UNIVERSITÀ DEGLI STUDI DI SALERNO

STUDI DI SCIENZE MATEMATICHE,
FISICHE E NATURALI

ITADINFO 2025

Terzo Convegno Italiano sulla Didattica dell'Informatica

A cura di
Maria Angela Pellegrino e Veronica Rossano

Ledizioni
The Innovative LEDIpublishing Company

Collana Scientifica dell'Università di Salerno

Studi di Scienze Matematiche, Fisiche e Naturali

INFORMATICA E SCUOLA
LABORATORIO NAZIONALE cini

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

APS Programma il Futuro

Progetto Nazionale "Informatica" del Piano Lauree Scientifiche

cini consorzio interuniversitario nazionale per l'informatica

[ITADINFO]

3° CONVEGNO ITALIANO SULLA DIDATTICA DELL'INFORMATICA

Salerno, 03-05 ottobre 2025

CON IL SOSTEGNO DI

Fondazione CRT | Progetto DIDEROT

SPONSOR ESPOSITORI

EDIZIONI THEMIS

axios italia

3D ROBOTICS

twinscape

BIMED

SPONSOR

seeweb THINK CLOUD

CON IL PATROCINIO DI

GRIN Società Informatica Italiana

REGIONE CAMPANIA

PROVINCIA DI SALERNO

COMUNE DI SALERNO

GII

I contributi qui pubblicati sono stati revisionati dal Comitato Scientifico del Convegno con *single blind peer-review*.

Via Boselli, 10 – 20136 Milano – Italy
www.ledizioni.it
info@ledizioni.it

ITADINFO 2025. Terzo Convegno Italiano sulla Didattica dell'Informatica
Impaginazione, editing e revisione del presente volume a cura di Maria Angela Pellegrino e Veronica Rossano

Prima edizione: ottobre 2025

ISBN cartaceo: 9791256005123
ISBN eBook: 9791256005130
ISBN PDF online: 9791256005147

Copertina e progetto grafico: ufficio grafico Ledizioni

Informazioni sul catalogo e sulle ristampe: www.ledizioni.it

Indice

RACCONTI DI ESPERIENZE SUL CAMPO E RELAZIONI SCIENTIFICHE PER LA SECONDARIA DI SECONDO GRADO

LABORATORI PER LA SECONDARIA DI PRIMO GRADO

LABORATORI PER LA SECONDARIA DI SECONDO GRADO

POSTER PER LA PRIMARIA

POSTER PER LA PRIMARIA E LA SECONDARIA DI PRIMO GRADO

POSTER PER LA SECONDARIA DI SECONDO GRADO

POSTER PER OGNI ORDINE E GRADO

VIDEO DIVULGATIVI

Prefazione

Il convegno italiano sulla Didattica dell'Informatica (ITADINFO) si propone come un punto d'incontro tra il mondo della scuola e quello della ricerca per favorire lo scambio di esperienze, pratiche e visioni sull'insegnamento dell'informatica come disciplina scientifica. ITADINFO, giunto nel 2025 alla sua **terza edizione**, si è svolto a **Salerno**, consolidando il suo ruolo di spazio aperto, partecipativo e inclusivo, dedicato a chi promuove la cultura informatica in ogni ordine e grado scolastico.

L'edizione 2024, tenutasi a Genova, ha dimostrato l'efficacia del formato basato su laboratori interattivi, relazioni scientifiche e racconti dal campo. L'interesse e la partecipazione crescenti hanno confermato il bisogno diffuso di strumenti, formazione e confronto per rendere l'informatica accessibile, significativa e didatticamente sostenibile. ITADINFO 2025 ha raccolto questa eredità offrendo un programma ancora più ricco e variegato, capace di rappresentare le diverse anime della comunità educante.

