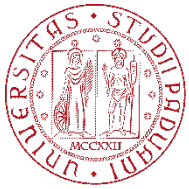

UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Psicologia dello Sviluppo e della Socializzazione

Corso di laurea triennale in Scienze Psicologiche dello Sviluppo, della Personalità e delle Relazioni Interpersonali

Tesi di laurea triennale

Il Coding e il Pensiero Computazionale

Coding and Computational Thinking

Relatore **Prof.ssa Barbara Arfé** ***Matricola: 1165604***

Laureando: Gabriele Tramontin

Anno Accademico 2022/2023

Sommario

Introduzione

Che cos'è il pensiero computazionale?

Il concetto di pensiero computazionale è un concetto introdotto da Janette Wing nel 2006.

La sua nascita ha generato numerose discussioni in campo scolastico tanto da essere presa in considerazione in molti stati europei e non quali: Inghilterra, Francia, Polonia, Finlandia, Australia e Stati Uniti.

Con il documento (Legge 107 del 2017) sulla buona scuola, il governo ha aperto il dibattito sull'introduzione dell'insegnamento del pensiero computazionale a scuola anche in Italia.

Il dibattito sul documento ha prodotto due proposte di curriculum (una per la scuola dell'obbligo e l'altra per le secondarie superiori) e due documenti di raccomandazioni (uno per l'introduzione dell'informatica nel curriculum e l'altro per l'introduzione del pensiero computazionale come competenza trasversale) (Governo Italiano, 2017).

La legge 1072 (2015) inserisce il pensiero computazionale tra gli obiettivi istruttivi della scuola (comma 7): qui la considerazione computazionale non è stata inequivocabilmente confrontata con altre materie scolastiche. Nel 2018 un *Comitato Nazionale delle Ricerche*, istituito dal Ministero dell'Università e della Ricerca (MIUR) redige un archivio denominato "*Norme nazionali e scenari inutilizzati*" per i moduli didattici della scuola materna e del ciclo primario di istruzione. Il documento distingue, tra gli insegnamenti da erogare nell'ambito scolastico, la presentazione al pensiero computazionale, introducendo formalmente un insegnamento moderno all'interno dei moduli didattici per l'istruzione obbligatoria.

Parallelamente, il MIUR ha supportato il "*Progetto il Futuro*" che dall'anno scolastico 2014-2015 sta sperimentando con esercizi di "*coding*" per la presentazione della considerazione computazionale a scuola. All'interno della circolare MIUR 08/10/20154 viene stimolata la presentazione della considerazione computazionale come segue: l'aspetto scientifico-culturale dell'innovazione dei dati, per altro chiamato "pensiero computazionale", fa la differenza per creare

attitudini coerenti e la capacità di vedere i problemi in un modo inventivo ed efficace, qualità che sono vitali per tutti i futuri cittadini.

ll modo migliore e più divertente per creare "pensiero computazionale" è attraverso il coding in un contesto di gioco. Come previsto anche all'interno del *National Computerized School Arrange*, un'adeguata attività di "pensiero computazionale", che va oltre l'istruzione avanzata introduttiva, è in realtà fondamentale per le generazioni moderne per poter confrontarsi con la società non come clienti distaccati e disinformati su progressi e amministrazioni, ma come soggetti attenti a tutte le prospettive in gioco e come attori che prendono effettivamente parte al loro miglioramento.

L'insegnamento del pensiero computazionale, ovvero l'acquisizione della considerazione logico-algoritmica e dei metodi mentali per la comprensione e l'approfondimento dei problemi, normali dell'informatica, punta a fornire al bambino un modo logico-analitico di considerare, che affina le sue attitudini di pensiero.

Capitolo 1

1. La Natura e i Concetti del Pensiero Computazionale

Il pensiero computazionale fornisce un sistema efficace per considerare l'informatica; tale sistema consente di riconoscere della computazione nel mondo che ci circonda e applicare apparati e strategie adeguati per ragionare su problemi o situazioni naturali e artificiali.

La computazione consente agli studenti di gestire i problemi, scomporli in parti fattibili e creare calcoli per risolverli. Il termine "pensiero computazionale" fu inizialmente utilizzato da Seymour Papert, ma la prof.ssa Jeannette Wing lo ha reso popolare sostenendo che il pensiero computazionale dovrebbe essere parte delle attitudini di tutti gli studenti universitari moderni (Wing, 2006).

L'autrice, inoltre, ha definito il pensiero computazionale come:

"*I processi mentali coinvolti nel formulare problemi e le loro soluzioni in modo che le soluzioni possano essere rappresentate in una forma che può essere efficacemente eseguita da un agente di elaborazione dell'informazione"* (Cuny, Snyder, Wing, 2010, citato in Wing, 2011, p.20).

La soluzione può essere eseguita da un essere umano o da una macchina, o, più in generale, da combinazioni di uomini e macchine (ibidem).

Così il pensiero computazionale si concentra sugli studenti che effettuano un processo di pensiero, non sulla produzione di artefatti o di prove; è lo sviluppo di capacità di pensiero che contribuiscono all'apprendimento e alla comprensione.

Anche l'articolo di Bucciarelli (2019) parla di *coding* (programmazione), e ne discute la relazione con le capacità di ragionamento. Diventa importante fare una precisazione: il *coding* è uno strumento del pensiero computazionale, ma non ne rappresenta l'essenza, esattamente come la conoscenza dei numeri risulta essere uno strumento del calcolo aritmetico, ma non coincide con il calcolo aritmetico o con il ragionamento matematico.

L’introduzione del pensiero computazionale dai primi anni di scuola ha l’obiettivo di insegnare a ragionare, a partire da problemi che prevedono soluzioni computazionali, cioè elaborazioni algoritmiche, fatte di istruzioni eseguibili da terzi (Nardelli e Ventre, 2015; Wing, 2006).

In questo commento, gli autori sviluppano le riflessioni di Bucciarelli circa l’insegnare a ragionare a scuola, parlando del ruolo che il pensiero computazionale può assumere in questo contesto.

Esso è infatti considerato un processo cognitivo che include un pensiero coerente attraverso il quale i problemi vengono risolti e gli artefatti, le strategie e le strutture sono capiti in modo migliore.

Il pensiero computazionale incorpora la capacità di pensare in termini di:

1. *Algoritmi*;
2. *Scomposizione*;
3. *Generalizzazioni*, individuando e facendo uso di schemi ricorrenti;
4. *Astrazioni*, scegliendo rappresentazioni appropriate;
5. *Valutazione*.

2. Ragionamento Logico

Il ragionamento logico permette agli alunni di dare un senso alle cose. Esso consente di fare e verificare previsioni e trarre conclusioni. Viene utilizzato ampiamente quando si prova, si depura (si esegue il debug), e si correggono gli algoritmi. Il ragionamento logico è l'applicazione dei concetti di pensiero computazionale per risolvere problemi.

Alcuni esempi potrebbero essere: scegliere i materiali per i diversi elementi del progetto di un modello di camion. Usando generalizzazioni quando si riconosce che le proprietà di uno strumento utilizzato in una situazione lo rendono adatto ad essere utilizzato in un contesto completamente diverso.

Un esempio di scomposizione è essere in grado di dividere il nuovo progetto in parti che richiedono strumenti diversi. Lo studente utilizzerà il ragionamento logico per la progettazione.
Si usa il ragionamento logico, ad esempio, quando si studia la forza di gravità impiegando un peso attaccato con un filo al coperchio di un barattolo di vetro. Prima di inclinare il vaso, gli studenti possono fare previsioni sul comportamento del peso sospeso, e possono quindi valutare i risultati delle loro prove. Questi ragionamenti e ipotesi permetteranno agli studenti di generalizzare il comportamento e riproporlo poi in altre situazioni, per esempio studiando il meccanismo di una gru. In questo caso si usa il ragionamento logico per capire una proprietà dei gravi.

Il ragionamento logico è fondamentale nel rimuovere gli errori (debug) di un programma. Durante questo processo, si potrà usare l'astrazione, la valutazione e il pensiero algoritmico. La correzione di errori nei programmi richiede ragionamento logico. Il pensiero computazionale riguarda una forma specifica di ragionamento: il ragionamento attorno ai problemi. Può coinvolgere processi di ragionamento deduttivo, induttivo, abduttivo e analogico.

