TEMA

Perché fare coding con le materie umanistiche

Stefano Penge

stefano.penge@gmail.com

Punto di partenza

Le riflessioni presentate in questo articolo sono partite circa trent'anni fa, in una scuola parificata di Roma, elementare e media. Mi era stato proposto di tenere un corso di "informatica" - che si intuiva sarebbe stata la materia del futuro - con una certa libertà sui contenuti; libertà dovuta semplicemente al fatto che non c'era un curriculum di informatica per la scuola dell'obbligo. Avevo declinato il corso in termini di programmazione, ovvero di quello che si chiama oggi "coding", introducendo il Logo dalla terza elementare e il BASIC per i più grandicelli delle medie. L'obiettivo di fine anno era quello di realizzare un videogioco; ma strada facendo avevamo toccato la geometria, la matematica, la logica e un po' di fisica. Confesso che non mi era del tutto chiaro quale fosse l'obiettivo ultimo: se l'apprendimento dello strumento oppure quello del contenuto disciplinare. Il mio punto di partenza era la lettura di Seymour Papert, che parlava di ambienti di esplorazione digitale, di ricerca di gruppo, di costruzione di modelli concettuali del mondo; ma per l'applicazione pratica di queste idee non avevo una ricetta precisa. Insomma, ho improvvisato.

Il corso andò bene: contenti tutti, dai ragazzi ai genitori, e persino il preside. Ma fu davvero utile? Non sono in grado di dirlo, perché ho perso i contatti con quei ragazzi - oggi quarantenni - e non so dire se quelle lezioni oltre che divertenti siano state davvero utili.

Comunque, da quella prima esperienza è nata la scelta di occuparmi professionalmente di linguaggi e ambienti digitali educativi, insomma dell'uso dei computer per l'apprendimento; ho passato tanti anni a scrivere software didattico e altri ancora a progettare ambienti di apprendimento digitale.

Tre anni fa, sull'onda delle discussioni nate intorno ai termini "coding", "pensiero computazionale" e "linguaggi visuali", ho ripensato a quell'esperienza temeraria. Cosa ho imparato? Cosa è cambiato da allora nel mondo? Quella che segue è una sintesi di queste riflessioni e delle proposte che mi sento di fare oggi ai docenti.

Ambienti e linguaggi per il coding

La prima riflessione in ordine di tempo – ma non di importanza - riguarda gli strumenti. Quale linguaggio va usato oggi per fare coding in classe?

Se nel 1980 il sole intorno a cui tutto girava era il Logo LCSI, un po' alla volta si è capito che in realtà si trattava di una stella composita, formata da tanti piccoli astri secondari (Logo Writer, StarLogo, NetLogo). E poi il firmamento si è arricchito da tanti altri fenomeni celesti: dalle meteore (ToonTalk) alle costellazioni (Squeak - Scratch – Snap!) fino alle nebulose per programmare smartphone (AppInventor & co.), magari direttamente dallo smartphone (PocketCode).

Alcuni ambienti sono testuali, altri visuali, altri misti; alcuni seguono un paradigma imperativo, altri funzionale o ad oggetti. Ce ne sono alcuni adatti solo ai piccoli e altri davvero potenti, che si possono estendere a piacere, o che poggiano su linguaggi "da grandi". Perché fermarsi al più diffuso? Sarebbe come studiare solo la letteratura inglese perché l'Inglese è la lingua più parlata.

In un piccolo ebook[1] nato in occasione di una serie di seminari in Piemonte ho provato a raccontare la storia dei linguaggi e degli ambienti di programmazione educativi, con lo scopo di ricostruire il contesto in cui sono nati e, chissà, far venire voglia ai docenti di sperimentare in classe qualcosa di nuovo.

