BRICKS | COMPETENZE e CERTIFICAZIONI

Scratch e il modulo ECDL Computing

a cura di:
Pierfranco Ravotto

Computational thinking, Pensiero computazionale, ECDL Computing, Scratch

Mi è capitato, in più occasioni, di sentire esprimere l'opinione che per "fare della vera informatica" occorra riferirsi ad un linguaggio di programmazione testuale. Linguaggi iconici come Scratch sono "carini", sicuramente adatti per far scrivere programmini ai bambini … ma la vera programmazione è un'altra cosa: questo il giudizio che ho sentito più volte esprimere.

Su un punto mi sentivo di dissentire: quando qualcuno rivendicava come vera informatica il dover stare attenti ai due punti o alle virgole, a non commettere i cosiddetti errori sintattici. Che in una lingua la sintassi sia importante è fuori di dubbio, ma l'essenza dell'informatica a me sembra stia nell'analisi del problema, nell'invenzione di una procedura risolutiva, nella sua verifica, non nella traduzione in uno specifico linguaggio. Pensavo, e continuo a pensare, che l'uso di un linguaggio grafico – come Scratch – possa permettere, più di un linguaggio testuale, di concentrarsi sull'algoritmo proprio perché libera dalla preoccupazione di virgole, punti e virgola o due punti.

Ma non avendo allora mai provato davvero Scratch non ero in grado di dire di più.

Poi la Fondazione ECDL ha lanciato una nuova certificazione, ***ECDL Computing***, e abbiamo, come AICA, progettato un corso online "Fare didattica con il coding", di cui ho già parlato nel numero di settembre dell'anno scorso (*La nuova certificazione "Computing" e un corso per i docenti sul pensiero computazionale*).

Il syllabus "Computing" non fa riferimento a nessun linguaggio specifico, ma molte domande d'esame necessitano di un riferimento ad un linguaggio ed è stato scelto Python. Di conseguenza anche i testi di preparazione all'esame fanno attualmente riferimento a Python.

Nel progettare il corso abbiamo provato a sviluppare in parallello applicazioni con Python e con Scratch e abbiamo sviluppato una duplice simulazione d'esame, una con riferimento al primo, l'altra con riferimento al secondo.

Operazione riuscita.

Abbiamo potuto proporre con Scratch le stesse attività proposte con Python. E non abbiamo avuto difficoltà a individuare domande equivalenti per la simulazione d'esame.

I risultati di questa verifica, uniti al fatto che la maggior parte dei partecipanti al corso hanno preferito le attività e la simulazione d'esame con Scratch, hanno portato AICA alla decisione di **implementare l'esame ECDL Computing anche in ambiente Scratch**. Così come, per esempio, l'esame del Modulo ECDL Word processing può essere sostenuto sia in ambiente Word che in ambiente OpenOffice o quello del modulo Computer essentials può essere svolto sia in ambiente Linux che in ambiente Windows, altrettanto l'esame del Modulo Computing può adesso essere svolto, a scelta, in ambiente Python o in ambiente Scratch.

Una novità già in essere: i primi esami ECDL Computing in ambiente Scratch si sono già svolti. Una novità che penso potrà interessare anche una parte dei nostri partner nella Fondazione ECDL.

Novità importante importante per un duplice motivo. Da un lato "sdogana" Scratch come "vero" ambiente di programmazione, non solo come adatto ai bambini. Dall'altro permette di proporre la certificazione – e un percorso formativo – sul *computational thinking* anche a insegnanti di scuole Primarie e Secondarie di 1° grado che hanno imparato a usare Scratch ma non hanno un retroterra

informatico. E questo può far fare loro un passo in avanti nella comprensione del pensiero computazionale, con ricadute positive sui loro allievi.

Penso ce ne sia bisogno. Perché non basta usare/far usare Scratch per educare gli studenti al pensiero computazionale. Non è Scratch a non permettere di fare vera programmazione. E' piuttosto una questione di uso che molti ne fanno.

