Insegnare l'informatica a scuola? Come e perché

Giovanni Salmeri

L'informatica è la nuova competenza di base?

Nel 1982 Neil Postman, noto soprattutto per i suoi studi critici sui mezzi di comunicazione di massa, pubblicò un libro dal titolo provocatorio *La scomparsa dell'infanzia*.[1] Le analisi lì sviluppate erano molto ricche e complesse, ma per quel che ci interessa ora possono essere riassunte in poche parole. Il concetto di «minore età» (molto imperfettamente tradotta con «infanzia»), sostiene Postman, è il prodotto dell'affermazione della scrittura e soprattutto della stampa: queste creano infatti una fascia di età che è prevalentemente dedicata ad acquisire la padronanza di quel mezzo, appunto la scrittura, che è in grado di far entrare nella società degli adulti, sia dal punto di vista delle conoscenze (molte informazioni sono mediate dai libri) sia della partecipazione attiva. Si rende quindi necessaria la scuola, un tempo dedicato quasi solo allo studio, che è una delle condizioni maggiori di sviluppo della civiltà. Ma tutto questo è stato radicalmente cambiato dall'avvento della televisione, che ha spodestato la stampa come principale mezzo di comunicazione e non richiede più lo sviluppo di nessuna competenza particolare per essere fruito: nel mondo contemporaneo dunque la «minore età» nel senso moderno sta scomparendo, con tutti i connessi rischi per lo sviluppo della civiltà. C'è una speranza? Postman (qui volevamo arrivare) la vede nell'avvento dei computer: essi infatti sono un nuovo strumento pervasivo, nuovamente (come la scrittura) molto difficile da usare: una nuova «minore età» potrà dunque nascere come il tempo destinato ad imparare ad usare i computer.

Letta a distanza di diversi decenni, questa previsione rischia di apparire incomprensibile (tant'è vero che non è raro vederla completamente fraintesa o dimenticata quando si discute dell'eredità di Postman). Per capirla basta però riandare con la mente agli anni 80 in cui veniva

[1] Neil Postman, *The Disappearance of Childhood*, Dell, New York 1982.

scritta: erano gli anni dell'esplosione dell'informatica personale (il «microcomputer», come si diceva allora, cominciava ad entrare in molte case), e i computer erano effettivamente difficili da usare, anche quando semplici ed economici. Un tipico computer personale dell'epoca era per esempio quello che una volta acceso presentava solo uno schermo nero pronto ad accettare comandi in un qualche dialetto del BASIC. Insomma, usare un computer significava essenzialmente saperlo *programmare*. Le riviste di informatica erano piene di «listati» di programmi da copiare, studiare, modificare. La previsione di Postman era perfettamente sensata: la percezione che l'informatica avrebbe cambiato il mondo era molto diffusa (e giusta), era dunque logico immaginare che saper usare il computer sarebbe diventato indispensabile quanto leggere, scrivere e far di conto e avrebbe assorbito buona parte degli sforzi educativi.

Le cose non sono andate in questo modo. I programmi applicativi (che non richiedono se non marginalmente programmazione) si sono diffusi, le interfacce «a riga di comando» sono state sostituite da quelle grafiche, e i computer sono sempre più apparsi come facili da usare. L'idea comune è che se un computer, o un programma, necessita di uno studio previo per essere usato, allora c'è qualcosa di sbagliato: tutti i film riflettono quest'ideale quando mostrano i protagonisti che si siedono di fronte a qualsiasi sconosciuto computer e non hanno mai alcuna esitazione nell'ottenere ciò che vogliono. Ma, se oggi perfino un bambino di pochissimi anni può usare un tablet, evidentemente non siamo lontani da quest'obiettivo. Come dunque spesso si legge: oggi i computer sono facili da usare, non c'è più bisogno di imparare i comandi del BASIC o del DOS. Giusto? Non proprio. *Mutatis mutandis*, sarebbe come dire: oggi la lingua italiana è molto più facile di un tempo, non c'è più bisogno di imparare a scrivere, ci sono tanti libri già pronti da acquistare in libreria. Il paragone è anzi ancora imperfetto, perché la lettura di un libro presuppone una comprensione almeno istintiva delle regole del linguaggio, mentre manipolare degli oggetti su uno schermo non implica la minima comprensione della logica soggiacente. Il paragone sarebbe semmai migliore tirando in causa la televisione: il computer di oggi si è per così dire avvicinato alla facilità del televisore, non richiede se non pochissime competenze per essere *in qualche modo* utilizzato.