Il programma di quest'anno si articola in **quattro sezioni principali**:

- **Relazioni scientifiche e racconti di esperienze**, che restituiscono pratiche didattiche sperimentate, attività interdisciplinari, riflessioni teoriche e percorsi strutturati per l'insegnamento dell'informatica;
- **Poster**, che arricchiscono lo spazio comune del convegno con una galleria di idee, prototipi, attività in corso e iniziative per la formazione e l'orientamento;
- **Laboratori formativi**, dedicati alla sperimentazione diretta di percorsi e strumenti, per supportare i docenti nella progettazione autonoma di attività in classe.
- Brevi **video divulgativi** per informare su temi dell'informatica, con esempi pratici, illustrazioni e testimonianze, così da rendere accessibili argomenti complessi a un pubblico ampio.

L'intero programma è attraversato da temi trasversali che testimoniano l'evoluzione delle esigenze educative e l'urgenza di una didattica dell'informatica aggiornata, critica e consapevole. L'intelligenza artificiale, la modellazione della realtà attraverso strumenti concettuali e computazionali, e l'integrazione di approcci educativi che combinano strumenti analogici e digitali rappresentano alcune delle direttrici che guidano le attività proposte. In questo contesto, l'informatica si conferma non solo come disciplina autonoma, ma anche come chiave di lettura per comprendere e agire nel mondo, promuovendo lo sviluppo del pensiero critico, creativo e computazionale. I contributi selezionati offrono un prezioso repertorio di attività, strumenti e riflessioni, rappresentando un punto di riferimento sia per chi si avvicina ora all'informatica come disciplina scolastica, sia per chi la insegna da anni e cerca nuove strade per farla vivere in classe. ITADINFO 2025 rinnova l'impegno a essere una comunità aperta, che valorizza il sapere dei docenti, la ricerca educativa e le sinergie tra scuola, università e territorio. A tutti i partecipanti, relatori e relatrici, un sentito ringraziamento per aver reso questo incontro una nuova tappa verso una didattica dell'informatica sempre più consapevole, condivisa e inclusiva.

Veronica Rossano
Maria Angela Pellegrino

Atti del convegno

ITADINFO 2025
Terzo Convegno Italiano sulla Didattica dell'Informatica

03-05 ottobre 2025
Laboratorio CINI Informatica e Scuola,
Università degli Studi di Salerno e APS Programma il Futuro

Comitato scientifico del convegno

Veronica Rossano, *Università degli Studi di Bari "Aldo Moro" (Presidente)*
Guglielmo Abbruzzese, *Università di Roma "Tor Vergata"*
Agnese Addone, *Università degli Studi di Salerno*
Giuliana Barberis, *Liceo Scientifico "Maria Curie" Pinerolo*
Claudia Canali, *Università di Modena e Reggio Emilia*
Sara Capecchi, *Università di Torino*
Antonella Carbonaro, *Università di Bologna*
Giorgio Delzanno, *Università di Genova*
Agnese Del Zozzo, *Università di Trento*
Valentina Di Noi, *Università di Torino*
Filomena Ferrucci, *Università degli Studi di Salerno*
Luca Forlizzi, *Università dell'Aquila*
Ilenia Fronza, *Università di Bolzano*
Maurizia Gai, *Istituto Comprensivo Vivaldi-Murialdo di Torino*
Enrica Gentile, *Università di Bari*
Giovanna Guerrini, *Università di Genova*
Gennaro Iaccarino, *I.I.S.S. "G. Galilei" di Bolzano, docente comandato presso la Dir. Istr. e Formazione italiana di Bolzano*
Luca Lamanna, *Università di Milano*
Michael Lodi, *Università di Bologna*
Violetta Lonati, *Università di Milano*
Sabrina Mantaci, *Università di Palermo*
Simone Martini, *Università di Bologna*
Giovanna Melideo, *Università dell'Aquila*
Claudio Mirolo, *Università di Udine*
Mattia Monga, *Università di Milano*
Alberto Montresor, *Università di Trento*
Anna Morpurgo, *Università di Milano*
Enrico Nardelli, *Università di Roma "Tor Vergata"*
Martina Palazzolo, *Istituto Comprensivo Ilaria Alpi di Milano*
Fabio Palomba, *Università degli Studi di Salerno*
Maria Angela Pellegrino, *Università degli Studi di Salerno*
Marcello Sarini, *Università di Milano – Bicocca*
Emanuele Scapin, *ITT Giacomo Chilesotti di Thiene*
Vittorio Scarano, *Università degli Studi di Salerno*
Ugo Solitro, *Università di Verona*
Tullio Vardanega, *Università di Padova*