Scelte, cicli, variabili, liste, eventi, subroutine con Scratch

E' noto che la programmazione strutturata prevede che si usino solo tre strutture: sequenza, scelta e ciclo. Solo tre, ma da usare, in linea di massima, tutte.

Invece ho notato che in molti casi i programmi fatti fare agli studenti o prodotti dagli insegnanti usano spesso solo la sequenza, in altri casi sequenza e scelta. Scratch è usato in molti casi per fare dello *storytelling*, per raccontare una storia che si sviluppa in modo lineare, con azioni in sequenza.

Ora, lo *storytelling* ha una notevole rilevanza pedagogica e imparare a far eseguire ad un computer una sequenza contribuisce ad aumentare la competenza digitale. Ma l'informatica non è una semplice automatizzazione di sequenze.

E, in ogni caso, se anche voglio semplicemente che il gattino faccia 3 passi, dal punto di vista dell'acquisizione del pensiero computazionale non è la stessa cosa ripetere 3 volte la sequenza per un passo o usare un **ciclo** per ripetere 3 volte la sequenza!

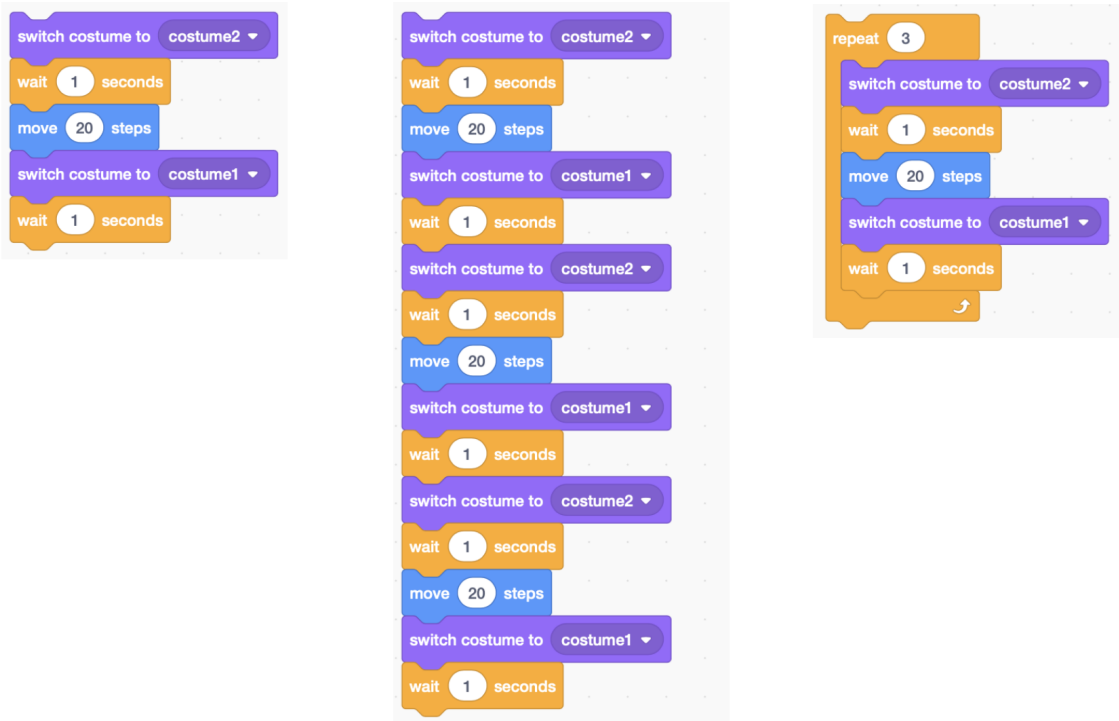


Figura 1 - A sinistra una sequenza per far fare un passo. Al centro: la sequenza viene scritta tre volte per far fare tre passi. A destra un ciclo fa ripetere tre volte la sequenza

E se il gattino deve in diversi punti della storia fare 3 passi: perché non usare una **subroutine (procedura)** richiamabile da più punti del programma?

