Di cosa parliamo quando parliamo di ‘programmi’*

Violetta Lonati[1,3], Claudio Mirolo[2,3] e Mattia Monga[1,3]
[1]Università degli Studi di Milano
[2]Università di Udine
[3]Laboratorio CINI Informatica e Scuola

Sommario

Il mondo della scuola si sta ormai convincendo che la programmazione debba avere un ruolo sempre più rilevante tra le competenze da acquisire a tutti i livelli e in tutti i percorsi formativi. Del resto è assai opportuno che una parte sempre più ampia della cittadinanza sia in grado di capire cosa significa progettare e realizzare elaborazioni automatizzate. Il rischio, tuttavia, è che la complessità tecnologica spinga a banalizzare gli obiettivi formativi o a soffermarsi su aspetti di dettaglio, perdendo di vista la ricchezza concettuale che la programmazione può dispiegare una volta colte le sue molteplici sfaccettature. Una chiara esposizione degli aspetti chiave dei programmi può aiutare insegnanti e altri operatori culturali a identificare le ragioni della centralità del *software* nella società attuale e a orientare al meglio l’azione educativa, affinché la pratica della programmazione dischiuda tutte le sue potenzialità come strumento di consapevolezza e cittadinanza attiva.

1 La programmazione nella scuola

La programmazione degli “elaboratori di informazioni” sta al cuore dell’informatica. In un certo senso è proprio la ragion d’essere della disciplina, il suo nucleo caratterizzante e motivante l’enorme varietà di approcci e sviluppi che ne vivacizzano lo studio e la pratica. Non stupisce quindi che in una società dominata dalle tecnologie dell’informazione, la programmazione si stia diffondendo sempre di più anche nelle attività scolastiche, a tutti i livelli e in tutti i percorsi formativi [12, 3]. In realtà c’è forse da chiedersi come mai solo ora si inizi a interrogarsi seriamente su come introdurre

*Le riflessioni riportate in questo articolo sono frutto del confronto e della discussione maturata all’interno del gruppo di lavoro WG5, nell’ambito dell’International Conference on Innovation and Technology in Computer Science Education (ITiCSE 2022) di cui fanno parte, oltre ai presenti autori, Tim Bell (NZ), Andrej Brodnik (SLO), Andrew Paul Csizmadia (UK), Liesbeth De Mol (FR), Henry Hickman (NZ), Therese Keane (AU).

compiutamente l'informatica e la programmazione nei percorsi di formazione degli insegnanti. La circostanza meriterebbe uno studio specifico, ma noi crediamo che il ritardo nell'instaurare un confronto e un dialogo approfonditi, volti a conciliare le sensibilità di orientamento pedagogico con quelle di orientamento scientifico, abbiano determinato una certa resistenza nel riconoscere il potenziale culturale ed educativo dell'informatica e, appunto, dell'attività di programmazione.

1. La confusione assai diffusa fra *uso* delle applicazioni informatiche e l'impresa *concettuale* di immaginarne l'utilità, progettarle, realizzarle, convalidarne il funzionamento e comprenderne l'impatto. Infatti, si è posta un'enfasi eccessiva sulla necessità di acquisire abilità di utilizzo di applicazioni specifiche, trascurando quasi completamente i principi scientifici che le hanno rese possibili. Un po' come se fosse sufficiente assaggiare molte meringhe con la panna per apprendere quanto la fisica, la chimica, o semplicemente la gastronomia siano cruciali in cucina: da questa enfasi mal riposta nemmeno i "pasticcieri" potrebbero trarre un beneficio significativo.
2. L'equivoco rapporto con discipline strutturate più stabilmente nella nostra tradizione scolastica, in particolare la matematica con la quale l'informatica ha spesso una relazione edipica (dando quasi l'impressione di volerla "superare" uccidendola), mentre sarebbe più proficuo se coltivasse le proprie potenzialità di nuovo strumento di pensiero, complementare e non alternativo ad altri più consolidati.
3. La percezione della programmazione come attività fortemente specialistica, utile solo a chi intende sviluppare una professionalità in settori tecnologici. Questa visione limitata trascura il fatto che l'elaborazione automatica di informazioni ha caratteristiche molto peculiari, ed è capace di influire grandemente sulla nostra consapevolezza del mondo, consapevolezza che è essa stessa basata su una elaborazione di informazioni — per definizione non automatica.

Le distorsioni a cui si è accennato sono emerse per motivi comprensibili, ma hanno avuto e tuttora hanno conseguenze negative. Per questo pensiamo possa essere utile mettere in luce, in una forma auspicabilmente accessibile a tutte le parti coinvolte in progetti educativi, gli aspetti centrali della programmazione, cioè *di che cosa parliamo* quando (noi informatici) sosteniamo che è importante fare l'esperienza della programmazione per capire alcuni aspetti del mondo che ci circonda senza subirne acriticamente l'impatto.