Ma *programmare* un computer e *usare* un computer sono cose differentissime tra loro, tanto quanto girare un film è diverso dal guardarlo, o imparare a suonare il pianoforte è diverso dall'ascoltare Glenn Gould. Che si faccia per esempio credere che le «conoscenze di base dell'informatica» siano quelle assicurate da una cosiddetta «patente europea», che peraltro è sfrenata pubblicità gratuita ai programmi applicativi di un'azienda americana, è una truffa culturale scandalosa.

Sarebbe però sbagliato vedere nella messa tra parentesi della programmazione l'effetto di una qualche congiura per istupidire l'umanità. Anzitutto, bisogna ovviamente dire che rendere più facile l'uso del computer è un'impresa intellettualmente difficile e nobilissima, che merita di essere sempre meglio perseguita. Un computer «facile» è nient'altro che un computer più vicino alle esigenze umane, del quale è più facile costruirsi un modello mentale. Il computer, del resto, se serve a qualcosa serve appunto a facilitare e rendere più veloce l'attività umana. In secondo luogo, la complessità dei programmi odierni (resa possibile dalla sempre maggiore velocità e capacità di memoria dei computer) rende molto meno visibile quel collegamento tra programmazione e uso che qualche decennio fa era invece ovvio. In altre parole, se un tempo un modesto studio di un linguaggio di programmazione consentiva a chiunque di usare nel pieno delle sue possibilità un computer, oggi anche programmare un'apparentemente semplice interfaccia grafica è un compito notevolmente difficile. Combiniamo le due cose, e non sorprende che il computer venga percepito sempre più come una scatola nera, abitata da meccanismi più o meno magici. Che si possa essere parte attiva nel funzionamento di quella macchina appare sempre più remoto, e perfino i corsi di laurea in informatica fanno fatica ad attrarre proprio quei giovani che fin da piccoli vivono in simbiosi con i computer.

Buoni motivi per insegnare informatica

Tutto ciò non sorprende, certo. Ma non cancella affatto la domanda sull'opportunità di fare un passo indietro, forse con uno sforzo maggiore rispetto a qualche decennio fa, e insegnare informatica anche al di fuori degli indirizzi scolastici che esplicitamente la prevedono come

materia. Il problema non è ovviamente quello della vita quotidiana (quanti sono i contenuti culturali della scuola che «servono» nella vita quotidiana? probabilmente pochissimi). Non è neppure l'opportunità per l'ingresso nel mondo del lavoro (la funzione della scuola, e anche dell'Università, è diversa rispetto alla preparazione immediata ad una professione, dove peraltro le competenze specifiche richieste possono evolvere molto rapidamente con il passare degli anni). Si tratta piuttosto di individuare alcuni dei valori culturali che sono inclusi nell'apprendimento dell'informatica e che possono essere formativi indipendentemente da un loro uso immediato in questo o quel campo. Ci pare che si possano dare almeno tre risposte diverse e complementari.