Le attività di pensiero computazionale possono essere sia digitali (usando il computer come esecutore terzo) che analogiche (usando, per esempio, il linguaggio naturale o il disegno per definire una sequenza di istruzioni necessarie per permettere a una terza persona di scoprire una regola o identificare una categoria di oggetti). In entrambi i casi, le attività coinvolte possono essere di *problem solving*, più o meno strutturate, che solitamente assumono la veste di giochi logici (Kalelioglu, 2015), (questo è il caso del ragionamento deduttivo e abduttivo) oppure attività di progettazione, volte a sviluppare idee o prodotti (ad esempio, un video-gioco, una storia), tipicamente meno strutturate (Bers, 2018; Strawhacker e Bers, 2019). Per entrambe queste attività il bambino impiega una gamma di processi di pensiero che includono diverse parti del ragionamento: la decomposizione, il pensiero algoritmico, la verifica logica (deduttiva), e l'astrazione (Yaşar, 2018).

3. Astrazione

L'astrazione è un meccanismo induttivo, che consiste nel districare e categorizzare le questioni, riconoscere gli elementi sottili da considerare e quelli da trascurare, parlare alle informazioni significative della questione in un modello mentale o concettuale (Yaşar, 2018). L'astrazione rende più semplice pensare a problemi o strutture; è il metodo per rendere più semplice un artefatto complesso rimuovendo sempre di più gli strati superflui nella sua descrizione.

La capacità di astrazione sta nello scegliere il dettaglio da ignorare in modo che la questione diventi meno impegnativa, senza perdere nulla di vitale. Una prospettiva chiave di questa abilità sta nella scelta di una grande rappresentazione della struttura. Le rappresentazioni distintive rendono meno impegnativo il funzionamento su cose diverse. Ad esempio, un programma per computer che gioca a scacchi è un'astrazione. Potrebbe essere un insieme limitato ed esatto di regole da seguire ogni volta che è il turno del computer di muoversi. È distante dalle forme di pensiero analogiche, appassionate, frazionarie, tipiche di un giocatore di scacchi umano. È un'astrazione poiché i punti di interesse superflui di tali forme vengono eliminati.

Si può osservare:

- Riduzione della complessità eliminando inutili dettagli.
- Scelta di un modo di rappresentare gli artefatti in modo che possano essere manipolati in maniera utile.
- Occultamento di tutte le complessità dell'artefatto (nascondere la complessità funzionale).
- Occultamento della complessità nei dati, ad esempio utilizzando strutture di dati.
- Identificazione di relazioni tra le astrazioni.
- Filtraggio delle informazioni nello sviluppo di soluzioni.

4. Valutazione

La valutazione è il metodo per garantire un ottimo accordo, sia esso un calcolo, un sistema o una preparazione: cioè che sia adatto allo scopo. Dovrebbero essere valutate diverse proprietà delle soluzioni.

Sono rettificate? Sono abbastanza veloci? Utilizzano le risorse con parsimonia? Sono facili da utilizzare per le persone? Danno soluzioni accettabili ed efficienti?

Dovrebbero essere fatti dei compromessi: raramente c'è un unico modo perfetto per tutte le circostanze. Nella valutazione, come utilizzata nella considerazione computazionale, c'è una considerazione specifica per i dettagli.

Per soddisfare le esigenze di più persone possibili, vengono continuamente sviluppate moderne interfacce per computer. Ad esempio, nel caso in cui si desideri un dispositivo medico che fornisca i farmaci a un paziente senza rischi, deve essere programmabile senza errori, in modo semplice, veloce e sicuro. La soluzione deve garantire che gli operatori sanitari possano modificare la dose senza sforzo errori, e non deve essere difficile o scomodo da utilizzare per i pazienti e gli infermieri. All'interno dell'interfaccia proposta ci sarà il giusto livello di compromessi tra velocità di inserimento dei numeri (efficienza) e controlli per evitare errori (adeguatezza e convenienza).

Il design dovrebbe essere valutato sulla base di raccomandazioni particolari di medici, legislatori e specialisti del piano di dispositivi medici (criteri) e regole comuni di buon design (euristica). I criteri, l'euristica e le esigenze del cliente consentono valutazioni precise e approfondite.

Nella valutazione l'attenzione va posta sulle seguenti azioni:

- Valutare che l'artefatto sia adatto allo scopo.
- Determinare che l'artefatto esegua la giusta funzione (correttezza funzionale).
- Progettare ed eseguire dei test programmati ed interpretarne i risultati (test).
- Valutare che le prestazioni dell'artefatto siano sufficientemente buone (utilità: efficacia ed efficienza).
- Confrontare le prestazioni degli artefatti che eseguono la stessa funzione.
- Eseguire dei compromessi tra esigenze contrastanti.
- Valutare se l'artefatto sia facile da usare per le persone (usabilità).
- Determinare se l'artefatto offre un'esperienza adeguatamente positiva quando utilizzato (esperienza dell'utilizzatore).
- Ripercorrere passo dopo passo i processi o gli algoritmi/la programmazione, per elaborare quello che eseguono (prova/traccia).
- Utilizzare una precisa argomentazione per giustificare che un algoritmo funzioni (prova).
- Utilizzare una precisa argomentazione per verificare l'usabilità o le prestazioni di un artefatto (valutazione analitica).
- Utilizzare metodi che includono l'osservazione dell'artefatto impiegato per valutarne l'usabilità (valutazione empirica).
- Valutare che il prodotto soddisfi i criteri generali di prestazione (euristiche).

5. Pensiero Algoritmico

Il pensiero algoritmico comprende nel caratterizzare un raggruppamento di attività essenziali o informative (o addirittura reiterate), ordini per un terzo agente, in grado di guidare alla sistemazione del problema o allo scopo. La creazione continua di algoritmi è possibile con, come afferma Bucciarelli (2019), il pensiero abduttivo (cioè in senso inverso, dagli impatti alle cause). Per quanto riguarda la ripetizione (che nella sua massima espressione corrisponde all'uso di informazioni ricorsive), Bucciarelli (2019) segnala quanto sia difficile e importante comprendere modalità più "sintetiche" di formulare questo costrutto in modo che possa riconoscersi nella particolare forma della "ricorsività", ovvero, quello di una funzione che si richiama più di una volta. Il pensiero Algoritmico è considerato un modo per raggiungere una conclusione attraverso una chiara definizione dei passaggi. Alcuni problemi sono del tipo una tantum; una volta risolto il primo, applicata la soluzione, possiamo passare al prossimo e così via. Il pensiero algoritmico entra in gioco quando questioni comparabili devono essere analizzate più e più volte. Una volta formulato l'algoritmo, non è fondamentale rielaborarlo da zero per ogni problema.

Il pensiero algoritmico è la capacità di pensare in termini di disposizioni e regole per affrontare problemi o comprendere le circostanze; parla della capacità di pensare in termini di raggruppamenti e regole, come un modo per approfondire le questioni. Potrebbe essere un'attitudine essenziale che gli studenti creano quando imparano a comporre i propri programmi per computer.

Nella sua applicazione si può osservare:

- *Formulazione di comandi* per ottenere un effetto desiderato.
- *Formulazione di istruzioni* da seguire in un dato ordine (sequenza).
- *Formulazione di comandi* che utilizzano operazioni aritmetiche e logiche.
- *Scrittura delle sequenze di comandi* che memorizzano e manipolano i dati (variabili e assegnazione).
- *Scrittura di comandi* che scelgono tra diverse istruzioni che li compongono (*selezione*).
- *Scrittura di comandi* che ripetono gruppi di istruzioni che li costituiscono (loop / ripetizione).

- *Raggruppamento e denominazione di un insieme* di comandi che eseguono un compito definito per creare una nuova istruzione (subroutine, procedure, funzioni, metodi).
- *Scrittura di comandi*, che prevedono sottoprogrammi che utilizzano repliche di sé stessi (algoritmo ricorsivo).
- *Scrittura di una serie di comandi* che possano essere seguiti contemporaneamente da diversi agenti (computer/persone, pensiero e processi paralleli, concorrenza).
- *Scrittura di un insieme di regole dichiarative* (coding in Prologo in un linguaggio di creazione di query su database).

Questo implica anche:

- L'utilizzo di un'appropriata dicitura per scrivere il codice che rappresenti uno dei comandi suddetti.
- La creazione di algoritmi per verificare un'ipotesi.
- La creazione di algoritmi che forniscono soluzioni basate sull'esperienza (euristiche).
- Creazione di descrizioni algoritmiche dei processi del mondo reale, in modo da averne una migliore comprensione. (modellizzazione computazionale).
- Progettazione di soluzioni algoritmiche che tengano conto delle capacità, limiti e esigenze delle persone che ne usufruiscono.

