenzialità educative della programmazione, riconoscendo nel coding un mezzo efficace per sviluppare abilità cognitive avanzate come il pensiero logico e la creatività. Attraverso un linguaggio di programmazione ideato da lui, LOGO, Papert introdusse un approccio pedagogico innovativo che permetteva ai bambini in età prescolare di interagire attivamente con i computer e di risolvere problemi utilizzando una tartaruga grafica. Questo metodo pionieristico spianò la strada all'introduzione della programmazione nei curriculum scolastici con l'obiettivo di trasformarla in uno strumento in grado di facilitare l'apprendimento e stimolare il pensiero critico e creativo.

Tuttavia, la ricerca sull'efficacia di questi metodi era ancora limitata, e diversi studiosi hanno messo in discussione le conclusioni di Papert, interrogandosi sul fatto

che la programmazione potesse effettivamente migliorare le abilità cognitive.

Due di questi, Pea and Kurland (1984) hanno cercato di esaminare se e in che modo l'apprendimento della programmazione potesse influenzare le abilità cognitive di chi lo praticava. Infatti, nonostante gli studi sui programmatori adulti mostrassero che la programmazione favoriva lo sviluppo di schemi di conoscenza complessi e strategie avanzate di problem-solving, all'epoca non c'erano prove concrete che l'apprendimento della programmazione fosse in grado di migliorare le abilità cognitive al di fuori del contesto della programmazione stessa. Secondo gli autori, ciò potrebbe essere dovuto alla complessità intrinseca della programmazione, la cui comprensione può essere suddivisa in quattro livelli di abilità:

- Utente del programma: esegue programmi senza comprenderne il funzionamento .
- Generatore di codice: scrive codice corretto, ma senza strategia.
- Generatore di programmi: comprende la logica dei programmi e affina il debug.
- Sviluppatore di software: risolve problemi complessi, ottimizzando prestazioni e usabilità.

Molti studenti si fermano ai primi tre livelli, limitando così l'impatto cognitivo dell'apprendimento. Infatti, secondo Pea, solo il quarto livello porta a effetti cognitivi che vanno oltre la programmazione stessa. Per questo motivo, l'autore sottolinea l'importanza di un insegnamento ben strutturato, che è essenziale per sviluppare appieno le potenzialità cognitive della programmazione e favorire il trasferimento delle competenze.

Il lavoro di Scherer si inserisce nel solco tracciato da Pea, riprendendo e approfondendo le riflessioni sulle possibilità di trasferimento delle abilità cognitive acquisite tramite il coding. In suo articolo del 2016 (Scherer (2016)),l'autore discute della scarsità di evidenze empiriche a sostegno di queste tesi. In particolare, Scherer sottolinea che, sebbene il pensiero computazionale venga frequentemente indicato come uno dei principali benefici dell'insegnamento della programmazione, le prove concrete a sostegno di questa affermazione sono limitate, a causa dei risultati contrastanti presenti in letteratura.

Nel tentativo di colmare queste lacune, Scherer ha successivamente condotto una rassegna sistematica nel 2018 (Scherer et al. (2018)) in cui ha esaminato, nuovamente, se l'apprendimento della programmazione possa migliorare le abilità cognitive generali. Gli obiettivi principali dello studio erano due: valutare il trasferimento delle competenze acquisite ad altri ambiti cognitivi e analizzare l'influenza di variabili come il linguaggio di programmazione, il livello di istruzione e la durata dell'intervento.

I risultati della ricerca hanno mostrato un effetto complessivo moderato del trasferimento, con effetti forti per il near transfer (abilità vicine alla programmazione) e moderati per il far transfer(abilità lontane alla programmazione). I miglioramenti maggiori sono stati osservati in creatività, abilità matematiche, meta-cognizione, competenze spaziali e ragionamento, mentre i benefici su rendimento scolastico e alfabetizzazione sono risultati minimi, fornendo così una solida evidenza empirica dell'impatto positivo della programmazione sulle capacità cognitive.

Nel 2021 (Scherer et al. (2021)) Scherer analizza per l'ultima volta il trasferimento attraverso un ibrido tra una rassegna della letteratura e una meta-analisi. Gli studi esaminati hanno confermato le conclusioni della sua SLR precedente, evidenziando un legame tra programmazione e il miglioramento delle competenze cognitive superiori. Tuttavia, è emerso che gli effetti del trasferimento possono variare in base a fattori come la metodologia didattica, la durata dell'insegnamento e la somiglianza tra la programmazione e altri compiti cognitivi.

A questo proposito, diversi studi hanno indagato gli effetti cognitivi dell'insegnamento della programmazione, ottenendo risultati che confermano le conclusioni di Scherer. In particolare, Grover , nel capitolo intitolato *Computational Thinking: A Competency Whose Time Has Come* del libro *Computer Science Education: Perspectives on teaching and learning* (Grover and Pea (2017)) ,fornisce un importante quadro teorico a supporto del legame tra programmazione e sviluppo del pensiero computazionale. Il pensiero computazionale, che si inspira al concetto di "pensare come un informatico", include elementi fondamentali della programmazione, come la logica, gli algoritmi, il riconoscimento di pattern, l'astrazione e l'automazione. Le pratiche principali associate al CT, come la decomposizione del problema, il debugging, lo sviluppo iterativo, la collaborazione e la creatività, sono tutte abilità che possono essere trasferite ad altri domini cognitivi. Secondo Grover, infatti, il CT è fondamentale non solo per le discipline STEM, ma anche per quelle umanistiche, come la musica, le scienze sociali, la storia e la linguistica. Questo concetto si allinea perfettamente con i risultati precedenti, confermando l'idea che la programmazione favorisca lo sviluppo di abilità cognitive trasferibili, rendendola un elemento chiave per il problem-solving in molteplici discipline.

Altri articoli, come quello di Penge (2019) ampliano ulteriormente il concetto di benefici cognitivi derivanti dall'insegnamento della programmazione. Sebbene il coding venga comunemente associato solo al pensiero computazionale e al problem solving, Penge sostiene che esso possa avere molteplici scopi educativi. Scrivere codice, infatti, non è solo un mezzo per risolvere problemi tecnici, ma anche un'opportunità per sviluppare strumenti pratici come dizionari digitali, app per l'analisi testuale e simulazioni per testare teorie scientifiche, o ancora per trasformare concetti astratti in modelli concreti. In questo modo, il coding si applica a una vasta gamma di

discipline, come la grammatica, la sintassi, la poesia, la musica e la storia dell'arte. Penge sottolinea come la formalizzazione del pensiero, sviluppata attraverso l'uso del coding, sia in grado di stimolare la creatività e il pensiero critico negli studenti. Inoltre, la programmazione aiuta, anche, a scomporre i problemi in parti più gestibili, migliorando così il ragionamento strutturato e organizzato. Penge conferma, quindi, che l'uso di diversi linguaggi di programmazione sia in grado di sviluppare competenze cognitive importanti e variegate , stimolando il pensiero astratto anche in contesti non strettamente tecnici, come le arti e le scienze sociali.

Anche Popat and Starkey (2019) esplorano come la programmazione possa influenzare non solo le abilità tecniche, ma anche le competenze trasversali. Analizzando 172 articoli, gli autori hanno individuato cinque categorie principali di competenze sviluppate tramite il coding scolastico:

- **Risoluzione dei problemi matematici**: La programmazione aiuta a migliorare il problem solving, soprattutto in matematica, sebbene dipenda dalle metodologie didattiche adottate.
- **Competenze sociali e collaborazione:** Il coding favorisce la collaborazione tra gli studenti, anche se non è chiaro se queste competenze derivino direttamente dalla programmazione.
- **Autogestione e apprendimento attivo:** La programmazione stimola l'autogestione, aumentando motivazione e impegno.
- **Pensiero critico:** Il coding favorisce il pensiero critico e lo sviluppo di competenze di livello superiore.
- **Competenze accademiche non informatiche:** Il coding può migliorare abilità come lettura e scrittura, sebbene in modo più limitato.

I risultati di questa ricerca confermano che la programmazione sviluppa competenze trasversali, non solo legate alla codifica, evidenziando il suo potenziale interdisciplinare, anche nelle materie umanistiche. Tuttavia,emerge che il suo successo non è automatico ma dipende dalla progettazione didattica e dall'utilizzo di attività mirate.

In linea con questi risultati, uno studio di Wang et al. (Wang et al. (2021)) ha esplorato come i bambini in età prescolare possano apprendere competenze di coding e come queste influenzino capacità cognitive come la pianificazione e il controllo inibitorio. I risultati hanno mostrato che i bambini con abilità superiori nel coding tendono ad avere migliori performance nel pensiero computazionale,ma non sempre questa associazione è garantita. Inoltre, le abilità nella programmazione sono state associate a una maggiore creatività, soprattutto nella fase iniziale di generazione di

idee. Tuttavia, un aspetto inusuale nella ricerca è che non è emersa una correlazione significativa tra coding e memoria di lavoro, nonostante da sempre la programmazione sia stata definita come un processo cognitivo che coinvolge memoria di lavoro e rappresentazioni mentali (Blackwell et al. (2019)).In conclusione, anche questo studio conferma che il coding può stimolare la creatività e il problem-solving , dimostrando inoltre che la programmazione può essere insegnata già in età prescolare in maniera efficiente.

La programmazione si dimostra utile anche in contesti che analizzano bambini con disabilità cognitive. Nello studio di Peppler and Warschauer (2012), ad esempio, è stato analizzato il percorso di Brandy, una bambina di nove anni con disabilità cognitive, che, grazie alla tecnologia e al coding, è passata da una condizione di emarginazione a quella di artista multimediale e mentore. Nel corso di due anni e mezzo, i ricercatori hanno raccolto dati misurando l'evoluzione nelle capacità della ragazzina. I principali risultati sono stati lo sviluppo di alfabetizzazioni multiple e delle capacità metalinguistiche,l' evoluzione della partecipazione sociale e un incremento della creatività. Anche questo studio ha evidenziato l'impatto positivo del coding su abilità cognitive quali la risoluzione dei problemi e il meta-pensiero, sottolineando come i suoi effetti siano talmente benefici da poter supportare l'apprendimento anche in bambini con difficoltà di lettura o scrittura.

In conclusione, i risultati evidenziano che la programmazione può potenziare il pensiero computazionale e le abilità cognitive, specialmente nei contesti educativi. Tuttavia, per comprendere appieno in che modo la programmazione influenzi il cervello umano è fondamentale l'analisi tramite fMRI, che permette di esplorare più a fondo i meccanismi neurologici sottostanti al coding. Queste ricerche si concentrano principalmente sull'identificazione delle aree cerebrali coinvolte nella comprensione del codice sorgente e sul confronto tra la programmazione e altri processi cognitivi, come il linguaggio naturale , il problem-solving e la matematica. La **Tabella 2.1** fornisce una sintesi dei cinque studi che sono stati trovati tramite la rassegna, evidenziandone gli obiettivi, le metodologie adottate e le principali conclusioni tratte da ciascuna ricerca, con il fine di metterne in risalto le similitudini e soprattutto le differenze.

Tabella 2.1: Sintesi degli studi con fMRI sulla comprensione del codice sorgente.

Studio	Obiettivo	Metodologia	Risultati	Conclusione
Siegmund et al. (2014)	Esplorare l'uso della risonanza magnetica funzionale per misurare l'attività cerebrale durante la comprensione del codice sorgente. Identificare le aree del cervello coinvolte e valutare se la metodologia supporta studi empirici sull'argomento.	17 studenti universitari con esperienza in Java hanno svolto due tipi di compiti: comprendere frammenti di codice (determinare output) e identificare errori di sintassi.	Sono state identificate cinque aree cerebrali attive, tutte nell'emisfero sinistro: Zona 6 (memoria di lavoro, attenzione, problem-solving), Zona 21 (memoria semantica, linguaggio), Zona 40 (memoria di lavoro verbale e numerica), Zona 44 (sintassi, integrazione linguistica), Zona 47 (selezione informazioni semantiche).	I risultati suggeriscono che la comprensione del codice coinvolge abilità cognitive complesse, tra cui la memoria di lavoro, l'attenzione e il linguaggio. I dati supportano l'ipotesi di Dijkstra: buone abilità linguistiche migliorano la capacità di programmare.

Continua nella pagina successiva

Studio	Obiettivo	Metodologia	Risultati	Conclusione
Floyd et al. (2017)	Esaminare come il cervello umano elabora le attività di ingegneria del software, confrontando l'analisi del codice con la lettura della prosa. Determinare se il codice è processato come un linguaggio naturale o se attiva regioni cerebrali distinte.	fMRI su programmatori esperti; tre compiti: comprensione del codice, rassegna del codice e rassegna della prosa.Lo studio utilizza il linguaggio di programmazione C per gli stimoli di codice analizzati nell'esperimento.	Lo studio ha rilevato che sia la lettura del codice che quella della prosa attivano principalmente le regioni prefrontali, responsabili della cognizione avanzata, del controllo esecutivo e del processo decisionale, ma con pattern distinti. L'area di Wernicke, coinvolta nella comprensione del linguaggio, si attiva in entrambi i casi, ma in modo diverso. La corteccia prefrontale dorsolaterale, legata al ragionamento logico e al problem solving, mostra un'attivazione più intensa durante l'analisi del codice rispetto alla lettura della prosa. Anche il giro frontale inferiore, associato alla memoria di lavoro, risulta coinvolto in entrambi i compiti, sebbene con livelli di attivazione variabili a seconda dello stimolo. Tuttavia, i programmatori esperti elaborano codice e prosa in modo più simile, riducendo il carico cognitivo.	Il codice attiva aree neurali diverse dal linguaggio naturale, ma nei programmatori esperti le differenze si attenuano, suggerendo un processo di adattamento cognitivo. Lo studio supporta l'idea che i linguaggi di programmazione influenzino il pensiero.

Continua nella pagina successiva

Studio	Obiettivo	Metodologia	Risultati	Conclusione
Ivanova et al. (2020)	Confrontare l'attivazione cerebrale tra codice, linguaggio naturale e matematica	fMRI su partecipanti con competenze intermedie; confronto tra lettura di codice (Python e Scratch), risoluzione di problemi matematici e linguistici.	L'elaborazione del codice attiva fortemente il sistema a domini multipli (MD), coinvolgendo il controllo cognitivo, la memoria di lavoro e il problem-solving, senza una lateralizzazione emisferica specifica. Al contrario, l'attivazione del sistema del linguaggio risulta debole, poiché la comprensione del codice non sembra coinvolgere in maniera significativa le aree cerebrali tipiche dell'elaborazione linguistica. Inoltre, non emergono aree cerebrali specifiche dedicate al codice, che viene invece elaborato attraverso circuiti già esistenti per il ragionamento astratto.	Lo studio suggerisce che la programmazione non è elaborata come un linguaggio naturale, ma richiede risorse cognitive più complesse.

Continua nella pagina successiva

Studio	Obiettivo	Metodologia	Risultati	Conclusione
Liu et al. (2020)	Indagare le regioni cerebrali coinvolte nella comprensione del codice e l'impatto dell'esperienza di programmazione sul carico cognitivo. Esaminare se l'apprendimento del linguaggio naturale e della programmazione siano correlati	Studio con fMRI su 15 programmatori esperti (media 11 anni di esperienza, minimo 5), testati in Python. Due tipi di prove: codice reale (funzioni Python) vs codice 'falso' (scrambling del codice reale).	La comprensione del codice attiva principalmente il network fronto-parietale sinistro, suggerendo una somiglianza con la logica formale e, in parte, con la matematica, sebbene coinvolga in misura minore le aree numeriche. L'attivazione del sistema del linguaggio risulta invece bassa, con una scarsa sovrapposizione rispetto alle aree tipiche dell'elaborazione linguistica, il che indica che il codice non viene processato come un linguaggio naturale. Inoltre, la sovrapposizione con i circuiti del controllo esecutivo è limitata, evidenziando che il coding non si riduce a un semplice compito di memoria di lavoro o attenzione. Infine, emergono pattern neurali distinti per diverse strutture del codice: i cicli for e le condizioni if attivano aree diverse, suggerendo una categorizzazione cerebrale della sintassi del codice.	Lo studio conferma i risultati di Ivanova (2020): la programmazione è più simile alla logica formale che al linguaggio naturale. Le differenze metodologiche potrebbero spiegare discrepanze minori (es. lateralizzazione sinistra del codice in questo studio, assente in Ivanova).

Continua nella pagina successiva

Studio	Obiettivo	Metodologia	Risultati	Conclusione
Peitek et al. (2020)	Indagare le regioni cerebrali coinvolte nella comprensione del codice e l'impatto dell'esperienza di programmazione sul carico cognitivo. Esaminare se l'apprendimento del linguaggio naturale e della programmazione siano correlati.	Lo studio di risonanza magnetica funzionale (fMRI) è stato condotto su 17 programmatori esperti in Java, ai quali sono stati assegnati due compiti: la comprensione del codice, contrassegno dell'output, e l'analisi sintattica. Per confermare la validità dei risultati, l'esperimento è stato successivamente replicato con un secondo gruppo di 11 partecipanti	Lo studio evidenzia cinque aree principali coinvolte: il giro frontale inferiore sinistro (BA 44 e 47) per la sintassi del linguaggio, il giro precentrale (BA 6) per memoria di lavoro e pianificazione, il lobo parietale inferiore (BA 40) per la manipolazione simbolica e numerica, e il giro temporale medio (BA 21) per l'elaborazione semantica. È emerso che maggiore è la complessità del codice, maggiore è la concentrazione richiesta. L'esperienza nella programmazione non riduce lo sforzo cognitivo, ma la familiarità con Java ne migliora l'elaborazione. Nel secondo esperimento, la replica del test ha confermato l'attivazione di BA 40, 21 e 44, ma ha rivelato differenze: BA 39, legata alla comprensione del linguaggio scritto, è risultata attiva, mentre BA 6 e 47 non hanno mostrato un'attivazione significativa.	I risultati suggeriscono che la lettura del codice non si limiti alla semplice decodifica di simboli, ma implica processi di elaborazione sintattica e semantica, analoghi a quelli coinvolti nella lettura di una lingua naturale.Il coinvolgimento di numerose aree cerebrali legate al linguaggio supporta l'ipotesi di Dijkstra, secondo cui una solida padronanza del linguaggio naturale favorisce l'apprendimento della programmazione.Tuttavia, a differenza del linguaggio naturale, il codice non attiva la corteccia temporale anteriore, associata alla narrazione. È emerso, inoltre, che i linguaggi di programmazione presentano un carico cognitivo specifico e che la competenza in uno non si trasferisce automaticamente ad altri.

