PENSIERO COMPUTAZIONALE

UNA GUIDA PER INSEGNANTI

ITD CNR

Questa guida è la versione italiana, di:

Computational thinking - A guide for teachers
di Computing at School (http://computingatschool.org.uk/computationalthinking)

Traduzione di:

Augusto Chioccariello
Consiglio Nazionale delle Ricerche
Istituto per le Tecnologie Didattiche

Sommario

Prefazione

Il pensiero computazionale è un concetto introdotto dalla Prof.ssa Janette Wing nel 2006. In un articolo breve, un "viewpoint" (a parer mio), la Wing propone il pensiero computazionale come una competenza originale e tipica dell'informatica, utile a tutti. Questo ha generato un vivace dibattito internazionale e la rimessa in discussione del curriculum della scuola. Non succede spesso che si rimetta in discussione l'impianto del curriculum scolastico; il pensiero computazionale è riuscito non solo a porre il problema, ma anche ad ottenerne la sua introduzione nel curriculum a partire dalla scuola primaria in Inghilterra (dall'anno scolastico 2014-2015), Australia, Francia, Polonia e Finlandia (dall'anno scolastico 2016-2017). Una recente legge federale degli Stati Uniti ha inserito l'informatica tra le materie che fanno parte di Science-Technology-Engineering-Math (STEM) aprendo le porte all'introduzione del pensiero computazionale nei curricula degli stati dell'Unione.

Con il documento sulla buona scuola, il governo ha aperto il dibattito sull'introduzione 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)[1].

La legge 107[2] (2015) include il pensiero computazionale tra gli obiettivi educativi della scuola (comma 7). Il successivo Piano Nazionale Scuola Digitale[3] ribadisce questa decisione e auspica una ridefinizione della "competenza digitale" e una revisione delle "Indicazioni per il curriculum" (pag. 74).

In parallelo, il MIUR ha promosso il progetto Programma il Futuro che dall'anno scolastico 2014-2015 sta sperimentando attività di "coding" per l'introduzione del pensiero computazionale a scuola.

Nella circolare MIUR 08/10/2015[4] l'introduzione del pensiero computazionale è così motivata:

> *"Nel mondo odierno i computer sono dovunque e costituiscono un potente strumento per la comunicazione. Per essere culturalmente preparato a qualunque lavoro uno studente vorrà fare da grande è indispensabile quindi una comprensione dei concetti di base dell'informatica. Esattamente com'è accaduto nel secolo passato per la matematica, la fisica, la biologia e la chimica.*
>
> *Il lato scientifico-culturale dell'informatica, definito anche "pensiero computazionale", aiuta a sviluppare competenze logiche e capacità di risolvere problemi in modo creativo ed efficiente, qualità che sono importanti per tutti i futuri cittadini.*
>
> *Il modo più semplice e divertente di sviluppare il "pensiero computazionale" è attraverso la programmazione (coding) in un contesto di gioco.*
>
> *Come previsto anche nel Piano Nazionale Scuola Digitale, un'appropriata educazione al "pensiero computazionale" che vada al di là dell'iniziale alfabetizzazione digitale, è infatti essenziale affinché le nuove generazioni siano in grado di affrontare la società del futuro non*

[1] https://labuonascuola.gov.it/costruiamo-insieme/pensiero-computazionale/
[2] http://www.gazzettaufficiale.it/eli/id/2015/07/15/15G00122/sg
[3] http://www.istruzione.it/scuola_digitale/allegati/Materiali/pnsd-layout-30.10-WEB.pdf
[4] http://www.istruzione.it/allegati/2015/prot2187.pdf

da consumatori passivi ed ignari di tecnologie e servizi, ma da soggetti consapevoli di tutti gli aspetti in gioco e come attori attivamente partecipi del loro sviluppo."