Una decina di anni fa un breve articolo di Jeannette M. Wing rese popolare l'espressione «pensiero computazionale»: con esso l'autrice intendeva sostenere che le capacità che vengono sviluppate dallo studio dell'informatica sono in realtà rilevanti per ogni attività umana.[2] In esso infatti viene sviluppata anzitutto uno stile nell'affrontare i problemi, prima che la loro applicazione pratica ai computer (forse è per questo motivo che la Wing sostiene che tale stile di pensiero è «più» che saper programmare, anche se tutte le indicazioni più specifiche che porta fanno inconfondibilmente riferimento ad aspetti dalla programmazione). Con questa tesi di fondo è possibile certo essere d'accordo. Ma essa può essere sostenuta ancora di più osservando che non è nuova: tutti gli elementi del «pensiero computazionale» in un modo o nell'altro sono stati individuati e articolati indipendentemente dall'esistenza dei computer. Facciamo solo un esempio. Si ritiene comunemente che il *Discorso sul Metodo* di Descartes rappresenti l'atto di nascita della filosofia moderna. Al suo cuore vi è l'enunciazione di quattro regole che dovrebbero condurre *ogni* ragionamento. È interessante che, eccettuata la prima che ha un significato chiaramente gnoseologico, le altre praticamente coincidono con alcuni principi di quella che dagli anni 60 viene chiamata «programmazione strutturata» e che è considerata come uno dei punti di non ritorno della storia dell'informatica: «dividere ogni problema in tante parti quante fosse possibile e richiesto per

[2] Jeannette M. Wing, «Computational Thinking», *Communications of the ACM*, vol. 49, n. 3 (marzo 2006), pp. 33-35.

risolverlo più agevolmente», «condurre ordinatamente i miei pensieri cominciando dalle cose più semplici e più facili a conoscersi», «fare in tutti i casi enumerazioni tanto perfette e tanto complete da essere sicuro di non omettere nulla». Ovviamente notare ciò non significa vedere né in Descartes un «anticipatore», né nell'informatica una «ripresa», ma semplicemente notare che questa si pone implicitamente, con le sue caratteristiche proprie, in una prestigiosa tradizione di pensiero rigoroso. Ovviamente non è assicurato che la capacità di analizzare un problema informatico immediatamente si ripercuota in altri campi (questa permeabilità è uno dei classici problemi della pedagogia): ma il meno che si possa dire è che tutto ciò che incoraggia il pensiero ordinato è bene, così come è bene imparare a scomporre ogni problema nei suoi costituenti elementari.

Oltre a quest'aspetto generale ne esiste ovviamente uno più specifico. Uno degli scopi fondamentali dell'educazione è far *comprendere* le cose. Se l'informatica è così pervasiva nel mondo di oggi, il compito di farla comprendere è essenziale, benché appunto più difficile nel momento in cui fin da bambini si è abituati ai computer come macchine da usare. Ciò è particolarmente urgente nei confronti di un campo scientifico e tecnico che ha raggiunto un livello di complessità mai prima raggiunto da nessun'altra produzione umana e che per la prima volta nella storia rischia di erodere la coscienza della specificità umana. Molti termini antropomorfici vengono usati evidentemente per comodità o pigrizia (il computer che «pensa», il computer che ha un «linguaggio», il computer che «segue una logica»): ma quando in interviste rilasciate da chi dovrebbe pur avere una precisa cognizione di causa si afferma che un computer «fra trent'anni sarà intelligente come noi» si sta dicendo una sciocchezza o riguardo all'informatica, o riguardo all'antropologia, o più probabilmente riguardo ad entrambe. Il che non toglie che il problema del rapporto dell'intelligenza umana con il computer sia serissimo, se non altro perché in forma seminale è stato posto sia da Alan Turing sia da Janós Neumann, due dei padri fondatori dell'informatica contemporanea: ma si tratta appunto di un problema da comprendere e studiare. Insomma, dalle disastrose ingenuità salva soltanto la conoscenza. E in generale, è difficile immaginare una maniera profonda di comprendere l'informatica se non studiare in qualche mo-

do, a qualche livello, la programmazione, che permette di sperimentare come la comprensione della soluzione di un problema all'adatto livello di astrazione sia appunto un processo tipicamente umano.