6. Scomposizione

La scomposizione è un meccanismo di natura analitica e deduttiva, che consiste nell'analizzare e scomporre una questione complessa in un insieme di sottoproblemi, o sottounità, che, se confrontati in raggruppamento, portano a una soluzione.

La scomposizione può essere un modo per considerare gli artefatti in termini di componenti. Le parti della persona possono essere comprese, risolte, sviluppate e valutate in modo indipendente. Questo approccio semplifica la risoluzione di problemi complessi permettendo altresì di capire situazioni nuove e facilita la creazione di progetti di grandi sistemi.

Si consideri lo sviluppo di un videogame: persone diverse possono progettare e creare i diversi livelli in modo indipendente, a condizione che gli aspetti chiave siano concordati in anticipo. A sua volta un livello di un videogioco potrebbe essere scomposto in più parti, come ad esempio il movimento realistico di un personaggio, lo scorrimento dello sfondo e l'impostazione delle regole su come i personaggi interagiscono.

Si può osservare:

1. La divisione degli artefatti nelle parti costituenti per renderli più facili da elaborare.
2. La scomposizione di un problema in versioni più semplici dello stesso;
   versioni che possono essere risolte nel medesimo modo (metodo *divide et impera*: questo approccio permette di affrontare in modo semplice problemi anche molto difficili).

7. Generalizzazione

La generalizzazione è la capacità di gestire somiglianze o progetti simili e ripetuti, e in questo modo scambiare procedure di risoluzione dei problemi con una raccolta di nuove soluzioni, attraverso una preparazione del pensiero analogico (Roman-Gonzalez et al., 2017). Potrebbe essere un modo per districare rapidamente problemi all'avanguardia sulla base dei problemi ora affrontati avendo anche imparato dal passato. È imperativo porre domande come: "È paragonabile a un problema che ho risolto fino ad ora?", "In che modo è diverso?" Altrettanto importante è imparare a riconoscere i

progetti ripetuti sia all'interno delle informazioni che all'interno delle forme e/o metodologie utilizzate. I calcoli che esplorano alcune questioni particolari possono essere adattati per illuminare un'intera lezione di questioni comparative, quindi quando viene sperimentato un problema di quella lezione, la disposizione comune può essere collegata.

Ad esempio, un alunno sta utilizzando la "tartaruga" per disegnare una disposizione di figure geometriche. Lo studente compone un programma per disegnare un quadrato e un triangolo. Lui a quel punto sceglie di disegnare un ottagono e un decagono. Lavorando con il quadrato e il triangolo, si rende conto che c'è una relazione tra il numero di lati della figura e i punti inclusi nel disegnarla con la tartaruga. A quel punto può comporre un calcolo che comunica questa proporzione e utilizzarlo per disegnare qualsiasi poligono regolare.

Delle caratteristiche della generalizzazione possiamo evidenziare:

- Identificazione di schemi e caratteristiche comuni negli artefatti.
- Adeguamento di soluzioni, o parti di esse, per applicarle a tutta una classe di problemi analoghi.
- Trasferimento di idee e soluzioni da una settore del problema ad un altro.

8. Tecniche Associate al Pensiero Computazionale

Nella guida *Computational thinking - A guide for teachers* (2015) si evidenzia quanto segue: “Esiste un certo numero di tecniche utilizzate per dimostrare e valutare il pensiero computazionale. Pensate a questo come al “fare computazionale”. Questi sono l'equivalente “informatico” dei "metodi scientifici”, sono gli strumenti con cui il pensiero computazionale viene reso operativo in aula, sul posto di lavoro, a casa”.

9. Riflettere

Nella stessa guida a proposito di riflessione si legge: “Riflettere è la capacità di esprimere giudizi (di valutare) che sono leali e onesti in situazioni complesse che non sono prive di valore.

Nell’informatica questa valutazione si basa sui criteri utilizzati per specificare il prodotto, sull’euristica (o regole empiriche) e sulle esigenze dell'utente che indirizzano i giudizi”.

10. Programmare (coding)

Tradurre il progetto in forma di codice e valutarlo per garantirne il corretto funzionamento in tutte le condizioni previste è un concetto essenziale nello sviluppo di qualsiasi sistema di computer. Rimuovere gli errori (debug) è l'applicazione sistematica di analisi e valutazione utilizzando abilità quali test, tracciamento e il pensiero logico per prevedere e verificare i risultati.

11. Progettare

Progettare significa definire la struttura, l'aspetto e la funzionalità degli artefatti; comporta la creazione di rappresentazioni del progetto, ivi incluse rappresentazioni leggibili da altre persone come diagrammi di flusso, storyboard, pseudo-codice, schemi di sistema, e altro.

Di frequente questo implica ulteriori attività di scomposizione, astrazione e progettazione di algoritmi.

12. Analizzare

Analizzare significa dividere in più parti (scomposizione), rimuovere complessità inutili (astrazione), individuare processi (algoritmi) e ricercare punti in comune o schemi ricorrenti (generalizzazione) nel programma. Si tratta di utilizzare il pensiero logico sia per comprendere meglio le cose sia per valutarle come idonee allo scopo.

13. Applicare

Ancora la guida per insegnanti (2015) suggerisce che “Applicare è l'adozione di soluzioni preesistenti per soddisfare le esigenze di un altro contesto. È una generalizzazione - l'identificazione di schemi ricorrenti, somiglianze e connessioni – che utilizza le caratteristiche della struttura o di una funzione degli artefatti. Per esempio, lo sviluppo di un sottoprogramma o algoritmo in un contesto che può essere riutilizzato in un contesto diverso”.

Capitolo 2

1. Utilità dell'apprendimento del coding

Nel 2018 Shahira Popat e Louise Starkey hanno pubblicato una rassegna sistematica di numerosi studi riguardanti l'efficacia e gli effetti dell'apprendimento del coding in campo scolastico.

La rassegna fornisce una sintesi della ricerca che esplora l'influenza del coding sui risultati educativi che sono stati classificati come pensiero di ordine superiore, pensiero di ordine inferiore e abilità personali che sono state incorporate in un modello generale (Fig 1). Il modello ha fornito esempi specifici di ciascuno di questi ampi risultati educativi in cinque temi: pensiero critico, risoluzione dei problemi, abilità sociali, autogestione e abilità o conoscenze accademiche.

Il risultato della rassegna mostra numerosi effetti positivi poiché si è visto quanto gli effetti del coding possono influenzare lo sviluppo di altre abilità quali ad esempio il problem solving in ambito matematico critico, abilità sociali, self-management e abilità accademiche.

Dieci articoli sono stati presi in analisi in questa rassegna nei quali il campione di studenti presi in considerazione è di un ampio range di età che va dai 5 e i 17 anni di numerose nazionalità quali: americani, neozelandesi, spagnoli, greci, irlandesi e turchi in modo da poter avere un quadro più completo e ampio possibile.

Questa rassegna si è concentrata sull'influenza del coding sui risultati educativi per i bambini in età scolare. La sintesi dei risultati ha portato a un modello generale che è stato sviluppato per rappresentare una visione complessiva dei risultati chiave identificati. Lo scopo di questa rassegna era trovare prove dei risultati educativi, oltre al coding, che sono stati influenzati dall'apprendimento del codice. Questi sono rappresentati nel terzo livello del modello generale e allineati con tre tipi di abilità: ordine superiore, ordine inferiore e abilità personali.

Imparare a programmare è al centro del modello generale (Fig. 1) poiché questo è un risultato chiave quando si impara attraverso la programmazione. Saez-López et al. (2016) hanno applicato un test quantitativo che ha rilevato che ci sono miglioramenti significativi nella capacità degli studenti

di comprendere i concetti di programmazione e le pratiche computazionali dopo aver utilizzato Scratch in classe. Tuttavia, gli studi forniscono prove che altri risultati educativi possono essere influenzati dalla progettazione delle attività di coding e dall'ambiente della classe. Questi possono essere alimentati attraverso il curriculum e la progettazione pedagogica all'interno dell'ambiente scolastico e attraverso la facilitazione degli insegnanti.