Comitato organizzatore del convegno

Presidenza e relazioni istituzionali
Enrico Nardelli, *Università di Roma "Tor Vergata" e direttore del Laboratorio CINI*
Filomena Ferrucci, *Università degli Studi di Salerno*
Vittorio Scarano, *Università degli Studi di Salerno*

Programma scientifico e Atti del convegno

Maria Angela Pellegrino, *Università degli Studi di Salerno*
Veronica Rossano, *Università di Bari*

Comunicazione

Michael Lodi, *Università di Bologna*

Sponsorizzazioni

Fabio Palomba, *Università degli Studi di Salerno*
Tullio Vardanega, *Università di Padova*

Ufficio stampa e comunicazione

Reputation Agency
Francesco Lacchia, *APS Programma il Futuro*

Filomena Ferrucci, *Università degli Studi di Salerno*
Fabio Palomba, *Università degli Studi di Salerno*
Maria Angela Pellegrino, *Università degli Studi di Salerno*
Vittorio Scarano, *Università degli Studi di Salerno*

Le *Notional machine* per l'apprendimento della programmazione, anche nell'era dell'AI generativa

Michael Lodi[1], *Simone Martini*[1]

[1] *Dipartimento di Informatica - Scienza e Ingegneria Alma Mater Studiorum - Università di Bologna Bologna, Italia. Laboratorio CINI Informatica e Scuola*

{*michael.lodi, simone.martini*}*@unibo.it*

Abstract

Imparare a programmare richiede di acquisire conoscenze di tipo sintattico, concettuale e strategico. La comprensione concettuale del comportamento dei programmi durante l'esecuzione è fondamentale per un apprendimento efficace. Questo tipo di conoscenza si basa sulla costruzione di modelli mentali adeguati. Gli insegnanti possono utilizzare una (o più) "notional machine", strumenti pedagogici che in modo semplificato ma preciso spiegano il funzionamento della macchina che esegue un programma scritto in un certo linguaggio. In assenza di una formazione esplicita, gli studenti sviluppano spontaneamente tali modelli, rischiando però misconcezioni. Per favorire lo sviluppo di modelli mentali accurati, sono cruciali attività di visualizzazione dell'esecuzione del codice, che includono tecniche di tracciamento manuale e strumenti automatici, e attività di lettura e comprensione del codice. L'avvento di strumenti di intelligenza artificiale generativa capaci di scrivere codice rende ancor più urgente che gli studenti padroneggino una robusta conoscenza concettuale: solo così potranno valutare criticamente e comprendere i programmi generati automaticamente, individuandone limiti e potenzialità.

1. Introduzione

Imparare a programmare richiede diversi tipi di conoscenza [1, 12]. In particolare, possiamo individuare tre categorie principali.

La prima è la conoscenza **sintattica**, cioè la capacità di utilizzare correttamente la sintassi di un linguaggio di programmazione. Questa conoscenza riguarda, ad esempio, come si scrive un determinato costrutto nel linguaggio scelto.

La seconda categoria è la conoscenza **concettuale**, che consiste nel comprendere la dinamica di un programma in esecuzione. In altre parole, significa sapere come funzionano realmente i costrutti e in che modo essi determinano l'esecuzione del programma (es. quante volte un ciclo verrà eseguito, quando termina, che effetto ha la modifica di un elemento di una lista, e così via...).

La terza categoria è la conoscenza **strategica**, ovvero la capacità di applicare sia la conoscenza sintattica che quella concettuale per risolvere problemi nuovi o raggiungere obiettivi

specifici. Si tratta, per esempio, di sapere quando utilizzare un particolare costrutto, perché sceglierlo rispetto ad altri e, più in generale, come affrontare e risolvere un problema utilizzando la programmazione.