Io stesso, dovendo scegliere un ambiente di coding per costruire delle attività didattiche da proporre alle scuole[2] ho esitato a lungo, e alla fine ne ho scelti *tre*: **LibreLogo**, una versione di Logo che vive nascosta tra le pagine di Libre Office Writer grazie al lavoro di un programmatore ungherese, Laszlo Nemeth; **Prolog**, il linguaggio francese dell'intelligenza artificiale e della programmazione logica; e **Kojo**, un ambiente didattico progettato da un professore di matematica e basato sul linguaggio svizzero Scala che comprende la geometria della tartaruga, GeoGebra e tanto altro.

Questi tre ambienti sono proposti perché l'insegnante possa scegliere quello più adatto al suo contesto, all'età dei ragazzi, alla loro (e sua) esperienza pregressa. Ma li ho proposti anche per mostrare le somiglianze e le differenze tra paradigmi e linguaggi. Non è vero che tutti i linguaggi sono equivalenti, così come non è vero che le lingue naturali sono tutte uguali: altrimenti non sarebbe così difficile tradurre una poesia...

Perché i linguaggi di programmazione (oggi ce ne sono almeno 8.000)[3] sono un intero universo: hanno dietro filosofie diverse, contesti storici e geografici, lingue naturali a cui sono ispirati, e soprattutto persone che li hanno inventati. Parlare di Logo significa

---

1 *Dietro il Coding*, http://steve.lynxlab.com/wp-content/uploads/2016/09/dietro_il_coding-1.pdf

2 Il risultato di questo lavoro è nel libro *Lingua, Coding e Creatività. Fare coding con le materie umanistiche*, Anicia, 2018. Trovate altre informazioni e risorse libere nel companion site del libro: http://www.anicialab.it/coding/lingua/

3 https://en.wikipedia.org/wiki/List_of_programming_languages ne raccoglie alcuni. Ma se volete navigare nella galassia dei Linguaggi, andate qui https://exploring-data.com/vis/programming-languages-influence-network/

parlare di Seymour Papert, di Wally Feurzeig e di Cynthia Solomon; Prolog porta con sé Alain Colmerauer e Philip Roussel; Kojo e Scala permettono di raccontare di Lalit Pant e di Martin Odersky. Persone, non bit. Persone con una vita, con i loro sogni, progetti, teorie, discussioni. E questo non andrebbe mai dimenticato.

Perché vale la pena scrivere codice sorgente

La seconda riflessione è relativa allo scopo. Spesso, se non sempre, si trovano associati *coding* e *pensiero computazionale*. Senza voler entrare qui nel merito delle diverse versioni (da quella storica di Papert a quella relativamente recente di Wing), questa associazione mette il coding al servizio di un modo di affrontare i problemi rigoroso ed efficace. Si fa coding, si dice, per imparare a risolvere problemi, magari anche senza computer. Questa caratteristica era stata associata in passato ad altre discipline, dalla matematica al latino, dalla filosofia agli scacchi; di qui le discussioni sulla *specificità* del coding.

Eppure sembra evidente che scrivere programmi a scuola può avere tanti altri obiettivi. Si possono creare piccoli strumenti utili al proprio autore: ad esempio, una app che cerchi le parole straniere su un dizionario. Oppure si possono scrivere (e leggere) programmi per diventare più consapevoli di cosa possono davvero fare i computer e gli smartphone con i nostri dati.[4]

Oppure ancora, ed è questa la cosa che mi interessa di più qui, si può creare un software *per fare didattica in maniera più efficace.*

Immaginate un nuovo campo di esperienza, uno qualsiasi. Lo affrontate insieme ai vostri ragazzi. Cercate di orientarvi insieme, di riconoscere delle regolarità, di definire elementi stabili e relazioni tra elementi, o di individuare processi che trasformano gli elementi in qualcos'altro. Alla fine credete di poter definire delle regole.

Come fate per *essere sicuri* che quelle regole siano giuste (vere, adeguate, sufficienti) cioè capaci di spiegare i fenomeni che avete davanti?