E se, invece, una volta il gattino deve fare 3 passi, un'altra solo 2 ed un'altra 5? Perché non usare una **variabile** per passare un **parametro** alla subroutine/procedura?

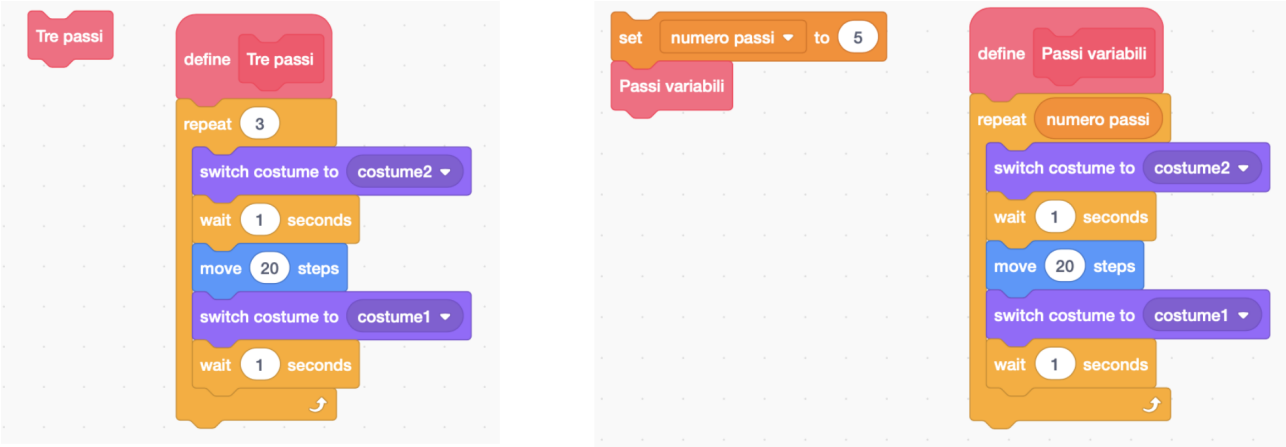


Figura 2 - A sinistra il programma principale chiama una subroutine che fa fare tre passi. A destra il programma principale attribuisce un valore alla variabile "numero di passi" e chiama la subroutine che fa fare il numero di passi indicato

Lo dico con altre parole, anzi con quelle di Vygotsky: "area di sviluppo prossimale". Pensate a una maestra elementare, magari di area letteraria, che ha imparato ad usare Scratch per far fare ai suoi bambini dello *storytelling*, usando uno strumento "attrattivo", e magari per contribuire alla competenza digitale. Cosa le diciamo? Passa a Python e impara la vera informatica? Usciremmo dalla sua area di sviluppo prossimale. Se, invece, le spieghiamo che il programma può essere scritto in modo più semplice usando i cicli, le subroutine, le variabili … agiamo nella sua area di sviluppo prossimale.

Potrei fare molti altri esempi, ma ne farò solo uno. Nel corso abbiamo preso in considerazione un programma – trovato nella *repository* di Scratch – che propone un test con autocorrezione su un argomento di storia. Era una lunga sequenza di:

<faccio la domanda>,

<acquisisco la risposta>

<IF la risposta è giusta THEN scrivo "vero" ELSE scrivo "falso">

Abbiamo proposto di modificarlo in modo da permettere allo studente di scegliere a quante domande sottoporsi e di permettere a un docente che non sappia programmare di riutilizzare lo stesso programma modificando domande e risposte o di aggiungendone altre. Facile da fare usando un **ciclo** e due **liste**, una per le domande e una per le risposte. Non c'è bisogno di passare a Python, si può farlo con Scratch.