I risultati di questi studi evidenziano una visione talvolta contrastante ed incerta sull'attività cerebrale durante la programmazione. Le discrepanze osservate potrebbero dipendere da diversi fattori. In primo luogo, sono stati utilizzati diversi linguaggi di programmazione, il che potrebbe aver influenzato le modalità di attivazione cerebrale rilevate. Inoltre, i partecipanti agli studi presentavano diversi livelli di esperienza: nello studio di Ivanova et al., i soggetti avevano competenze intermedie mentre nello studio di Floyd erano presenti anche programmatori esperti, i quali probabilmente adottano strategie diverse di lettura del codice più automatizzate, influenzando così i risultati. Un ulteriore elemento da considerare è il numero ristretto di partecipanti in ciascuno studio, che probabilmente ha influito sulla generalizzabilità dei risultati e potrebbe aver contribuito alle discrepanze emerse.

Ciò che appare particolarmente rilevante, tuttavia, è che, nonostante le divergenze metodologiche e interpretative, tutti gli studi concordano sul coinvolgimento di aree cognitive avanzate, tra cui quelle associate alla logica formale, alla memoria di lavoro e al ragionamento, nel processo di comprensione del codice. Questo dato riconferma il ruolo della programmazione non solo come competenza tecnica, ma anche come catalizzatore di abilità cognitive di ordine superiore.

Parallelamente a questi studi , emergono altre due ricerche che utilizzano fMRI ma che si discostano dagli approcci precedenti, concentrandosi sui cambiamenti neuroplastici indotti dall'apprendimento del coding. Il primo studio , di Hongo et al. (Hongo et al. (2023)) , ha analizzato le modifiche strutturali nel cervello di studenti universitari che imparavano a programmare per la prima volta. Dopo un corso di 15 settimane su Processing, è stato rilevato un aumento della materia grigia in otto aree cerebrali, tra cui il polo frontale destro (associato a persistenza e obiettivi), il giro frontale mediale (coinvolto nel ragionamento deduttivo), il pallidum (legato alla motivazione e alla ricompensa) e il cuneo sinistro (coinvolto nell'elaborazione visuo-spaziale). In particolare, il volume del polo frontale destro è risultato correlato al successo nel coding, suggerendo un legame tra la persistenza e le prestazioni. Il secondo studio, condotto da Hiskikawa et al. (Hishikawa et al. (2023) ha invece esaminato i meccanismi neurali dell'apprendimento della programmazione e l'evoluzione dell'attività cerebrale con l'acquisizione delle competenze di coding. I risultati hanno mostrato l'attivazione di diverse aree corticali e subcorticali, tra cui il giro frontale inferiore bilaterale (coinvolto nella comprensione del codice e nel problem-solving), i lobi parietali e temporali (associati alla memoria di lavoro e all'analisi delle informazioni), nonché i nuclei caudati e il cervelletto (relazionati all'apprendimento motorio e alla pianificazione). Il giro frontale inferiore destro è l'unica area che ha mostrato un aumento significativo di attività. Mentre, sebbene il giro frontale inferiore sinistro fosse stato particolarmente attivo durante la programmazione, non ha subito cambiamenti dovuti all'apprendimento né ha mostrato una correlazione con il miglioramento delle prestazioni. Questo suggerisce che l'apprendimento della programmazione modifichi selettivamente il giro frontale inferiore destro, rendendolo cruciale per le migliori performance. Lo studio evidenzia, anche, che l'attivazione cerebrale indotta dalla programmazione riguarda principalmente l'acquisizione di competenze pratiche, piuttosto che la comprensione teorica del concetto. L'attività

osservata sembra quindi legata più alla capacità di scrivere codice e risolvere problemi che a una comprensione astratta della programmazione. Inoltre, l'attivazione dell'area fronto-parietale sinistra ha indicato il coinvolgimento di processi linguistici nella comprensione della programmazione. Tuttavia, a differenza della lingua naturale, la programmazione sembra fare un uso maggiore del sistema di richiesta multipla, suggerendo similitudini con l'apprendimento di una seconda lingua. Secondo l'ipotesi classica, infatti, nelle fasi iniziali dell'acquisizione di una seconda lingua in età adulta si verifica uno spostamento dell'attività cerebrale dalla destra alla sinistra.

I risultati di entrambi gli studi rivelano che il coding provoca cambiamenti neuroplastici sia strutturali che funzionali, dimostrando così il suo impatto sulle abilità cognitive e sulla riorganizzazione delle reti neurali.

In conclusione, questi risultati non solo evidenziano il ruolo della programmazione come abilità tecnica, ma la consacrano come un vero e proprio catalizzatore di trasformazioni cognitive e neurali. Il coding, dunque, non è soltanto un mezzo per interagire con le macchine, ma un'attività che modella attivamente la struttura e il funzionamento del cervello, potenziandone le capacità di ragionamento, problem-solving e adattamento. Nella **Figura 2.1** sono state sintetizzate tutte le abilità che sembrano essere particolarmente influenzate dalla programmazione, così come emerse dall'analisi della letteratura.Tuttavia, i benefici attribuiti al coding non sono automatici: senza un insegnamento adeguato, la programmazione rischia di non fornire gli effetti cognitivi positivi per cui è spesso celebrata. Affinché l'inserimento del coding nei curriculum scolastici sia realmente efficace, è essenziale che venga supportato da metodologie didattiche mirate, basate su evidenze scientifiche e adattate ai diversi livelli di apprendimento. L'insegnamento della programmazione, infatti, dovrebbe essere progettato in modo da stimolare non solo l'acquisizione di competenze tecniche, ma anche il pensiero critico e la risoluzione di problemi complessi.

Perciò, nonostante i progressi compiuti nella comprensione dell'impatto della programmazione sul cervello, la ricerca è tutt'altro che conclusa. Rimangono ancora numerose incertezze, sia riguardo all'intensità e sia riguardo alla durata dei cambiamenti cognitivi indotti dal coding, soprattutto in relazione al miglioramento di abilità periferiche, ovvero quelle non direttamente legate alla programmazione stessa. La scarsità di studi longitudinali e la difficoltà nel confrontare i diversi paradigmi sperimentali rendono ancora complesso delineare un quadro univoco. In particolare, gli studi basati su fMRI rappresentano un'area di ricerca promettente, ma attualmente limitata sia per la sua ridotta popolarità sia per la variabilità dei risultati. Per questo motivo, è auspicabile lo sviluppo di ulteriori ricerche che colmino le lacune attuali. Sarebbe particolarmente utile condurre studi su campioni più ampi e diversificati, adottare metodologie standardizzate e approfondire il confronto tra i diversi linguaggi di programmazione, così da ottenere una comprensione più chiara e sistematica del rapporto tra coding e processi cognitivi. Solo attraverso un approccio interdisciplinare e rigoroso sarà possibile chiarire definitivamente in che misura e in quali condizioni la programmazione possa effettivamente rappresentare un motore di trasformazione cognitiva e neuroplastica.

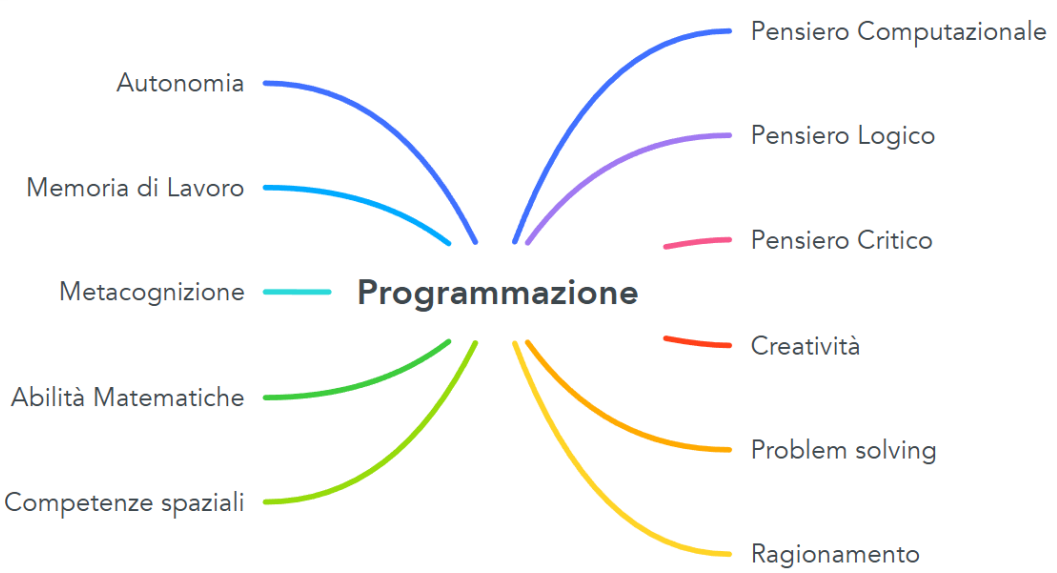

Figura 2.1: Capacità cognitive migliorate con la programmazione

2.2 Il coding è simile al linguaggio naturale?

La programmazione è stata tradizionalmente collocata all'intersezione tra linguaggio,matematica e logica. Sin dalle sue origini , essa è stata considerata non solo un mezzo per la scrittura di istruzioni eseguibili da una macchina, ma anche come un sistema formale che incorpora strutture sintattiche e semantiche tipiche del linguaggio, dei principi matematici e dei metodi di ragionamento logici.

Da un lato, i linguaggi di programmazione sono strettamente legati alla logica e alla matematica perché si basano su regole formali e strutture rigorose per eseguire operazioni. La logica booleana governa il flusso decisionale dei programmi, mentre l'algebra booleana gestisce i bit e le operazioni sui dati. Inoltre, concetti matematici come la teoria degli insiemi, le funzioni e i grafi costituiscono le fondamenta delle strutture dati. Per anni, questo legame è stato considerato così forte da far ritenere le competenze matematiche un requisito essenziale per il successo nella programmazione, in particolare nello sviluppo del pensiero computazionale (Hadjerrouit and Hansen (2020)).

Dall'altro, i linguaggi di programmazione presentano una grammatica e una sintassi che li accomuna con i linguaggi naturale. Già nel Tractatus Logico-Philosophicus, Wittgenstein definiva il linguaggio perfetto come un sistema rigoroso e privo di ambiguità, in cui ogni proposizione corrisponde a un fatto del mondo in modo univoco. Questo ideale di precisione trova una forte risonanza nella programmazione, dove i linguaggi devono essere formalmente definiti per garantire l'esecuzione corretta delle istruzioni.

Questa prospettiva ha spinto molti studiosi a interrogarsi sulla relazione tra questi due mondi, così strettamente connessi ma, al tempo stesso, distinti. Primo fra questi, Zemanek (1966) che nel 1966 si inserisce in una più ampia corrente di pensiero che, già in quegli anni, analizzava la distinzione tra sintassi, semantica e pragmatica nei linguaggi naturali. Egli applica questa distinzione ai linguaggi di programmazione, riconoscendone non solo la struttura formale, ma anche le implicazioni pratiche e comunicative. Inoltre, l'autore individua un'ulteriore somiglianza tra i linguaggi di programmazione e il linguaggio naturale: la loro componente comunitaria e conversazionale. Nel caso dei linguaggi di programmazione, questa dimensione è duplice, poiché la comunicazione avviene sia tra uomo e macchina sia tra programmatori, attraverso il codice scritto.

Questa prospettiva sulla pragmatica ha contribuito a sfatare l'idea che i linguaggi di programmazione siano semplicemente linguaggi formali tecnici, mettendo in luce il loro ruolo come strumenti di comunicazione tra programmatori ed esecutori di lavoro. Un'idea già diffusa in quegli anni, come evidenziato da Caracciolo (1966), e che continua a essere rilevante anche in epoca moderna, Casey (2019).

Più di vent'anni dopo, in linea con questo interesse per le similitudini tra linguaggi naturali e linguaggi di programmazione, si inserisce l'articolo di Fedorenko et al. (2019), nel quale gli autori sostengono che, nonostante le differenze tecniche tra linguistica e informatica, entrambi i tipi di linguaggio si basano su strutture significative analoghe. Nel linguaggio naturale, infatti, le parole si combinano per formare frasi significative, mentre nei linguaggi di programmazione funzioni e variabili si uniscono per generare codice. Inoltre, entrambi i sistemi linguistici sono caratterizzati da vincoli che regolano la combinazione di questi elementi, al fine di generare significati complessi.

Dato che queste somiglianze strutturali esistono, è lecito chiedersi se la padronanza del linguaggio naturale possa avere un ruolo nell'apprendimento della programmazione. Proprio su questo aspetto si concentra lo studio di Prat et al. (2020) che ha esplorato se l'attitudine linguistica possa essere un predittore significativo del successo nel coding, al pari delle competenze matematiche. Dalla ricerca è emerso che l'abilità linguistica è in grado di predire il successo nell'apprendimento della programmazione, spiegando il 17% della varianza nei risultati; al contrario, le competenze matematiche spiegavano solo il 2% della varianza. Questo risultato è stato confermato anche dallo studio di Navarro-Cota et al. (2024) che, analizzando i principali fattori cognitivi associati al successo nella programmazione, ha evidenziato il ruolo dell'intelligenza linguistica e del bilinguismo.

Tuttavia, non tutti gli studi concordano su questa visione. Hartman nel suo studio del 2024 (Hartmann et al. (2024)) mette in discussione l'ipotesi di una forte connessione tra linguaggio naturale e programmazione, indagando se l'apprendimento di una lingua artificiale (Brocanto) prima di imparare a programmare possa attivare strategie linguistiche utili per facilitare la comprensione del codice. I risultati non hanno mostrato nessun miglioramento immediato nelle competenze di coding per il gruppo Brocanto rispetto al gruppo di controllo. Tuttavia, gli studenti con Brocanto si sono sentiti più sicuri nell'affrontare le nuove regole sintattiche e hanno percepito Python come più intuitivo, suggerendo possibili

benefici a lungo termine. Di fronte a questi esiti, l'autore propone due ipotesi: la prima è che la durata limitata dell'esperimento (una sola settimana) potrebbe non essere stata sufficiente per rilevare effetti concreti; la seconda implica che esista una differenza sostanziale tra linguaggi naturali e linguaggi di programmazione, una differenza che Hartman identifica nella semantica. Nel linguaggio naturale la semantica è immediata e il significato emerge direttamente dalle parole, mentre nel codice il significato dipende dall'esecuzione del programma.

Queste differenze ,tra le due tipologie di linguaggio, vengono evidenziate anche dallo stesso Zemanek (Zemanek (1966)) . L'autore sostiene che ciò che distingue in maniera drastica i linguaggi naturali da quelli di programmazione è la formalizzazione di questi ultimi. Secondo Zemanek però questa formalizzazione è essenziale, poiché un codice deve produrre sempre lo stesso risultato, indipendentemente da chi lo scrive o lo legge. Inoltre, la ridondanza, tipica del linguaggio naturale, lo renderebbe più lento. Al contrario, nella sua forma formale, il codice risulta più facile da analizzare e tradurre. Tuttavia,proprio questa sua diversità rispetto al linguaggio naturale , sostiene Zemanek, permette al programmatore di riflettere in modo critico e preciso sulle istruzioni da dare alla macchina.

Casey (2019) approfondisce ulteriormente questa distinzione, focalizzandosi sulle somiglianze e le differenze tra codice e linguaggio naturale, attraverso un'analisi sulla ripetitività e prevedibilità. I risultati dimostrano che il codice presenta una maggiore ridondanza e una minore varietà rispetto al linguaggio naturale. Un altro esito interessante riguarda il fatto che la ripetizione nel codice non dipenda solo dalla sintassi. L'autore ha analizzato le possibili cause di questa ripetitività, suggerendo che ci siano fattori cognitivi e pratici alla sua base. Infatti, la complessità della programmazione porta gli sviluppatori ad utilizzare strutture note per ridurre lo sforzo mentale. L'uso di design patterns standardizzati e il riutilizzo di codice da fonti esterne contribuiscono ulteriormente a questo fenomeno. Tuttavia, la ripetizione può anche avere vantaggi, rendendo il codice più prevedibile e facilitando debugging e manutenzione. Per avvalorare queste ipotesi, Casalnuovo ha esaminato uno studio che ha rivelato come gli studenti di inglese tendano a scrivere in modo più ripetitivo e prevedibile rispetto ai madrelingua. Questo comportamento supporta l'idea che, quando una lingua presenta una maggiore complessità cognitiva, le persone sono spinte a utilizzare un numero più limitato di forme per semplificare l'espressione.

Negli ultimi anni anche le neuroscienze cognitive hanno cercato di analizzare la relazione tra linguaggio naturale e programmazione (studi già analizzati nella risposta prima e visibili nella **Tabella 2.1**). Alcuni di questi studi, basati sulla risonanza magnetica funzionale (fMRI), come quelli di Siegmund et al. (2014) , Peitek et al. (2020) e Hishikawa et al. (2023), hanno mostrato che alcune delle aree cerebrali coinvolte nella comprensione del linguaggio naturale si attivano anche durante la lettura e la scrittura di codice, nonostante negli ultimi due processi sia presente anche l'attivazione di aree non relative al linguaggio , come quelle riservate alla logica e al problem solving. Hishikawa, in particolare, ha notato che, nonostante la forte attivazione delle aree del linguaggio durante la lettura del codice, le uniche aree modificate plasticamente erano quelle dell'emisfero destro. Ha quindi

riconosciuto che i processi cognitivi coinvolti nell'apprendimento della programmazione potrebbero essere simili a quelli necessari per apprendere una seconda lingua.

A differenza di quanto osservato da questi studi , Floyd et al. (2017) ha rilevato che la programmazione attiva maggiormente le regioni prefrontali, responsabili della cognizione di alto livello, del problem-solving e del controllo esecutivo, mentre le aree responsabili del linguaggio presentavano complessivamente una bassa attivazione. Tuttavia, con il crescere dell'esperienza del programmatore, queste differenze di attivazione si appianavano, minimizzando così la differenza cognitiva tra la lettura della prosa e quella del codice.