L'associazione Computing at School,, composta da insegnanti e ricercatori inglesi, che sta cooperando con il governo per la formazione in servizio degli insegnanti dopo l'introduzione del nuovo curriculum sul "computing", ha prodotto un'utile guida per insegnanti sul pensiero computazionale: "Computational thinking - A guide for teachers". Il presente documento ne è la versione italiana con alcuni adattamenti al contesto nazionale; i riferimenti al curriculum inglese per il "computing" sono stati rimossi e in questa introduzione si dà conto della situazione italiana.

Un ringraziamento particolare al CAS e agli autori della guida inglese che scegliendo una licenza Creative Commons hanno semplificato la realizzazione di questa versione. Elisa Pastore ha collaborato alla realizzazione della versione italiana.

Augusto Chioccariello
Genova, marzo 2016

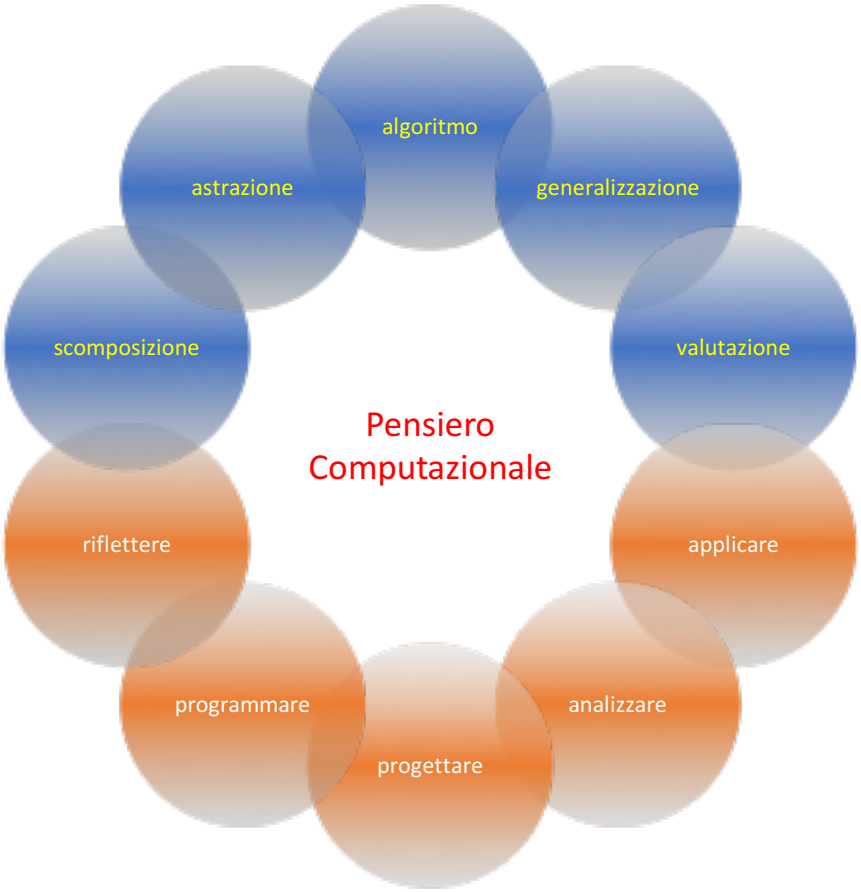
La natura del pensiero computazionale

Il pensiero computazionale fornisce un quadro potente per studiare l'informatica. La sua applicazione, però, va ben al di là dell'informatica: è il processo di riconoscere aspetti della computazione nel mondo che ci circonda, e nell'applicare strumenti e tecniche informatiche per capire e ragionare su sistemi e processi naturali, sociali e artificiali. Permette agli studenti di affrontare problemi, di scomporli in pezzi risolvibili e di elaborare gli algoritmi per risolverli. La locuzione "pensiero computazionale" è stata usata per primo da Seymour Papert, la prof.ssa Jeannette Wing ha reso popolare l'idea sostenendo che il pensiero computazionale dovrebbe far parte delle competenze di tutti i nuovi studenti universitari (Wing, 2006). 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."*
>
> (Wing, 2011, p. 20).

L'enfasi è chiara. Si concentra sugli studenti che effettuano un processo di pensiero, non sulla produzione di artefatti o di prove. Il pensiero computazionale è lo sviluppo di capacità di pensiero che contribuiscono all'apprendimento e alla comprensione.