Da qui un terzo elemento: comprendere l'informatica, seppure ad un livello elementare, significa spostarsi almeno idealmente dalla posizione dell'utente a quella del creatore. La domanda giusta non è qui *quanti* studenti diventeranno in futuro informatici di professione (benché ovviamente la questione abbia la sua rilevanza: non dare una minima esposizione ad un campo della cultura significa inevitabilmente condannare molti potenziali talenti a spegnersi). Si insegna a scrivere pur sapendo che pochi diventeranno scrittori di professione, o si insegna a far musica o sport pur sapendo che pochi diventeranno musicisti o atleti di professione. Lo scopo dell'educazione non è (se non in maniera molto subordinata) selezionare «eccellenze», ma piuttosto coltivare tutte le facoltà in modo che ciascun essere umano possa essere alla fine più compiuto e più spiritualmente ricco. Essere capaci di concepire e scrivere un semplice programmino informatico è un elemento di realizzazione umana tanto quanto lo è comporre una bella pagina, risolvere un problema di matematica, o fare canestro. E questo pare un aspetto tanto più importante quanto più la diffusione dell'informatica è congiunta ad una comprensibile e fortissima spinta commerciale, in cui il ruolo umano viene presentato come quello dell'acquirente e del consumatore. Se non è la scuola che anzitutto si oppone a tutto questo, è difficile immaginare chi lo possa fare.

A che cosa serve il computer nell'insegnamento

Una volta stabiliti alcuni perché, si tratta di porsi il problema della maniera in cui insegnare l'informatica. Da decine d'anni sono molti gli studi e le esperienze al riguardo, ma a volte ciò che sembra carente è una riflessione che congiunga gli aspetti pedagogici a quelli più propriamente tecnici, che certamente sono qui coinvolti in una forma che spesso sfida le normali competenze di chi si occupa d'insegnamento. Tentiamo qui di formulare solo alcune osservazioni preliminari. La prima e decisiva: di per sé l'insegnamento della programmazione non implica l'uso effettivo del computer. Da questo punto di vista *programma-*

re e *usare* un computer sono cose ancor più differenti che imparare a suonare il pianoforte e ascoltare Glenn Gould. Programmare infatti non significa altro che descrivere un procedimento (nel lessico matematico si direbbe: un algoritmo) usando un codice simbolico univocamente definito. Che poi esista una macchina in grado di interpretare questo codice simbolico (e dunque, tipicamente, di eseguirlo ad una velocità incomparabilmente superiore a quella possibile ad un essere umano) cambia tutto dal punto di vista pratico, ma nulla dal punto di vista teorico. È questa un'esagerazione ipotetica? Assolutamente no. Edsger Dijkstra, il teorico della programmazione strutturata, impartiva i suoi corsi senza mai far toccare un computer, non toccandolo neppure lui, e sottolineando che la correttezza di un programma andava sempre *dimostrata* con carta e penna, come si fa con un teorema matematico.[3] Il linguaggio ALGOL 60, che ha avuto un'influenza enorme nello sviluppo dell'informatica, venne descritto e definito *prima* di essere realizzato su qualsiasi computer, anzi con il sospetto che le macchine del tempo non fossero potenti a sufficienza per supportarlo. Alcuni anni più tardi venne descritto un altro linguaggio che ebbe enormi ripercussioni: ISWIM, che non venne *mai* realizzato su nessun computer. Ciò non toglie, appunto, che fosse possibile scrivere programmi perfettamente corretti e dunque «funzionanti» in esso.