Sebbene la sintassi costituisca una difficoltà tipicamente solo nella fase iniziale dell'apprendimento della programmazione (abbastanza presto chi è alle prime armi riesce a scrivere programmi sintatticamente corretti – ma spesso semanticamente insensati), la maggior parte dei libri di testo si concentra sulla conoscenza sintattica (es. il suo indice è strutturato sulla base dei costrutti che vengono insegnati in ciascun capitolo). Meno attenzione è posta alla conoscenza concettuale (tema di cui tratteremo in questo articolo) e ancor meno alla conoscenza strategica.

2. Le *Notional machine*

Tutti noi creiamo modelli mentali (semplificati) di ciò che ci circonda, che servono per spiegarci come funziona qualcosa in situazioni diverse e fare previsioni. Si possono avere modelli mentali di noi stessi, di un'altra persona, di oggetti, di sistemi. Grazie a questi modelli, possiamo eseguire nella nostra mente delle simulazioni. Queste simulazioni possono porre l'oggetto del modello mentale in contesti diversi, o, per usare una metafora informatica, istanziare alcune variabili del modello per simulare mentalmente cosa accadrebbe in una specifica condizione. Queste simulazioni impiegano la limitata memoria di lavoro che l'essere umano ha a disposizione, e dunque non possono includere troppi stati o variabili. Ovviamente, le simulazioni fatte sui nostri modelli mentali sono *fallibili*, nel senso che non sempre predicono correttamente la realtà [15].

Sappiamo che un programma scritto in un linguaggio di programmazione è eseguito da una certa macchina: può essere una macchina fisica, ma, più probabilmente, una macchina astratta, cioè un modello concettuale che descrive precisamente "un qualsiasi insieme di strutture dati e di algoritmi che permettano di memorizzare ed eseguire programmi" [6]. Come spesso accade in Informatica, per i linguaggi ad alto livello abbiamo solitamente a che fare con una gerarchia di macchine astratte, che "scendono" fino ad arrivare alla macchina fisica - che davvero "farà i conti" usando elettroni e transistor [10]. Già da tempo la ricerca sull'apprendimento della programmazione sostiene che sia comunque indispensabile comprendere almeno1 il funzionamento del livello "immediatamente inferiore" [2, 17].

Chi apprende deve quindi formarsi un modello mentale della macchina che sta imparando a controllare, nel suo ruolo di esecutrice di programmi (scritti nel linguaggio studiato). Il modello deve essere sufficientemente astratto, ma preciso, per permettere di spiegarsi correttamente come i programmi sono eseguiti.

Le studentesse e gli studenti si formeranno il modello in ogni caso. Se non viene insegnato loro esplicitamente, se lo costruiranno inferendo esclusivamente da ciò che vedono: la sintassi e gli esempi di programmi che vengono presentati loro. Specialmente nei linguaggi ad alto livello moderni, molto leggibili dagli esseri umani, numerosi dettagli sul funzionamento a tempo di esecuzione sono nascosti. Questa "astrazione" dai livelli inferiori è ovviamente un bene, ma può portare a modelli inaccurati e incompleti - che causano delle incomprensioni, chiamate in letteratura *misconcezioni*. Ad esempio, vedendo solo esempi in cui le variabili hanno sempre nomi di una sola lettera, chi apprende potrà pensare che nomi più lunghi rappresentino parole

1 Esistono progetti didattici che, con un approccio bottom-up, provano a risalire l'intera gerarchia di astrazione: uno tra tutti è il famoso "From NAND to Tetris" (https://www.nand2tetris.org/).

riservate (in analogia a quelle che impara, come *for, if, def*); potrà inoltre pensare che l'esecuzione di un *while* termini nel momento esatto in cui la guardia diventa negativa, e non quando il controllo "ripassa" dalla guardia per rivalutarla dopo aver eseguito il corpo del ciclo. La letteratura ha documentato centinaia di queste misconcezioni [3].