In contrasto con tutti i precedenti studi, invece, Ivanova et al. (2020) e Liu et al. (2020) hanno riscontrato un'importante attivazione del sistema di domanda multipla MD(coinvolto nel problem-solving, nella memoria di lavoro e nel controllo cognitivo) , mentre l'attivazione delle aree coinvolte nel linguaggio era debole, se non addirittura inesistente. Entrambi gli studi, infatti, sono giunti alla medesima conclusione: la programmazione è più vicina al problem-solving che alla linguistica.

Abbiamo già discusso delle discrepanze di questi risultati in risposta alla RQ precedente. Tuttavia, è possibile che i risultati non siano necessariamente in contraddizione: i principianti e gli intermedi potrebbero fare maggiore affidamento sul sistema di domanda multipla (MD) per processare il codice, mentre gli esperti potrebbero integrare maggiormente le aree linguistiche. Questo sarebbe coerente con la teoria dell'automatizzazione delle competenze, secondo cui, con l'esperienza, la programmazione potrebbe diventare più simile alla lettura di un testo, almeno per alcuni aspetti. Tuttavia, queste incertezze nelle conclusioni evidenziano la necessità di ulteriori approfondimenti in questo ambito di ricerca ancora in fase di sviluppo.

In conclusione, la letteratura attuale non è stata in grado di trovare una risposta definitiva alla complessa e sfaccettata relazione tra il linguaggio naturale e quello di programmazione. Sotto alcuni aspetti, i due risultano estremamente simili, tanto che, come è emerso, le abilità linguistiche sono in grado di influenzare il successo nell'apprendimento della programmazione, in misura maggiore perfino della matematica. Inoltre, dai risultati del neuroimaging sembra emergere che, sebbene vi siano differenze rispetto al linguaggio naturale , durante la programmazione si attivino talvolta le stesse aree cognitive. Ciò suggerisce che la programmazione sia simile al linguaggio naturale oppure che, quantomeno, il suo apprendimento segua dinamiche analoghe a quelle di una seconda lingua.

Tuttavia, non è ancora chiaro fino a che punto le differenze tra questi due mondi influenzino la programmazione, né se siano abbastanza significative da distinguerla nettamente dal linguaggio, collocandola in una categoria completamente nuova e autonoma. Di conseguenza, nonostante le ricerche svolte, il rapporto tra programmazione e linguaggio rimane difficile da definire: la programmazione è sufficientemente simile al linguaggio da poter essere considerata, almeno in parte, analoga? Oppure le sue differenze sono tali da renderla un'attività cognitiva indipendente dai processi del linguaggio naturale? È importante che questi interrogativi trovino risposta attraverso studi futuri, poiché possono offrire mag-

giore chiarezza sul ruolo cognitivo della programmazione e, di conseguenza, contribuire a migliorare sia la progettazione dei linguaggi di programmazione sia l'insegnamento del coding.

Se la programmazione si rivelasse abbastanza simile al linguaggio significherebbe che il suo impatto cognitivo è ancora più profondo di quello evidenziato nella risposta alla SQR1. Infatti, così come il linguaggio sembra influenzare il pensiero, guidando la percezione e modellando la categorizzazione e l'interpretazione della realtà (Francis (2023)), i linguaggi di programmazione, con i loro diversi paradigmi, potrebbero plasmare il ragionamento in modo profondo e distinto, arrivando, persino a modificare profondamente il modo dei programmatori di approcciare e risolvere i problemi. Data la portata di queste implicazioni, esplorare questa direzione appare non solo utile, ma necessario.

2.3 I diversi linguaggi di programmazione modellano il pensiero in modi differenti?

Abbiamo visto che la programmazione non è solo un'abilità tecnica, ma un'attività che influisce sui processi cognitivi, migliorando il problem solving, la logica e persino la creatività. Tuttavia, il modo in cui questi effetti si manifestano potrebbe dipendere dal linguaggio di programmazione utilizzato.

Se scrivere in inglese o in giapponese può modellare la nostra percezione del mondo (Francis (2023)), allora anche programmare in Python o in Haskell potrebbe influenzare il nostro modo di ragionare. Alcuni linguaggi potrebbero rendere il pensiero più strutturato e logico, altri stimolare la creatività e l'intuizione, mentre alcuni possono perfino ostacolare certe strategie cognitive.

I paradigmi di programmazione, ancora più che i linguaggi stessi, potrebbero rappresentare veri e propri stili di pensiero. Essi si basano su un insieme di regole, logiche, approcci e funzionalità che guidano il programmatore nella risoluzione dei problemi informatici. In questo senso, i paradigmi fungono da modelli logici e approcci pratici che influenzano il modo in cui un programmatore affronta una determinata sfida. Ogni paradigma, infatti, potrebbe richiedere strategie di pensiero diverse e stimolare il ragionamento in modi differenti, a seconda della struttura e degli strumenti che offre, diventando a tutti gli effetti un mezzo per comprendere e osservare un problema Shaft and Vessey (2006).

Queste ipotesi trovano una prima conferma nella riflessione di Zemanek, che suggerisce come il linguaggio di programmazione non sia un semplice strumento tecnico, ma un mezzo che guida il nostro approccio alla risoluzione dei problemi (Zemanek (1966)). Secondo Zemanek, è proprio la rigidità formale della sintassi e della semantica di un linguaggio di programmazione a modellare il pensiero del programmatore: le sue strutture e regole impongono schemi cognitivi precisi, forzando determinati approcci di problem solving e rendendone altri meno intuitivi o accessibili.

Questi livelli di rigidità, tuttavia , non sono gli stessi per ciascun linguaggio di programmazione. Come evidenziato dal concetto di sintonicità affrontato da Watt (1998) dove alcuni linguaggi richiedono meno carico cognitivo. L'autore riprende il concetto di sintonicità, introdotto da Seymour Papert per descrivere il modo in cui i bambini possono creare e comprendere concetti utilizzando il linguaggio di programmazione Logo, da lui sviluppato. La sintonicità è considerata un'estensione della capacità umana di attribuire stati mentali a oggetti e agenti, come programmi e linguaggi di programmazione. Può manifestarsi in due forme: quella corporea e quella dell'ego. Nel contesto della programmazione, essa riguarda principalmente l'ego, poiché tendiamo a concepire i programmi come entità psicologiche piuttosto che fisiche. Watt sostiene che maggiore è la sintonicità di un linguaggio, più risulta facile da apprendere e si riduce il carico cognitivo. L'autore analizza quattro linguaggi di programmazione: Logo, Prolog,StarLogo e Cocoa, e giunge alla conclusione che i linguaggi con metafore intuitive rendono la programmazione più accessibile, soprattutto per i principianti. I linguaggi visivi come Logo,infatti, favoriscono una sintonicità corporea, mentre i linguaggi di programmazione logica e dichiarativa, come Prolog, incoraggiano una sintonicità dell'ego. Tuttavia, i linguaggi più astratti e complessi, come Cocoa, riducono la sintonicità dell'ego e risultano più difficili da apprendere. In definitiva, i linguaggi con metafore più intuitive e una sintonicità più marcata tendono a ridurre il carico cognitivo, rendendo l'esperienza di programmazione più naturale e immediata. Per questo motivo, Watt sostiene che i linguaggi dovrebbero essere progettati considerando anche la loro afferrabilità psicologica, oltre alla loro logica computazionale. Questa visione della sintonicità ci offre un importante spunto per riflettere su come i diversi linguaggi di programmazione, in base alla loro struttura e al loro grado di sintonicità, possano influenzare in modo differente i processi cognitivi.

Se da un lato la sintonicità di un linguaggio può renderlo più intuitivo e ridurre il carico cognitivo, dall'altro White and Sivitanides (2002) evidenziano che i diversi paradigmi di programmazione attivano processi cognitivi differenti. Secondo White, ogni paradigma richiede una particolare modalità di pensiero e diversi livelli di sviluppo cognitivo. L'autore analizza il legame tra i linguaggi di programmazione , lo stile e i requisiti cognitivi(utilizzando la teoria dello sviluppo di Piaget), di quattro paradigmi principali:

- **Procedurale:** Questo paradigma mette in risalto la logica sequenziale, utilizzando variabili e assegnazioni. Richiede un pensiero formale e razionale, che stimola soprattutto l'emisfero sinistro del cervello. Gli studenti con una spiccata capacità di astrazione tendono a eccellere con questo approccio. Per padroneggiarlo completamente, è utile aver raggiunto lo sviluppo operatorio formale, meglio se a livello avanzato.

- **Orientato agli Oggetti:** In questo paradigma, si utilizzano oggetti che combinano dati e metodi, rendendo particolarmente efficace il problem-solving per compiti complessi. Coinvolge entrambi gli emisferi cerebrali, stimolando così diverse capacità cognitive. È caratterizzato da un'astrazione profonda, una solida organizzazione concettuale e una struttura gerarchica ben definita. Rispetto al paradigma procedu-

rale, risulta essere più intuitivo. Per padroneggiarlo appieno, è necessario sviluppare una buona capacità di pensiero formale. Questa visione è stata supportata anche da Hall et al. (2006) che ha identificato tra i predittori del successo nella programmazione orientata agli oggetti competenze cognitive quali il pensiero astratto, analitico e il ragionamento teorico.

- **Programmazione visiva:**Utilizza un'interfaccia grafica per rendere la programmazione più semplice e intuitiva, rendendola accessibile anche a chi ha uno sviluppo cognitivo meno avanzato. Questo paradigma funge da ponte verso linguaggi di programmazione più complessi, facilitando l'apprendimento graduale.

- **Basato su script:**Caratterizzato da una struttura semplice e un'alta leggibilità, questo paradigma richiede meno astrazione. È ideale per chi ha un pensiero operativo concreto e una forte inclinazione creativa, che stimola l'emisfero destro del cervello. Tuttavia, quando c'è una lateralità mista, le prestazioni tendono ad essere inferiori.

Questa classificazione suggerisce che la scelta di un linguaggio di programmazione non riguarda solo aspetti tecnici, ma può anche influenzare il modo in cui un programmatore pensa e affronta la risoluzione dei problemi. Inoltre, ogni paradigma richiede specifiche caratteristiche cognitive per essere utilizzato al meglio.

Castro (2020) va oltre questi risultati, analizzando come l'esperienza pregressa in un paradigma possa condizionare il modo di pensare di un programmatore nel tempo. L'autore sostiene che il paradigma influisce sulle decisioni di design più del linguaggio di programmazione, influenzando così l'architettura del software e l'approccio alla risoluzione dei problemi. Secondo Castro, il paradigma con cui un programmatore ha maggiore familiarità diventa il modello mentale dominante e, per questo, cambiare paradigma richiede un enorme sforzo cognitivo e un significativo adattamento mentale, ancora più che cambiare linguaggio di programmazione.

Queste ipotesi trovano,inoltre, supporto indiretto dagli studi neuroscientifici, come quelli basati su fMRI, che, come già abbiamo analizzato (**Tabella 2.1**). Hongo et al. (2023) infatti, hanno analizzato le differenze tra i loro studi e quelli presenti in letteratura, riflettendo sulle divergenze emerse. Per esempio, alcuni studi precedenti avevano osservato un'attivazione cerebrale nelle aree del linguaggio con Java, mentre nel loro caso era stato utilizzato Processing, un linguaggio più orientato alla grafica e all'animazione, e l'attivazione delle aree del linguaggio risultava quasi assente. Secondo gli autori, ciò suggerisce che il tipo di linguaggio di programmazione potrebbe essere la causa delle differenze e, di conseguenza, essere in grado di influenzare la neuroplasticità e le attivazioni cerebrali in modi differenti.

Di tutt'altra opinione, tuttavia, è la rassegna sistematica della letteratura di Scherer et al. (2018) nella quale non emerge una differenza significativa tra linguaggi visuali (Scratch, Alice) e testuali (Java, C) nel trasferimento delle competenze.Ciò suggerisce che, forse, non

è il paradigma di programmazione a determinare i miglioramenti cognitivi, ma il processo stesso di programmazione.

In conclusione, sebbene emergano indizi significativi riguardo al modo in cui i diversi linguaggi di programmazione possano influenzare i processi cognitivi e il pensiero, la letteratura attuale risulta ancora limitata e non del tutto uniforme. Le ricerche condotte finora offrono spunti importanti, ma i risultati sono spesso contrastanti, talvolta difficili da generalizzare e non sufficienti per fornire risposte concrete, permettendoci al massimo di abbozzare alcune ipotesi. La letteratura in questo campo è ancora in fase di sviluppo, ma le prospettive sono promettenti. Già alcune università hanno avviato studi sull'argomento(Mitchum (2017)), riconoscendo che i risultati potrebbero trasformare significativamente il nostro approccio ai linguaggi di programmazione, influenzando sia il modo in cui li creiamo sia fornendo implicazioni importanti per il settore cognitivo. A differenza delle lingue naturali, infatti, i linguaggi di programmazione sono Turing-completi, il che significa che qualsiasi programma scritto in un linguaggio può essere tradotto e fatto funzionare in un altro e ciò potrebbe essere estremamente interessante.

Alla luce di queste evidenze, la ricerca futura dovrebbe approfondire diversi aspetti ancora poco esplorati. In particolare, sarebbe utile indagare in modo più sistematico l'effetto dei paradigmi di programmazione sulle strategie cognitive, utilizzando metodologie neuroscientifiche e studi longitudinali per verificare come il pensiero dei programmatori si sviluppa nel tempo in relazione al linguaggio utilizzato.

Il paradigma dichiarativo potrebbe favorire un ragionamento più astratto e generale, dato che richiede di concentrarsi su cosa ottenere piuttosto che su come farlo. Questo potrebbe rendere i programmatori più inclini a pensare in termini di regole e proprietà piuttosto che di sequenze operative.

Il paradigma imperativo, invece, abitua a un pensiero più sequenziale e orientato all'azione, dove il focus è sul controllo esplicito dello stato. Chi usa principalmente questo approccio potrebbe sviluppare una mentalità più operativa e procedurale, facilitando la gestione passo-passo dei problemi. Il paradigma procedurale, essendo una specializzazione dell'imperativo, rafforza il pensiero algoritmico e strutturato,incoraggiando una mentalità analitica, dove la suddivisione in sotto-problemi e l'organizzazione gerarchica delle operazioni diventano fondamentali.

Il paradigma orientato agli oggetti potrebbe promuovere un pensiero più sistemico e relazionale, portando i programmatori a vedere i problemi come insiemi di entità interconnesse. Questo potrebbe sviluppare una maggiore attitudine alla modellazione e all'astrazione delle strutture del mondo reale.

Il paradigma logico potrebbe allenare il pensiero deduttivo e formale, poiché si basa sulla definizione di regole e sulla derivazione automatica di conclusioni. I programmatori abituati a questo approccio potrebbero essere più predisposti a pensare in termini di vincoli e condizioni piuttosto che di azioni sequenziali.

Infine, il paradigma funzionale potrebbe potenziare la capacità di pensare in modo matematico e trasformazionale. L'uso di funzioni pure e l'immutabilità potrebbero portare a un approccio più astratto e modulare ai problemi, sviluppando così una tendenza a vedere le operazioni come combinazioni di trasformazioni sui dati.

Un altro aspetto particolarmente interessante riguarda l'effetto dei linguaggi multi-paradigma, come Python. Se i linguaggi che aderiscono a un unico paradigma tendono a favorire schemi di pensiero specifici, quelli che permettono di mescolare diversi paradigmi potrebbero rendere il ragionamento più flessibile. Tuttavia, questo potrebbe anche richiedere uno sforzo cognitivo maggiore per adattare differenti stili di programmazione all'interno dello stesso codice. Questo aspetto rappresenta una sfida ulteriore per chi studia l'influenza della programmazione sul pensiero: non solo il paradigma adottato, ma anche il modo in cui il programmatore alterna diversi paradigmi potrebbe giocare un ruolo chiave.

2.4 Programmazione e creatività

La programmazione è tradizionalmente percepita come un'attività tecnica, basata su logica e rigore formale. Eppure, il cuore del lavoro di un programmatore è la risoluzione di problemi, un processo che richiede immaginazione e creatività. Per definizione, creativo è ciò che è capace di creare, e creare significa produrre attraverso abilità immaginativa. Questo è esattamente ciò che fa un programmatore: immagina soluzioni. Non nel senso artistico tradizionale, ma nel senso di dare forma, attraverso il codice, a idee nuove e funzionali. Nel software, le soluzioni non sono predeterminate. Il codice può essere scritto in modi diversi, senza vincoli rigidi, se non quelli imposti dallo scopo del progetto e dal linguaggio di programmazione scelto. In questo senso, la programmazione può essere considerata una disciplina che fonde logica e creatività.

Diversi studi, tra cui quello di Pasini et al. (2017), hanno evidenziato come la creatività e il pensiero divergente siano positivamente correlati alle performance di programmazione.Ciò suggerisce che le persone con una maggiore capacità di pensare in modo originale e trovare soluzioni fuori dagli schemi hanno più probabilità di eccellere nello sviluppo software.

Questa relazione tra creatività e programmazione non è solo unidirezionale: oltre a richiedere creatività, il coding stesso può favorirne lo sviluppo. Clements (1995) ha dimostrato che l'uso di software creativi ,tra cui Logo, nell'educazione può migliorare il pensiero divergente e la produzione creativa. Tuttavia, il supporto dei docenti è cruciale affinché questi strumenti siano pienamente efficaci; solo strategie didattiche adeguate, infatti, sono in grado di massimizzare il potenziale creativo negli alunni.

Questo risultato è successivamente stato rafforzato dagli studi di Peppler and Warschauer (2012) , dove Scratch si è dimostrato uno strumento accessibile e innovativo per l'apprendimento creativo, anche da parte di bambini con disabilità cognitive. In particolare, Brandy, una bambina di 9 anni con difficoltà superiori alla media nella lettura e nella scrittura, attraverso un percorso personalizzato con Scratch, ha progressivamente sviluppato una

maggiore creatività, che le ha consentito di acquisire nuove competenze e di rafforzare la propria identità sociale. Questo studio rappresenta inoltre uno dei pochi approcci longitudinali nel campo, poiché analizza l'evoluzione della creatività nel corso di due anni e mezzo.