Tutto ciò non serve solo a relativizzare l'importanza del «computer in classe» (finalmente una buona notizia per il Ministero delle Finanze: l'informatica si può insegnare a costo zero!). Serve anche a collocare l'informatica nel suo corretto alveo disciplinare, che è quello della matematica e della logica. Quando si osserva (giustamente) che in genere i programmi più semplici da scrivere sono appunto quelli matematici, non si fa altro che notare che in questo campo si è già così abituati a ragionare in termini algoritmici, che spesso la traduzione in un programma informatico non necessita altro che la riscrittura in un codice

---

[3] *E. W. Dijkstra Archive. The manuscripts of Edsger W. Dijkstra. 1930–2002*, http://www.cs.utexas.edu/users/EWD/. Le idee qui richiamate si trovano diverse volte nei suoi manoscritti, per esempio in «On the cruelty of really teaching computing science», EWD 1036. Si noti la dizione «computing science», che Dijkstra indica come tipica della tradizione europea e preferisce alla più comune «computer science».

simbolico più o meno simile a quello abituale in matematica. Notiamo che ciò non ha però nulla a che fare con l'uso di programmi che eventualmente facilitino l'apprendimento della matematica: questo è un problema di didattica della matematica in senso stretto. Piuttosto bisognerebbe dire che vi sono campi della logica e della matematica contigui all'informatica e che forse oggi potrebbero almeno affacciarsi nel bagaglio culturale di qualsiasi ordine di scuola.

Una volta detto che i computer non sono *necessari* per insegnare l'informatica, bisogna dunque comprendere perché e in qual modo possano essere utili. La domanda è paradossalmente complessa. La risposta più evidente è che un computer permette di provare che il programma che si è scritto è «giusto»: ciò che spesso nei libri di matematica è offerto sotto forma di risultato degli esercizi, in questo caso sarebbe invece essenzialmente parte dello strumento stesso. Tale risposta è però purtroppo anche sbagliata, almeno per due motivi. Il primo è che mettere alla prova un programma può soltanto mostrare che ci sono errori, mai invece che essi non ci sono (i cosiddetti *bachi* sono il più delle volte errori che compaiono solo in condizioni che non sono state sperimentate: e le condizioni sono potenzialmente infinite): è per questo che un programma scritto su un foglio di carta mentre si viaggia in metropolitana rischia di essere più corretto di uno scritto e provato al computer, e precipitosamente ritenuto universalmente corretto la prima volta che offre un risultato giusto. Il secondo motivo è che della scrittura di un programma fanno parte *essenzialmente* anche aspetti come la chiarezza, la modularità, la documentazione, l'eleganza, che evidentemente non hanno influenza immediata sul corretto funzionamento. Eliminata dunque una risposta netta e sbagliata, quelle giuste sono più sfumate, ma non meno importanti. La prima: il computer costringe ad affrontare problemi che restano sullo sfondo finché la programmazione viene condotta come un'operazione puramente logico-matematica, e che ciononostante hanno fondamento matematico: per esempio l'efficienza (quale algoritmo è più veloce?). La seconda: il computer svolge *in parte* una funzione autocorrettiva (nel senso per esempio del materiale montessoriano). È vero che esso non può garantire la correttezza, però in molti casi può segnalare un errore, sia esso sintattico o logico. Il computer svolge quindi una funzione di aiuto

nell'apprendimento, come anche di rinforzo e motivazione: *vedere* funzionare un programma dà senza dubbio soddisfazione.

Considerazioni di questo tipo dovrebbero far avanzare seri dubbi sull'aggiunta di aspetti motivazionali estrinseci nello studio dell'informatica. Talvolta per esempio si sostiene che per un bambino il contesto migliore di apprendimento è quello della creazione di «videogiochi» (o per le ragazze l'elaborazione di «storie»). Può essere che i dati empirici dimostrino che in queste condizioni si impari più rapidamente. Ma rimane il fatto che così facendo le motivazioni dell'apprendimento sono poste in elementi estranei, o almeno del tutto collaterali, rispetto alla natura dell'informatica, e questo non può che lasciare perplessi. Chi mai si chiederebbe quale motivazione aggiungere, per esempio, per indurre allo studio di uno strumento? non è forse la bellezza della musica un motivo sufficiente? Aggiungere motivazioni estrinseche sembra anzi controproducente, perché ha l'inevitabile effetto collaterale di mettere in ombra quelle intrinseche, mostrandole come poco degne di attenzione. Far comprendere che un programma informatico possa essere bello e attraente non solo nel suo risultato, ma nella sua scrittura, è evidentemente difficile: ma qui ci pare che sta o cade uno dei principali scopi dell'educazione.