Il coding fornisce proprio questo: un metodo per *mettere alla prova una teoria.*

In poche parole: si crea una versione digitale di quel campo di esperienza, e al suo interno si rappresentano gli elementi reali come oggetti. Si codificano le regole come funzioni (o metodi, o vincoli) in un linguaggio di programmazione. E poi si dà vita a questo ambiente di simulazione per verificare se gli esiti sono quelli previsti.[5]

A quale campo avete pensato? Fisica? Biologia? Bene, ottima idea. Ma perché non l'Italiano, o l'Inglese? O la Storia dell'Arte? O la Musica?

In tutte le discipline c'è spazio per la costruzione e l'applicazione di regole.

Proprio per dimostrarlo, in "Lingua, coding e creatività" ho presentato dieci attività didattiche da svolgere in classe, tutte appartenenti al campo dello studio della lingua

---

4 Come ho provato a raccontare qui: https://www.slideshare.net/stefanopenge/programma-o-sarai-programmato

5 E questo, ne sono sicuro, è quello che aveva in mente Papert quando scriveva: "Il computer è il Proteo delle macchine. La sua essenza è la sua universalità, il suo potere di simulazione". Introduzione a "Mindstorms: Children, Computers, and Powerful Ideas", Basic Books, 1980 (trad. nostra)

in senso più generale. Si parte dalla grammatica di base (come si mette l'articolo davanti ad un nome?) per toccare la fonetica, le coniugazioni, l'alfabeto Braille, le poesie. Per finire con la sintassi dei giochi, le trasformazioni di melodie e la creazione di animazioni basate su stringhe di comandi.

Ogni volta si inizia cercando di immaginare un automa (cioè un programma) in grado di svolgere un compito specifico, e si procede per aggiustamenti successivi, risolvendo i problemi che mano a mano si presentano, fino ad ottenere una versione sufficientemente generale, per poi lasciare spazio a ipotesi di sviluppo futuro. Learning by doing, capire facendo.

Costruire in classe un automa in grado di associare un articolo indeterminativo maschile ad un nome in base alla sua lettera iniziale è soprattutto un modo di riflettere insieme sulle competenze linguistiche apprese da ciascuno, di scomporle, di rappresentarle in forma di regole. Regole che possono essere applicate anche ad altri casi (quando serve l'apostrofo? il femminile, il plurale) e poi generalizzate ulteriormente a problemi simili (l'articolo determinativo, gli articoli partitivi, i dimostrativi...).

Meta-apprendimento

Ma fare coding non serve solo a imparare l'italiano. Cosa si impara *in maniera trasversale* dalle attività di coding, almeno da quelle che ho descritto sopra?

Prima di tutto, le attività affrontano il tema delle *grammatiche*. Grammatiche intese non come maniere pedanti di affrontare la lingua, in termini di adeguamento ad una norma di cui ormai nessuno capisce il senso, ma come una maniera efficace di affrontare tutte le possibili varianti di una istanza comunicativa senza dover procedere ad un'enumerazione completa. Categorizzare un gruppo di fenomeni attraverso una regola significa poter trattare casi diversi da un unico punto di vista. Arrivare a scrivere questa regola in forma di algoritmo (cioè in forma eseguibile da un computer) significa essere in grado di distinguere le parti necessarie e quelle opzionali, le parti fisse e quelle varianti; insomma aver capito come funziona il lavoro scientifico.

D'altra parte, la regola può essere usata sia per *analizzare* un fenomeno, sia per *riprodurlo*. Ad esempio, la rappresentazione formale della struttura metrica di un sonetto può servire sia a verificare che un sonetto sia formalmente corretto, sia a produrre dei simil-sonetti originali. Per far questo, occorre avere un lessico e un motore che peschi a caso degli elementi e permetta di realizzare delle variazioni su un tema. Qui si tocca uno dei punti, a mio parere, più interessanti: il limite tra caso e libertà, il punto in cui un meccanismo sembra produrre risultati creativi. Non c'è intelligenza nelle macchine, lo sappiamo; ma interrogarsi su cosa distingue un Limerick prodotto da un programma e uno scritto da Edward Lear (o da Rodari) può essere un'attività che aiuta a districarsi nelle discussioni tanto di moda sulla dittatura dell'Intelligenza Artificiale che sembra attenderci dietro l'angolo.