A tal fine abbiamo formato un gruppo di lavoro [23] che si è riunito a partire da marzo 2022 e che comprende studiosi con una lunga esperienza non solo nella didattica dell'informatica, ma anche nella storia e nella filosofia della disciplina, nonché nella diffusione delle tecnologie digitali. Ne è risultato un documento [24], che qui intendiamo riassumere, rivolto principalmente agli insegnanti e agli altri attori coinvolti nelle politiche scolastiche. L'obiettivo è fornire uno strumento che consenta di sviluppare una visione più chiara e articolata della *natura* dei programmi [2] e delle attività connesse alla loro realizzazione, e che inoltre possa fungere da guida ai fini della scelta dei temi più rilevanti per garantire l'efficacia delle azioni educative e pedagogiche.

2 Caratterizzazione dei programmi nella letteratura

A prima vista potrebbe sembrare facile definire *che cos'è un programma*: in molti manuali, per esempio, è introdotto semplicemente come una sequenza di istruzioni per un *computer*. E in effetti termini come *coding*, molto usato anche nei documenti del Ministero dell'Istruzione, trasmettono di fatto questa idea riduttiva [25, 4]. Ma in realtà si tratta di un concetto molto più sfaccettato e ricco, come si può evincere da una lettura attenta della storia dell'informatica e da un'analisi di come l'elaborazione automatica di informazioni descritta dai programmi sia ormai inestricabilmente legata

alla maggior parte delle nostre attività quotidiane, dalla guida delle automobili all'espressione artistica. È ormai difficile, infatti, trovare ambiti rispetto ai quali la commistione con la programmazione non abbia influito in modi anche radicali.

Alla luce della centralità che i programmi e la programmazione assumono nell'informatica, ci si dovrebbe aspettare che esista una definizione chiara e condivisa di che cos'è un programma. In realtà, a partire degli esordi della disciplina, se ne possono trovare svariate. Per esempio, una definizione influente da parte di Grace Hopper apparve nel *First glossary of programming terminology* pubblicato dall'ACM nel 1954 [14], definizione in seguito ripresa in diversi glossari:

> **Programma:** un piano per risolvere un problema. Un programma completo comprende i piani per la trascrizione dei dati, la codifica per il computer e i piani per l'acquisizione dei risultati nel sistema. La lista delle istruzioni codificate è denominata **routine**.

Una caratterizzazione in termini di pianificazione sequenziale di istruzioni codificate, introdotta nell'ambito del progetto ENIAC (uno dei primi elaboratori digitali), ispira anche quelle prevalenti oggi [5]; in particolare, fu adottata in relazione ai primi linguaggi di programmazione di alto-livello come ALGOL:

> Sequenze di istruzioni e dichiarazioni, combinate opportunamente, sono chiamate programmi. [26]

Negli anni '50 e nei primi anni '60 del Novecento i campi di applicazione della programmazione erano ancora piuttosto circoscritti, orientati prevalentemente al calcolo scientifico e aziendale, e l'odierna ubiquità dei programmi era inimmaginabile per i costi dell'hardware, accessibili solo a grandi organizzazioni. Malgrado il modo in cui oggi si sviluppano e si utilizzano i programmi si sia certamente evoluto rispetto agli albori dell'informatica, alcune concezioni che risalgono a quell'epoca restano ancora molto popolari fra gli informatici.

La natura dei programmi assume inoltre un ruolo chiave nel contesto più ampio della discussione riguardo alla natura dell'informatica, a partire da alcuni contributi significativi che sono apparsi negli anni '80 e '90 (per esempio [20, 7, 29]), discussione stimolata in particolare dalla domanda se l'informatica — computer *science* nel mondo anglosassone — sia effettivamente una scienza. Contributi più recenti al riguardo sono riportati in varie fonti, fra cui si possono segnalare [6, 9, 27, 17, 28, 30, 1]. Da questo dibattito emerge che l'informatica non ha un'identità chiara e tende piuttosto a delinearsi come un'area interdisciplinare.