A supporto di questa prospettiva, Romero et al. (2017) ha esaminato lo sviluppo del pensiero computazionale attraverso la programmazione creativa. Gli esiti dello studio hanno messo in luce come la programmazione non sia solo un esercizio tecnico, ma una vera e propria forma di creatività, capace di stimolare il problem-solving in modo innovativo. Inoltre, è emersa chiaramente l'influenza della creatività sul pensiero computazionale, dimostrando così che quest'ultimo non si limita alla complessità algoritmica, ma include anche la capacità di adattare e modellare soluzioni creative per risolvere problemi. Gli autori , dopo aver osservato i risultati della ricerca, hanno proposto di includere la programmazione creativa nei curriculum scolastici per sviluppare competenze cognitive trasversali, introducendo il concetto di Zona di Creatività Prossimale (ZPC), ovvero il livello ottimale di creatività che un'attività di programmazione dovrebbe stimolare. Suggeriscono inoltre che i linguaggi di programmazione dovrebbero essere valutati non solo in base alla loro complessità tecnica, ma anche per la capacità di favorire strategie cognitive creative, sottolineando l'importanza di nuovi studi sull'argomento.

L'idea che la programmazione possa essere un mezzo di espressione è stata esplorata anche da Penge (2019) che con il suo articolo ha esaminato la relazione tra discipline umanistiche e coding, evidenziando come la programmazione possa diventare uno strumento per il ragionamento strutturato, la creatività e il pensiero critico. Inoltre, se utilizzato nel modo giusto, il coding può diventare uno strumento espressivo al pari della scrittura o della musica. L'aspetto fondamentale è scegliere il linguaggio di programmazione adeguato, poiché ogni linguaggio enfatizza aspetti differenti come, per esempio, la manipolazione grafica piuttosto che la logica e il ragionamento simbolico.

In quest'ottica ,Amini et al. (2024) evidenzia come programmatori più creativi sviluppino soluzioni più dinamiche e astratte, rafforzando l'idea che i linguaggi di programmazione flessibili (come quelli orientati agli oggetti o funzionali) possano incentivare un approccio più creativo rispetto a linguaggi più rigidi.

Questi risultati sono stati ulteriormente confermati da analisi più recenti. Scherer, in due revisioni sistematiche della letteratura (SLR) condotte nel 2018 e nel 2021, ha analizzato gli effetti positivi della programmazione sulle abilità cognitive (Scherer et al. (2018, 2021)) e ha riscontrato come essa possa favorire lo sviluppo della creatività soprattutto negli aspetti di flessibilità, fluidità, elaborazione e originalità , oltre a potenziare il pensiero critico e le capacità di problem-solving.

Inoltre, studi come quello di Wang et al. (2021) hanno trovato una correlazione positiva tra l'apprendimento della programmazione e lo sviluppo della creatività nei bambini in età prescolare, in particolare sulla fluency e sull'originalità, suggerendo che il coding è in grado di stimolare la generazione di nuove idee anche nei bambini piccoli.

Anche Sendag et al. (2023) ha analizzato il rapporto tra l'insegnamento della programmazione e il pensiero creativo. I risultati hanno dimostrato che l'insegnamento tradizionale della programmazione ha migliorato la creatività del 15% mentre l'integrazione di tecniche creative ha portato ad un incremento del 26%. L'originalità è risultata l'abilità creativa che ha beneficiato maggiormente della programmazione. Dallo studio è emerso anche che il pensiero computazionale e la creatività risultano collegati , infatti chi nell'esperimento ha seguito i corsi che integravano le tecniche creative ha registrato miglioramenti anche nelle capacità computazionali. Lo studio sottolinea , quindi, che il coding sia in grado di stimolare il pensiero creativo ma che il metodo di insegnamento influisce significativamente sull'entità di questi effetti, suggerendo che una didattica ottimizzata possa massimizzare i benefici.

Dello stesso parere è Aytekin and Topcu (2024) , il quale ha riscontrato miglioramenti nella creatività e nel pensiero computazionale attraverso l'insegnamento tradizionale di Scratch ai bambini delle scuole elementari. Tuttavia, è emerso che l'utilizzo di strategie come giochi e attività artistiche ha portato a risultati nettamente migliori. Questo probabilmente perché, soprattutto nei bambini più piccoli, queste attività sono state in grado di coinvolgerli maggiormente sia a livello fisico che emotivo. Inoltre, i bambini non avevano alcuna familiarità pregressa con la programmazione, aspetto che potrebbe aver influenzato i risultati. Nonostante ciò, i risultati hanno dimostrato che, sebbene la programmazione sia di per sé uno strumento efficace, a fare la differenza sono i metodi di insegnamento adottati.

In conclusione, l'analisi della letteratura ha evidenziato che la creatività non si tratta di un'abilità esclusiva di alcune persone, ma di una competenza che può essere stimolata in determinati contesti o favorita da specifiche attività, come il coding. La programmazione, infatti, non è solo scrittura di codice, ma un processo che coinvolge analisi, modellazione e design thinking. Inoltre, non solo i programmatori più creativi tendono a ottenere risultati migliori, ma, in ambito educativo, è stato dimostrato che la programmazione può essere utilizzata per sviluppare capacità creative e di risoluzione dei problemi, trasformandosi in una vera e propria strategia pedagogica anziché in un semplice strumento tecnico. Questo ridisegna la percezione della programmazione, che non può più essere considerata una mera disciplina tecnica, ma assume una dimensione più ampia, vicina all'espressione artistica e all'innovazione concettuale. Infine, è emerso il ruolo cruciale dell'educazione nel permettere alla programmazione di esprimere al massimo il suo potenziale.

Attualmente, non esistono studi sistematici che mettano a confronto diretto diversi linguaggi di programmazione in relazione alla creatività. Tuttavia, alcuni degli studi analizzati suggeriscono che i linguaggi con una sintassi più flessibile o orientati al pensiero astratto potrebbero favorire la creatività più di quelli con regole rigide e strutturate. I linguaggi visuali, ad esempio, sembrano facilitare l'apprendimento e la sperimentazione, mentre quelli funzionali enfatizzano l'astrazione e la modellazione concettuale. Anche i linguaggi formali, che pongono l'accento sulla teorizzazione e sull'astrazione, potrebbero stimolare il pensiero creativo e affinare le capacità di problem solving avanzate. I linguaggi orientati agli oggetti, invece, si basano su strutture chiamate oggetti e adottano principi

come incapsulamento, ereditarietà e polimorfismo. Queste caratteristiche favoriscono la modularità e il riuso del codice, rendendo lo sviluppo software più organizzato e intuitivo, poiché permettono di modellare il codice in modo simile alla realtà. Al contrario, i linguaggi più rigidi e a basso livello impongono vincoli stringenti che, da un lato, potrebbero apparire limitanti per l'espressione creativa, ma dall'altro – come sosteneva ad esempio Zemanek (1966)– questa rigidità potrebbe stimolare l'uso di nuove tecniche di risoluzione dei problemi, favorendo un maggiore focus su ottimizzazione ed efficienza. Infine, alcuni linguaggi specificamente progettati per applicazioni artistiche e musicali, come Processing, dimostrano come la programmazione possa diventare un vero e proprio mezzo di espressione creativa, avvicinandosi al mondo dell'arte e discipline umanistiche.

Un possibile quadro teorico che potrebbe essere utilizzato nel futuro per interpretare la relazione tra linguaggi di programmazione e creatività è il concetto di Cognitive Fit (Shaft and Vessey (2006)). Questa teoria evidenzia come la corrispondenza tra il formato dell'informazione e le capacità cognitive dell'utente influenzi la performance nella risoluzione dei problemi. Applicata alla programmazione, significa che i paradigmi che incoraggiano un elevato livello di cognitive fit possono migliorare significativamente le prestazioni di problem solving. Questo suggerisce che linguaggi con strutture più flessibili , intuitive e dal basso carico cognitivo possono favorire forme diverse di creatività,facilitare la comprensione del codice e migliorare le performance nelle modifiche, rispetto a linguaggi più strutturati e rigidi.

La letteratura, quindi, dovrebbe approfondire queste direzioni per comprendere meglio come utilizzare i linguaggi di programmazione per stimolare la creatività e come insegnarli in modo da sfruttarne al meglio il potenziale.

Capitolo 3

Conclusione

L'analisi della letteratura ha dimostrato che imparare a programmare può portare a diversi benefici cognitivi. Da un lato, aiuta a migliorare le competenze di programmazione e a sviluppare il pensiero computazionale. Dall'altro, favorisce anche abilità cognitive di alto livello, come memoria, capacità di analisi, ragionamento e creatività.

Uno dei risultati più significativi è stato verificare che la programmazione non solo influisce sui processi cognitivi, ma può anche modificare la struttura del cervello grazie alla neuroplasticità. Gli studi recenti di neuroimaging, infatti, hanno dimostrato che l'apprendimento del coding induce cambiamenti in alcune aree cerebrali coinvolte nella memoria di lavoro e nel problem-solving, avvalorando le ipotesi sul suo potenziale come strumento di sviluppo cognitivo.

Inoltre, la ricerca ha dimostrato che la programmazione non è un'abilità riservata esclusivamente alle discipline STEM. Come evidenziato da Katai (2014) , gli studenti umanistici sono in grado di sviluppare competenze algoritmiche molto simili a quelle degli studenti scientifici, confermando che il pensiero computazionale non è esclusivo delle scienze dure. Questo suggerisce che la programmazione potrebbe essere integrata anche in ambiti umanistici, favorendo nuove forme di apprendimento interdisciplinare e potenziando le capacità analitiche e creative in diversi campi del sapere.

In conclusione, alla luce di questi risultati,la programmazione si è confermata come un'ottima aggiunta per i curriculum scolastici, anche per i bambini in età pre-scolare, in quanto permette di sviluppare efficacemente abilità estremamente richieste nel XXI secolo e considerate essenziali per il mondo del lavoro, quali:il pensiero logico, il problem solving, la creatività, la capacità di pianificazione , la resilienza, la collaborazione e l'alfabetizzazione digitale.La crescente domanda di professionisti con competenze di coding non fa altro che confermare il valore della programmazione non solo come disciplina tecnica , ma anche come catalizzatore di abilità trasversali applicabili in molteplici contesti.

Tuttavia, diversi aspetti rimangono ancora poco esplorati. L'impatto specifico dei diversi linguaggi di programmazione sul pensiero e sulla creatività non è del tutto chiaro. Dalla rassegna è emerso che i linguaggi di programmazione possono essere considerati veri e propri mezzi attraverso i quali i programmatori ragionano e strutturano le soluzioni ai problemi. Di conseguenza, i loro fondamenti, le loro architetture e le loro regole potrebbero essere in grado di stimolare o, al contrario, limitare il ragionamento, la creatività e il problem-solving. Nonostante queste ipotesi,la scarsità di ricerche sistematiche sull'argomento non ha permesso di ottenere risposte definitive.

Di fronte a queste incertezze, emergono alcune aree di studio che meritano maggiore attenzione. In particolare, è essenziale considerare: :

1. **I limiti della ricerca attuale**, che evidenziano la necessità di ulteriori studi per comprendere meglio l'impatto del coding sul cervello.
2. **Le nuove direzioni di ricerca**, che dovrebbero esplorare l'interazione tra linguaggi, paradigmi e processi cognitivi attraverso studi più approfonditi.
3. **Il ruolo dell'insegnamento**, poiché la programmazione, senza una didattica strutturata, rischia di non produrre i benefici sperati.
4. **Le implicazioni e applicazioni pratiche**, che potrebbero portare a innovazioni sia in ambito educativo sia nel settore dello sviluppo software.

Nelle sezioni seguenti, analizzeremo più nel dettaglio ciascuno di questi aspetti.

3.0.1 Limiti dello studio e direzioni future

L'analisi ha evidenziato alcune limitazioni nella letteratura. Come abbiamo già visto, la ricerca sulla relazione tra linguaggi di programmazione e processi cognitivi è ancora in fase di sviluppo e spesso caratterizzata da approcci metodologici eterogenei. Molti studi si concentrano su singoli linguaggi o paradigmi, senza considerare un confronto sistematico tra essi, rendendo difficile generalizzare i risultati.

Inoltre, la maggior parte delle ricerche si basa su esperimenti a breve termine, lasciando aperte molte domande sugli effetti duraturi della programmazione sul pensiero e sulla creatività.

Per il futuro, sarebbe utile condurre studi longitudinali che analizzino come la programmazione influenzi il pensiero nel corso del tempo, per osservare se i suoi effetti diminuiscano o si consolidino nel tempo. Inoltre, sarebbero utili analisi comparative tra linguaggi di programmazione, per verificare se alcuni paradigmi stimolino maggiormente la creatività o il problem-solving rispetto ad altri e ricerche che indaghino sull'influenza del contesto. Il modo in cui i programmatori pensano, infatti, potrebbe essere modellato non solo dal linguaggio utilizzato, ma anche dalle modalità di apprendimento, dall'esperienza pratica

e dall'ambiente di lavoro. Sviluppare questi studi permetterebbe di delineare un quadro più completo degli effetti della programmazione sul cervello umano.

È probabile che le scoperte più interessanti arrivino dalle neuroscienze, soprattutto grazie alle tecniche di neuroimaging. Questi strumenti potrebbero aiutarci a capire meglio se e come linguaggi di programmazione diversi attivano aree differenti del cervello, offrendoci una nuova prospettiva su come pensiamo e ragioniamo quando scriviamo codice.

3.0.2 Il ruolo dell'insegnamento

Un aspetto centrale emerso in molte ricerche riguarda il ruolo dell'apprendimento e dell'istruzione strutturata. Numerosi studi trattati hanno evidenziato l'importanza dell'insegnamento come fattore determinante nel miglioramento delle abilità cognitive, dimostrando che il semplice atto di programmare non garantisce automaticamente un trasferimento efficiente delle competenze. Infatti, il successo nell'apprendimento varia in base a fattori come il linguaggio scelto, il livello di istruzione e la durata dell'intervento.

L'insegnamento e la creatività

Tra tutte le abilità cognitive potenziate dalla programmazione, la creatività sembra essere quella più influenzata dall'insegnamento. Infatti, in Sendag et al. (2023), integrare nell'insegnamento attività creative come il brainstorming, gli input casuali e le parole tabù ha quasi raddoppiato il pensiero creativo rispetto ai risultati ottenuti con la didattica tradizionale dell'informatica. Questo risultato è stato confermato anche da Aytekin and Topcu (2024), dove l'approccio unplugged—basato su giochi e attività artistiche, senza l'uso di Scratch o altri ambienti digitali—ha migliorato significativamente la capacità di risoluzione creativa dei problemi rispetto al metodo tradizionale. Questo è accaduto perché, soprattutto nei bambini, il coinvolgimento fisico ed emotivo favorisce un maggiore sviluppo creativo. Per chi non ha familiarità con i linguaggi di programmazione, l'iniziale elevato carico cognitivo richiesto per apprenderli potrebbe rallentare lo sviluppo della creatività, rendendo più efficace un approccio più libero e intuitivo soprattutto nelle fasi iniziali dell'apprendimento.

L'Insegnante come Mediatore Cognitivo

L'insegnante gioca un ruolo cruciale nel processo di apprendimento della programmazione, soprattutto nella capacità degli studenti di trasferire le competenze acquisite a contesti diversi. Gli studi hanno dimostrato che l'insegnamento strutturato e mirato è essenziale per massimizzare l'impatto della programmazione sullo sviluppo cognitivo e sulle capacità di problem-solving.

L'apprendimento della programmazione non può essere lasciato alla sola esplorazione individuale dello studente. Le ricerche sottolineano che, senza una guida adeguata, molti studenti faticano a sviluppare una comprensione profonda dei concetti computazionali.

L'insegnante, quindi, deve assumere il ruolo di facilitatore, aiutando gli studenti a costruire modelli mentali solidi e a sviluppare strategie efficaci per la risoluzione dei problemi.

L'Importanza di un Insegnamento Mirato

Un approccio didattico generico non è sufficiente per garantire un apprendimento efficace della programmazione. L'insegnamento deve essere adattato alle capacità cognitive degli studenti e deve seguire una progressione strutturata. Tra le strategie più efficaci si evidenziano:

- **Progressione graduale della difficoltà**: introdurre concetti semplici prima di affrontare quelli complessi, evitando un sovraccarico cognitivo.
- **Uso di esempi concreti**: facilitare l'apprendimento attraverso la connessione con situazioni reali, migliorando la comprensione e il trasferimento delle conoscenze.
- **Metodologie attive**: incoraggiare gli studenti a sperimentare, sbagliare e correggere, sviluppando così il pensiero critico e la capacità di problem-solving.
- **Personalizzazione dell'insegnamento**: adattare i metodi alle diverse esigenze degli studenti, riconoscendo che ognuno ha uno stile di apprendimento differente.

Tecniche di apprendimento utili

Nella rassegna è emerso che il metodo più efficace per l'apprendimento della programmazione è quello Attivo che ha radici nel Costruzionismo proposto da Papert negli anni '80.

Secondo Papert, l'esperienza concreta gioca un ruolo centrale nei processi cognitivi. L'apprendimento, infatti, risulta più efficace e veloce quando il soggetto è coinvolto nella creazione di artefatti tangibili e condivisibili con gli altri.

Il costruzionismo, di conseguenza, integra in un processo in cui l'attività mentale e la costruzione pratica si influenzano reciprocamente. Questo approccio si basa sul principio del "learning by doing" (imparare facendo), ovvero un apprendimento che avviene attraverso l'esperienza diretta e la collaborazione con gli altri. Apprendere in questo modo significa realizzare artefatti, condividerli, discuterli e modificarli grazie al confronto con la collettività.

Il costruzionismo si distingue dall'approccio didattico tradizionale perché propone un percorso flessibile e dinamico, in cui lo studente sviluppa capacità di problem solving, elaborando concretamente soluzioni e adattandole agli ostacoli incontrati. Inoltre, è stato dimostrato che questo metodo potenzia anche le capacità di pianificazione e di gestione delle risorse necessarie per raggiungere un obiettivo(redazione di IONOS (2020))

Nella didattica per bambini, l'apprendimento attivo ha dimostrato di essere particolarmente efficace: attraverso esperienze dirette, giochi e strategie creative, gli studenti mantengono un alto livello di coinvolgimento emotivo e sviluppano sia la creatività sia il pensiero computazionale. In questo contesto, Scratch si è affermato come uno degli strumenti più utilizzati per introdurre i concetti di coding in modo intuitivo, senza il peso della sintassi formale. La sua interfaccia visiva favorisce un'interazione diretta e semplice, rendendolo adatto sia ai bambini sia ai principianti. Inoltre, riducendo il carico cognitivo e incentivando un approccio basato sulla manipolazione corporea, facilita l'apprendimento delle basi del pensiero algoritmico in modo naturale e accessibile.