Concetti del pensiero computazionale

Il pensiero computazionale è un processo cognitivo che coinvolge il **ragionamento logico** attraverso cui i problemi sono risolti, gli artefatti, le procedure e i sistemi sono meglio compresi.

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

- **algoritmi**;
- **scomposizione**;
- **generalizzazioni**, individuando e facendo uso di **schemi ricorrenti**;
- **astrazioni**, scegliendo rappresentazioni appropriate;
- **valutazione**.

Ragionamento logico

Il ragionamento logico permette agli alunni di dare un senso alle cose, analizzando e verificando i fatti attraverso un ragionamento chiaro e preciso. Esso consente agli alunni di disegnare sia sui propri modelli di conoscenza sia interni per fare e verificare previsioni e trarre conclusioni. Gli alunni lo utilizzano ampiamente quando provano, debuggano, e correggono algoritmi. Il ragionamento logico è l'applicazione di altri concetti di pensiero computazionale per risolvere problemi.

Studenti di un corso di design stanno scegliendo i materiali per i diversi elementi del progetto di un modello di camion. Stanno usando generalizzazioni quando riconoscono che le proprietà di un materiale utilizzato in una situazione lo rendono adatto ad essere utilizzato in un contesto completamente diverso. Essere in grado di dividere il nuovo progetto in parti, che richiedono materiali diversi, è un esempio di scomposizione. Lo studente utilizzerà il ragionamento logico per la progettazione di un camion.

Gli alunni utilizzano il ragionamento logico quando studiano 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. Essi possono essere in grado di generalizzare il comportamento ad altre situazioni, per esempio ad una gru. Si usa il ragionamento logico per capire una proprietà dei gravi.

Il ragionamento logico è fondamentale nel rimuovere gli errori (debug) di un programma. Possono lavorare in gruppo per controllare a vicenda i loro programmi, isolare gli errori (bug), e suggerire correzioni. Durante questo processo, potranno usare l'astrazione, la valutazione e il pensiero algoritmico. La correzione di errori nei programmi richiede ragionamento logico.

Astrazione

L'astrazione fa sì che risulti più facile ragionare su problemi o sistemi. L'astrazione è il processo di rendere un artefatto più comprensibile attraverso la rimozione di dettagli superflui alla sua descrizione. Un esempio classico è la cartina della metropolitana. Per esempio la metropolitana di Milano è un sistema complesso. Particolari rappresentazioni di Milano (di solito mappe o schemi) aiutano i diversi utenti. La mappa della metropolitana di Milano è un'astrazione molto raffinata con solo le informazioni essenziali per il viaggiatore a navigare nella rete metropolitana. Sono omesse informazioni come la distanza e l'esatta posizione geografica, che risulterebbero un inutile appesantimento. La mappa è una rappresentazione che contiene tutte le informazioni necessarie per pianificare un percorso da una stazione all'altra - e non di più!
L'abilità nell'astrazione sta nello scegliere il dettaglio da nascondere in modo che il problema diventi più facile, senza perdere niente di ciò che è importante. Un aspetto fondamentale di questo processo

è nella scelta di una buona rappresentazione del sistema. Rappresentazioni diverse rendono più semplice operare su cose diverse.

Per esempio, un programma per computer che gioca a scacchi è un'astrazione. Si tratta di un insieme finito e preciso di regole da seguire ogni volta che è il turno del computer di muovere. È lontano dai processi mentali analogici, emozionali, parziali e distratti tipici di un giocatore umano di scacchi. È un'astrazione perché i dettagli inutili di tali processi sono rimossi.

Valutazione

La valutazione è il processo per garantire che una soluzione, sia essa un algoritmo, un sistema, o un processo, è buona: cioè che sia adatta allo scopo. Varie proprietà delle soluzioni devono essere valutate. Sono corrette? Sono abbastanza veloci? Usano le risorse con parsimonia? Sono facili da utilizzare per le persone? Promuovono un'esperienza adeguata? Bisogna fare compromessi, raramente vi è un'unica soluzione ideale per tutte le situazioni. Nella valutazione usata nel pensiero computazionale c'è una specifica attenzione alla cura del dettaglio.