I diversi modi di concepire i programmi e la programmazione spesso ne enfatizzano solo un aspetto particolare. Il progettista di un linguaggio di programmazione, per esempio, tenderà principalmente a guardare i programmi dal punto di vista della notazione e della semantica formale; un informatico teorico spesso li considera come procedure astratte; un ingegnere del software, d'altro canto, sarà più interessato alla complessità dei programmi e alle rispettive relazioni con il mondo reale. Donald Knuth, per esempio, ha inteso i programmi innanzitutto come implementazioni di algoritmi [19], ma è interessante notare che più tardi lo stesso Knuth prefigura una caratterizzazione abbastanza diversa, assimilando i programmi a "opere letterarie", da cui l'approccio che oggi è conosciuto come *literate programming*:

> Invece di immaginare che il nostro compito principale sia di istruire un computer su quello che deve fare, [dovremmo] piuttosto concentrarci [sul compito di] spiegare agli esseri umani quello che vogliamo che il computer faccia. [21]

Edsger Dijkstra, come altri studiosi, si è concentrato sulla natura matematica dei programmi [10]. In polemica con questo punto di vista, il filosofo James Fetzer enfatizzò la distinzione fra programmi come entità matematiche *astratte* e programmi come modelli *causali* in quanto oggetti fisici:

> Da un punto di vista metodologico, si può dire che i programmi sono congetture, mentre le esecuzioni sono tentativi di refutazioni — che hanno fin troppo spesso successo. [13]

Secondo la prospettiva ingegneristica i programmi sono componenti di sistemi più complessi [8], e quindi non sono riducibili alla loro notazione o a una struttura matematica. Per Jackson [16], in particolare, la terna *requisiti – specifica - programma* è centrale al fine di mettere in relazione la macchina e il mondo. Altri autori hanno proposto tassonomie a supporto di una comprensione più diversificata dei programmi. Un esempio ben conosciuto, sempre nell'ambito dell'ingegneria del software, è il lavoro di Lehman sulle leggi alla base dell'evoluzione dei programmi [22], dove i programmi sono classificati in termini di complessità e "intreccio" con il mondo.

Definizioni e classificazioni sono questioni tradizionali anche per la filosfia analitica. Un'idea chiave alla base della caratterizzazione di Colburn [6] fa riferimento al ruolo peculiare dell'astrazione in informatica e nella programmazione. Nel saggio di Turner [30] i programmi sono invece analizzati attraverso la lente della filosofia degli artefatti tecnologici, basata sulla dicotomia struttura/funzione, e del processo di progettazione. Più specificamente, per Turner i programmi sono definiti dalla terna *specifica - programma simbolico - processo fisico*. Eden [11], d'altro canto, discute le contrastanti caratterizzazioni ontologiche, metodologiche ed epistemologiche dei programmi secondo le diverse prospettive ispirate alla matematica, all'ingegneria e alla scienza. Per portare un ulteriore esempio, Irmak [15] confronta invece i programmi con la musica, osservando che, analogamente alla musica, i programmi hanno una natura duale. L'idea della natura duale dei programmi è comunque ricorrente; come osservato da Alan Kay:

> Un messaggio intangibile incorporato in un supporto materiale è l'essenza del software [...]. Il computer è un veicolo da guidare o un saggio da scrivere? Gran parte della confusione deriva dal tentativo di risolvere la questione a questo livello. La natura proteiforme del computer sta nel fatto che può comportarsi come una macchina o come un linguaggio a cui dare forma per trarne beneficio. [18]

Le caratterizzazioni esemplificate mostrano come una certa concezione dei programmi di solito tenda a confermare una particolare visione dell'informatica. Di conseguenza, proprio come per l'informatica in generale, ci ritroviamo di fronte a un ventaglio variegato di definizioni, ciascuna delle quali getta luce su un aspetto diverso. Non deve quindi sorprendere che anche nella didattica dell'informatica si ritrovino diverse prospettive su come i programmi debbano essere insegnati e con quali obiettivi. Forse ciò spiega in parte la proliferazione di termini quali *coding*, *computational thinking* e *programmazione*.

Recentemente la natura poliedrica dei programmi è stata riconosciuta in diversi contesti. Il lavoro collaborativo [2] prende le mosse proprio a partire dall'osservazione che i "programmi" sono oggetti complessi che comprendono molti aspetti: sono reali — hanno effetti sulla nostra vita; sono astratti — elaborano entità astratte; sono concreti — occupano uno spazio nella memoria dei dispositivi digitali e possono essere copiati, trasferiti o corrotti. Il progetto PROGRAMme, che si rivolge a storici, filosofi e studiosi di informatica, propone un'analisi dei programmi in termini delle loro diverse modalità: fisica, socio-tecnica e notazionale (https://programme.hypotheses.org/). Si può quindi concludere che un approccio olistico ai programmi sia il più adatto al fine di garantire una "alfabetizzazione" *generale*, ovvero, con le parole di Alan Kay:

> Indirizzare la magia del mezzo verso il perseguimento delle proprie finalità, anziché contro di esse, significa acquisire l'alfabetizzazione. [18]

3 Le sei "facce" di un programma

Attraverso le discussioni maturate in seno al gruppo di lavoro [23], alla luce delle prospettive maturate negli ambiti filosofico e didattico disciplinare, abbiamo identificato sei aspetti chiave che, a

nostro avviso, sono i più salienti per caratterizzare la natura dei programmi e per spiegarne l'impatto rivoluzionario nella società. Non intendiamo certamente avanzare alcuna pretesa di esaustività: anche la visione d'insieme qui proposta è probabilmente riduttiva da certi punti di vista. Tuttavia, ci sembra sufficiente a motivare l'interesse che la Scuola dovrebbe dedicare all'argomento.