In questa direzione, Li et al. (2023) ha sottolineato l'importanza di un approccio didattico equilibrato, in cui un ambiente stimolante rafforza la motivazione, le attività pratiche incentivano la partecipazione senza compromettere la riflessione critica e l'autoregolazione degli studenti ottimizza l'apprendimento della programmazione. Questo modello non solo facilita l'acquisizione di competenze tecniche, ma promuove anche lo sviluppo di abilità di pensiero superiore, come il pensiero computazionale, il problem-solving, la creatività e il pensiero critico.

Infine, l'uso di metafore narrative, come il concetto di Choice-Point in Prolog, gioca un ruolo fondamentale nel rendere più intuitiva la comprensione di meccanismi complessi, in particolare per chi si avvicina per la prima volta alla programmazione, diventando un mezzo per coinvolgere gli studenti in un percorso di comprensione più naturale e facile, trasformando concetti formali e astratti in immagini e storie facilmente assimilabiliWatt (1998).

Conclusione

Affinché la programmazione abbia un impatto significativo sul pensiero umano, è essenziale un insegnamento guidato e strutturato per massimizzare i benefici cognitivi. Questo tema è di grande rilevanza e dovrebbe essere approfondito attraverso ulteriori ricerche e revisioni, poiché individuare le strategie più efficaci per favorire l'apprendimento della programmazione è essenziale per sfruttare appieno le potenzialità.

3.0.3 Implicazioni e applicazioni pratiche

Studiare come i linguaggi di programmazione influenzano il pensiero e comprendere finalmente le differenze del loro impatto, evidenziando così i loro limiti e le loro potenzialità potrebbe fornire un'enorme opportunità per migliorare sia gli strumenti di sviluppo che le metodologie di insegnamento. La ricerca in questo ambito può portare a innovazioni per rendere la programmazione più accessibile, efficiente e intuitiva. Continuare a esplorare queste differenze è fondamentale per progettare linguaggi che facilitino il ragionamento, riducano gli errori e favoriscano nuove forme di espressione algoritmica.

Di seguito, vedremo in che modo gli sviluppi della ricerca in questo campo possono offrire benefici concreti.

Migliorare gli strumenti di sviluppo

Se, per esempio, determinati linguaggi comportano una maggiore complessità cognitiva o si presentano difficili da comprendere e interiorizzare, comportando così declini nelle prestazioni creative e nell'efficienza , la ricerca potrebbe favorire la creazione di strumenti di debugging e di editor più sofisticati, in grado di aiutare i programmatori a superare questi ostacoli. In questo contesto, potrebbe essere inserita l'ai per rendere gli strumenti "intelligenti" e adattabili al profilo cognitivo del programmatore, offrendo così suggerimenti contestuali o visualizzazioni semplificate del flusso di esecuzione, migliorando così l'efficienza e riducendo il rischio di errori.

Progettazione di nuovi linguaggi di programmazione

Se si riesce a comprendere come i paradigmi di programmazione influenzano il modo in cui i programmatori pensano, questo può diventare una guida fondamentale per progettare nuovi linguaggi che siano :

- **Più intuitivi:** Creando linguaggi che riflettono meglio il modo naturale di pensare dei programmatori .
- **Più facili d'apprendere:**Se certi paradigmi si dimostrano più accessibili a livello cognitivo , come per esempio quelli visivi come evidenziato da White and Sivitanides (2002), questi possono essere integrati per progettare linguaggi incentrati sull'apprendimento rapido.
- **Più specializzati:** Una volta evidenziati quali caratteristiche dei paradigmi favoriscono la creatività piuttosto che il pensiero computazionale , si potrebbero sviluppare linguaggi di programmazione che, oltre a essere efficienti, puntino consapevolmente a potenziare una o più abilità cognitive, raffinando il coding in uno strumento di crescita intellettuale mirata e studiata.

Migliorare la didattica

Dal punto di vista educativo , comprendere come i linguaggi influenzano specifiche capacità cognitive potrebbe permettere di sviluppare materiali didattici specializzati, rendendo così l'apprendimento del coding più efficace e accessibile, soprattutto per i principianti. Riuscire a trovare delle tecniche universali e riconosciute per sfruttare i benefici cognitivi portati dal coding permetterebbe finalmente di sbloccare la programmazione a 360 gradi, rendendola più accessibile ed efficace, e di costruire una didattica realmente fruttuosa, capace di sviluppare competenze profonde e durature.

Inoltre, la ricerca permetterebbe di identificare i linguaggi con il minor carico cognitivo, offrendo agli insegnanti la possibilità di strutturare l'apprendimento in modo più efficace. Partendo da linguaggi più semplici e intuitivi e aumentando gradualmente il livello di complessità, si potrebbe favorire una progressione naturale delle competenze, evitando

che i principianti si sentano sopraffatti dall'intrinseca complessità della programmazione e rendendo l'acquisizione delle abilità più accessibile e motivante.

Ottimizzazione della produttività e creatività aziendale

Anche le aziende e i team di sviluppo potrebbero beneficiare da queste ricerche, poiché comprendere l'impatto dei diversi linguaggi sul pensiero , sulla creatività e sul problem-solving permetterebbe loro di effettuare scelte più consapevoli e mirate in base alle esigenze specifiche dei progetti. Ad esempio , ambienti che richiedono originalità e innovazione potrebbero preferire quei linguaggi che stimolano maggiormente la creatività, mentre i progetti che necessitano di elevata precisione e rigore potrebbero optare per linguaggi ottimizzati nella risoluzione dei problemi e nell'efficienza computazionale. In questo modo la ricerca potrebbe contribuire in positivo alle performance generali nello sviluppo software.

Definire il profilo dei programmatori

Infine, la ricerca permetterebbe di comprendere meglio il modo di ragionare dei programmatori veterani. Ad esempio, sapere che un programmatore ha trascorso vent'anni a lavorare con il linguaggio C potrebbe suggerire che abbia sviluppato abilità analitiche avanzate, una memoria affinata, una forte attenzione ai dettagli e una notevole capacità di astrazione e di pensiero matematico. D'altra parte, un programmatore con anni di esperienza in paradigmi orientati agli oggetti potrebbe aver coltivato un pensiero strutturato, una spiccata capacità di generalizzazione, un'abilità nell'organizzazione e nel problem-solving, oltre a un marcato pensiero sistemico e una maggiore creatività. Questo potrebbe rendere i due profili molto diversi tra loro, ognuno con i propri punti di forza e le sue debolezze che, a seconda del contesto, potrebbero risultare decisive, sia nel mondo del lavoro che nell'ambito privato.

Riferimenti bibliografici

Amini, M., Olson, J. A., and Sharafi, Z. (2024). Coding with a creative twist: Investigating the link between creativity scores and problem-solving strategies. In *46th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER'24)*, page 21–25. Association for Computing Machinery.

Aytekin, A. and Topcu, M. S. (2024). Improving 6th grade students' creative problem solving skills through plugged and unplugged computational thinking approaches. *Journal of Science Education and Technology*, 33(6):867–891.

Blackwell, A. F., Petre, M., and Church, L. (2019). Fifty years of the psychology of programming. *International Journal of Human-Computer Studies*, 131:52–63.

Bloom, P. and Keil, F. C. (2001). Thinking through language. *Mind & Language*, 16(4):351–367.

Caracciolo, A. (1966). Some preliminary remarks on theoretical pragmatics. *Commun. ACM*, 9(3):226–227.

Casey, C. (2019). Studying the difference between natural and programming language. *Empirical Software Engineering*, 24(4):1823–1868.

Castro, L. M. (2020). It was never about the language: Paradigm impact on software design decisions. In Silhavy, R., Silhavy, P., and Prokopova, Z., editors, *Software Engineering Perspectives in Intelligent Systems*, pages 159–169, Cham. Springer International Publishing.

Clements, D. H. (1995). Teaching creativity with computers. *Educational Psychology Review*, 7(2):141–161.

Fedorenko, E., Ivanova, A., Dhamala, R., and Bers, M. U. (2019). The language of programming: A cognitive perspective. *Trends in Cognitive Sciences*, 23(7):525–528.

Floyd, B., Santander, T., and Weimer, W. (2017). Decoding the representation of code in the brain: An fmri study of code review and expertise. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*.

Francis, J. M. (2023). The relationship between language and thought: How does language shape human perception of the world? *Literature and Linguistics Journal*, 2(2):12–19.

Grover, S. and Pea, R. (2017). *Computational Thinking: A Competency Whose Time Has Come*. Taylor and Francis Ltd.

Hadjerrouit, S. and Hansen, N. K. (2020). Students engaging in mathematical problem-solving through computational thinking and programming activities: A synthesis of two opposite experiences. In *17th International Conference on Cognition and Exploratory Learning in Digital Age (CELDA 2020)*, page 8.

Hall, D. J., Cegielski, C. G., and Wade, J. N. (2006). Theoretical value belief, cognitive ability, and personality as predictors of student performance in object-oriented programming environments. *Decision Sciences Journal of Innovative Education*, 4(2):237–257.

Hartmann, E. M., Bergum, A., Gorgosch, D., Peitek, N., Apel, S., and Siegmund, J. (2024). Tapping into the natural language system with artificial languages when learning programming. *arXiv preprint*.

Hishikawa, K., Yoshinaga, K., Togo, H., Hongo, T., and Hanakawa, T. (2023). Changes in functional brain activity patterns associated with computer programming learning in novices. *Brain Structure and Function*, 228(7):1691–1701.

Hongo, T., Yakou, T., Yoshinaga, K., Kano, T., Miyazaki, M., and Hanakawa, T. (2023). Structural neuroplasticity in computer programming beginners. *Cerebral Cortex*, 33(9):5375–5381.

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'Reilly, U.-M., Bers, M. U., and Fedorenko, E. (2020). Comprehension of computer code relies primarily on domain-general executive brain regions. *eLife*.

Katai, Z. (2014). The challenge of promoting algorithmic thinking of both sciences- and humanities-oriented learners. *Journal of Computer Assisted Learning*, 31.

Kitchenham, B. (2004). Procedures for performing systematic reviews. Technical report, Keele University, Keele, UK. Technical Report.

Li, W., Huang, J.-Y., Liu, C.-Y., Tseng, J. C., and Wang, S.-P. (2023). A study on the relationship between students' learning engagements and higher-order thinking skills in programming learning. *Thinking Skills and Creativity*, 49:101369.

Liu, Y.-F., Kim, J., Wilson, C., and Bedny, M. (2020). Computer code comprehension shares neural resources with formal logical inference in the fronto-parietal network. *eLife*, 9.

Lupyan, G. and Bergen, B. (2016). How language programs the mind. *Topics in Cognitive Science*, 8:408–424.

Mitchum, R. (2017). Computer programming languages can impact science and thought.

Navarro-Cota, C., Molina, A. I., Redondo, M. A., and Lacave, C. (2024). Individual differences in computer programming: a systematic review. *Behaviour & Information Technology*, 44(2):357–375.

Pasini, M., Solitro, U., Brondino, M., Burro, R., Raccanello, D., and Zorzi, M. (2017). Psychology of programming: The role of creativity, empathy and systemizing. *Methodologies and Intelligent Systems for Technology Enhanced Learning*, pages 83–89.

Pea, R. D. and Kurland, D. M. (1984). On the cognitive effects of learning computer programming. *New Ideas in Psychology*, 2(2):137–168.

Peitek, N., Siegmund, J., Apel, S., Kastner, C., Parnin, C., Bethmann, A., Leich, T., Saake, G., and Brechmann, A. (2020). A look into programmers' heads. *IEEE Transactions on Software Engineering*, 46:442–462.

Penge, S. (2019). Perché fare coding con le materie umanistiche. *Bricks*, 9(1). Analisi tenuta da Stefano Penge: autore, formatore e sviluppatore di software. Non si tratta di un articolo scientifico.

Peppler, K. A. and Warschauer, M. (2012). Uncovering literacies, disrupting stereotypes: Examining the (dis)abilities of a child learning to computer program and read. *International Journal of Learning and Media*, 3:15–41.

Popat, S. and Starkey, L. (2019). Learning to code or coding to learn? a systematic review. *Computers & Education*, 128:365–376.

Prat, C. S., Madhyastha, T. M., Mottarella, M. J., and Kuo, C.-H. (2020). Relating natural language aptitude to individual diferences in learning programming languages. *Scientific Reports*, pages 1–10.

redazione di IONOS, L. (2020). Paradigmi di programmazione: quali sono i principi di programmazione esistenti?

Romero, M., Lepage, A., and Lille, B. (2017). Computational thinking development through creative programming in higher education. *International Journal of Educational Technology in Higher Education*, 14(42):1–15.

Scherer, R. (2016). Learning from the past–the need for empirical evidence on the transfer effects of computer programming skills. *Sec. Educational Psychology*, 7.

Scherer, R., Siddiq, F., and Sánchez-Scherer, B. (2021). Some evidence on the cognitive benefits of learning to code. *Sec. Educational Psychology*, 12:1–5.

Scherer, R., Siddiq, F., and Viveros, B. S. (2018). The cognitive benefits of learning computer programming: A meta-analysis of transfer effects. *Journal of Educational Psychology*, online:1–15.

Sendag, S., Yakin, I., and Gedik, N. (2023). Fostering creative thinking skills through computer programming: Explicit or integrated teaching? *Education and Information Technologies*, 28:10819–10838.

Shaft, T. M. and Vessey, I. (2006). The role of cognitive fit in the relationship between software comprehension and modification. *MIS Quarterly*, 30(1):29–55.

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., and Brechmann, A. (2014). Understanding understanding source code with functional magnetic resonance imaging. In *Proceedings of the 36th International Conference on Software Engineering*, page 378–389. Association for Computing Machinery.

Wang, L., Fengji, G., Xiaoxin, H., Donglin, S., Tengfei, W., and Yan, L. (2021). Measuring coding ability in young children: relations to computational thinking, creative thinking, and working memory. *Current Psychology*, 42.

Watt, S. N. (1998). Syntonicity and the psychology of programming. In *Proceedings of the 10th Annual Workshop of the Psychology of Programming Interest Group (PPIG)*, pages 1–12.

White, G. L. and Sivitanides, M. P. (2002). A theory of the relationships between cognitive requirements of computer programming languages and programmers' cognitive characteristics. *Journal of Information Systems Education*, 13(1):59–66.

Zemanek, H. (1966). Semiotics and programming languages. *Communications of the ACM*, 9:139 – 143.

Zlatev, J. and Blomberg, J. (2015). Language may indeed influence thought. *Frontiers in Psychology*, 6:1631.

Appendici

Appendice A

Rassegna degli articoli raccolti

Autore	Obiettivi	Risultati	Limiti	Riflessioni di collegamento
Zemanek (1966)	- Applicare la semiotica ai linguaggi di programmazione. - Analizzare sintassi, semantica e pragmatica nei linguaggi artificiali. - Confrontare linguaggi naturali e di programmazione. - Esaminare come la formalizzazione influenzi il pensiero degli sviluppatori.	- I linguaggi di programmazione modellano il pensiero computazionale. - La sintassi definisce la struttura, la semantica il significato e la pragmatica l'uso del linguaggio. - Il successo di un linguaggio dipende anche dalla sua adattabilità alla comunità degli sviluppatori. - La formalizzazione garantisce precisione e riproducibilità, a differenza dei linguaggi naturali.	- Approccio teorico senza dati empirici. - Non analizza specifici linguaggi di programmazione.	-Analisi interdisciplinare tra semiotica e programmazione -La pragmatica è cruciale per il successo dei linguaggi di programmazione -I linguaggi influenzano il pensiero degli sviluppatori e la risoluzione dei problemi. -Le scelte sintattiche e semantiche modellano il modo in cui i programmatori concepiscono le soluzioni.
Pea (1984)	- Esaminare se e in che modo l'apprendimento della programmazione influenzi il pensiero logico, il problem-solving e altre abilità cognitive. - Integrare teorie della psicologia dello sviluppo con elementi cognitivi per analizzare il trasferimento delle competenze. - Superare una visione riduzionistica dell'apprendimento della programmazione, evidenziando la complessità dei processi cognitivi coinvolti.	- Le ipotesi sulla programmazione come strumento per migliorare abilità cognitive trovano poco supporto empirico. - Il trasferimento spontaneo delle competenze acquisite con la programmazione a contesti non informatici è limitato. - Gli studenti devono ricevere istruzioni esplicite su come applicare il pensiero computazionale in altri ambiti per massimizzare l'impatto cognitivo. - L'apprendimento della programmazione dipende da abilità cognitive pregresse, come logica, matematica e pensiero procedurale. - Molti studenti non superano le prime fasi dell'apprendimento della programmazione e non sviluppano competenze avanzate. - Senza un approccio strutturato e mirato, la programmazione da sola non garantisce il miglioramento delle capacità cognitive.	- Il contesto storico dello studio (anni '80) potrebbe limitare la sua applicabilità alle tecnologie e metodologie educative odierne. - Non viene esplorato in dettaglio il ruolo delle differenze individuali (es. background matematico, predisposizione al ragionamento logico) nell'apprendimento della programmazione. - Il focus principale è sulla programmazione con Logo, il che potrebbe non essere generalizzabile ad altri linguaggi e ambienti di apprendimento.	- Il paper è ancora rilevante per la SLR in quanto evidenzia i limiti del trasferimento delle competenze acquisite nella programmazione, un tema di dibattito attuale. - La necessità di un insegnamento strutturato e mirato è un elemento centrale nei lavori analizzati nella SLR. - Studi recenti (Scherer, 2016; Scherer, 2021; Hartmann, 2024) confermano che il coding, senza un supporto didattico adeguato, non porta automaticamente a un miglioramento generale delle capacità cognitive.
Clements (1995)	Analizzare il ruolo dei computer nello sviluppo della creatività nei bambini, distinguendo tra software che stimolano il pensiero divergente e quelli che lo limitano.	I software progettati per la creatività (es. Logo, word processing, software di design) favoriscono il pensiero divergente, l'autonomia e l'apprendimento. Tuttavia, è necessario un contesto educativo adeguato e il supporto degli insegnanti per massimizzare l'efficacia.	Il linguaggio Logo, pur avendo avuto un impatto positivo, è oggi obsoleto.	Il paper è una delle prime evidenze sull'impatto positivo dei computer sulla creatività e sul pensiero divergente. Rimane attuale nel sottolineare che la programmazione da sola non basta: deve essere accompagnata da strategie didattiche adeguate.