Continuamente vengono sviluppate nuove interfacce di computer per soddisfare le esigenze dei diversi utenti. Per esempio, se è necessario un dispositivo medico che fornisce automaticamente i farmaci ad un paziente, questo deve essere programmabile senza errori in modo semplice, rapido e sicuro. La soluzione deve assicurare che gli infermieri possanno regolare il dosaggio facilmente senza errori e che non sarà frustrante da usare per pazienti e infermieri. Nel progetto proposto ci sarà il giusto livello di compromessi tra velocità di inserimento dei numeri (efficienza) e controlli per evitare errori (efficacia e usabilità). Il design dovrebbe essere giudicato sulla base delle specifiche proposte da medici, legislatori ed esperti di design dei dispositivi medici (criteri) e le norme generali in materia di buon design (euristica). Criteri, euristica ed esigenze degli utenti consentono di effettuare valutazioni in modo sistematico e rigoroso.

Pensiero algoritmico

Il pensiero algoritmico è un modo di arrivare a una soluzione attraverso una chiara definizione dei passaggi. Alcuni problemi sono del tipo una tantum; una volta risolti, si applica la soluzione, e si passa ad affrontare il successivo. Il pensiero algoritmico entra in gioco quando problemi analoghi devono essere risolti più e più volte. Essi non devono essere analizzati ex novo ogni volta, è necessaria una soluzione che funzioni ogni volta. Un esempio sono gli algoritmi per fare la moltiplicazione o la divisione che si imparano a scuola. Se le regole sono eseguite con precisione, da un computer o da una persona, si può trovare il risultato di ogni moltiplicazione. Una volta che ci si è impadroniti dell'algoritmo, questo non deve essere elaborato da zero per ogni nuovo problema.

Il pensiero algoritmico è la capacità di pensare in termini di sequenze e regole per risolvere problemi o capire situazioni. Si tratta di una competenza di base che gli alunni sviluppano quando imparano a scrivere i loro programmi per il computer.

Scomposizione

La scomposizione è un modo di pensare ad artefatti in termini delle loro componenti. Le singole parti possono essere comprese, risolte, sviluppate e valutate separatamente. Questo approccio rende più facile risolvere problemi complessi, permette di comprendere meglio situazioni nuove e facilita la progettazione di grandi sistemi.

Ad esempio, preparare la colazione può essere scomposta in attività separate come: fare toast; preparare il tè; far bollire delle uova; ecc. Ognuno di queste, a sua volta, potrebbe essere ulteriormente suddivisa in una serie di passaggi. Attraverso la scomposizione del compito originale ogni parte può essere sviluppata e integrato più avanti nel processo.

Si consideri lo sviluppo di un gioco: 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.

Generalizzazione

La generalizzazione è associata al saper identificare schemi ricorrenti, somiglianze, connessioni, e a sfruttare queste caratteristiche. È un modo di risolvere rapidamente problemi nuovi sulla base delle soluzioni di problemi già affrontati, a capitalizzare le precedenti esperienze. È importante porsi domande del tipo: "È simile ad un problema che ho già risolto?", "In cosa è diverso?". Altrettanto importante è imparare a riconoscere schemi ricorrenti sia nei dati che nei processi e/o strategie che vengono utilizzati. Algoritmi che risolvono alcuni problemi specifici possono essere adattati per risolvere tutta una classe di problemi simili. Quindi, quando si incontra un problema di quella classe si può applicare la soluzione generale.

> Per esempio, uno studente sta usando la "tartaruga" per disegnare una serie di figure geometriche. L'alunno scrive un programma per disegnare un quadrato e un triangolo. Decide quindi di disegnare un ottagono e un decagono. Dal lavoro con il quadrato e il triangolo, si accorge che esiste una relazione tra il numero di lati della figura e gli angoli coinvolti nel disegnarla con la tartaruga. Può quindi scrivere un algoritmo che esprime questo rapporto e utilizzarlo per disegnare qualsiasi poligono regolare.