Per limitare le misconcezioni, è importante che gli insegnanti facciano esplicito riferimento a una o più *notional machine* (NM)[2]. La letteratura ha proposto diverse definizioni (si vedano ad esempio [16, 7, 5, 13]), ma in generale possiamo dire che si tratta di un dispositivo pedagogico (uno "scaffolding") che l'insegnante fornisce allo studente affinché egli maturi un proprio modello dell'esecutore [13] adeguato a "ragionare sul comportamento di un programma e scrivere programmi che funzionano" [7].

Possiamo informalmente pensare a una *notional machine* come a un modello semplificato dell'esecutore che l'insegnante fornisce. La *notional machine* non deve essere necessariamente fedele a quel che realmente succede dietro le quinte (nella macchina astratta e/o nella macchina fisica), purché raggiunga i suoi obiettivi (far sì che gli studenti si spieghino correttamente il comportamento osservabile di un programma e riescano a scriverne uno corretto). Esistono *notional machine* diverse per scopi didattici e per livelli di astrazione diversi[3] (es. che rappresentano la programmazione ad oggetti come uno scambio di messaggi oppure come la manipolazione di riferimenti ad oggetti in memoria, e così via).

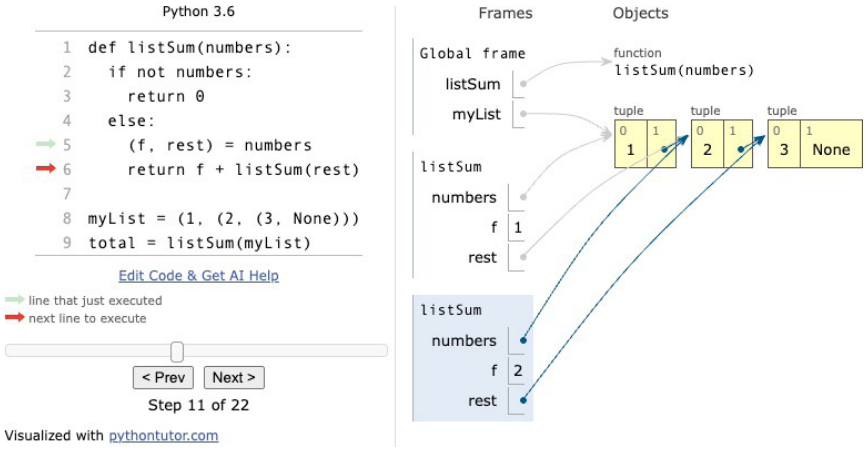

Figura 1: Pythontutor.com

2 L'uso di notional pare indicare il fatto che sia "semplificata" e "solo parzialmente vera" [5]: in questo senso, e per mantenere esplicito il suo intento pedagogico, si potrebbe tradurre come "macchina ideale", "macchina pedagogica" o anche "macchina astratta didattica". Un'alternativa è "macchina concettuale" per legarsi alla conoscenza concettuale di cui abbiamo parlato nell'introduzione (a quanto pare di recente anche in inglese qualcuno utilizza "conceptual machine" [5]). Un problema di questa traduzione è, però, che si può confondere la notional machine con il "modello concettuale", che solitamente indica invece un modello estremamente preciso e fedele alla realtà [5].

3 Una collezione di notional machine, distinte in "rappresentazioni della macchina" e "analogie" è disponibile a https://notionalmachines.github.io/