Quale linguaggio di programmazione

Le numerose proposte ed esperienze riguardo al «primo linguaggio di programmazione» adatto all'insegnamento dànno invece l'impressione che il problema sia stato spesso sopravvalutato. I linguaggi di programmazione concepiti esplicitamente per questo scopo sembrano aver avuto in generale vita difficile. L'elenco di quelli praticamente abbandonati dopo una pur accurata progettazione è lungo (per citarne solo alcuni: GRAIL, Phrogram, Blue, Picky, Elan). I motivi sono stati diversi, ma in generale possono riassumersi proprio nel fatto che non hanno mostrato dal punto di vista didattico vantaggi così evidenti rispetto ai comuni linguaggi di programmazione da compensare il loro difetto di non essere, appunto, reali e di risultare quindi nettamente più limitati dall'uno o l'altro punto di vista. Le eccezioni sono importanti ma apparenti. C'è per esempio il caso del BASIC, che ha avuto un

successo enorme (seppure spesso in forme alquanto rozze e deprecate dagli inventori János Kémeny e Thomas E. Kurtz): ma si tratta di un linguaggio nato non direttamente per scopi didattici, ma piuttosto per rendere accessibile a tutti la programmazione: ed è in questa veste che infatti si è diffuso e dagli anni 70 ha goduto di un successo travolgente. C'è subito dopo il caso del Pascal, creato questa volta esplicitamente per motivi didattici: esso però da una parte si inseriva in una tradizione consolidata (grosso modo può essere definito come un ALGOL semplificato), dall'altra si salvò dalla marginalizzazione solo grazie alla comparsa nel 1983 del Turbo Pascal, un efficientissimo compilatore e ambiente di sviluppo che lo rese un linguaggio reale e non più solo didattico (e infatti i linguaggi successivamente ideati dallo stesso creatore del Pascal, Niklaus Wirth, senza una spinta simile hanno avuto una diffusione limitatissima). C'è l'anomalo caso del Logo, un raffinato linguaggio elaborato in connessione con le teorie di Jean Piaget, che ha avuto sì grande eco nel campo educativo, ma quasi esclusivamente per il suo aspetto più superficiale della «grafica della tartaruga» (ciò che quindi difficilmente può essere ritenuto un successo). Insomma: si deve sospettare che cercare di ideare il «miglior linguaggio educativo possibile» sia un vicolo cieco, tanto più che molte delle caratteristiche che dovrebbero far parte di questo modello ideale sono già alla base di diversi linguaggi di programmazione comuni.

Merita un'analisi anche l'idea secondo cui un'utile propedeutica all'apprendimento di un linguaggio di programmazione sarebbe la scrittura dei cosiddetti «diagrammi di flusso». Il fatto è che essi, per quanto molto usati nei primi anni dell'informatica (furono perfino oggetto di uno dei primi standard ECMA nel 1964 e poi nel 1966), hanno svolto un ruolo sempre più marginale e sono stati poi innumerevoli volte deprecati in quanto incapaci di rappresentare proprio le forme essenziali della programmazione strutturata (per tacere di quella funzionale o ad oggetti). Bisognerebbe dunque ideare diagrammi di flusso adattati ai principi moderni della programmazione? In effetti, nel 1973 due giovani informatici, Ben Shneidermann e Isaac Nassi, proposero a questo scopo diagrammi alternativi a quelli tradizionali, che hanno goduto di