In sintesi, la caratterizzazione della natura dei programmi può essere così schematizzata:

1. I programmi sono *strumenti* utilizzabili;
2. I programmi sono artefatti tecnologici *opera dell'uomo*;
3. I programmi sono *oggetti fisici*;
4. I programmi sono *entità astratte*;
5. I programmi sono *eseguibili automaticamente*;
6. I programmi sono *artefatti linguistico-notazionali*.

Le sei "facce" così introdotte sono sintetizzate iconicamente nella figura 1, che inoltre presenta ai lati opposti di un esagono coppie di aspetti *duali*: essi caratterizzano la natura dei programmi proprio nella tensione dialettica che li contrappone.

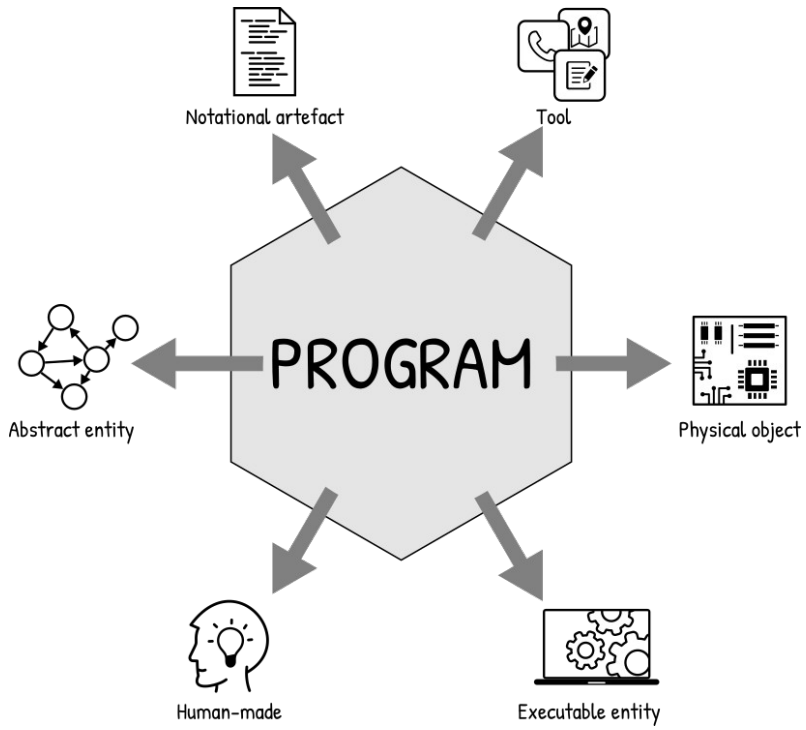

Figura 1: I sei aspetti chiave della natura di un programma
(figura tratta dal documento originale [24])

3.1 Programmi come strumenti

Questo è forse l'aspetto più evidente per molte persone. I programmi sono gli strumenti attraverso i quali svolgiamo moltissime delle nostre attività. In larga misura la società dipende dalla disponibilità di questi strumenti e i programmi ci permettono di affrontare compiti che altrimenti sarebbero al di fuori dalle nostre possibilità individuali, anche amplificando le nostre capacità cognitive. Grazie ai programmi lo stesso dispositivo fisico (p. es. uno *smartphone*) può assolvere molte funzioni diversissime fra loro: dalla contabilità aziendale, all'ascolto della musica, dalla fotografia al montaggio cinematografico, e così via. Tuttavia, è bene tenere presente che uno strumento non è mai neutrale, ma incorpora scelte e valori impliciti: occorre tener conto di questo anche quando la nostra esperienza è mediata da un programma — benché sia facile dimenticarsene a causa dell'ubiquità, della flessibilità e dell'invisibilità degli strumenti *software*.

3.2 Programmi come opera dell'uomo

I programmi sono un'opera umana, realizzati intenzionalmente per soddisfare qualche esigenza umana, con uno spettro molto ampio: si può scrivere un programma per risolvere un problema matematico, o portare a termine una ricerca in una banca dati, ma anche per soddisfare un impulso artistico o semplicemente per… imparare a programmare. In ogni caso si tratta di qualcosa costruito dall'uomo con uno scopo, che non sempre è allineato con gli scopi di chi il programma poi lo usa: chi sviluppa programmi che permettono la formazione di *social network* ha in genere obiettivi piuttosto diversi (p. es. raccogliere informazioni utili per il mercato pubblicitario) rispetto a chi usa tali programmi per condividere informazioni multimediali. La realizzazione di un programma richiede spesso un'attività complessa, svolta cooperativamente da gruppi di lavoro eterogenei che devono rispondere a moltissimi interessi e pulsioni, oltre a rispettare i vincoli tecnologici.