I temi dei risultati educativi individuati sono raggruppati in tre grandi titoli; capacità di pensiero di ordine inferiore come comprensione e comprensione, capacità di pensiero di ordine superiore di pensiero critico e *problem solving* attraverso concetti matematici e abilità personali che includono abilità sociali e autogestione.

Le abilità di pensiero di ordine inferiore e superiore sono state utilizzate per le abilità cognitive identificate che sono state analizzate rispetto alla tassonomia rivista di Bloom ("*Revised Bloom's Taxonomy*" David Krathwohl 2001).

Le abilità personali sono state viste come buoni indicatori per comunicare, collaborare e autogestirsi poiché queste abilità sembravano essere utilizzate naturalmente attraverso le esperienze e la pratica degli studenti. Le abilità sotto ogni intestazione sono interconnesse. Ad esempio, per risolvere i problemi gli studenti devono pensare in modo critico e possono farlo in modo collaborativo.

Nel complesso, la letteratura suggerisce che attraverso l'apprendimento del codice, gli studenti utilizzano una gamma di abilità oltre la codifica. Tuttavia, queste abilità vengono ulteriormente sviluppate attraverso l'insegnamento e l'inclusione deliberati nel curriculum e nella progettazione pedagogica.

Nelle sezioni seguenti approfondiremo i punti del modello qui sotto riportato singolarmente.

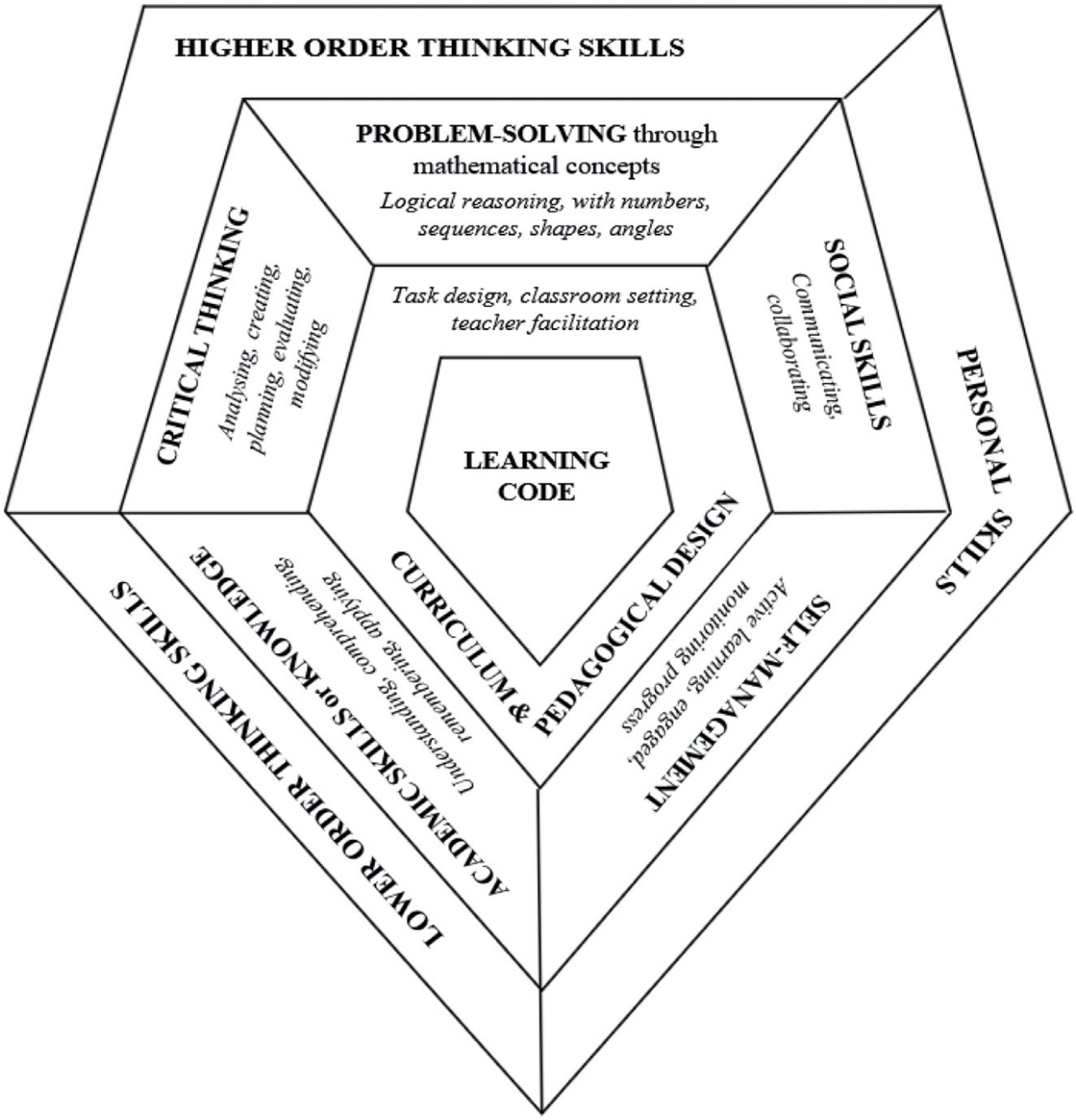

Figura 1. Tratta da "Learning to code or coding to learn? A systematic review" di Shahira Popat e Louise Starkey (2019)

2. Problem solving

Questo tema era evidente in tutti e dieci gli studi. Nove studi hanno testato attivamente la capacità di risoluzione dei problemi che doveva essere applicata attraverso i concetti matematici presentati all'interno del set di attività di coding. Il decimo studio ha riconosciuto che nel processo di apprendimento della programmazione, gli studenti hanno appreso abilità matematiche che hanno migliorato la risoluzione dei problemi.

In questa rassegna, il *problem solving* è stato definito come l'atto di risoluzione di un problema tramite concetti matematici. Le abilità di coding o di risoluzione dei problemi computazionali erano quelle in cui uno studente poteva progettare ed eseguire una procedura algoritmica (cioè dire al computer cosa fare per risolvere il problema). Tuttavia, come risultato educativo al di fuori della programmazione, la risoluzione dei problemi attraverso concetti matematici era un'abilità più generica che poteva essere analizzata in diverse situazioni. Ad esempio, il ragionamento logico, che includeva la capacità di interpretare schemi, sequenze numeriche o la relazione tra forme per risolvere problemi.

Gli studi rivelano che nell'apprendimento del problem solving il coding, nonostante sia efficace, non presenta molti vantaggi rispetto a uno studio diretto della materia. Si pensa che a lungo termine ci possano essere più miglioramenti anche se attualmente non abbiamo certezze. (Falloon, 2016; Hayes and Stewart's, 2016; Kalelioğlu, 2015; Kalelioğlu and Gülbahar, 2014; Psycharis and Kallia, 2017).

Questa rassegna non ha studiato il pensiero computazionale, ma ha scoperto che la risoluzione dei problemi matematici era un risultato educativo chiave attraverso l'apprendimento del codice che potrebbe essere dovuto a una correlazione stretta con la risoluzione dei problemi computazionali.

La letteratura esaminata suggerisce che la risoluzione di problemi matematici è un risultato educativo quando si impara a programmare. Tuttavia, mentre le abilità matematiche, come l'uso della geometria per risolvere i problemi, possono essere migliorate imparando il coding, altri metodi di apprendimento producono gli stessi miglioramenti o rimangono ugualmente efficaci (Hayes & Stewart, 2016; Kalelioğlu & Gülbahar, 2014). Pertanto, se l'obiettivo accademico è che gli studenti imparino a risolvere problemi matematici, insegnare queste abilità direttamente è più efficace che impararle attraverso la programmazione.

È stato scoperto che gli studenti che hanno idee sbagliate non migliorano la loro capacità di risoluzione dei problemi rispetto ai coetanei senza idee sbagliate (Mayer & Fay, 1987). Ciò

suggerisce che è necessario che gli insegnanti intervengano quando si presentano “idee sbagliate” (*misconception*) e che considerino le opportunità di apprendimento collaborativo durante lo sviluppo della risoluzione di problemi matematici.

3. Abilità Sociali e Collaborazioni

Gli studi hanno evidenziato alcuni eventi inattesi; uno di questi è lo sviluppo delle abilità sociali e della collaborazione, poiché nello svolgersi delle attività di *coding* gli allievi tendevano a condividere le conoscenze e ad aiutarsi fra loro in modo da poter avere una migliore resa nel compito affidatogli (Falloon, 2016; Fessakis et al., 2013; Kalelioğlu, 2015).