3. "Rappresentare" la *notional machine*

Una delle tecniche più utilizzate per favorire la creazione di un modello mentale *adeguato* a comprendere e scrivere programmi è quella di rendere *visibile* ciò che visibile non è: ciò che accade a *runtime* quando un programma viene eseguito. La tecnica più ovvia è quella di disegnare / annotare alla lavagna lo stato interno quando si traccia l'esecuzione di un programma (utilizzando ad esempio rettangoli etichettati per tenere traccia dei valori legati ai nomi, o usando frecce per riferimenti, o una "pila" per rappresentare i frame). È importante chiedere a chi apprende di svolgere anche attivamente questo tipo di tracciatura su un foglio di carta: se ciò non avviene, potrebbero interpretare i disegni solo come uno strumento di spiegazione, e non un'abitudine che devono costruirsi loro stessi [4]. Esistono poi strumenti automatici, che permettono di visualizzare lo stato interno eseguendo passo passo un programma. Uno dei più famosi è Python Tutor[4] – che, a dispetto del nome, può visualizzare l'esecuzione in moltissimi linguaggi moderni (Figura 1). Ad una granularità più fine (e quindi più adatto per programmi brevi, iniziali) è il debugger di Thonny[5], un IDE didattico per Python. Il debugger (Fig. 2) permette di visualizzare il risultato della valutazione delle singole espressioni in Python, il passaggio dei parametri (con il legame tra formali e attuali) in caso di chiamata di una funzione, e così via.

Funzioni analoghe sono presenti negli IDE per Java BlueJ[6] e Greenfoot[7] (anche per il linguaggio "a frame" Stride[8]).

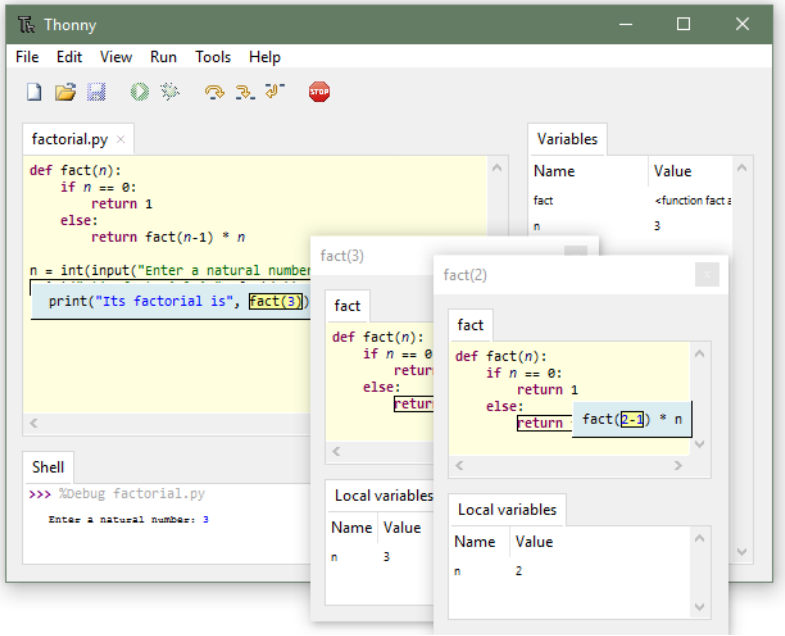

Figura 2: Thonny.org

4 https://pythontutor.com/
5 https://thonny.org/
6 https://bluej.org/
7 https://www.greenfoot.org/home
8 https://stride-lang.net/

Anche nei linguaggi a blocchi sono presenti semplici strumenti: quando si crea una variabile in Scratch[9] e nei suoi derivati, automaticamente il suo nome e valore corrente vengono visualizzati nello Stage (Fig. 3). In Snap![10] è possibile eseguire "passo passo" uno script facendo sì che il blocco eseguito in quel momento "si illumini", di modo da collegare il blocco all'effetto grafico che produce (Fig. 3).

Un aspetto cruciale per la creazione di modelli mentali adeguati è l'utilizzo di attività di *program comprehension* [8], in cui non si chiede di scrivere codice, ma di leggerlo, analizzarlo, comprenderlo (ma anche completarlo, riordinarlo, ecc.): un esplicito allenamento per la conoscenza concettuale. Metodologie di scaffolding per l'introduzione alla programmazione quali UMC [9] e PRIMM [14] si basano tra gli altri proprio su questo tipo di attività.

4. *Notional machine* nell'era dell'AI generativa

Gli strumenti di AI generativa quali gli LLM esibiscono ogni giorno di più capacità di scrivere programmi corretti, impostare progetti, trovare e correggere errori sulla base di prompt scritti in linguaggio naturale. La macchina che interpreta tali prompt può essere vista come una "black box" statistica, allenata su una mole di dati a predire "la prossima parola", spesso con elementi di randomicità.