Watt (1998)	- Esplorare il concetto di sintonicità nella programmazione. - Analizzare il legame tra linguaggi di programmazione e processi cognitivi. - Esaminare come la sintonicità influisca sulla comprensione dei linguaggi. - Confrontare diversi modelli di esecuzione e linguaggi educativi.	- La sintonicità facilita l'apprendimento e la comprensione dei linguaggi. - Linguaggi con metafore intuitive (es. Logo, StarLogo) sono più accessibili. - Il modello narrativo (Choice-Point) in Prolog migliora la comprensione rispetto al Byrd Box. - Cocoa, meno sintonico, è più difficile da apprendere. - La sintonicità dell'ego è più rilevante della sintonicità corporea nella programmazione.	Analizza linguaggi oggi meno utilizzati.	- Mostra come i linguaggi influenzano i processi cognitivi. - Conferma che la sintonicità semplifica l'interazione con il codice. - Sottolinea il valore delle metafore nella progettazione dei linguaggi. - Applicabile a linguaggi moderni come Python e Scratch. - Suggerisce che la sintonicità può favorire l'esplorazione creativa.
Bloom (2001)	- Esplorare se e come il linguaggio influenza il pensiero. - Analizzare le teorie di Sapir-Whorf e Vygotskij. - Distinguere effetti reali da ipotesi non supportate. - Valutare l'impatto del linguaggio sulla memoria, il ragionamento e la categorizzazione.	- Il linguaggio modella il pensiero, ma non ne è il determinante principale. - Effetti universali: il linguaggio arricchisce la cognizione (memoria, calcolo, consapevolezza visiva). - Effetti specifici delle lingue meno rilevanti (critica all'ipotesi forte di Sapir-Whorf). - Il linguaggio facilita la formazione di concetti, ma non li crea da solo. - La capacità cognitiva esiste anche senza linguaggio, come dimostrano bambini pre-verbali e animali.	Non discute direttamente il rapporto tra pensiero e linguaggi di programmazione.	- Contribuisce al dibattito sull'influenza del linguaggio sulla cognizione. - Mostra che il linguaggio supporta, ma non determina, il pensiero. - Evidenzia il ruolo del linguaggio nella memoria e categorizzazione. - Rilevante teorica per l'analisi di linguaggi di programmazione e processi cognitivi.
White (2002)	-Esaminare la relazione tra paradigmi di programmazione e capacità cognitive richieste per apprenderli. -Verificare se il successo nell'apprendimento dipenda dal livello e dallo stile cognitivo dell'individuo. -Formulare una teoria che colleghi i requisiti cognitivi dei linguaggi alle caratteristiche cognitive degli studenti, basandosi su Piaget (sviluppo cognitivo) e McCarthy (dominanza emisferica).	1. Programmazione e sviluppo cognitivo: I linguaggi di programmazione richiedono livelli diversi di sviluppo cognitivo secondo Piaget. 2. Dominanza emisferica e programmazione: Il paradigma procedurale favorisce l'emisfero sinistro (logico e sequenziale), mentre quello OO coinvolge entrambi gli emisferi, combinando logica e organizzazione concettuale. 3. Differenze tra paradigmi: - Procedurale → Richiede pensiero logico e formale, legato all'emisfero sinistro. - OO → Coinvolge logica e astrazione, più accessibile di quello procedurale. - Visuale → Minore richiesta cognitiva, utile per sviluppare il pensiero astratto. - Scripting → Accessibile a studenti con pensiero concreto, legato all'emisfero destro. 4. Implicazioni didattiche: Un mismatch tra la complessità cognitiva del linguaggio e le capacità dello studente può causare frustrazione o noia. L'equilibrio ottimizza motivazione e apprendimento.	- Non tiene conto della diffusione di linguaggi ibridi come Python. - Il concetto di dominanza emisferica del cervello è ancora rilevante seppur è stato ridimensionato: si ritiene ora che entrambi gli emisferi lavorino insieme più di quanto si credesse in passato, sebbene permangano distinzioni tra discipline	-Offre intuizioni pionieristiche ancora attuali (es. apprendimento interattivo, coding visivo, metodologie basate sul gioco). -È uno dei pochi a riflettere sulle differenze cognitive tra i paradigmi di programmazione. -Aiuta a evitare l'uso di linguaggi troppo complessi per gli studenti, riducendo burnout e frustrazione. -Analizza come i linguaggi influenzano il pensiero dei programmatori, la creatività e il carico mentale.

Hall (2006)	- Analizzare il rapporto tra prestazioni nella programmazione OO e tre fattori: (1) credenza nel valore teorico (2) abilità cognitiva (3) personalità. - Colmare una lacuna nella letteratura sui predittori del successo in ambienti OO.	- La combinazione di credenza nel valore teorico, abilità cognitiva e personalità predice il successo in programmazione OO. - Credenza nel valore teorico è il predittore più forte: gli studenti con inclinazione al ragionamento formale ottengono risultati migliori. - Personalità influisce significativamente: autostima e autoefficacia migliorano le prestazioni; un livello moderato di nevroticismo può incentivare l'impegno. - Abilità cognitiva ha un impatto minore del previsto, sfidando l'idea tradizionale che sia l'elemento chiave per il successo nella programmazione OO.	\	-Fattori individuali nel successo della programmazione: Lo studio evidenzia che il successo non dipende solo dalle abilità cognitive, ma anche da credenze e tratti di personalità. - Programmazione OO e pensiero astratto: L'uso degli OOPL richiede competenze specifiche come l'astrazione e il ragionamento teorico, indicando possibili difficoltà per studenti con un profilo cognitivo meno incline a tali processi. - Implicazioni educative: La didattica della programmazione potrebbe essere migliorata tenendo conto dei profili individuali, sviluppando strategie per supportare studenti con diverse attitudini cognitive e motivazionali.
Shaft(2006)	-Analizzare il concetto di *cognitive fit* nello sviluppo e modifica del software. -Esaminare come le caratteristiche dei programmatori e del software influenzano la comprensione e la modifica -del codice. Verificare se un alto *cognitive fit* porta a migliori prestazioni e se un disallineamento causa interferenze nei compiti di programmazione.	- Un elevato adattamento cognitivo facilita la comprensione del codice e migliora le performance nelle modifiche. - Quando il cognitive fit è basso, aumenta il carico cognitivo, causando interferenze nei compiti di comprensione e modifica. - I programmatori esperti sono in grado di gestire un basso adattamento cognitivo meglio dei principianti. - Le rappresentazioni mentali del codice sono influenzate dal linguaggio di programmazione utilizzato e dal tipo di informazioni che il linguaggio enfatizza.	-Il campione dello studio è ristretto (24 programmatori), il che potrebbe limitare la generalizzabilità dei risultati. -Si è utilizzato solo il linguaggio COBOL, limitando l'applicabilità dei risultati ad altri linguaggi di programmazione.	- Rileva come la rappresentazione delle informazioni e il tipo di compito influenzano i processi cognitivi dei programmatori, supportando l'idea che i linguaggi di programmazione siano strumenti cognitivi. - Le scelte del linguaggio di programmazione e la compatibilità tra la rappresentazione mentale e il compito di modifica sono determinanti per la prestazione nel problem solving.
Peppler (2012)	- Esplorare il ruolo dell'uso creativo del coding da parte di bambini con disabilità cognitive. - Studio di Brandy, una bambina di 9 anni con QI basso e difficoltà di lettura/scrittura, che attraverso Scratch diventa una creativa multimediale. - Analizzare l'impatto del coding sulla consapevolezza metalinguistica e sull'identità sociale della bambina. - Studio etnografico in un Computer Clubhouse per bambini svantaggiati, con raccolta di dati per due anni e mezzo.	- Il coding ha portato allo sviluppo di alfabetizzazioni multiple, permettendo a Brandy di acquisire competenze sia tradizionali che digitali. - Ha migliorato la sua partecipazione sociale, passando da un'iniziale marginalità a un ruolo di mentore. - Brandy ha sviluppato una creatività crescente, con progetti ispirati ai suoi interessi e al suo ambiente.		-Dimostra che il coding può essere uno strumento di apprendimento accessibile per bambini con difficoltà cognitive. -Evidenzia come la programmazione favorisca la creatività, il meta-pensiero e la risoluzione di problemi. -Supporta l'idea che i linguaggi di programmazione possano essere strumenti di inclusione sociale e sviluppo dell'autonomia.

Katai (2014)	- Esaminare le differenze tra studenti di discipline scientifiche e umanistiche nell'apprendimento del pensiero algoritmico (AT).	- Le strategie didattiche mirate, come l'uso di rappresentazioni visive e attività pratiche, hanno ridotto il divario tra gruppi. - Gli studenti umanistici hanno acquisito abilità algoritmiche simili a quelle degli studenti scientifici, suggerendo che l'AT non è esclusivo delle discipline STEM. - Un fattore chiave per l'apprendimento è stato il coinvolgimento attivo e la motivazione, supportata da una progressione graduale nella difficoltà.	-Campione limitato -Apprendimento di un solo algoritmo -Durata limitata dell'esperimento	- Rafforza l'ipotesi che i linguaggi di programmazione possano modellare il pensiero al di là delle STEM, estendendo il pensiero computazionale anche a studenti umanistici. - Sottolinea l'importanza di strategie didattiche personalizzate, come l'uso di analoghe artistiche e strumenti e-learning interattivi, per rendere il pensiero algoritmico accessibile a tutti. - Conferma che il pensiero algoritmico influenza le capacità di problem-solving e il ragionamento strutturato anche fuori dal contesto informatico.
Siegmund (2014)	- Usare la risonanza magnetica funzionale (fMRI) per misurare l'attività cerebrale durante la lettura di codice. - Identificare le aree del cervello coinvolte nella comprensione del codice. - Valutare se la metodologia fMRI possa supportare studi empirici nell'ingegneria del software.	- La comprensione del codice coinvolge cinque aree cerebrali dell'emisfero sinistro (memoria di lavoro, linguaggio, problem-solving). - Sovrapposizione significativa tra aree del cervello coinvolte nella programmazione e nella comprensione del linguaggio naturale (Brodmann 21, 44, 47). - La memoria di lavoro è fondamentale per la programmazione, suggerendo un legame tra carico cognitivo e linguaggio di programmazione. - Conferma l'ipotesi di Dijkstra: buone abilità linguistiche migliorano la capacità di programmare.	-Campione limitato -I partecipanti avevano esperienza in Java	- Supporta l'idea che i linguaggi di programmazione siano più simili ai linguaggi naturali che alla matematica pura. - Evidenzia che la programmazione richiede processi cognitivi complessi, suggerendo che diversi linguaggi possano influenzare il pensiero in modi distinti. - Rafforza la teoria che la programmazione modella la mente, adattando il carico cognitivo richiesto per la risoluzione di problemi. - Sottolinea l'importanza della memoria di lavoro nella programmazione, confermando il ruolo chiave del pensiero computazionale.
Zlatev (2015)	- Confutare le quattro principali obiezioni che ostacolano le ricerche sull'influenza del linguaggio sul pensiero. - Dimostrare che è possibile e necessario condurre ricerche in quest'ambito. - Chiarire lo spazio semiotico del dibattito, senza proporre una teoria definitiva.	Il linguaggio influenza il pensiero, ma in modi diversi e non assoluti. Esistono molte sfumature di influenza.	Pur chiarendo il dibattito, non fornisce una soluzione definitiva alla complessità del rapporto tra linguaggio e pensiero.	-Integrazione con Bloom (2001): Offre una prospettiva più recente e meno drastica sulla teoria dell'influenza del linguaggio. - Applicazione ai linguaggi di programmazione: Sostiene che i sistemi simbolici possono influenzare i processi cognitivi, modellando il modo in cui i problemi vengono rappresentati e risolti. - Quadro teorico utile: Fornisce una tassonomia originale e una prospettiva interdisciplinare per future ricerche empiriche.
Lupyan (2016)	- Analizzare il linguaggio non solo come mezzo di comunicazione, ma come strumento di programmazione cognitiva. - Esaminare come il linguaggio modella la cognizione in quattro aree principali:	- Il linguaggio non è solo etichettatura, ma modifica attivamente le rappresentazioni mentali. - Le parole guidano la percezione e l'organizzazione concettuale, influenzando il modo in cui pensiamo e apprendiamo.		Se il linguaggio naturale modella il pensiero, i linguaggi di programmazione potrebbero favorire un pensiero più analitico, sistematico o creativo, a seconda delle loro caratteristiche (es. sintassi rigorosa vs. flessibile).

	1. Apprendimento e insegnamento (trasmissione di conoscenza e linguaggio interiore). 2. Categorizzazione e allineamento concettuale (costruzione di una realtà condivisa). 3. Uso di concetti astratti (costruzione di relazioni tra idee). 4. Evocazione del conosciuto e dell'ipotetico (simulazione mentale e immaginazione). - Approccio basato su evidenze neuroscientifiche ed esperimenti cognitivi.	- Il linguaggio attiva aree senso motorie del cervello, collegando parole ed esperienze fisiche. - La comunicazione ha evoluto il linguaggio per "programmare" le menti, facilitando apprendimento e collaborazione.		
Scherer (2016)	- Esaminare se la programmazione migliora problem-solving, creatività e pensiero computazionale. - Analizzare studi empirici sul trasferimento delle abilità cognitive. - Identificare limiti metodologici della ricerca attuale.	-Scarse prove empiriche a supporto della trasferibilità delle abilità cognitive. - Risultati contrastanti su problem-solving e creatività. - Necessità di studi controllati e aggiornati su linguaggi moderni. - Identifica tre problemi nella ricerca: troppa teoria, debolezza metodologica, focus su linguaggi obsoleti.		- Evidenzia le lacune nella letteratura sulla programmazione come strumento cognitivo. - Conferma che servono nuovi studi empirici per determinare gli effetti reali. - Rileva che il trasferimento delle abilità cognitive è plausibile ma non ancora dimostrato con rigore scientifico.
Floyd (2017)	- Esaminare con fMRI come il cervello umano processa il codice rispetto al linguaggio naturale. - Valutare la relazione tra esperienza nella programmazione e attivazione cerebrale. - Analizzare se il codice è elaborato come un linguaggio naturale o in modo distinto. - Studio su task sperimentali: - Comprensione del codice - Revisione del codice (GitHub pull request) - Revisione della prosa (modifiche testuali).	- Il codice di programmazione è processato in modo distinto dal linguaggio naturale. - Il cervello attiva principalmente regioni prefrontali (cognizione di alto livello, problem-solving, controllo esecutivo), con pattern diversi tra codice e prosa. - Maggiore esperienza = minore differenziazione neurale tra codice e prosa, suggerendo che il codice diventa più "linguistico" con l'esperienza. - Classificazione delle attività cerebrali tramite Machine Learning (GPC): - 79,17% accuratezza nel distinguere codice vs. prosa. - 70,83% codice vs. revisione della prosa. - 61,84% revisione del codice vs. comprensione del codice.	-Piccolo campione di partecipanti -Analizzato solo un linguaggio di programmazione	- Supporta l'idea che la programmazione influenza il pensiero e le capacità cognitive. - Indica che il codice può essere appreso in modo simile a una lingua naturale, ma con differenze significative nei processi neurali.
Grover (2017)	- Analizzare il pensiero computazionale (CT) come competenza chiave del XXI secolo. - Esaminare i concetti e le pratiche fondamentali del CT. - Approfondire l'integrazione del CT in diverse discipline, oltre l'informatica.	- Il CT è fondamentale per STEM e ambiti umanistici, sviluppando competenze trasversali. - Definito come il processo di formulazione e risoluzione di problemi in modo eseguibile da un computer (umano o macchina). - Gli elementi chiave del CT includono logica, algoritmi, pattern recognition, astrazione, automazione. - Le pratiche chiave includono decomposizione del problema, debugging, sviluppo iterativo, collaborazione, creatività. - Il CT è applicabile a discipline come musica, matematica,		- Fornisce un quadro teorico essenziale per la SLR, supportando il legame tra programmazione e sviluppo del CT. - Conferma che la programmazione favorisce abilità cognitive trasferibili, rendendola un elemento chiave per il problem-solving in più discipline. - Sottolinea l'importanza di integrare il CT in diversi contesti educativi, non solo informatici.