un discreto successo fino ai nostri giorni.[4] Quattro anni più tardi lo stesso Shneidermann contribuì come prima firma ad uno studio sperimentale finalizzato ad evidenziare i vantaggi dei diagrammi di flusso, inclusi quelli da lui stesso coideati, nell'attività di programmazione.[5] Il problema è che gli autori, sconsolati, dovettero ammettere che di vantaggio non ne non avevano visto neppure uno, semmai il contrario. Si può dunque ragionevolmente ritenere che il problema del-l'utilità didattica dei diagrammi di flusso sia stato definitivamente risolto e che una loro certa sopravvivenza sia dovuta solo all'attrazione esercitata dalle immagini, a cui però in questo caso non corrisponde alcun reale vantaggio. Non è in fondo una novità che il linguaggio, anche se artificiale, ha nel suo genere una chiarezza e una ricchezza irraggiungibile, e l'amore anche dei bambini per esso non va sottovalutato.

Ma, eliminate le due strade del linguaggio «didattico» e dei diagrammi di flusso, quale tra i linguaggi reali scegliere per l'insegnamento? È oggi comune sostenere che la scelta è tutto sommato indifferente.

L'affermazione è corretta quando viene posta nella giusta prospettiva: la maggior parte dei linguaggi *oggi* più utilizzati (C, Perl, Python, PHP, Lua, Delphi, VisualBasic) fanno parte in maniera più o meno stretta della famiglia ALGOL. Fanno eccezione solo i linguaggi funzionali (Scheme, Haskell, OCaml) e quelli orientati primariamente alla programmazione ad oggetti (C++, Java, JavaScript, Ruby), che meriterebbero entrambi un discorso a parte. Ciò significa che riguardo a semplici algoritmi le differenze tra i più comuni linguaggi sono solo superficiali e passare dall'uno all'altro richiede solo un mutamento di sintassi. Sicuramente insomma esistono buoni motivi per preferire l'uno all'altro nel contesto educativo, ma essi non sono decisivi come all'epoca in cui si potevano accusare alcuni linguaggi di indurre cattive abitudini poi difficili da abbandonare. Per riprendere l'espressione prima introdotta, tutti quelli oggi più diffusi costringono alla buona prassi della «pro-

---

[4] Isaac Nassi, Ben Shneidermann, «Flowchart techniques for structured programming», *SIGPLAN Notices*, agosto 1973, pp. 12-26.

[5] Ben Shneidermann et al., «Experimental Investigations of the Utility of Detailed Flowcharts in Programming», *Communications of the ACM*, vol. 20, n. 6 (giugno 1977), pp. 373-381.

grammazione strutturata» e potrebbero essere guardati con sufficiente simpatia da Descartes.

I problemi aperti sembrano piuttosto di altra natura. Un primo decisivo problema venne bene messo in evidenza da un celebre articolo polemico, scritto nel 2006 dallo scrittore di fantascienza David Brin.[6] In esso egli raccontava come, per far imparare a suo figlio con soddisfazione i fondamenti della programmazione, fosse stato costretto alla fine a comprare di seconda mano un vecchissimo computer di un quarto di secolo prima: solo con esso era possibile programmare nel semplicissimo BASIC, percependo chiaramente il legame con il funzionamento reale della macchina e senza essere travolti da una quantità esorbitante di funzioni: è solo partendo da quella semplicità, argomentava, che si può incoraggiare il pensiero creativo e la competenza tecnica sui computer. Benché le argomentazioni lì presentate avessero numerose sbavature, il punto sollevato era rilevante: per imparare qualcosa bisogna iniziare da qualcosa di semplice, che sia possibile dominare.
L'ambiente operativo del BASIC degli anni 70 e 80 (o, analogamente, delle prime versioni del Turbo Pascal) era da questo punto di vista esemplare. Bisogna costatare che oggi è molto difficile trovare soluzioni che tengano conto del progresso dell'informatica (un ritorno al BASIC da questo punto di vista non pare ideale), ma contemporaneamente mantengano la semplicità maggiore possibile. La cosiddetta *featuritis*, cioè la moltiplicazione smodata delle funzioni dei programmi informatici, è del resto un problema che non riguarda solo il campo dell'educazione.

