# Un ambiente di calcolo evoluto per lo sviluppo del pensiero computazionale

Marisa Di Luca[1][0000-0001-5919-096X] and Marina Marchisio[0000-0003-1007-5404]

[1] Istituto Istruzione Superiore A. Volta Via A. Volta n. 15 65129 Pescara, Italia
[2] Università di Torino, Dipartimento di Matematica, Via Carlo Alberto 10, 10123 Torino, Italia
marisadl@libero.it, marina.marchiiso@unito.it

**Abstract.** Competenze, Coding, pensiero computazionale, Information Comunication Tecnology, Comunità di apprendimento, Didattica laboratoriale, Problem Posing, sono solo alcuni dei termini che sono entrati di prepotenza nelle nostre scuole negli ultimi anni e che identificano concetti molto forti che sono la base per un rinnovamento della prassi didattica. Questo lavoro cercherà di mettere in evidenza come l'utilizzo di una Ambiente di Calcolo Evoluto nella didattica sia particolarmente adatto allo sviluppo del pensiero computazionale da un lato e all'introduzione al coding (programmazione) dall'altro in classi della scuola secondaria di secondo grado.

**Keywords:** Ambiente di Calcolo Evoluto, Coding, Computational Thinking, Problem Posing, Problem Solving,

## 1 Introduzione

Mai come in questi ultimissimi anni si è sentito parlare in ambito scolastico di coding, computational thinking, programmazione, problem posing, digital literacy e così via. A volte, a onor del vero, ci sono stati da un lato ambiguità dall'altro abuso di concetti che nell'insegnamento/apprendimento delle discipline tecnico-scientifiche sono sempre stati in realtà utilizzati anche se non con continuità e senza progettazione. Anche del concetto di laboratorialità si è abusato; l'attività "pratica" è fondamentale nelle attività formative, ma altrettanto essenziale è il concetto che sta alla base e che governa il processo. E' importante che si arrivi alla concettualizzazione, alla formalizzazione di quanto osservato/sperimentato altrimenti non si può parlare di apprendimento, ancor meno di competenze perché semplicemente ci si è impossessati di una tecnica.

Perché tanto interesse della scuola nei confronti di questi temi? Principalmente per passare dalla scuola dei contenuti a quella delle competenze. Lavorare sul coding e sullo sviluppo del pensiero computazionale con metodologie di problem posing, impostando una didattica laboratoriale e utilizzando, in modo critico e riflessivo, la tecnologia, crea le condizioni ottimali per tale passaggio. Parlando di competenze un ruolo importante è quello delle competenze chiave (imparare a imparare, progettare, comunicare, agire in modo autonomo e responsabile, risolvere problemi, individuare collegamenti e relazioni e interpretare l'informazione) che sono coinvolte in modo

deciso in un'attività laboratoriale che utilizza Ambienti di Calcolo Evoluto (ACE), ormai da anni un riferimento importante nell'impostazione della didattica per competenze. Questo lavoro cercherà, dopo aver inquadrato lo stato dell'arte, di mostrare attraverso esempi concreti come l'utilizzo di un ACE con gli studenti delle classi di un primo biennio della scuola secondaria di secondo grado sia particolarmente adatto allo sviluppo del pensiero computazionale e all'introduzione al coding.

## 2 Stato dell'arte

Dalle raccomandazioni del Parlamento Europeo e del Consiglio del 18 dicembre 2006 si legge: "Le competenze chiave sono considerate ugualmente importanti, poiché ciascuna di esse può contribuire a una vita positiva nella società della conoscenza. Molte delle competenze si sovrappongono e sono correlate tra loro: aspetti essenziali a un ambito favoriscono la competenza in un altro. La competenza nelle abilità fondamentali del linguaggio, della lettura, della scrittura e del calcolo e nelle tecnologie dell'informazione e della comunicazione (ICT) è una pietra angolare per l'apprendimento, e il fatto di imparare a imparare è utile per tutte le attività di apprendimento. Vi sono diverse tematiche che si applicano nel quadro di riferimento: pensiero critico, creatività, iniziativa, capacità di risolvere i problemi, valutazione del rischio, assunzione di decisioni e capacità di gestione costruttiva dei sentimenti svolgono un ruolo importante per tutte e otto le competenze chiave". E' evidente come ci sia un nesso molto forte con una organizzazione della didattica sopra descritta.

Già dall'ultima riforma sia nelle linee guida sia nelle indicazioni nazionali si può dedurre l'importanza della didattica laboratoriale da un lato e dell'utilizzo degli strumenti tecnologici dall'altro. Questi ultimi non certo come semplice meccanicismo, ma piuttosto come un "nuovo ambiente di apprendimento" learner centred o problem solving oriented, aspetto che può essere ritenuto sicuramente un nodo cruciale del ruolo che riveste la tecnologia nella scuola oggi, vista ancora come semplice strumento di supporto, facilitatore e non come un ambiente di apprendimento da progettare e costruire. Gli "ambienti di apprendimento" sono l'approccio didattico adeguato quando si vuole promuovere un "apprendimento significativo" in luogo di uno meccanico, quando si persegue la comprensione e non la memorizzazione, la produzione di conoscenza invece che la sua mera riproduzione, l'utilizzo dei contenuti didattici piuttosto che la loro ripetizione [13].