3.3 Programmi come oggetti fisici

I programmi sono anche oggetti fisici, un aspetto che è facile sottovalutare, ma che ha invece una grande rilevanza. Il programma che stiamo usando deve risiedere da qualche parte per poter essere eseguito da una macchina. È potenzialmente soggetto ad alterazioni, volontarie e involontarie, che possono impedirne il funzionamento oppure determinare effetti inaspettati/indesiderati. Inoltre, l'esecuzione di programmi e talvolta anche la semplice conservazione consumano energia.

3.4 Programmi come entità astratte

I programmi sono poi entità astratte: rappresentano manipolazioni di simboli, cioè di segni privi di significato intrinseco, cosicché il significato prende forma esclusivamente nella mente di chi li concepisce o ne prefigura il funzionamento. L'interprete (meccanico/elettronico), nel dare corso all'elaborazione descritta dal programma, non fa altro che operare in accordo con i suoi principi costruttivi, facendo evolvere lo stato dei suoi componenti in maniera prevedibile. L'interesse dell'elaborazione scaturisce dunque dalle proprietà astratte, algoritmiche dei programmi, che occorre imparare ad apprezzare e a mettere in relazione con la realtà concreta, valutandone le semplificazioni implicate e le potenziali alternative, oltre che la correttezza rispetto ai risultati attesi.

3.5 Programmi come entità eseguibili automaticamente

I programmi vengono eseguiti automaticamente, in contesti in cui non è previsto alcun intervento da parte dei progettisti, e ciò ne costituisce una caratteristica fondamentale. In altri termini, il programmatore deve immaginare ciò che accadrà o che potrebbe accadere durante l'esecuzione, ma una volta realizzato, l'esecuzione del programma è fuori dal suo controllo: per cambiare qualcosa

serve produrre una nuova versione. L'elaborazione dei dati di input (manuali o provenienti da dispositivi collegati e sensori) avviene nella maniera cieca e meccanica descritta al punto precedente; qualsiasi valutazione di ragionevolezza o affidabilità è responsabilità di coloro che hanno concorso allo sviluppo del sistema.

3.6 Programmi come artefatti linguistico-notazionali

I programmi sono inoltre uno strumento per comunicare idee e strategie non solo agli interpreti che li eseguono passivamente, ma anche ad altre persone che possono analizzarli, apprezzandone tutti i dettagli grazie alla precisione imposta dalla possibilità dell'esecuzione automatica. La varietà dei linguaggi di programmazione dimostra un'inesauribile ricerca di modalità espressive che si prestino alla modellazione e alla risoluzione dei problemi che si vogliono affrontare. Nello stesso tempo, le notazioni utilizzate per rappresentare l'informazione tendono intrinsecamente a condizionare il modo in cui riusciamo a ragionare per mezzo di esse.

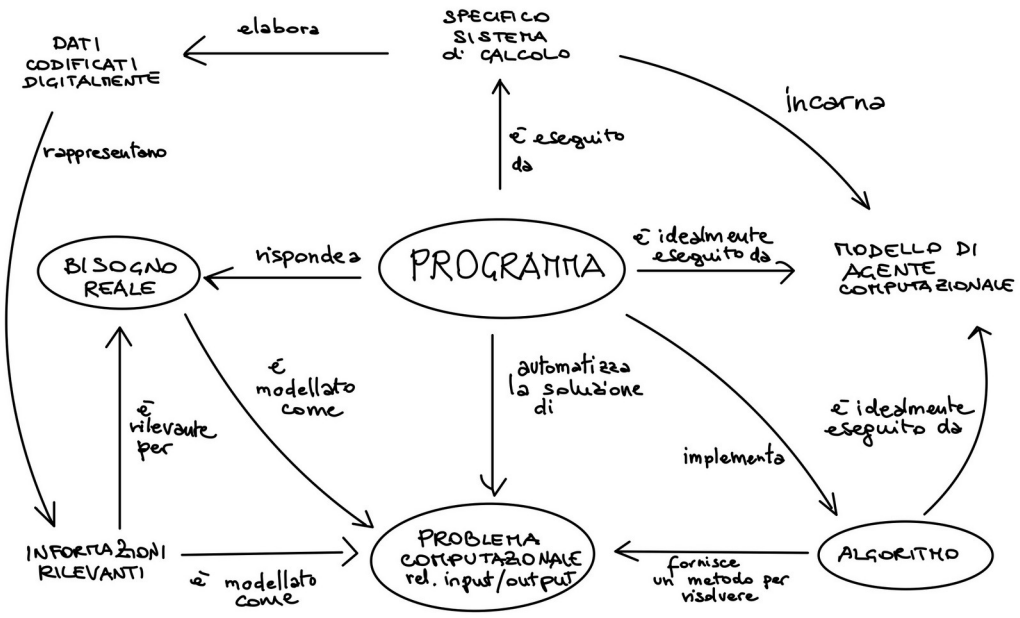

Figura 2: Una mappa concettuale che illustra l'attività della programmazione nel suo complesso (figura tradotta dal documento originale [24])