		scienze sociali, storia e linguistica.		
Pasini (2017)	- Esaminare il legame tra creatività, pensiero divergente e prestazioni di programmazione. - Analizzare il ruolo del tipo di cervello (Empathizing-Systemizing, E-S) nella programmazione. - Valutazione di: - Pensiero divergente (fluidità, flessibilità, originalità) - Personalità creativa (curiosità, immaginazione, complessità) - Stile cognitivo (EQ-SQ, teoria E-S) - Performance nella programmazione misurata con test teorici e pratici.	- Nessuna correlazione significativa tra EQ/SQ e performance complessiva nella programmazione. - Tra i maschi, SQ è positivamente correlato alla programmazione pratica, ma non a quella teorica. - EQ non ha impatto sulla programmazione, ma tra le studentesse emerge una correlazione negativa tra EQ e SQ. - Creatività e pensiero divergente sono positivamente correlati alle performance di programmazione. - Il campione ridotto limita la generalizzabilità dei risultati.	-Campione piccolo	- Supporta la RQ1 e la RQ2 della SLR, evidenziando che creatività e pensiero divergente migliorano la programmazione.
Romero (2017)	- Esaminare lo sviluppo del pensiero computazionale (CT) attraverso la programmazione creativa. - Analizzare il ruolo della creatività nel problem-solving informatico. - Confrontare valutazione automatizzata (Dr. Scratch) e umana (modello #5c21) per misurare il CT.	- La programmazione non è solo un'attività tecnica, ma uno strumento per sviluppare creatività e problem-solving. - La creatività nel CT non è solo innovazione, ma anche parsimonia (uso efficiente delle risorse). - Dr. Scratch sottovaluta la creatività, valutando solo la complessità algoritmica, mentre il modello #5c21 rileva una maggiore diversità nelle strategie risolutive. - Proposta della Zona di Creatività Prossimale (ZPC): un livello ottimale di creatività stimolabile attraverso la programmazione. - I linguaggi di programmazione dovrebbero essere valutati non solo per la complessità tecnica, ma anche per la loro capacità di favorire strategie cognitive e creative.	-Il campione comprendeva solo studenti senza esperienza -Utilizzato solo un linguaggio, Scratch	- Supporta l'idea che la programmazione non è solo sintassi e algoritmi, ma anche una forma di modellazione cognitiva. - Collega il CT con la creatività, suggerendo che linguaggi più flessibili possono favorire soluzioni più originali rispetto a linguaggi più rigidi. -Conferma che la programmazione influisce sulle capacità cognitive e può essere una strategia pedagogica trasversale.
Scherer (2018)	Analizzare se e in che misura l'apprendimento della programmazione migliori le abilità cognitive generali. Valutare il trasferimento delle competenze a diversi domini cognitivi e il ruolo di variabili contestuali (tipo di linguaggio, livello di istruzione, durata dell'intervento).	1. Trasferimento delle competenze: L'apprendimento della programmazione migliora abilità cognitive generali. Il near transfer (abilità direttamente collegate alla programmazione) mostra un effetto forte, mentre il far transfer (abilità più generali) un effetto moderato. 2. Miglioramenti specifici: Effetti positivi su creatività, abilità matematiche, metacognizione, competenze spaziali e ragionamento. Benefici minimi sul rendimento scolastico e alfabetizzazione. 3. Variabili contestuali: Nessuna differenza significativa tra linguaggi visuali (Scratch, Alice) e testuali (Java, C) nel trasferimento delle competenze. Il miglioramento dipende dal processo di programmazione e		- Solida evidenza empirica dell'impatto della programmazione sulle capacità cognitive. - Conferma il ruolo della programmazione nel potenziamento del pensiero computazionale e della creatività. - Dimostra che il trasferimento delle competenze varia in base al dominio cognitivo di riferimento. - Si evince che non è il paradigma di programmazione a determinare i miglioramenti cognitivi, ma il processo stesso di programmazione.

		dalle strategie cognitive adottate, non dal paradigma usato. 4. Bias di pubblicazione: Studi pubblicati tendono a riportare effetti maggiori rispetto alla letteratura grigia. Il trasferimento è più evidente quando confrontato con gruppi di controllo non trattati.		
Beckwella (2019)	- Fornire una riflessione storica sulla psicologia della programmazione dagli anni '70 a oggi. - Analizzare come il concetto di "programmazione" si sia evoluto includendo attività non tradizionali (es. configurazione dispositivi, fogli di calcolo). - Esaminare le principali teorie cognitive sulla programmazione.	- La psicologia della programmazione si è evoluta attraverso 5 decenni: dagli studi cognitivi individuali (anni '70-'80) alla programmazione come attività sociale (anni 2000). - Le ricerche hanno influenzato usabilità dei linguaggi, collaborazione nei team e programmazione end-user. - Emergono diverse teorie chiave, tra cui: - *Programming as Information Processing* (Brooks, 1977) - *Cognitive Dimensions Framework* (Green & Petre, 1996) - *Attention Investment Model* (Blackwell, 2002) - La programmazione non è solo scrittura di codice, ma un'attività che modella il pensiero e le strategie di problem-solving. - Rilevanza crescente di tematiche come Gender HCI e self-efficacy.		- Fornisce un quadro teorico e storico essenziale per la SLR sulla programmazione come abilità cognitiva. - Supporta l'idea che la programmazione sia un processo cognitivo complesso e non solo una questione tecnica. - Evidenzia il legame tra linguaggi di programmazione e modelli cognitivi, utile per comprendere come il codice influenzi il ragionamento. - Il *Cognitive Dimensions Framework* è particolarmente rilevante per analizzare l'impatto della sintassi e della struttura dei linguaggi sulla risoluzione dei problemi.
Casalnuovo (2019)	- Analizzare il codice sorgente non solo come un mezzo di comunicazione con le macchine, ma anche tra esseri umani. - Esplorare le somiglianze e le differenze tra codice e linguaggio naturale, concentrandosi sulla ripetitività e prevedibilità. - Metodologia: confronto tra corpora di codice Java e testi in inglese, utilizzando modelli linguistici statistici e reti neurali LSTM.	- Il codice sorgente è più ripetitivo e prevedibile rispetto alla lingua naturale, con un'entropia media più bassa. - La ripetitività del codice non dipende solo da vincoli sintattici, ma anche da fattori semantici e cognitivi. - La prevedibilità del codice è vantaggiosa per la collaborazione, il debugging e la manutenzione. - L'uso di pattern standardizzati, idiomi comuni e il riutilizzo di codice contribuiscono alla ripetizione. - La ripetitività facilita la comunicazione tra programmatori, riducendo il carico cognitivo.		- Supporta l'idea che i linguaggi di programmazione abbiano una forte componente comunicativa tra esseri umani. - Rilevante per il tema "coding as literacy", poiché mostra come il codice sia modellato da esigenze sociali e collaborative. - Evidenzia come i linguaggi di programmazione possano modellare il pensiero e le abitudini cognitive dei programmatori. - Ricollega alla discussione sulle differenze tra programmazione e linguaggio naturale, sottolineando il ruolo della prevedibilità e della struttura del codice.
Fedorenko (2019)	Esaminare la relazione tra programmazione e linguaggio naturale, indagando i processi cognitivi e neurali coinvolti. - Metodologia: revisione della letteratura esistente, con riferimenti a studi di neuroimaging e sperimentazioni con bambini. - Analisi delle somiglianze tra linguaggi di programmazione e linguaggi naturali: entrambi sistemi simbolici strutturati	- Le strutture linguistiche e di programmazione condividono elementi comuni come regole sintattiche e composizione gerarchica. - Studi pilota mostrano che imparare a programmare potrebbe potenziare le capacità di elaborazione del linguaggio nei bambini. - Studi di neuroimaging suggeriscono una sovrapposizione parziale tra le		- Supporta l'ipotesi che la programmazione sviluppi abilità cognitive legate al linguaggio piuttosto che alla matematica. - Importante per comprendere come il coding influenzi il pensiero e la cognizione. - Rilevante per discussioni su "coding as literacy" e l'inclusione della

	con regole sintattiche e semantiche.	aree cerebrali coinvolte nell'elaborazione del linguaggio e quelle attivate nella programmazione. - La programmazione potrebbe essere considerata non solo come problem-solving ma anche come espressione simbolica, simile alla scrittura.		programmazione nei curricula educativi. - Può orientare future ricerche su metodi didattici basati su analogie con l'apprendimento linguistico.
Penge (2019)	-Esplora l'intersezione tra coding e materie umanistiche, evidenziando il suo valore educativo oltre il problem-solving. -Analizzare come la programmazione possa favorire il pensiero critico, creativo e astratto nelle discipline umanistiche.	1. Il coding non è solo problem-solving: Tradizionalmente associato al pensiero computazionale e alla logica, il coding ha un valore educativo anche nelle discipline umanistiche. 2. Apprendimento interdisciplinare: La programmazione può essere utilizzata per creare strumenti educativi (dizionari digitali, app per l'analisi testuale), esplorare concetti linguistici, storici e musicali, e sviluppare narrazioni interattive. 3. Formalizzazione del pensiero e consapevolezza cognitiva: Scrivere codice aiuta gli studenti a scomporre problemi complessi in parti più semplici, migliorando il ragionamento strutturato e organizzato. 4. Il coding come espressione creativa: La programmazione, se usata nel modo giusto, può diventare uno strumento espressivo, al pari della scrittura o della musica. 5. Importanza del linguaggio di programmazione scelto: Diversi linguaggi enfatizzano aspetti differenti: - Logo: manipolazione grafica. - Prolog: logica e ragionamento simbolico. - Kojo: programmazione visuale e interattiva.		Rilevanza per la SLR: - Coding e materie umanistiche: Il paper dimostra che il coding non è esclusivo delle STEM, ma può arricchire anche l'apprendimento umanistico. - Influenza del linguaggio di programmazione sul pensiero: L'uso di linguaggi diversi sviluppa competenze differenti, mostrando che il coding non è solo una tecnica, ma un approccio cognitivo. - Stimolo alla creatività e al pensiero critico: L'autore evidenzia come la programmazione possa diventare un mezzo per esplorare nuove modalità di apprendimento e rappresentazione della conoscenza. Implicazioni per studi futuri: Sarebbe interessante approfondire come il coding possa supportare lo sviluppo di abilità astratte in contesti educativi non-STEM e quali metodologie didattiche possano massimizzare il suo impatto.
Popat (2019)	Indagare se la programmazione a scuola serva principalmente per apprendere il coding o se favorisca lo sviluppo di altre competenze educative. Analizzare studi empirici per identificare le competenze trasversali acquisite attraverso il coding.	1. Il coding sviluppa competenze oltre la programmazione: Gli studenti non solo imparano a programmare, ma sviluppano anche abilità cognitive, sociali e accademiche. 2. Cinque categorie di competenze emerse: - Problem-solving matematico: 9 studi su 10 confermano che la programmazione aiuta a sviluppare abilità matematiche, anche se l'efficacia dipende dalle metodologie didattiche. - Competenze sociali e collaborazione: Due studi riportano un aumento dell'interazione sociale, ma non è chiaro se sia dovuto al coding o al contesto di apprendimento. - Autogestione e apprendimento attivo: Tre studi evidenziano maggiore coinvolgimento e autonomia negli studenti.		Rilevanza per la SLR: Questo paper conferma che il coding non è solo un'abilità tecnica, ma influisce sul modo di pensare e risolvere problemi. - Impatto cognitivo: La programmazione aiuta gli studenti a riconoscere pattern, gestire la complessità e strutturare il pensiero. - Interdisciplinarità: Il coding può essere un ponte tra diverse discipline, favorendo un apprendimento più ampio. - Dipendenza dalla didattica: I benefici del coding variano a seconda della progettazione educativa, sottolineando l'importanza di metodologie didattiche mirate. Implicazioni per studi futuri: Servono ricerche per esplorare come l'esposizione

		- Pensiero critico: Cinque studi mostrano che il coding aiuta gli studenti ad analizzare problemi, testare soluzioni e migliorare il proprio lavoro. - Competenze accademiche extra-matematiche: Alcuni studi riportano benefici in materie come storia e arte, ma i miglioramenti riguardano principalmente abilità di ordine inferiore nella tassonomia di Bloom. 3. Un modello globale per classificare l'impatto del coding: Gli autori propongono tre categorie di abilità sviluppate: - Pensiero di ordine inferiore: Comprensione e memorizzazione. - Pensiero di ordine superiore: Pensiero critico e problem-solving. - Abilità personali: Collaborazione e autogestione. 4. Necessità di studi a lungo termine: Mancano ricerche longitudinali per valutare gli effetti del coding sullo sviluppo cognitivo e sociale degli studenti nel tempo.		prolungata alla programmazione possa influenzare il ragionamento logico e la flessibilità mentale nel lungo termine.
Castro (2020)	Indagare l'importanza del paradigma di programmazione rispetto alla scelta del linguaggio di programmazione nelle decisioni di progettazione software. Analizzare come i paradigmi influenzano l'architettura del software e la concezione delle soluzioni.	1. Il paradigma guida le decisioni di design più del linguaggio di programmazione: Il modo di pensare del programmatore è modellato dal paradigma con cui ha più esperienza, influenzando l'architettura del software. 2. Differenze tra paradigmi: - Il paradigma imperativo porta a un'architettura centralizzata con gestione diretta dello stato, ma con maggiore complessità negli errori e scalabilità. - L'OOP offre modularità e riusabilità, ma introduce una complessità strutturale aggiuntiva. - Il paradigma funzionale favorisce testabilità, parallelizzazione e tolleranza ai guasti grazie all'assenza di effetti collaterali. 3. Sforzo cognitivo nel cambio di paradigma: Passare da un paradigma all'altro richiede un significativo adattamento mentale, poiché ciascuno impone vincoli e strutture specifiche alla risoluzione dei problemi.		Implicazioni per l'industria e l'educazione: Le aziende dovrebbero scegliere tecnologie in base al paradigma più adatto alle proprie esigenze, anziché partire da un linguaggio specifico. Nell'insegnamento, è più vantaggioso apprendere più paradigmi piuttosto che focalizzarsi su singoli linguaggi, per sviluppare maggiore flessibilità nel problem-solving. Rilevanza per la SLR: Questo paper supporta l'idea che la programmazione non è solo una questione tecnica, ma cognitiva, poiché ogni paradigma modella il pensiero del programmatore. Si ipotizza che l'esposizione a paradigmi differenti possa influenzare la flessibilità mentale e il ragionamento logico nel lungo periodo, aprendo la strada a future ricerche sulla relazione tra paradigmi e processi cognitivi.
Hadjerrouit (2020)	Analizzare il ruolo del pensiero computazionale (CT) nella risoluzione di problemi matematici, esplorando come la programmazione in MatLab possa supportare l'apprendimento matematico. Indagare le difficoltà e i vantaggi dell'integrazione del	1. Il pensiero computazionale è efficace solo con una solida base matematica: senza una chiara comprensione del problema, gli studenti faticano a sviluppare strategie algoritmiche. 2. Differenze nei metodi di risoluzione: Uno studente con scarso background matematico ha avuto difficoltà nel collegare		Conferma che il coding è uno strumento cognitivo oltre che tecnico: La programmazione aiuta gli studenti a sviluppare capacità di decomposizione, astrazione e formulazione algoritmica, facilitando la transizione dal pensiero matematico a quello computazionale.

	coding nei corsi universitari di matematica.	il problema alla soluzione computazionale, mentre uno studente con una buona preparazione ha affrontato il problema in modo strutturato e autonomo. 3. Importanza dell'usabilità del linguaggio di programmazione: Un buon linguaggio deve essere intuitivo, rapido da apprendere e fornire feedback chiari per supportare gli studenti nella correzione degli errori. 4. Necessità di un approccio didattico personalizzato: Gli studenti con meno autonomia nel problem-solving traggono beneficio dal supporto del docente.		Implicazioni per l'insegnamento: Integrare la programmazione nei corsi di matematica può migliorare la risoluzione dei problemi, ma richiede un'adeguata progettazione didattica e strumenti visivi per facilitare la comprensione. Limiti dello studio: Campione ristretto e dati qualitativi limitano la generalizzabilità dei risultati, suggerendo la necessità di ricerche su un numero più ampio di studenti.
Ivanova (2020)	Indagare i processi cognitivi e neurali alla base della comprensione del codice, verificando se essa sia supportata dal sistema del linguaggio o dal sistema di domanda multipla (MD), responsabile del problem-solving e del ragionamento astratto. Utilizza fMRI per confrontare l'attivazione cerebrale nella lettura del codice(phyton e Scratch) con quella di problemi matematici e linguistici.	1. Forte attivazione del sistema MD: Il codice attiva le regioni frontali e parietali bilaterali coinvolte nel problem-solving, nella memoria di lavoro e nel controllo cognitivo. 2. Bassa attivazione del sistema del linguaggio: A differenza del linguaggio naturale, la lettura del codice non mostra attivazione nelle aree linguistiche tradizionali, suggerendo che il codice non venga elaborato come una lingua naturale. 3. Nessuna area cerebrale specializzata per la programmazione: Il codice viene processato attraverso circuiti cerebrali esistenti per il ragionamento astratto, senza evidenza di regioni dedicate esclusivamente alla programmazione. 4. Differenze con studi precedenti: Risultati in contrasto con Siegmund (2014) e Floyd (2017), probabilmente per differenze nei linguaggi testati, nei livelli di esperienza dei partecipanti e nei metodi di analisi.	-Utilizzo di solo due linguaggi Python e Scratch -Campione piccolo	-Conferma che la programmazione è più vicina al problem-solving che alla linguistica: la comprensione del codice coinvolge il sistema MD più che quello del linguaggio, indicando un forte legame con la logica e la memoria di lavoro. Implicazioni per l'insegnamento: Se il codice viene elaborato principalmente tramite ragionamento astratto e problem-solving, l'educazione alla programmazione potrebbe beneficiare di metodi didattici orientati alla logica piuttosto che alla semplice memorizzazione della sintassi. -Le discrepanze con ricerche precedenti suggeriscono che il livello di esperienza possa influenzare il coinvolgimento del sistema del linguaggio nella comprensione del codice.
Liu(2020)	Studiare le basi cognitive e neurali della comprensione del codice (in questo caso Phyton), testando se i linguaggi di programmazione riciclano meccanismi del linguaggio naturale o dipendono da reti fronto-parietali condivise con la logica e la matematica.	-Attivazione del network fronto-parietale sinistro, coinvolto nella logica formale e nella manipolazione simbolica, con un pattern simile a quello osservato nei compiti di logica, ma distinto dalla matematica. -Bassa attivazione del sistema linguistico, con una sovrapposizione minima tra le aree della comprensione del codice e quelle deputate al linguaggio naturale.Questo suggerisce che il coding non venga elaborato come una lingua naturale. -Sovrapposizione parziale con il sistema di controllo esecutivo, indicando che la programmazione non è	-Campione ridotto. -Analizzato un solo linguaggio.	-Il paper offre prove a favore dell'idea che i circuiti neurali specializzati (come il sistema linguistico) non si adattino facilmente a nuove invenzioni culturali, mentre il network fronto-parietale mostra maggiore plasticità. -Coerenza con Ivanova (2020): conferma il ruolo della rete fronto-parietale e la bassa attivazione del sistema del linguaggio, con qualche differenza nella lateralizzazione. -Contrasto con Floyd (2017) e Siegmund (2014): questi studi avevano trovato coinvolgimento delle aree della lettura nei programmatori esperti.

		semplicemente un compito di memoria di lavoro o attenzione. -Distinzione tra diverse strutture sintattiche del codice: l'analisi MVPA ha mostrato pattern cerebrali distinti per cicli for e condizioni if, suggerendo che il cervello riconosce e categorizza il codice in base alla sintassi. -Lateralizzazione sinistra nella comprensione del codice, sebbene questo non fosse stato osservato in studi precedenti come quello di Ivanova (2020).		
Prat (2020)	Indagare se l'attitudine linguistica sia un predittore significativo dell'apprendimento della programmazione, rispetto alle competenze matematiche.	- L'abilità linguistica predice il successo nell'apprendimento della programmazione, spiegando il 17% della varianza nei risultati. - L'intelligenza fluida e la memoria di lavoro sono i migliori predittori generali, spiegando rispettivamente 34% e alta correlazione con il successo nella programmazione. - Il ruolo della numeracy è minore: le abilità matematiche tradizionali spiegano solo il 2% della varianza. - L'attività cerebrale (EEG) predice il successo: alta attività nelle bande beta e gamma nei lobi frontali-temporali è correlata a migliori prestazioni. - Differenze individuali significative: il miglior studente ha imparato 2,5 volte più velocemente del più lento.		- Mette in discussione il ruolo della matematica nell'apprendimento della programmazione e sottolinea il peso delle abilità linguistiche e cognitive. -Suggerisce che i linguaggi di programmazione possano essere appresi con strategie simili alle lingue naturali, influenzando il modo in cui viene insegnato il coding. - Supporta l'idea che la programmazione non sia solo un processo matematico, ma un'attività cognitiva complessa che coinvolge sintassi, memoria e ragionamento fluido.
Pietek (2020)	Esplorare i processi cognitivi della comprensione del codice tramite fMRI, analizzando le regioni cerebrali coinvolte e l'impatto dell'esperienza di programmazione sul carico cognitivo. Focus su Java e su due compiti: previsione dell'output e analisi sintattica.	1. Attivazione di aree cerebrali associate al linguaggio e alla memoria di lavoro: BA 44, 47 (sintassi), BA 6 (memoria e attenzione), BA 40 (manipolazione simbolica), BA 21 (elaborazione semantica). 2. Maggiore complessità del codice = maggiore sforzo cognitivo. 3. L'esperienza generale nella programmazione non riduce l'attivazione cerebrale, ma la familiarità con un linguaggio specifico sì. 4. Il secondo esperimento conferma in gran parte i risultati, ma introduce BA 39 (integrazione semantica), indicando possibili strategie cognitive diverse tra i gruppi.	-Campione ridotto.	-Conferma il legame tra codice e linguaggio naturale: attivazione delle aree del linguaggio suggerisce un'elaborazione sintattico-semantica del codice.Tuttavia, la corteccia temporale anteriore, legata alla comprensione del discorso, non viene attivata, evidenziando una differenza fondamentale con il linguaggio naturale. -Ogni linguaggio di programmazione ha un proprio carico cognitivo e l'esperienza generale nella programmazione non riduce automaticamente lo sforzo richiesto nell'apprendimento di un nuovo linguaggio. -Conferma dell'intuizione di Dijkstra: una buona padronanza del linguaggio naturale facilita l'apprendimento della programmazione.