Falloon (2016) suggerisce che per facilitare la collaborazione, gli studenti dovrebbero essere organizzati in coppie per condividere la loro conoscenza incoraggiare l'accettazione delle opinioni di altre persone.

4. Self-management e apprendimento attivo

Un tema individuato da tre dei dieci studi era il self-management e l'apprendimento attivo.

Solo uno di questi tre studi ha misurato l'apprendimento attivo come risultato, gli altri due hanno trovato prove attraverso osservazioni e dati qualitativi.

Si definisce “apprendimento attivo” il processo per cui gli studenti possono impegnarsi nelle attività e avere il controllo sul proprio apprendimento. Kalelioğlu (2015) afferma che gli studenti hanno cercato di completare più fasi e più livelli al proprio ritmo utilizzando *code.org* rispetto alle normali attività in classe. L'insegnante ha osservato che gli studenti erano disposti a continuare attraverso le fasi delle attività e coloro che in precedenza non erano interessati alla lezione hanno partecipato e hanno mostrato progressi "sorprendenti" utilizzando code.org. Uno studente ha dichiarato: "Voglio completare tutte le fasi del sito code.org" (Kalelioğlu, 2015, p. 208).

Agli studenti è stato insegnato il contenuto attraverso video e come rivedere i loro progressi.

Falloon (2016) ha osservato che i giovani studenti del suo studio trascorrevano molto tempo discutendo e pianificando ciò che dovevano fare. Attraverso le tecniche dell'osservazione e dell'analisi dei questionari Sáez-López, Román-González e Vázquez-Cano (2016) hanno concluso che i bambini hanno riconosciuto il proprio approccio attivo all'apprendimento.

Queste ricerche suggeriscono, dunque, che l'insegnamento del coding potrebbe consentire agli studenti di diventare più attivi nel loro apprendimento.

Tuttavia, negli studi è anche riconosciuto che sono necessari aspetti pedagogici come l'insegnamento specifico delle materie e la progettazione dei compiti per sviluppare la comprensione degli studenti su come svolgere con successo le attività di apprendimento al proprio ritmo per consentire loro di essere discenti attivi e autogestiti.

5. Pensiero Critico

Il pensiero critico è stato identificato in cinque degli studi dei dieci presi in considerazione da Shahira Popat e Louise Starkey (2019).

C'è un'importante sovrapposizione tra lo sviluppo del pensiero critico e lo sviluppo della capacità di risoluzione dei problemi, poiché per risolvere i problemi i bambini devono pensare in modo critico e applicare il ragionamento. In questa rassegna, il tema del *problem solving* è definito come l'atto di risolvere un problema attraverso concetti matematici, mentre il pensiero critico è definito come la capacità di svolgere alcune o tutte le seguenti azioni: analizzare un'attività, creare e attuare un piano, valutare i risultati, identificare e applicare le azioni necessarie per migliorare o correggere le prestazioni e valutare se il risultato desiderato è stato raggiunto.

Il pensiero critico è stato identificato come una caratteristica chiave in diversi studi.

Falloon (2016) ha constatato che gli studenti che utilizzano *Scratch* prevedevano un possibile risultato dell'esecuzione del proprio codice prima di eseguirlo. Kalelioğlu (2015) ha testato

quantitativamente aspetti del pensiero critico: è stato riscontrato un aumento lieve ma significativo nella differenza media tra i risultati pre- e post-test per la valutazione.

Tale risultato è stato interpretato come la capacità di fare un passo indietro e pensare in modo critico a come vengono risolti i problemi (Kalelioğlu, 2015), implicando così che gli studenti potrebbero correggere la propria prestazione. Il miglioramento delle capacità di ragionamento è stato identificato come risultato dell'apprendimento della programmazione informatica (Hayes & Stewart, 2016; Psycharis & Kallia, 2017). L'evidenza suggerisce che gli aspetti del pensiero critico possono essere sviluppati attraverso l'attività di coding e che la progettazione dei compiti dovrebbe incoraggiare gli studenti a testare, valutare e modificare il codice per sviluppare le proprie capacità di pensiero critico. Essere in grado di testare, valutare e modificare il codice è stato valutato, rispetto alla tassonomia rivista di Bloom (2001), come capacità di pensiero di ordine superiore.

6. Capacità o conoscenze accademiche (escluse competenze matematiche o informatiche)

Le abilità o conoscenze accademiche sono state identificate in due dei dieci studi esaminati. Esse riguardano l'aumento del livello di competenza nelle aree disciplinari legate alla scuola.

Le aree tematiche non incluse in questo tema sono la matematica e la coding/informatica poiché queste sono evidenti in tutti gli studi e sono state discusse separatamente.

Nello studio di Hayes e Stewart (2016), studenti di età compresa tra 10 e 11 anni che hanno ricevuto istruzioni utilizzando la programmazione *Scratch* sono stati testati su abilità accademiche come lettura e ortografia.

I risultati pre-test e post-test hanno mostrato un miglioramento all'interno di queste abilità accademiche; tuttavia, non è chiaro se i miglioramenti siano dovuti alla specifica metodologia utilizzata.

Saez-López et al. (2016) hanno implementato la programmazione nel programma scolastico di storia dell'arte e hanno scoperto che gli studenti potevano capire e comprendere una serie di concetti di arte e storia utilizzando un *software* di programmazione

Pertanto, c'è qualche rimando che il coding possa influenzare le abilità accademiche, tuttavia questo era all'interno della progettazione pedagogica dell'attività e quando è stato poi analizzato attraverso la tassonomia revisionata di Bloom (2001) sembrava riguardare il pensiero di ordine inferiore, come ad esempio, il richiamo di informazioni.

Il pensiero critico è stato raggruppato in abilità di pensiero di ordine superiore nel modello generale basato sulla tassonomia rivista di Bloom, e includeva abilità come creare e valutare.

Sebbene pensare in modo creativo possa essere un'abilità chiave da sviluppare attraverso il coding (Resnick e Siegel, 2015), un tale sviluppo non è stato identificato dagli studi esaminati, poiché molti studi che propongono attività legate all'apprendimento e uso del coding non supportano l'uso di espressioni creative (Resnick e Siegel, 2015). Risulta evidente in questa rassegna, in cui la maggior parte dei pensieri sugli studi indagati per programmare implicano semplici abilità di *problem solving* che richiedono attitudini di giudizio ma limitano il pensiero creativo.

L'introduzione degli studenti agli standard dell'informatica attraverso la programmazione in altre aree disciplinari come la musica può consentire una migliore verbalizzazione di impieghi più ampi di capacità di risoluzione dei problemi e di pensiero creativo (Alano et al., 2017).

7. Curriculum e progettazione pedagogica

Il curriculum e la progettazione pedagogica sono state le caratteristiche più importanti in tutta la letteratura, fornendo il ponte tra l'apprendimento del codice e lo sviluppo di risultati educativi chiave. Il curriculum e la progettazione pedagogica includono ciò che viene insegnato nella programmazione e come viene insegnato. Questo tema riguarda un aspetto che influenza i risultati educativi piuttosto che un risultato in sé. I primi cinque studi presentavano prove sufficienti per

suggerire che il curriculum e la progettazione pedagogica fossero importanti nell'influenzare i risultati di apprendimento identificati. Ciò includeva l'insegnamento della programmazione in aree disciplinari diverse dall'informatica, dalla progettazione di attività o dalla collaborazione in coppia o in gruppo. Ad esempio, l'integrazione del *coding* in aree disciplinari come matematica e storia dell'arte ha mostrato che gli studenti non solo hanno ottenuto miglioramenti significativi nella programmazione, ma anche nei contenuti relativi ai corsi (Psycharis & Kallia, 2017; Sáez-López et al., 2016).

Certe attività di *coding* usando *Scratch* danno l'opportunità di risolvere problemi di matematica legati al *problem solving* e di pensiero critico. coding.org ha permesso di evidenziare quanto, con video tutorial e il suo sistema di self monitoring, si possa sviluppare l'auto-apprendimento. Gli studenti sono stimolati a questo poiché gli viene concesso un enorme controllo sul quadro generale di ciò che stanno svolgendo (Kalelioğlu, 2015). È altresì rilevante, tramite la presenza di docenti che si rendono disponibili all'aiuto e all'assistenza degli studenti, lo sviluppo di abilità collaborative e di comunicazione. L'attività di apprendimento studiata potrebbe essere stata progettata per insegnare ai bambini non solo la programmazione, ma anche la storia dell'arte (Sáez-López et al., 2016) o la risoluzione di problemi matematici (Bernardo & Morris, 1994).