Ho scritto che molti docenti usano Scratch per lo *storytelling*. Altri lo usano per progettare videogiochi, secondo il famoso suggerimento di Barack Obama alla presentazione della Computer Science Education Week del 2013: *"Non limitarti a comprare un videogioco. Realizzane uno"*. E' certamente più facile realizzare videogiochi con Scratch, che mette a disposizione *sprite* e semplici comandi per gestire gli eventi e sensori, che non con Python. Ma questa semplicità gioca a favore dello sviluppo del pensiero computazionale perché, permettendo di aumentare il numero di oggetti da far interagire e le modalità di interazione, richiede di concentrarsi sulla progettazione di algoritmi via via più complessi.

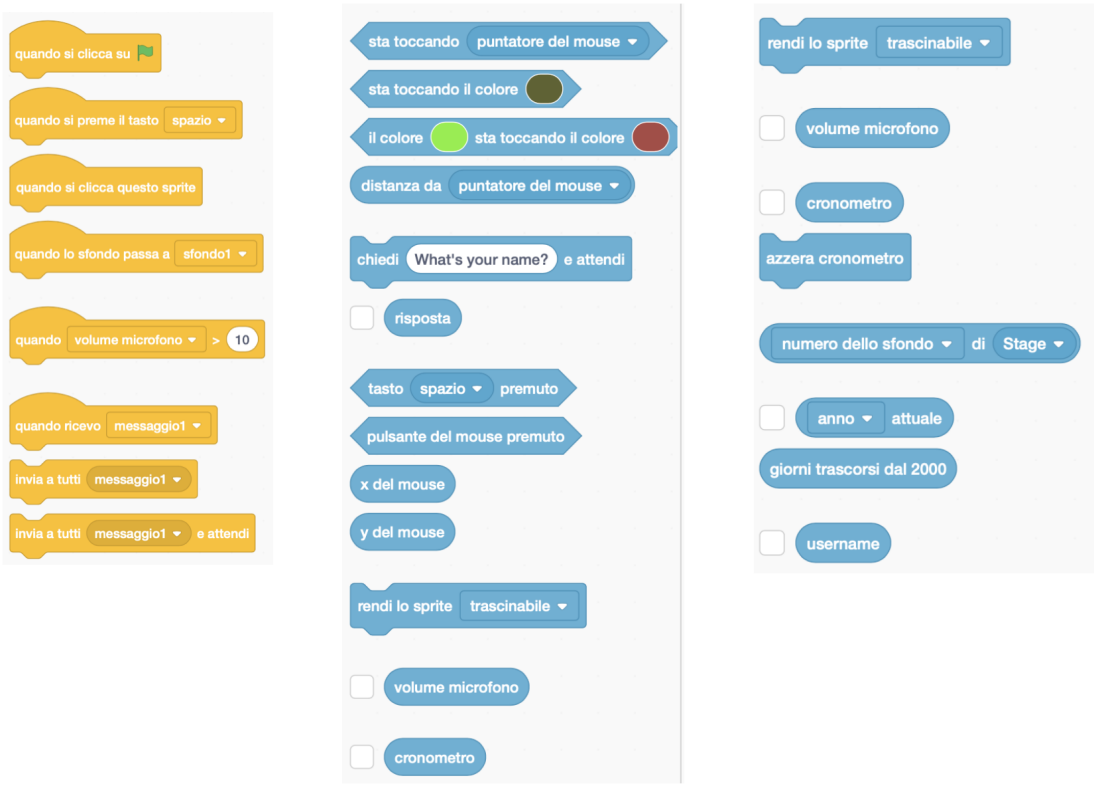


Figura 3 - In giallo le istruzioni per gli eventi, in celeste quelle per i sensori

Scratch permette, inoltre, l'interfacciabilità con robot fisici e quindi l'apertura alla robotica. Ma questo è un discorso che va oltre il Modulo ECDL Computing.