Figura 3: In linguaggi come Scratch e Snap!, il nome con relativo valore delle variabili è solitamente visualizzato sullo stage. In Snap!, è possibile eseguire step-by-step i blocchi di istruzioni. L'istruzione correntemente eseguita è illuminata di azzurro (in questo caso: attendi 0.8 secondi)

Al giorno d'oggi, l'output è ancora - nella maggior parte dei casi - un programma scritto in un linguaggio di programmazione ad alto livello (es. Python). Riprendendo i tre tipi di conoscenza di cui abbiamo parlato nell'introduzione, questi programmi sono quasi sempre corretti sintatticamente. Invece, anche per la natura stocastica delle reti neurali alla base dei moderni sistemi, talvolta contengono errori importanti a livello concettuale e strategico. Dunque, almeno per il momento, è importante (anzi, *più importante* di prima) che chi apprende acquisisca la conoscenza concettuale, formandosi un modello mentale adeguato a leggere e comprendere un programma scritto da una intelligenza artificiale – per poterne valutare ad esempio la correttezza, l'efficienza, l'opportunità. Se un LLM è in grado di scrivere programmi, occorre che chi li utilizza per questo scopo sappia leggerli e comprenderli, anche per motivi etici, visto che sono i

9 https://scratch.mit.edu/

10 https://snap.berkeley.edu/

programmatori umani (e non le IA) ad avere la responsabilità sociale e giuridica dei programmi che, con qualsiasi strumento prodotti, rendono disponibili [11].

In futuro però, potrebbe esserci un vero e proprio "step-up" nel livello di astrazione: il prossimo linguaggio di programmazione potrebbe "essere l'inglese". In tale scenario, il prompt diventa il nuovo programma, e l'output potrebbe essere direttamente un comportamento o un artefatto computazionale. Non è ancora chiaro quale sia il livello "immediatamente inferiore" alla programmazione via prompt: la macchina che media tra questi due livelli non è più una macchina deterministica tradizionale, ma un modello statistico complesso. Sarà quindi necessario chiarire i meccanismi interni rilevanti da esplicitare per favorire la comprensione. Gli insegnanti dovranno avvalersi di nuove, e molto diverse, notional machine: su questo la ricerca è solo agli albori. La sfida per la didattica dell'informatica nei prossimi anni sarà dunque duplice: da un lato, potenziare l'insegnamento della conoscenza concettuale tradizionale, per garantire una comprensione dei programmi; dall'altro, aiutare gli studenti a comprendere e governare sistemi generativi non deterministici.

Ringraziamenti

Ringraziamo Violetta Lonati per i preziosi suggerimenti alla stesura della versione finale.

Fondi

Il lavoro di M. Lodi è supportato dallo Spoke 1 "FutureHPC & BigData" del Centro Nazionale di Ricerca in "High Performance Computing, Big Data and Quantum Computing" (ICSC) finanziato da MUR Missione 4 Componente 2 Investimento 1.4: Potenziamento strutture di ricerca e creazione di campioni nazionali di R&S (M4C2-19) - Next Generation EU (NGEU). Il lavoro di S. Martini è parzialmente finanziato da: (a) SERICS (PE00000014) nell'ambito del Piano Nazionale di Ripresa e Resilienza del MUR, finanziato dal progetto "NextGenerationEU" dell'Unione Europea; (b) INdAM GNSAGA; (c) supporto finanziario ricevuto nell'ambito del Piano Nazionale di Ripresa e Resilienza, Missione 4, Componente 2, Investimento 1.1, Bando No. 104, pubblicato il 2/02/2022 dal Ministero dell'Università e della Ricerca, finanziato dal progetto "NextGenerationEU" dell'Unione Europea, titolo del progetto "Learning Informatics" – CUP E53D23007720006, decreto di assegnazione del finanziamento No. 959 adottato il 22/04/2022 dal Ministero dell'Università e della Ricerca

Bibliografia

[1] Piraye Bayman and Richard E. Mayer. Using conceptual models to teach BASIC computer programming. Journal of Educational Psychology, 80(3):291–298, September 1988.