Scherer (2021)	Analizzare l'effetto di trasferimento delle competenze sviluppate con la programmazione ad altri ambiti cognitivi e professionali, combinando revisione della letteratura e meta-analisi.	- Il coding non è solo una competenza tecnica, ma coinvolge processi cognitivi come: 1) Astrazione e generalizzazione di pattern 2) Elaborazione sistematica delle informazioni 3) Uso di simboli e rappresentazioni 4) Pensiero algoritmico 5) Debugging e rilevamento degli errori - Studi evidenziano un legame tra programmazione e miglioramento di pensiero critico, metacognizione, abilità matematiche e spaziali. - Gli effetti sono variabili, dipendono da: 1) metodologia didattica 2) durata dell'apprendimento 3) somiglianza tra programmazione e altri compiti cognitivi. - Il problem-solving e la logica mostrano un trasferimento più forte rispetto alla lettura e comprensione del testo. - Il trasferimento delle abilità non è automatico, ma dipende da strategie didattiche e fattori individuali (memoria di lavoro, metacognizione).		- Altamente pertinente perché analizza il legame tra coding e processi cognitivi. - Evidenzia il potenziale del coding per la risoluzione di problemi complessi e il pensiero computazionale. - Discute le condizioni necessarie per il trasferimento efficace delle competenze, contribuendo alla comprensione del reale impatto del coding.
Wang (2021)	-Indagare la capacità dei bambini in età prescolare di apprendere concetti di coding. -Sviluppare uno strumento standardizzato per misurare le competenze di coding nei bambini. -Esplorare le connessioni tra il coding e altre abilità cognitive (pensiero computazionale, creatività, memoria di lavoro).	- Il test è risultato valido e affidabile, con un buon coinvolgimento dei bambini. - Le abilità di coding sono risultate correlate al pensiero computazionale, confermando che la programmazione aiuta nello sviluppo del problem-solving e del ragionamento logico. - Il coding è legato alla creatività, in particolare alla fluency e all'originalità, suggerendo che stimola la generazione di nuove idee. Tuttavia, non sembra influenzare fasi più avanzate della creatività. - Non è emersa alcuna correlazione tra coding e memoria di lavoro, contraddicendo alcuni studi precedenti su programmatori adulti (es. Ivanova, 2020).Questo potrebbe dipendere dal carico cognitivo dei giochi utilizzati nello studio.		- Dimostra che la programmazione può essere insegnata già in età prescolare e che ha effetti sullo sviluppo cognitivo, suggerendo che il coding potrebbe essere un vero e proprio linguaggio cognitivo. - Conferma che la programmazione potenzia il pensiero computazionale, sostenendo l'ipotesi che il coding migliori il problem-solving e la logica. - Mostra che la programmazione stimola alcune dimensioni della creatività, aprendo nuove prospettive su come il coding possa essere usato per sviluppare il pensiero divergente nei bambini. - Solleva dubbi sul rapporto tra coding e memoria di lavoro, suggerendo che l'effetto possa variare in base all'età e al tipo di compiti proposti.
Fracis (2023)	Esplorare il legame tra linguaggio e percezione attraverso una revisione sistematica della letteratura, analizzando come il linguaggio modella il pensiero e viceversa.	- Il linguaggio e il pensiero sono interconnessi: il linguaggio struttura il pensiero e la percezione, mentre il pensiero influenza la produzione linguistica. - Il bilinguismo è legato a una maggiore flessibilità cognitiva.		- Esplora come il linguaggio influenzi i processi cognitivi. -Base teorica per supportare l'ipotesi su come i linguaggi di programmazione potrebbero influenzare la risoluzione di problemi complessi attraverso strutture specifiche.

		- Il pensiero metaforico influenza la comprensione di concetti astratti. - Il linguaggio è plasmato dalla cultura e guida la percezione attraverso categorie cognitive specifiche. - L'ipotesi di Sapir-Whorf (in forma debole) è supportata: il linguaggio influenza la categorizzazione e l'interpretazione della realtà. - L'acquisizione del linguaggio è fondamentale per lo sviluppo di abilità cognitive come memoria, attenzione e ragionamento		- Essendo una revisione sistematica recente, fornisce una panoramica aggiornata e ben documentata sulle connessioni tra linguaggio e pensiero.
Hiskikawa (2023)	Esplorare i meccanismi neurali dell'apprendimento della programmazione attraverso fMRI, analizzando le aree cerebrali coinvolte e verificando se l'attività cerebrale cambia con l'acquisizione delle competenze di coding.	- L'attività cerebrale durante i compiti di programmazione ha coinvolto diverse aree corticali e subcorticali, tra cui il giro frontale inferiore bilaterale (comprensione del codice e problem-solving), i lobi parietali e temporali (memoria di lavoro e analisi delle informazioni), e i nuclei caudati e cervelletto (apprendimento motorio e pianificazione). - Il giro frontale inferiore destro ha mostrato un aumento significativo dell'attività tra l'inizio e la fine del corso, suggerendo il suo ruolo chiave nel miglioramento delle prestazioni. - Il giro frontale inferiore sinistro, pur essendo attivo, non ha mostrato variazioni significative, suggerendo una differenza nel modo in cui il cervello processa il coding rispetto ai linguaggi naturali. - L'attivazione dell'area fronto-parietale sinistra suggerisce un collegamento tra la programmazione e i processi linguistici tradizionali, ma con una maggiore attivazione del sistema di richiesta multipla, tipico dell'apprendimento di una seconda lingua. - Nei principianti, l'attivazione dell'emisfero destro sembra predominante, mentre negli esperti (come evidenziato in studi precedenti) l'attività si sposta verso l'emisfero sinistro.	-Campione ristretto -Analisi di un solo linguaggio di programmazione	- Lo studio è particolarmente rilevante per la SLR perché dimostra che l'apprendimento della programmazione induce cambiamenti neuroplastici funzionali, confermando che il coding ha un impatto misurabile sulle capacità cognitive. - Rafforza l'ipotesi che la programmazione possa essere assimilata all'apprendimento di una seconda lingua, con un'evoluzione dell'attivazione cerebrale simile a quella osservata nei bilingui. - Evidenzia il ruolo del linguaggio di programmazione utilizzato: le differenze tra linguaggi potrebbero influenzare le aree cerebrali attivate, spiegando la variabilità nei risultati degli studi fMRI.
Hongo (2023)	-Indagare i cambiamenti strutturali nel cervello di studenti universitari che apprendono a programmare per la prima volta, al fine di comprendere meglio i meccanismi cognitivi e neurali dell'apprendimento del coding.	- Dopo un corso di 15 settimane su Processing, gli studenti hanno mostrato un aumento della materia grigia in 8 aree cerebrali, tra cui il Poli frontali sinistro e destro(Persistenza nei compiti e raggiungimento degli obiettivi),Giro frontale mediale destro (Ragionamento deduttivo),Cuneo sinistro (Elaborazione visuo-spaziale),Pallidum destro e sinistro (Motivazione e rinforzo dell'apprendimento), Cerebello	-Campione ristretto -Analisi di un solo linguaggio di programmazione	-Dimostra che il coding modifica la struttura cerebrale, influenzando la persistenza, il problem-solving e il ragionamento deduttivo. -Suggerisce che il tipo di linguaggio di programmazione possa influenzare la neuroplasticità in modi diversi.

		laterale sinistro (Coordinazione e sequenzialità delle azioni) e Cerebello mediale (Controllo motorio e gestione del carico cognitivo). - Il volume del polo frontale destro era correlato al successo nel coding, suggerendo un legame tra persistenza e performance. - Un'analisi fMRI su un sottogruppo di partecipanti ha evidenziato attivazione nelle aree del linguaggio, ma senza cambiamenti strutturali significativi. - I risultati suggeriscono che il linguaggio di programmazione utilizzato potrebbe influenzare diversamente la neuroplasticità: Processing (grafico/animazione) non ha prodotto gli stessi effetti di Java, più orientato alla sintassi.		
Li (2023)	-Indagare il legame tra il coinvolgimento cognitivo degli studenti nell'apprendimento della programmazione e lo sviluppo delle competenze di pensiero superiore (pensiero computazionale, problem-solving, creatività, pensiero critico). -Analizzare come le diverse forme di coinvolgimento (comportamentale, emotivo, cognitivo) influenzano tali competenze.	-Il coinvolgimento comportamentale, emotivo e cognitivo influisce positivamente sulle competenze di pensiero superiore. - Il coinvolgimento emotivo rafforza il legame tra coinvolgimento cognitivo, pensiero computazionale e creatività. - Un elevato coinvolgimento comportamentale può indebolire il legame tra coinvolgimento cognitivo e pensiero critico. - Un ambiente positivo favorisce prestazioni cognitive migliori, ma un'eccessiva partecipazione pratica può ridurre la riflessione critica.	-Misurazione basata su questionari self-report che potrebbero introdurre bias soggettivi nelle risposte degli studenti.	-Conferma che la programmazione favorisce lo sviluppo del pensiero computazionale, del problem-solving, della creatività e del pensiero critico. -Evidenzia l'importanza di un approccio didattico bilanciato, che combini attività pratiche con momenti di riflessione per ottimizzare l'apprendimento. -Sottolinea che l'ambiente emotivo gioca un ruolo chiave nel rafforzare l'acquisizione delle competenze cognitive avanzate.
Sengad (2033)	Esaminare se l'insegnamento della programmazione migliori il pensiero creativo e se l'integrazione di tecniche creative possa potenziarne ulteriormente lo sviluppo.	- Entrambi i metodi (insegnamento esplicito e programmazione creativa) migliorano il pensiero creativo, ma in modo diverso. - L'insegnamento esplicito della programmazione ha migliorato la creatività del 15%, mentre l'integrazione di tecniche creative ha portato a un incremento del 26%. - Il pensiero computazionale e la creatività risultano collegati: chi ha seguito la programmazione creativa ha mostrato miglioramenti anche nelle capacità computazionali. - L'originalità è l'abilità creativa che ha beneficiato maggiormente della programmazione.	-Lo studio utilizza solo due linguaggi di programmazione (Scratch e Alice), senza confronti con altri strumenti.	- Rilevante per la SLR perché fornisce prove empiriche del legame tra programmazione e sviluppo del pensiero creativo. - Supporta l'idea che il coding non solo sviluppi competenze computazionali, ma anche capacità di problem-solving innovativo e generazione di idee originali. - Conferma che il metodo di insegnamento influisce significativamente sugli effetti cognitivi della programmazione, suggerendo che una didattica ottimizzata possa massimizzare i benefici. - Si collega ad altri studi nella SLR che evidenziano il ruolo delle strategie didattiche nel amplificare l'impatto della programmazione sulle abilità cognitive.

Aytekin (2024)	-Analizzare il rapporto tra pensiero computazionale e creatività. -Valutare l'efficacia di due approcci didattici (plugged e unplugged) nello sviluppo delle capacità di problem-solving creativo negli studenti di sesta elementare. -Confrontare i risultati con un metodo di insegnamento tradizionale.	-Entrambi gli approcci (plugged e unplugged) hanno migliorato le capacità di risoluzione creativa rispetto al metodo tradizionale. -Il gruppo unplugged ha ottenuto i migliori risultati, con una differenza significativa rispetto al gruppo di controllo. -Il gruppo plugged ha mostrato un miglioramento, ma senza una differenza statisticamente significativa rispetto al gruppo di controllo. -La programmazione digitale può creare barriere cognitive per i bambini inesperti, rallentando il processo creativo. -Il metodo unplugged favorisce un maggiore coinvolgimento emotivo e fisico, stimolando la creatività attraverso materiali fisici e attività interattive. -Il metodo unplugged incentiva la collaborazione e una maggiore motivazione rispetto all'ambiente digitale.	-Lo studio è limitato a un periodo di 4 settimane, senza un'analisi a lungo termine. -Non viene analizzato l'impatto della familiarità pregressa con la programmazione sui risultati ottenuti.	-Dimostra la connessione tra pensiero computazionale e creatività. -Evidenzia come il pensiero computazionale aiuti a scomporre problemi complessi in parti gestibili, favorendo soluzioni innovative. -Sottolinea che gli strumenti digitali non sono sempre la scelta migliore per insegnare il pensiero computazionale. -Suggerisce che un approccio didattico ben progettato possa massimizzare lo sviluppo delle competenze cognitive legate alla programmazione.
Amini (2024)	-Esplorare il ruolo della creatività nella programmazione, ampliando il focus oltre l'ingegneria dei requisiti e analizzando l'aspetto umano del problem-solving nel coding. -Studiare le strategie di problem-solving adottate dagli sviluppatori. -Valutare la correlazione tra pensiero creativo e prestazioni nella programmazione.	- Il pensiero divergente è associato a un approccio più esplorativo e a strategie transformational. - I programmatori più creativi tendono a riformulare i problemi da prospettive innovative. - Nessun legame significativo tra creatività e rapidità di completamento. -I partecipanti con maggiore pensiero divergente impiegano più tempo, adottando un approccio più esplorativo. -Non emerge una correlazione significativa tra pensiero convergente e velocità di completamento.	-Il campione di 77 partecipanti è relativamente ridotto, limitando la generalizzabilità dei risultati. -L'analisi si concentra su un'unica attività di programmazione (implementazione di una funzione di punteggio per Whack-a-Mole). -Non vengono considerate fasi cruciali come debugging o progettazione di algoritmi complessi.	-Approfondisce il legame tra creatività, programmazione e problem-solving, un tema poco esplorato in letteratura. -Suggerisce che i programmatori più creativi tendono a sviluppare soluzioni più dinamiche e astratte.
Hartman (2024)	Investigare se l'apprendimento di una lingua artificiale (Brocanto) prima della programmazione possa attivare strategie linguistiche utili per facilitare la comprensione del codice.	- Nessun miglioramento immediato nelle competenze di coding per il gruppo Brocanto rispetto al gruppo di controllo. - Gli studenti Brocanto si sono sentiti più sicuri nell'affrontare nuove regole sintattiche e alcuni hanno percepito Python come più intuitivo, suggerendo possibili benefici a lungo termine. - I risultati non confermano un trasferimento automatico tra competenze linguistiche e programmazione, ma indicano la necessità di studi a lungo termine.	-L'esperimento è stato di breve durata (una settimana), quindi non ha potuto valutare effetti a lungo termine. -Campione di studenti limitato, possibile variabilità individuale nei risultati.	- Studio rilevante per la SLR perché esplora il legame tra apprendimento del linguaggio naturale e coding. sulla struttura cognitiva della programmazione, suggerendo che potrebbe sviluppare abilità distinte rispetto al linguaggio naturale. - Collegato alle ricerche neuro-cognitive sul processing del codice, come quelle che evidenziano l'attivazione di aree cerebrali comuni tra linguaggio naturale e programmazione.

Navarro-Cota (2024)	-Analizzare l'influenza delle differenze individuali tra i programmatori sulle loro capacità di apprendimento e performance nel coding. -Proporre una tassonomia dei fattori umani rilevanti per il successo nella programmazione. -Comprendere il paradosso dell'elevato tasso di abbandono nei corsi di programmazione nonostante la crescente richiesta di programmatori.	-I tratti cognitivi specifici (ragionamento logico, abilità spaziali, memoria di lavoro, pensiero matematico, computazionale, riflessivo e locale) sono fortemente correlati al successo nella programmazione. -L'intelligenza multifattoriale, in particolare quella linguistica, gioca un ruolo importante nell'apprendimento della programmazione.		-Fornisce una tassonomia dei fattori cognitivi più rilevanti, utile per migliorare l'insegnamento della programmazione e la selezione dei programmatori. -Sostiene l'ipotesi che imparare a programmare possa potenziare abilità cognitive e suggerisce un legame tra coding e apprendimento linguistico.