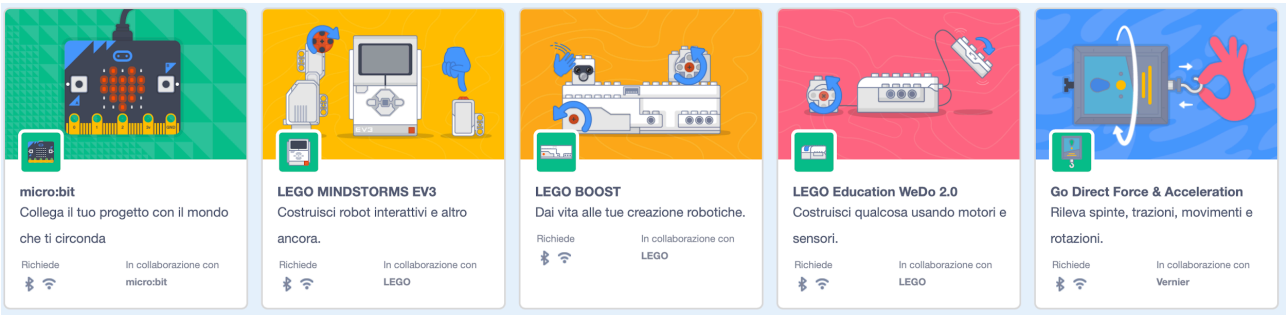


Figura 4 - Estensioni. Accesso alle istruzioni per comandare robot.

Il Syllabus ECDL Computing

Il syllabus ECDL Computing è articolato in 5 sezioni, articolate in 12 temi (poi ulteriormente specificati in 54 argomenti):

1. Termini informatici
    - 1.1 Concetti fondamentali
2. Metodi di pensiero computazionale
    - 2.1 Analisi del problema
    - 2.2 Algoritmi
3. Iniziare a programmare
    - 3.1 Per iniziare
    - 3.2 Variabili e tipi di dati
4. Costruire con il codice
    - 4.1 Logica
    - 4.2 Iterazione
    - 4.3 Condizionalità
    - 4.4 Procedure e funzioni
    - 4. 5 Eventi e comandi
5. Test, debug e rilascio
    - 5.1 Esecuzione, test e debug
    - 5.2 Rilascio

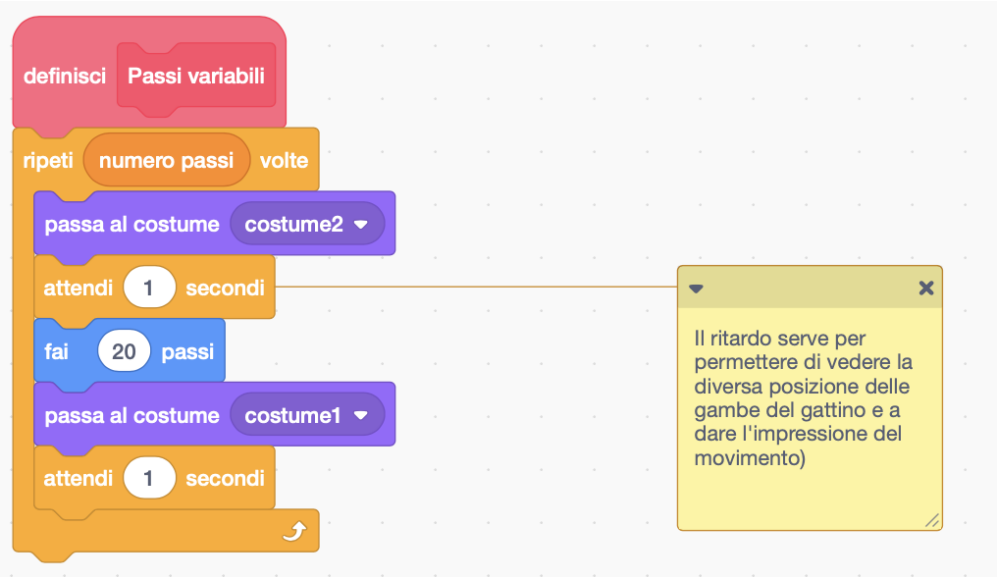


Figura 5 - Esempio di commento

Le sezioni che richiedono il riferimento ad un linguaggio di programmazione sono la 3, la 4 e la 5. E, come ho già detto, abbiamo verificato che si possono affrontare, allo stesso livello in Python o in Scratch. Solo un argomento non può essere proposto con Scratch (e di conseguenza non può essere oggetto di una domanda d'esame). Si tratta del 5.1.4: *"Identificare e correggere un errore sintattico in un programma, quale errore di ortografia, punteggiatura mancante".* Ma non è certo lì la sostanza del pensiero computazionale.