4 L'impresa concettuale delle elaborazioni automatiche

I programmi si scrivono per esprimere soluzioni automatiche di problemi computazionali e quindi assumono significato in relazione alle varie entità schematizzate nella mappa concettuale illustrata in figura 2. I problemi computazionali sono il corrispettivo in termini di elaborazione dell'informazione (ovvero, nel caso digitale, di manipolazione di rappresentazioni simboliche) di un'esigenza nel mondo

reale. I problemi computazionali modellano i bisogni nel mondo reale rappresentando le informazioni rilevanti con dati opportuni. Un programma può essere eseguito automaticamente — in altre parole un sistema fisico (per esempio un computer o uno smartphone) può dare corso all'elaborazione di informazioni descritta dal programma. Normalmente questi sistemi sono in grado di trattare solo rappresentazioni digitali, cioè simboliche, dei dati. I dati, perciò, devono essere codificati tramite sequenze di simboli tratti da un alfabeto predefinito.

Un algoritmo è l'idea astratta di un metodo (ossia una procedura precisa che può essere portata a termine seguendola in modo *prescrittivo*) per risolvere un problema computazionale. L'algoritmo viene progettato pensandone l'esecuzione da parte di un agente computazionale idealizzato, capace di eseguire un insieme ridotto di azioni primitive. Un sistema di calcolo reale (potrebbe essere un computer o un altro dispositivo hardware, ma anche un altro programma, come l'interprete di un linguaggio di programmazione) è un esemplare specifico di tale agente computazionale idealizzato. Affinché possa essere eseguito automaticamente da un sistema di calcolo reale, un algoritmo deve essere implementato, cioè codificato in un linguaggio di programmazione rispettandone le rigide regole necessarie a evitare ambiguità e adattandolo ai vincoli tecnologici specifici del sistema: solo così un algoritmo si traduce effettivamente in un programma.

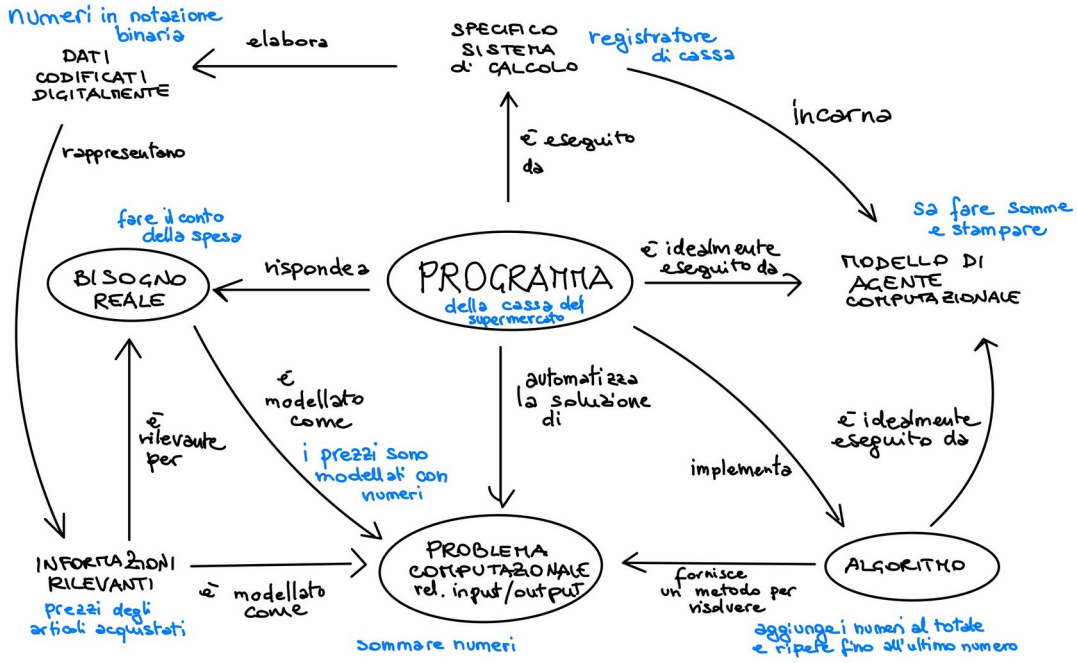

Figura 3: Un esempio: il programma della cassa del supermercato
(figura tradotta dal documento originale [24])