Capitolo 3

Altre ricerche

Come possiamo leggere in *Trends in Cognitive Sciences*, (Rule, Tenenbaum, Piantadosi, 2020), lo sviluppo cognitivo umano è qualitativamente unico.

Sebbene gli esseri umani nascano insolitamente indifesi, accumulano un repertorio cognitivo senza precedenti, tra cui: teorie intuitive per domini come la fisica e la biologia, teorie formali in matematica e scienze, comprensione e produzione del linguaggio e complesse abilità percettive e motorie.

I bambini imparano anche ad apprendere, producendo conoscenze di ordine superiore che arricchiscono i concetti esistenti e migliorano l'apprendimento futuro. Le prestazioni simili a quelle umane in uno qualsiasi di questi domini sembrano sostanzialmente al di là degli attuali sforzi nell'intelligenza artificiale. Tuttavia, i bambini umani acquisiscono essenzialmente queste capacità simultaneamente e universalmente. Potrebbero persino scoprire nuove idee che alterano radicalmente la comprensione del mondo da parte dell'umanità.

I campi fondamentali delle scienze cognitive, tra cui filosofia, psicologia, neuroscienze e informatica stanno affrontano una sfida radicale nel tentare di spiegare pienamente la ricchezza dello sviluppo umano.

Per aiutare ad affrontare questa sfida, introduciamo l'idea del "*bambino come hacker*", come un'ipotesi sulle rappresentazioni, i processi e gli obiettivi di un apprendimento tipicamente umano. Come la visione del "*bambino come scienziato*", il bambino come hacker è sia una metafora fertile che una fonte di ipotesi concrete sullo sviluppo cognitivo.

Questa idea suggerisce anche una tabella di marcia per quello che potrebbe essere un resoconto formale unificante dei principali fenomeni in via di sviluppo. Una parte fondamentale dell'idea del bambino-hacker è il parallelismo tra l'apprendimento e la programmazione, che sostiene che i programmi simbolici (cioè il codice) forniscono la migliore rappresentazione formale della

conoscenza che abbiamo. L'apprendimento diventa quindi un processo di creazione di nuovi programmi mentali. Esaminiamo il supporto per l'apprendimento come programmazione e sosteniamo che, sebbene su un terreno sempre più solido come teoria a livello computazionale, rimane sottospecificato. Estendiamo l'idea dell'apprendimento come programmazione traendo ispirazione dall'hacking, un approccio alla programmazione guidato internamente che enfatizza i diversi obiettivi e i mezzi che gli esseri umani usano per migliorare il codice.

La nostra affermazione principale è che le rappresentazioni, le motivazioni, i valori e le tecniche specifiche dell'hacking formano un ricco insieme di ipotesi sull'apprendimento, in gran parte non verificate.

L'hacking rappresenta una raccolta di valori epistemici e pratiche adattate all'organizzazione della conoscenza utilizzando i programmi e vi sono prove crescenti che i programmi sono un buon modello di rappresentazioni mentali.

Il bambino come hacker combina queste idee in una sorta di percorso verso un resoconto computazionale dell'apprendimento e dello sviluppo cognitivo; fa affermazioni verificabili su una classe generale di bias induttivi che gli esseri umani dovrebbero avere, vale a dire quelli relativi alla sintesi, all'esecuzione e all'analisi delle informazioni come programmi. L'hacking identifica anche concretamente le rappresentazioni, gli obiettivi e i processi che supportano l'apprendimento con quelli degli hacker adulti. Infine, fa un'affermazione unificante su come questi tre potrebbero essere implementati rispettivamente come codice, procedure per la valutazione del codice e procedure per la rassegna del codice.

Nell'introdurre l'idea del bambino come hacker, l'obiettivo principale della rassegna è stato quello di offrire un percorso per rispondere alle sfide centrali dell'apprendimento umano e dello sviluppo cognitivo che riformula le domande classiche e ci aiuta a porre nuove domande. I recenti lavori nelle scienze cognitive sul pensiero costruttivo, la neuroscienza della programmazione (Tenenbaum, J.B. et al., 2011) e la modellazione dello sviluppo di domini fondamentali come la fisica intuitiva

utilizzando motori di gioco rappresentano promettenti passi complementari (Ullman, T.D. et al. (2017)). I recenti sviluppi nell'induzione di programmi e nelle tecniche di sintesi dei programmi dall'informatica stanno anche iniziando a rendere operativi aspetti di specifiche tecniche di *hacking,* incluso il lavoro sul concatenamento all'indietro di obiettivi e sotto-obiettivi, sintesi guidata neurale, *refactoring* interattivo, programmazione incrementale e apprendimento di modelli probabilistici generativi. Questi sforzi hanno il potenziale per evolvere l'idea del bambino come hacker, rendendo questa metafora un resoconto computazionale funzionante e verificabile dello sviluppo cognitivo. Ma sono solo i primi passi: attendiamo con impazienza tutto il lavoro che resta da fare per capire come è possibile che i bambini modifichino le proprie rappresentazioni mentali per costruire strumenti di pensiero ancora ineguagliabili.

Gli insegnanti generalmente non hanno familiarità con la *Computational Thinking* (CT) e hanno difficoltà a trovare collegamenti tra la CT e i loro programmi di studio attuali. La progettazione e lo sviluppo di una valutazione CT affidabile e valida è la chiave per un'istruzione di successo della CT incorporata in più discipline (Grover et Pea, 2013). La mancanza di una definizione standard per rendere operativa la CT, tuttavia, porta di conseguenza a ricerche in cui le misurazioni variano notevolmente tra gli studi, il che rende i risultati meno convincenti e certamente difficili da confrontare. Un altro problema che deve essere risolto riguarda la difficoltà nel giudicare se gli interventi di CT in classe ottengono effettivamente i risultati desiderati (Settle et al., 2012).

Ancora una volta, i ricercatori hanno sottolineato la difficoltà di valutare la CT nelle aule e hanno chiesto una valutazione in tempo reale per fornire dati sulla progressione didattica per studente per supportare l'istruzione data dagli insegnanti (Bers et al., 2014; Wolz et al. 2011). Il quadro che abbiamo proposto può essere una linea guida per sviluppare compiti di valutazione che evidenziano prove per specifiche abilità CT. Gli esempi che presentiamo di come viene misurata la CT in Scratch, Zoombinis e FACT hanno lo scopo di evidenziare queste possibilità.

Questa rassegna di Valerie J. Shute et al. 2017 della letteratura ha messo in luce una serie di lacune

nella ricerca sulla CT. Ad esempio, in letteratura manca una definizione generalmente condivisa di CT, insieme a una specifica dei suoi componenti (Wing, 2008). L'uso incoerente dei termini è presente in molti articoli, come la confusione tra informatica, programmazione informatica e CT (Czerkawski & Lyman, 2015; Israel et al., 2015).

Sono inoltre necessari studi sul trasferimento di CT (Bers et al., 2014). Un problema è come identificare l'applicazione della CT in altri domini (Czerkawski & Lyman, 2015). Gli sforzi sono stati fatti da alcuni ricercatori, come Ioannidou et al. (2011), che ha cercato di esaminare il trasferimento della CT acquisita dai giochi ai corsi di matematica e scienze. Allo stesso modo, Repenning et al. (2015) mirava a indagare le abilità CT trasferite dai giochi alla creazione di simulazioni di fenomeni scientifici. Grover et al. (2015) hanno scoperto che gli studenti potrebbero applicare i concetti computazionali appresi da Scratch a specifici linguaggi di programmazione basati su testo, il che è piuttosto promettente. In ogni caso, la conservazione a lungo termine delle competenze CT, insieme all'applicazione delle competenze CT ad altri contesti e domini, è poco studiata.