Nella programmazione con linguaggi come Python si insiste, giustamente, sulla documentazione. Nel syllabus l'argomento 3.1.5 recita *"Definire il termine "commento" dei costrutti di programmazione. Illustrare lo scopo di un commento in un programma"*. Qualcuno mi ha fatto notare che invece con Scratch non si documenta. Ciò non è vero: cliccando sul tasto destro su un'istruzione è possibile inserire un commento.

Test, debug e rilascio

Per motivare la scelta di proporre l'esame ECDL Computing potrei fermarmi qui. Ma alla luce dell'esperienza del corso "Fare didattica con il coding" - che nelle sue due edizioni ha coinvolto 430 docenti provenienti dai diversi ordini di scuola e da molti test center privati - penso importante soffermarmi sulla quinta sezione del Syllabus: Test, debug e rilascio.

In fase di progettazione del corso pensavamo che si dovesse attirare l'attenzione degli insegnanti sulla necessità di lavorare con gli studenti sul controllo e la verifica del programma. Perché il controllo della soluzione progettata è un elemento essenziale del Computational thinking.

Controllare il programma non significa solo verificare che "giri", Occorre verificare che faccia effettivamente quanto previsto. E che lo faccia in tutti i casi possibili. Nel video-tutorial su questo argomento cito il *Millennium bug*: prima di rilasciare un programma nel 1977 si sarebbe dovuto controllare che fosse in grado di funzionare non solo allora e negli anni successivi ma anche il 1° gennaio 2000!