[2] Mordechai Ben-Ari. Constructivism in computer science education. Journal of computers in Mathematics and Science Teaching, 20(1):45–73, 2001.

[3] Luca Chiodini, Igor Moreno Santos, Andrea Gallidabino, Anya Tafliovich, André L. Santos, and Matthias Hauswirth. A curated inventory of programming language misconceptions. In Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1, ITiCSE '21, page 380–386, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Kathryn Cunningham, Sarah Blanchard, Barbara Ericson, and Mark Guzdial. Using tracing and sketching to solve programming problems: Replicating and extending an analysis of what students draw. In Proceedings of the 2017 ACM Conference on International Computing Education Research, ICER '17, page 164–172, New York, NY, USA, 2017. Association for Computing Machinery.

[5] Sally Fincher, Johan Jeuring, Craig S. Miller, Peter Donaldson, Benedict du Boulay, Matthias Hauswirth, Arto Hellas, Felienne Hermans, Colleen Lewis, Andreas Mühling, Janice L. Pearce, and Andrew Petersen. Notional machines in computing education: The education of attention. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, ITiCSE-WGR '20, page 21–50, New York, NY, USA, 2020. Association for Computing Machinery.

[6] Maurizio Gabbrielli and Simone Martini. Linguaggi di programmazione. Principi e paradigmi. McGraw-Hill, Milano, 2011.

[7] S. Grover. Computer Science in K-12: An A-To-Z Handbook on Teaching Programming. Edfinity, 2020.

[8] Cruz Izu, Carsten Schulte, Ashish Aggarwal, Quintin Cutts, Rodrigo Duran, Mirela Gutica, Birte Heinemann, Eileen Kraemer, Violetta Lonati, Claudio Mirolo, and Renske Weeda. Fostering program comprehension in novice programmers - learning activities and learning trajectories. In Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education, ITiCSE-WGR '19, page 27–52, New York, NY, USA, 2019. Association for Computing Machinery.

[9] Irene Lee, Fred Martin, Jill Denner, Bob Coulter, Walter Allan, Jeri Erickson, Joyce Malyn-Smith, and Linda Werner. Computational thinking for youth in practice. ACM Inroads, 2(1):32–37, February 2011.

[10] Michael Lodi and Simone Martini. Algoritmi, programmi e linguaggi. Nuova Secondaria, Anno XLIII, 2025/2026. To appear.

[11] Simone Martini. Teaching programming in the age of generative AI. In Proceedings of the 2024 Conference on Innovation and Technology in Computer Science Education V. 1, ITiCSE 2024, pages 1–2, New York, NY, USA, 2024. Association for Computing Machinery

[12] Tanya J. McGill and Simone E. Volet. A conceptual framework for analyzing students' knowledge of programming. Journal of Research on Computing in Education, 29(3):276–297, 1997.

[13] Bhagya Munasinghe, Tim Bell, and Anthony Robins. Computational thinking and notional machines: The missing link. ACM Trans. Comput. Educ., 23(4), December 2023.

[14] Sue Sentance and Jane Waite. PRIMM: Exploring pedagogical approaches for teaching text-based programming in school. In Proceedings of the 12th Workshop on Primary and Secondary Computing Education, WiPSCE '17, page 113–114, New York, NY, USA, 2017. Association for Computing Machinery.

[15] Juha Sorva. Visual program simulation in introductory programming education. PhD thesis, Aalto University, 2012.

[16] Juha Sorva. Notional machines and introductory programming education. ACM Trans. Comput. Educ., 13(2), July 2013.

[17] Matti Tedre, Tapani Toivonen, Juho Kahila, Henriikka Vartiainen, Teemu Valtonen, Ilkka Jormanainen, and Arnold Pears. Teaching machine learning in k–12 classroom: Pedagogical and technological trajectories for artificial intelligence education. IEEE Access, 9:110558–110572, 2021.