Un'ultima area che merita attenzione nella ricerca riguarda le differenze di genere nello sviluppo della CT (Teaching from coding) . Le femmine sono spesso sottorappresentate nelle materie relative alle STEM, in particolare una volta che raggiungono l'università. I ricercatori possono prendere in considerazione l'utilizzo della CT per motivare gli studenti, in particolare le ragazze, a specializzarsi in campi scientifici (ad es. Grover & Pea, 2013; Kazimoglu et al. 2012; Repenning et al., 2015). Tuttavia, i risultati degli studi che esaminano le differenze di genere sono incoerenti. Atmatzidou e Demetriadis (2016) hanno riportato che le abilità CT delle ragazze sono migliorate in modo significativo dopo l'intervento e che alla fine ragazze e ragazzi hanno raggiunto lo stesso livello di abilità. Le ragazze tendono a dedicare molto più tempo all'apprendimento online dopo la scuola, ottenendo prestazioni migliori rispetto ai ragazzi (Grover et al., 2015). Tuttavia, nessuna differenza

di genere è stata segnalata da Werner et al. (2012) e Yadav et al. (2014).

Effetti del coding sulle abilità di pianificazione

Le abilità predittive sono abilità pratiche e quotidiane che facilitano la gestione dei carichi di lavoro, l'esecuzione di attività e la collaborazione con gli altri. Ci sono vari esempi di capacità di pianificazione, tra cui il pensiero critico, l'attenzione ai dettagli e la comunicazione. Nella rassegna di Arfè e collaboratori (2020) ci si aspettava proporre il coding ai bambini incrementasse il tempo che questi ultimi dedicano alla pianificazione, migliorando così la loro capacità (ovvero la loro precisione) di risolvere problemi di *coding*. Tuttavia, il tempo che i bambini hanno dedicato alla pianificazione è invece diminuito in seguito all'intervento. Una possibile interpretazione di questa scoperta inaspettata è che la pratica con la piattaforma *Code.org* ha aumentato la familiarità dei bambini con i giochi di coding e gli strumenti, riducendo così il tempo necessario per esplorare il compito. Inizialmente i compiti di *coding* erano totalmente nuovi per i bambini, sia per tipo di compito (sviluppare istruzioni come algoritmi) che per struttura (fornire istruzioni spostando blocchi visivi da un menu a una tela).

Al pretest della rassegna sopracitata, i bambini hanno trascorso la maggior parte del loro tempo di pianificazione esplorando l'interfaccia e cercando di capire cosa fosse richiesto dal compito.

Dopo l'intervento, essendo più familiari con l'attività, i comandi e i blocchi di istruzioni, hanno avuto bisogno di meno tempo per esplorare gli strumenti e l'interfaccia. Pertanto, il tempo di latenza ridotto al post-test potrebbe semplicemente riflettere la crescente familiarità dei bambini con il compito, piuttosto che una reale diminuzione della pianificazione. È probabile che i bambini del gruppo sperimentale abbiano dedicato abbastanza tempo alla pianificazione, come dimostrato dal fatto che le loro prestazioni hanno mostrato un miglioramento significativo dell'accuratezza dal pre-test al post-test.

Al contrario, sul test Elithorn, i bambini del gruppo sperimentale, che si erano esercitati con la piattaforma *Code.org*, sono stati trovati a dedicare più tempo alla pianificazione al post-test, il che probabilmente ha influito anche sul loro maggiore aumento di precisione rispetto al gruppo di controllo. L'Elithorn è un labirinto di carta e matita simile ai labirinti che i bambini possono trovare nelle riviste. I bambini hanno richiesto quindi relativamente poco tempo per capire come svolgere questo compito. In questo caso, il tempo di latenza rifletteva principalmente la tendenza del bambino a pianificare in anticipo e i bambini esposti al coding probabilmente hanno sviluppato questo atteggiamento in misura maggiore.

Conclusioni

Limitazioni e sviluppi futuri

Nello studio appena esaminato di Arfè e colleghi (2020), gli effetti del *coding* sono stati valutati su un solo gruppo di età (bambini di 5-6 anni). Studi trasversali e longitudinali sarebbero importanti per valutare come gli effetti delle attività di coding a scuola influenzino le capacità cognitive dei bambini in diversi momenti dello sviluppo. Per valutare le abilità cognitive in questo studio sono stati utilizzati dei compiti *Executive Functions* (EF) standard, ampiamente utilizzati nella ricerca precedente (Arfe et al., 2018; Brooks et al., 2009; Heine et al., 2010; Leuzzi et al., 2004; Pina et al. al., 2015; Unterrainer et al., 2013). Tuttavia, alcune delle misure considerate hanno mostrato solo una moderata affidabilità test-retest alla nostra valutazione. Altri ricercatori hanno correlato la moderata affidabilità dei compiti EF alla natura specifica di questi compiti. La capacità di esercitare un controllo cognitivo sui nostri comportamenti può essere infatti più instabile di giorno in giorno rispetto ad altre abilità cognitive (Paap & Sawi, 2016; Weafer et al., 2013). Ciò rappresenta una sfida principale nella valutazione degli EF, specialmente quando sono coinvolti bambini, poiché i loro EF possono risultare più instabili nel tempo. Un modo per aumentare l'affidabilità della valutazione EF consiste nell'utilizzare più di un'attività per la stessa misura EF. Questa è stata la strategia adottata in questo studio.

Anche il confronto diretto dell'efficacia di diverse attività CT (ad esempio, decomposizione, astrazione e *debug*), compiti (ad esempio, programmazione virtuale o robotica) o strumenti di *coding* (Scratch o Code.org) sulle capacità cognitive e di *problem solving* dei bambini sono cruciali per informare le decisioni didattiche su quando e come introdurre le diverse componenti della CT nel curriculum scolastico. In molte scuole, il coding non fa ancora parte del curriculum e questo periodo di transizione rappresenta un'opportunità inestimabile per prendere decisioni informate e basate sull'evidenza su come insegnare la CT e il coding a scuola.

La CT è considerata una capacità di *problem solving* strettamente legata allo sviluppo cognitivo del bambino (Roman-Gonzalez et al., 2017). Questo studio ha fornito prove di conferma a sostegno di questa ipotesi. Non solo la pratica con il coding tramite Code.org ha migliorato in modo misurabile la capacità dei bambini di risolvere i problemi di coding (ad es. abilità CT), ma ha anche influenzato positivamente gli EF dei bambini. I bambini esposti alle attività di coding hanno aumentato il tempo e l'accuratezza della pianificazione e hanno ridotto il tasso di errori di inibizione.

In generale, gli effetti dell'intervento di *coding* erano globalmente più evidenti nell'ambito della a pianificazione che nell'ambito dell'inibizione della risposta. Questa scoperta può essere spiegata in due modi. Da un lato, la risoluzione dei problemi implica la capacità di scomporre i problemi e sviluppare un piano d'azione eseguibile verso la soluzione (Chen et al., 2017). Quindi, lavorare alla soluzione dei problemi di coding avrebbe potuto migliorare direttamente le capacità di pianificazione dei bambini. L'inibizione della risposta è coinvolta nella pianificazione (Luciana et al., 2009).

Gli effetti dell'apprendimento del codice sull'inibizione della risposta, dunque, avrebbero potuto essere più indiretti e, quindi, meno forti. D'altra parte, l'inibizione della risposta potrebbe essere più vulnerabile a fattori situazionali ed esterni (ad esempio: stanchezza, umore) rispetto alla pianificazione, che essendo un processo cognitivo più complesso, può comportare strategie compensatorie per superare i limiti nel controllo inibitorio.

Bibliografia

Anderson, L. W., & Krathwohl, D. R. (2001). A taxonomy for learning, teaching, and assessing: A revision of Bloom's taxonomy of educational objectives. Longman.

Alano, J., Babb, D., Bell, J., Booker-Dwyer, T., DeLyser, L. A., McMunn Dooley, C., & Phillips, R. (2016). K–12 computer science framework. *K–12 Computer Science Framework*.

Arfé, B., Vardanega, T., & Ronconi, L. (2020). The effects of coding on children's planning and inhibition skills. *Computers & Education*, *148*, 103807.

Arfé, B., Montanaro, M., Mottura, E., Scaltritti, M., Manara, R., Basso, G., ... & Colombatti, R. (2018). Selective difficulties in lexical retrieval and nonverbal executive functioning in children with HbSS sickle cell disease. *Journal of pediatric psychology*, *43*(6), 666-677.

Atmatzidou, S., & Demetriadis, S. (2016). Advancing students' computational thinking skills through educational robotics: A study on age and gender relevant differences. *Robotics and Autonomous Systems*, *75*, 661-670.

Bernardo, M. A., & Morris, J. D. (1994). Transfer effects of a high school computer programming course on mathematical modeling, procedural comprehension, and verbal problem solution. *Journal of research on computing in education*, *26*(4), 523-536.