Un secondo problema che pare irrisolto è connesso alla cosiddetta «programmazione letterata», il paradigma presentato da Donald Knuth nel 1984 e basato su un cambiamento di prospettiva: la programmazione non dovrebbe più consistere nel dire ad un computer che cosa debba fare, ma nello spiegare ad una persona che cosa si voglia che un computer faccia.[7] Il programma, insomma, diventa un'opera letteraria.

---

[6] David Brin, «Why Johnny can't code», *Salon*, 14 settembre 2006, http://www.salon.com/2006/09/14/basic_2/.

[7] Donald Knuth, «Literate Programming», *The Computer Journal. British Computer Society*, vol. 27, n. 2 (1984), pp. 97–111.

Sarebbe lungo entrare nei dettagli tecnici di come ciò sia possibile: il problema è che questi dettagli sono molti e decisamente complessi: scrivere un programma in programmazione letterata significa utilizzare *contemporaneamente*, incrociandoli senza errori, tre diversi linguaggi di programmazione o di marcatura, e poi seguire un procedimento complesso per l'esecuzione del programma risultante e la generazione della documentazione. Knuth stesso ammetteva che ciò che stava proponendo era destinato solo a pochi informatici di professione. La situazione è solo marginalmente migliorata nella maggior parte dei sistemi consimili seguiti. Ciò condurrebbe ad archiviare senza rimpianti l'ipotesi di considerare la programmazione letterata un punto di partenza nell'insegnamento, se non fosse che i vantaggi educativi di questo approccio in sé appaiono immensi: qui infatti nella programmazione viene incorporata la riflessione su ciò che si sta facendo e lo sforzo di spiegarlo chiaramente. Il già citato Edsger Dijkstra sosteneva che la prima qualità di un buon programmatore è un perfetto dominio della propria lingua materna. In questo modo insomma l'informatica si mostra a pieno titolo una disciplina ponte tra quelle scientifiche, quelle tecniche e quelle letterarie. Appare quindi una singolare mancanza che non vi siano stati ancora sforzi per rendere la programmazione letterata semplice e pedagogicamente accessibile. Questa è una possibile sfida per il futuro.

Torniamo a Postman. Dopo che abbiamo attraversato alcuni dei punti critici dell'insegnamento dell'informatica, la sua idea appare in realtà meno bizzarra. È vero che le cose sono andate diversamente, che i computer non sono restati oggetti difficili: ma è anche vero che questo è avvenuto solo perché la loro difficoltà è stata nascosta. E tuttavia il problema della comprensione di ciò che avviene in questo mondo sempre più percorso dall'informatica è inalterato, anzi è aumentato. Insegnare informatica nella scuola non è certo indispensabile, vi sono tanti modi per favorire una cultura e un'intelligenza equilibrata. Sarebbe però miope non percepire l'importanza della sfida: in gioco vi sono aspetti essenziali di comprensione della nostra civiltà e di capacità di elaborazione critica e creativa. Qualcosa di analogo, appunto, a ciò che è avvenuto con la scrittura e la stampa. Le difficoltà sembrano però diversamente orientate rispetto a questo caso paradigmatico: la scrittura

era tutto sommato facile da insegnare (innumerevoli maestri e maestre lo hanno fatto senza problemi), difficile da imparare (apprendere a scrivere bene e a leggere capendo tutto non è per niente facile e richiede lunghi anni), ma esisteva una fortissima motivazione sociale a farlo. Nel caso dell'informatica anzitutto sembra difficile insegnare e la motivazione sociale è molto più tenue, perlomeno finché essa viene orientata da motivi commerciali: in fondo è sufficiente che il bravo utente clicchi, non serve che egli comprenda pure. Proprio per questo la scuola ha un'occasione preziosa per andare controcorrente.