4.1 Un esempio

La figura 3 illustra i concetti che appaiono nella mappa di figura 2 facendo riferimento a un programma (quello della "cassa del supermercato") progettato per gestire la produzione degli scontrini di un negozio. I prezzi degli articoli venduti possono essere modellati come numeri. Perciò,

l'elaborazione di uno scontrino corrisponde al generico problema computazionale di "sommare una serie di numeri". Il programma della cassa del supermercato è eseguito da un registratore di cassa, un dispositivo in grado di acquisire i prezzi degli articoli in vendita (digitati da un operatore o letti interpretando un codice a barre), quindi di stampare gli scontrini corrispondenti. Il dispositivo, in realtà, manipola i numeri rappresentati tramite sequenze di simboli binari — basati cioè su due stati chiaramente distinguibili dei suoi componenti elettronici.

L'algoritmo tipico per sommare una serie di numeri è il seguente:

1. usa una variabile per tener traccia del totale, assegnandovi inizialmente il valore zero;
2. fintantoché la lista dei numeri non è esaurita, ripeti l'istruzione seguente:
   2.1. somma il prossimo numero alla variabile che tiene traccia del totale;
3. stampa il valore della variabile che tiene traccia del totale e termina.

Durante la progettazione di questo algoritmo, assumiamo che l'esecutore sia in grado di ricevere in input qualsiasi numero, sommare due numeri e stampare qualsiasi numero. Affinché il registratore di cassa possa eseguire *fisicamente* queste operazioni occorre tener conto di alcuni vincoli specifici, per esempio i limiti ai valori ed eventualmente alla quantità di numeri oggetto dell'elaborazione. L'algoritmo probabilmente viene implementato in un linguaggio di programmazione specializzato per i registratori di cassa. Sia il problema computazionale che l'algoritmo, però, sono molto generali e possono essere facilmente adattati ad altri scopi analoghi, come per esempio calcolare la popolazione totale di un Paese sulla base delle popolazioni delle regioni che lo costituiscono. Perciò, un'implementazione in un linguaggio di programmazione non specifico permetterebbe di usare lo stesso programma, magari con qualche piccolo aggiustamento, anche in contesti piuttosto diversi.

5 Conclusione

Lo scopo del gruppo di lavoro è fornire una chiara esposizione degli aspetti chiave dei programmi che possa aiutare gli insegnanti e altri operatori culturali a identificare le ragioni della centralità del *software* nella società attuale e a orientare al meglio gli interventi educativi. La versione italiana preliminare del documento è disponibile all'indirizzo https://aladdin.unimi.it/naturadeiprogrammi.pdf. La versione definitiva è attesa per la fine del 2022 e crediamo che possa diventare una risorsa importante ai fini di progettare percorsi formativi in cui la pratica della programmazione e la riflessione sui programmi vengano intesi innanzitutto come strumenti di consapevolezza e cittadinanza attiva.

Riferimenti bibliografici

[1] Tim Bell, Paul Tymann, and Amiram Yehudai. The Big Ideas in Computer Science for K–12 Curricula. Bull. EATCS 124, 11 pages, 2018.

[2] Andrej Brodnik, Andrew Csizmadia, Gerald Futschek, Lidija Kralj, Violetta Lonati, Peter Micheuz, and Mattia Monga. Programming for All: Understanding the Nature of Programs. CoRR abs/2111.04887, 2021, 25 pages. arXiv:2111.04887

[3] Francesca Caena and Christine Redecker. Aligning teacher competence frameworks to 21st century challenges: The case for the European Digital Competence Framework for Educators (Digcompedu). European Journal of Education, 2019, 54. https://doi.org/10.1111/ejed.12345.

[4] Isabella Corradini, Michael Lodi, and Enrico Nardelli. An investigation of italian primary school teachers' view on coding and programming. In Proc. of Informatics in Schools. Fundamentals of Computer

Science and Software Engineering – ISSEP 2018. Springer International Publishing, 2018, pp. 228–243, S.N. Pozdniakov and V. Dagienė (Eds.), Cham, Switzerland.

[5] Liesbeth De Mol and Maarten Bullynck. Roots of 'program' Revisited. Commun. ACM, 64(4), 2021, pp. 35–37. https://doi.org/10.1145/3419406

[6] Timothy Colburn. Philosophy and Computer Science. Routledge (Taylor & Francis), 1999, New York, NY.

[7] Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R Young. Computing as a discipline. Computer, 22(2), 1989, pp. 63–70.

[8] Peter J. Denning. The Field of Programmers Myth. Commun. ACM, 47 (7), 2004, pp. 15–20. https://doi.org/10.1145/1005817.1005836

[9] Peter J. Denning and Craig H. Martell. Great Principles of Computing. The MIT Press, 2015, Cambridge, MA.