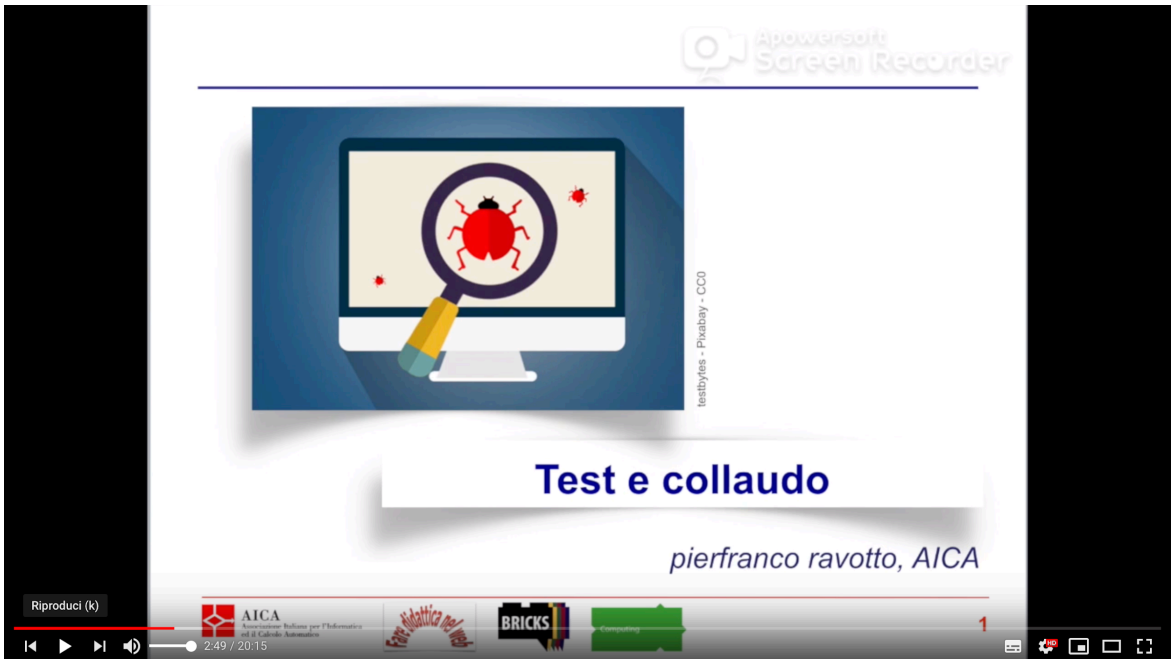


Figura 6 - Un videotutorial del corso - https://youtu.be/NYoJbbJGDkQ

Pensavamo, dunque, di dover attirare l'attenzione degli insegnanti su questo aspetto perché ne tenessero conto nel lavoro con i propri allievi. Ma, con stupore, ci siamo resi conto che tanti insegnanti sono i primi a non preoccuparsi del controllo. Il corso prevedeva delle "consegne" e, tra queste, dei "programmi" da sviluppare. Ebbene, spesso i programmi consegnati:

- non erano visibili (in Scratch si erano dimenticati di renderli pubblici),
- non "giravano",
- funzionavano in alcuni casi, non in altri,
- non davano risultati corretti (per esempio in "indovina il numero" era richiesto che il programma indicasse il numero di tentativi fatti, ebbene spesso il risultato era errato perché il contatore contava le risposte errate ma non l'ultima giusta),
- non facevano tutto quanto era richiesto.

In tanti casi non si trattava di un errore difficile da individuare (come quello che sarebbe avvenuto al cambio di millennio); in alcuni casi era sufficiente provare ad eseguire il programma prima di inviarlo!

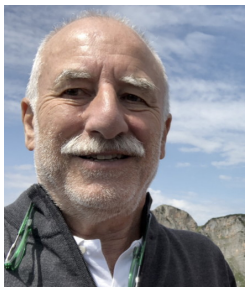
Per cercare di lavorare su questo aspetto, abbiamo proposto alcuni programmi con errori chiedendo di individuarli. Anche qui siamo rimasti davvero sorpresi da molte risposte dei corsisti da cui emergeva un atteggiamento sbrigativo del tipo: "se il programma gira è giusto", una mancanza di metodo nell'organizzazione dei test, e soprattutto un ragionare sulle istruzioni senza "provare" concretamente. Un esempio: in "indovina il numero" se uno ragiona sulle istruzioni magari non individua l'errore, ma se prova il programma e conta i tentativi si accorge per forza che il risultato è errato. Poi, sapendo che è errato (e magari avendo constatato che è sempre errato per un -1), si mette alla ricerca dell'errore.

Quindi un corso di formazione per gli insegnanti oltre a suggerire fortemente l'importanza del *debugging* con gli studenti, deve condurre, mediante esercitazioni *ad hoc*, a far provare loro in prima persona cosa significhi saltare o minimizzare la fase di *debugging*.

A proposito di "vera informatica": essa è qui, molto più che nelle virgole.

E anche in questo caso: è un tema su cui si può lavorare sia con Scratch che con Python. Anzi, a dirla tutta, con Python si rischia di pensare di aver esaurito il *debugging* una volta controllata l'ortografia. Con Scratch è certo che non si tratta di ortografia ma di ben altro.

— L'autore —



Pierfranco Ravotto

email: pierfranco.ravotto@gmail.com

Laureato in Ingegneria elettronica si è dedicato all'insegnamento iniziando con corsi di formazione per apprendisti e con le 150 ore per proseguire poi per oltre trent'anni in scuole secondarie superiori quale docente di Elettrotecnica, Elettronica e Automazione. Si è occupato di Orientamento, Alternanza scuola-lavoro, Corsi post-diploma, Scambi con alternanza all'estero e Progetti europei di ricerca sull'eLearning.

E' stato Project manager del progetto Ensemble per l'Università di Firenze. Attualmente è consulente AICA per la formazione sulla didattica digitale e per i progetti europei.

E' membro del Direttivo AICA Lombardia e del Direttivo SIe-L e co-direttore della rivista Bricks.