Tecniche associate al pensiero computazionale

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.

Riflettere

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.

Programmare (coding)

Un elemento essenziale dello sviluppo di qualsiasi sistema di computer è tradurre il progetto in forma di codice e valutarlo per garantirne il corretto funzionamento in tutte le condizioni previste. Il debugging è l'applicazione sistematica di analisi e valutazione utilizzando abilità quali test, tracciamento, e il pensiero logico per prevedere e verificare i risultati.

Progettare

Progettare significa definire la struttura, l'aspetto e la funzionalità degli artefatti. Essa comporta la creazione di rappresentazioni del progetto, ivi incluse rappresentazioni leggibili da altre persone come diagrammi di flusso, story board, pseudo-codice, schemi di sistema, ecc. Questo implica ulteriori attività di scomposizione, astrazione e progettazione di algoritmi.

Analizzare

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

Applicare

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.

Pensiero computazionale in classe

Ognuno dei concetti del pensiero computazionale sopra descritti (il pensiero algoritmico, la scomposizione ecc.) viene associato a diversi comportamenti da parte degli studenti, che possono essere osservati in classe.

Pensiero algoritmico

Il Pensiero algoritmico rappresenta la capacità di pensare in termini di sequenze e regole, come modo di risolvere i problemi. Si tratta di una competenza di base che gli alunni sviluppano quando imparano a scrivere i propri programmi per computer. In classe 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, muovono 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 (codifica in Prolog o in un linguaggio di creazione di query su database).

Questo implica anche:

- Utilizzo di un'appropriata dicitura per scrivere il codice che rappresenti uno dei comandi suddetti.
- Creazione di algoritmi per verificare un'ipotesi.
- 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.

Scomposizione

La scomposizione è un modo di pensare agli artefatti in termini delle parti che li compongono. Le singole parti possono essere comprese, risolte, sviluppate e valutate separatamente. In classe si può osservare:

- Divisione degli artefatti nelle parti costituenti per renderli più facili da lavorare.
- Scomposizione di un problema in versioni più semplici dello stesso, che possono essere risolte nel medesimo modo (metodo divide et impera e ricorsivo).

Generalizzazione

La generalizzazione è un modo di risolvere nuovi problemi sulla base delle soluzioni ai precedenti. Si tratta di identificare e sfruttare i modelli. In classe si può osservare:

- 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.

Astrazione

L'astrazione è il processo di creazione di un artefatto di maggior comprensione nascondendone i dettagli. In classe 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.

Valutazione

La valutazione è il processo di verifica che la soluzione sia esatta: che sia adatta allo scopo. Durante la valutazione basata sul pensiero computazionale, vi è una specifica, e spesso estrema, attenzione alla cura dei dettagli. In classe si può osservare:

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

Bibliografia

Department for Education. 2014. *The National Curriculum in England, Framework Document*. Reference: DFE-00177-2013. (https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/335116/Master_final_national_curriculum_220714.pdf)

Dorling, M., Selby, C. & Woollard, J. 2015. Evidence of Assessing Computational Thinking. IFIP 2015, *A New Culture of Learning: Computing and Next Generations*. Vilnius, Lithuania. (http://eprints.soton.ac.uk/377856)

Selby, C. & Woollard, J. 2013. *Computational thinking: the developing definition*. (http://eprints.soton.ac.uk/356481/)

Wing, J. 2006. Computational Thinking. *Communications of ACM*, 49, 3, 33-35. (http://dl.acm.org/citation.cfm?id=1118215)

Wing, J. 2011. Research Notebook: Computational Thinking - What and Why? *The Link*. Pittsburgh, PA: Carneige Mellon. (http://www.cs.cmu.edu/sites/default/files/11-399_The_Link_Newsletter-3.pdf)


astrazione
scomposizione
Computazionale
valutazione
algoritmo
generalizzazione
Pensiero