[10] Edsger W. Dijkstra. On the cruelty of really teaching computer science. Personal communication, 1988. https://www.cs.utexas.edu/users/EWD/ewd10xx/EWD1036.PDF

[11] Amnon H. Eden. Three Paradigms of Computer Science. Minds Mach, 17 (27), 2007, pp. 135–167. https://doi.org/10.1007/s11023-007-9060-8

[12] Katrina Falkner, Sue Sentance, Rebecca Vivian, Sarah Barksdale, Leonard Busuttil, Elizabeth Cole, Christine Liebe, Francesco Maiorana, Monica M. McGill, and Keith Quille. An international comparison of K–12 computer science education intended and enacted curricula. In Proceedings of the 19th Koli Calling International Conference on Computing Education Research, Koli Calling '19, New York, USA, 2019. Association for Computing Machinery.

[13] James H. Fetzer. Program Verification: The Very Idea. Commun. ACM, 31 (9), 1988, pp. 1048–1063. https://doi.org/10.1145/48529.48530

[14] Grace M. Hopper, chairman. ACM first glossary of programming terminology. Report to the ACM from the Committee on Nomenclature, 1954.

[15] Nurbay Irmak. Software is an Abstract Artifact. Grazer Philosophische Studien, 86(1), 2012, pp. 55–72.

[16] Michael Jackson. The World and the Machine. In Proceedings of the 17th International Conference on Software Engineering – ICSE '95, Seattle, USA, 1995. Association for Computing Machinery, New York, NY, USA, pp. 283–292. https://doi.org/10.1145/225014.225041

[17] Lars-Erik Janlert. The program is the solution — what is the problem? European Conference on Computing and Philosophy, 2006.

[18] Alan Kay. Computer software. Scientific American, 251, 1984, pp. 53–59.

[19] Donald E. Knuth. The Art of Computer Programming (1st Edition) Vol. 1. Addison-Wesley, 1968, Boston, USA.

[20] Donald E. Knuth. Computer Science and its Relation to Mathematics. The American Mathematical Monthly, 81(4). 1974, pp. 323–343. https://doi.org/10.1080/00029890.1974.11993556

[21] Donald E. Knuth. Literate Programming. Comput. J., 27(2), 1984, pp. 97–111. https://doi.org/10.1093/comjnl/27.2.97

[22] Meir M. Lehman. Programs, life cycles, and laws of software evolution. In Proc. IEEE, 68(9), 1980, pp. 1060–1076. https://doi.org/10.1109/PROC.1980.11805

[23] Violetta Lonati, Andrej Brodnik, Tim Bell, Andrew Paul Csizmadia, Liesbeth De Mol, Henry Hickman, Therese Keane, Claudio Mirolo, Mattia Monga, and Matti Tedre. Characterizing the nature of programs for educational purposes. In Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education Vol. 2, ITiCSE '22, pp. 572–573, New York, NY, USA, 2022. Association for Computing Machinery.

[24] Violetta Lonati, Andrej Brodnik, Tim Bell, Andrew Paul Csizmadia, Liesbeth De Mol, Henry Hickman, Therese Keane, Claudio Mirolo, and Mattia Monga. The nature of programs, or: What we talk about when we talk about programs. https://drive.google.com/file/d/1lhVPDhSHu3ivRXvcK7BVEhSmpqz5izow (ultimo accesso, novembre 2022).

[25] Violetta Lonati, Dario Malchiodi, Mattia Monga, and Anna Morpurgo. Is coding the way to go? In Proc. of the 8th International Conference on Informatics in Schools: Situation, Evolution, and Perspective – ISSEP 2015. LNCS, Vol. 9378, pp. 165–174, 2015. Springer International Publishing, A. Brodnik and J. Vahrenhold (Eds.).

[26] Alan J. Perlis and Klaus Samelson. Preliminary report – International algebraic language. Commun. ACM, 1(12), 1958, pp. 8–22. http://www.softwarepreservation.org/projects/ALGOL/report/Algol58_preliminary_report_CACM.pdf

[27] William J. Rapaport. Philosophy of Computer Science: An Introductory Course. Teaching Philosophy, 4(28), 2005, pp. 319–341.

[28] Matti Tedre. The Science of Computing: Shaping a Discipline. CRC Press, 2014, Boca Raton, FL.

[29] Sherry Turkle and Seymour Papert. Epistemological pluralism: Styles and voices within the computer culture. Signs: Journal of women in culture and society, 16(1), 1990, pp. 128–157.

[30] Raymond Turner. Computational Artifacts: Towards a Philosophy of Computer Science. Springer, 2018, Cham, Switzerland.