Bers, M. U., Flannery, L., Kazakoff, E. R., & Sullivan, A. (2014). Computational thinking and tinkering: Exploration of an early childhood robotics curriculum. *Computers & Education*, *72*, 145-157.

Bers, M. U. (2018, April). *Coding, playgrounds and literacy in early childhood education: The development of KIBO robotics and ScratchJr*. In 2018 IEEE global engineering education conference (EDUCON) (pp. 2094-2102).

Bucciarelli, M. (2019). *Imparare a ragionare... e continuare a farlo. Giornale italiano di psicologia*, 46(4), 743-760.

Chen, G., Shen, J., Barth-Cohen, L., Jiang, S., Huang, X., & Eltoukhy, M. (2017). Assessing elementary students' computational thinking in everyday reasoning and robotics programming. *Computers & education*, *109*, 162-175.

Czerkawski, B. C., & Lyman, E. W. (2015). Exploring issues about computational thinking in higher education. *TechTrends*, *59*(2), 57-65.

Falloon, G. (2016). An analysis of young students' thinking when completing basic coding tasks using Scratch Jnr. On the iPad. *Journal of Computer Assisted Learning*, *32*(6), 576-593.

Fessakis, G., Gouli, E., & Mavroudi, E. (2013). Problem solving by 5–6 years old kindergarten children in a computer programming environment: A case study. *Computers & Education*, *63*, 87-97.

Grover, S., & Pea, R. (2013). Computational thinking in K–12: A review of the state of the field. *Educational researcher*, *42*(1), 38-43.

Grover, S., Pea, R., & Cooper, S. (2015). Designing for deeper learning in a blended computer science course for middle school students. *Computer science education*, *25*(2), 199-237.

Hayes, J., & Stewart, I. (2016). Comparing the effects of derived relational training and computer coding on intellectual potential in school-age children. *British Journal of Educational Psychology*, *86*(3), 397-411.

Heine, A., Tamm, S., De Smedt, B., Schneider, M., Thaler, V., Torbeyns, J., ... & Jacobs, A. (2010). The numerical stroop effect in primary school children: a comparison of low, normal, and high achievers. *Child Neuropsychology*, *16*(5), 461-477.

Ioannidou, A., Bennett, V., Repenning, A., Koh, K. H., & Basawapatna, A. (2011). Computational Thinking Patterns. *Online Submission*.

Israel, M., Pearson, J. N., Tapia, T., Wherfel, Q. M., & Reese, G. (2015). Supporting all learners in school-wide computational thinking: A cross-case qualitative analysis. *Computers & Education*, *82*, 263-279.

Kalelioğlu, F. (2015). *A new way of teaching programming skills to K-12 students: Code. org. Computers in Human Behavior*, 52, 200-210.

Kalelioglu, F., & Gülbahar, Y. (2014). The Effects of Teaching Programming via Scratch on

Problem Solving Skills: A Discussion from Learners' Perspective. *Informatics in education*, *13*(1), 33-50.

Kazimoglu, C., Kiernan, M., Bacon, L., & MacKinnon, L. (2012, October). Experimental evaluation results of a game based learning approach for learning introductory programming. In *E-Learn: World Conference on E-Learning in Corporate, Government, Healthcare, and Higher Education* (pp. 636-647). Association for the Advancement of Computing in Education (AACE).

Leuzzi, V., Pansini, M., Sechi, E., Chiarotti, F., Carducci, C. L., Levi, G., & Antonozzi, I. (2004). Executive function impairment in early-treated PKU subjects with normal mental development. *Journal of inherited metabolic disease*, *27*(2), 115-125.

Luciana, M., Collins, P. F., Olson, E. A., & Schissel, A. M. (2009). Tower of London performance in healthy adolescents: The development of planning skills and associations with self-reported inattention and impulsivity. *Developmental Neuropsychology*, *34*(4), 461-475.

Mayer, R. E., & Fay, A. L. (1987). A chain of cognitive changes with learning to program in Logo. *Journal of Educational Psychology*, *79*(3), 269.

Nardelli, E., & Ventre, G. (2015). *Introducing computational thinking in Italian schools: A first report on "programma il futuro" project*. In INTED2015 Proceedings (pp. 7414-7421).

Paap, K. R., & Sawi, O. (2016). The role of test-retest reliability in measuring individual and group differences in executive functioning. *Journal of Neuroscience Methods*, *274*, 81-93.

Pina, V., Castillo, A., Cohen Kadosh, R., & Fuentes, L. J. (2015). Intentional and automatic numerical processing as predictors of mathematical abilities in primary school children. *Frontiers in Psychology*, *6*, 375.

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

Psycharis, S., & Kallia, M. (2017). The effects of computer programming on high school students' reasoning skills and mathematical self-efficacy and problem solving. *Instructional science*, *45*(5), 583-602.

Repenning, A., Webb, D. C., Koh, K. H., Nickerson, H., Miller, S. B., Brand, C., ... & Repenning, N. (2015). Scalable game design: A strategy to bring systemic computer science education to schools through game design and simulation creation. *ACM Transactions on Computing Education (TOCE)*, *15*(2), 1-31.

Resnick, M., & Siegel, D. (2015). A different approach to coding. *International Journal of People-Oriented Programming*, *4*(1), 1-4.

Román-González, M., Pérez-González, J. C., & Jiménez-Fernández, C. (2017). Which cognitive abilities underlie computational thinking? Criterion validity of the Computational Thinking Test. Computers in human behavior, 72, 678-691.

Rule, J. S., Tenenbaum, J. B., & Piantadosi, S. T. (2020). The child as hacker. Trends in cognitive sciences, 24(11), 900-915.

Sáez-López, J. M., Román-González, M., & Vázquez-Cano, E. (2016). Visual programming languages integrated across the curriculum in elementary school: A two year case study using "Scratch" in five schools. *Computers & Education*, *97*, 129-141.

Settle, A., Franke, B., Hansen, R., Spaltro, F., Jurisson, C., Rennert-May, C., & Wildeman, B. (2012, July). Infusing computational thinking into the middle-and high-school curriculum. In *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education* (pp. 22-27).

Shute, V. J., Sun, C., & Asbell-Clarke, J. (2017). Demystifying computational thinking. *Educational Research Review*, *22*, 142-158.

Strawhacker, A., & Bers, M. U. (2019). *What they learn when they learn coding: investigating cognitive domains and computer programming knowledge in young children*. Educational Technology Research and Development, 67(3), 541-575.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., & Goodman, N. D. (2011). How to grow a mind: Statistics, structure, and abstraction. *science*, *331*(6022), 1279-1285.

Ullman, T. D., Spelke, E., Battaglia, P., & Tenenbaum, J. B. (2017). Mind games: Game engines as an architecture for intuitive physics. *Trends in cognitive sciences*, *21*(9), 649-665.

Unterrainer, J. M., Ruh, N., Loosli, S. V., Heinze, K., Rahm, B., & Kaller, C. P. (2013). Planning steps forward in development: in girls earlier than in boys. *PloS one*, *8*(11), e80772.

Weafer, J., Baggott, M. J., & de Wit, H. (2013). Test–retest reliability of behavioral measures of impulsive choice, impulsive action, and inattention. *Experimental and clinical psychopharmacology*, *21*(6), 475.

Werner, L., Denner, J., Campe, S., & Kawamoto, D. C. (2012, February). The fairy performance assessment: Measuring computational thinking in middle school. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 215-220).

Wing, J. (2011). *Research notebook: Computational thinking—What and why*. The link magazine, 6, 20-23.

Wing, J. M. (2006). *Computational thinking*. Communications of the ACM, 49(3), 33-35.

Wing, J. M. (2008). Computational thinking and thinking about computing. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 366(1881), 3717-3725.

Wolz, U., Stone, M., Pearson, K., Pulimood, S. M., & Switzer, M. (2011). Computational thinking and expository writing in the middle school. *ACM Transactions on Computing Education (TOCE)*, *11*(2), 1-22.

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in K-12 classrooms. *TechTrends*, *60*(6), 565-568.

Yaşar, O. (2018). *A new perspective on computational thinking*. Communications of the ACM, 61(7), 33-39.

Sitografia

Governo Italiano. (2018). https://labuonascuola.gov.it/costruiamo-insieme/pensiero-computazionale/ [Consultato il 10/09/2022]