Inoltre, tutte le attività descritte fanno parte della grande classe delle *trasformazioni*: trasformazioni di lettere, di parole, di frasi, di stringhe. Non solo all'interno di uno stesso dominio, ma anche tra un dominio (quello linguistico) e un altro (quello sonoro

o quello visivo). Una volta espressa una melodia con una sequenza di simboli, possiamo trasformarla in un disegno o, meglio, in un'animazione.

Tradurre, trasformare, ri-mediare: sono competenze davvero fondamentali, non solo per affrontare un testo in un'altra lingua, ma anche per spiegarsi, per adattare e far comprendere un insieme di informazioni in un contesto diverso da quello dove sono nate. Competenze di cui, soprattutto oggi, sembra esserci un grande bisogno.

L'ultima riflessione riguarda uno dei punti più delicati nelle pratiche di coding: il ruolo del docente. Sia per le competenze che il docente dovrebbe avere (tecniche? organizzative? pedagogiche?), sia per la maniera in cui dovrebbe relazionarsi con gli studenti in un contesto laboratoriale in cui al centro dell'attenzione c'è uno schermo.

Il docente è la guida, che apre la discussione, aiuta a vagliare le ipotesi, suggerisce parallelismi e approfondimenti. Il punto importante – e sembra strano doverlo scrivere nel terzo millennio - non è come trasmettere conoscenze agli studenti, ma come arrivare a costruire insieme agli studenti una conoscenza stabile, condivisibile.

Per questo il codice sorgente va *costruito insieme* ai ragazzi, un pezzetto alla volta, discutendone la scrittura, provando versioni diverse, analizzando gli errori per imparare insieme e poi condividendo i risultato. E questa è forse l'acquisizione metodologica più importante e duratura, tanto per gli studenti che per il docente: *imparare a costruire insieme conoscenza in forma pubblica.*

Conclusioni

Se l'onda del coding non si esaurisce sul bagnasciuga dei problemi quotidiani della scuola, può essere una grande occasione da cavalcare per prendere il largo verso terre ancora da esplorare. Potremmo essere finalmente in grado di rinnovare la didattica in maniera costruttiva, collaborativa, ma insieme rigorosa, con degli effetti misurabili non solo in termini di generica capacità logica.

I computer verrebbero allora trasformati da costosi giocattoli dall'obsolescenza troppo veloce in strumenti cognitivi insostituibili per l'apprendimento.

Una nota finale proprio sulla natura di questi strumenti e sul nostro rapporto con essi.

Finché i linguaggi (e gli ambienti di programmazione) vengono visti solo come tool neutri, come macchine indifferenti, si rischia di restare impantanati nel fossato che divide il terreno umanistico da quello scientifico. I linguaggi di programmazione sono qualcosa di radicalmente diverso da ogni altro strumento, proprio perché sono basati su simboli, parole, frasi; e come ogni altro sistema di simboli, i linguaggi di programmazione si sposano felicemente con una tendenza fondamentale del genere umano: quella di *raccontare delle storie.*

Insomma, il codice sorgente è una parte fondamentale della cultura della fine del millennio scorso, e presumibilmente lo sarà anche del millennio attuale. E' ora di cominciare a studiare il codice sorgente come una *forma particolare di testo*, con la stessa attenzione e gli stessi metodi con cui studiamo i brogliacci dei Promessi Sposi.[6]

[6] Questo obiettivo è quello che ha portato alla nascita del progetto *Codexpo.org: il primo Museo del Codice Sorgente*. Vedi http://www.codexpo.org