Successivamente alla riforma Gelmini del 2008 il Piano Nazionale Scuola Digitale, PNSD (legge 107/2015), ha formalizzato in maniera molto puntuale l'uso della tecnologia nelle scuole italiane. Tra i tantissimi aspetti uno dei cardini fondamentali è proprio lo sviluppo del pensiero computazionale, quindi di attività di coding, a partire dalla scuola primaria e dalla secondaria di primo grado. In questi ordini di scuola, grazie anche alla piattaforma www.programmailfuturo.it, gestita dal MIUR, alle "giornate" dedicate al coding (Europe Code Week, l'Ora del codice) e all'ambiente visuale Scratch, molti passi avanti sono stati compiuti. Le azioni dove si trovano i riferimenti essenziali nel PNSD: #14 Framework comune per le competenze digitali: […] di 21 competenze descritte per conoscenze, abilità e atteggiamenti, comprese in 5

aree: Informazione, Comunicazione, Creazione di contenuti, Sicurezza e Problem Solving. […]; #16 Una research unit per le competenze del XXI secolo: […] competenze trasversali e della capacità di muoversi nell'ambiente digitale: alfabetizzazione informativa e digitale (information literacy e digital literacy). Per competenze trasversali si intendono: problem solving, il pensiero laterale e la capacità di apprendere. […]. #18: Aggiornare il curricolo di Tecnologia alla scuola secondaria di primo grado. Il PNSD da un lato non ha potuto ignorare la dimensione della formazione degli studenti europei dall'altro ha recepito l'importanza che assumono oggi le ICT nella vita di un individuo inserito in quella che oggi si chiama "società della conoscenza". Non è strano, quindi, che sempre più si facciano riferimenti alle competenze digitali o, in maniera più generale, alla digital literacy, termine che indica competenze non strettamente legate al semplice utilizzo dello strumento (hardware o software) tecnologico, ma che mettono in evidenza le esigenze che ogni cittadino dovrà avere per potersi inserire non solo in ambienti lavorativi, ma anche sociali [5], [8].

Nell'Agenda Digitale europea, una delle iniziative della strategia "Europa 2020" (con obiettivi come la crescita dell'occupazione, della produttività e della coesione sociale in Europa da raggiungere entro il 2020) la Commissione Europea indica la strada per un utilizzo delle ICT per l'integrazione, il progresso e la formazione. Una delle prime azioni concrete è quella dell'alfabetizzazione e dello sviluppo di competenze digitali. Il mondo della scuola, ovviamente, non può ignorare questa forte necessità sia per quanto riguarda la preparazione all'inserimento nel mondo del lavoro e/o al prosieguo degli studi, ma soprattutto perché l'utilizzo riflessivo e intelligente della tecnologia è funzionale al lifelong learning; non se ci impossessa di tecniche, che peraltro diventerebbero obsolete in brevissimo tempo, ma di una mentalità, di un approccio analitico "spendibile" sempre e dovunque.

Nel contesto europeo numerose sono le iniziative di introdurre l'uso di un ACE nell'insegnamento e nell'apprendimento della Matematica. Significativo il caso della Danimarca che a tutti gli studenti, a partire dagli 11 anni, fornisce un computer portatile dotato di un ACE per poter apprendere meglio le discipline scientifiche.

La scuola secondaria di secondo grado italiana si occupa, a volte, solo marginalmente di questi temi. Probabilmente solo negli istituti tecnici si imposta una didattica diversa (ad esempio con la robotica educativa) e si continua nello sviluppo, o meglio potenziamento, delle competenze digitali e del pensiero computazionale. Dall'anno scolastico 2012/13 il progetto ministeriale Problem Posing and Solving (PP&S), che vede in partnership MIUR, Università di Torino, Politecnico di Torino, AICA e CNR, ha tra gli obiettivi principali anche quello di sviluppare uno spazio di formazione integrata che interconnetta logica, matematica e informatica, [1] e [3], e offre la possibilità ai docenti di matematica, di informatica, di fisica di sperimentare come, utilizzando un ACE, si possa lavorare nella direzione del CODING quindi della programmazione, nello sviluppo del pensiero computazionale. Il progetto PP&S ha contribuito alla formazione di una comunità di docenti non solo di buone pratiche, ma di apprendimento.

## 3 Pensiero Computazionale e Problem Solving

Cosa si nasconde dietro questi due termini? Perché oggi è così importante impadronirsi di questa attitudine? Perché la scuola mostra tanto interesse? Anche il Presidente Barak Obama (2013) pone l'accento sull'importanza della programmazione. La sua celebre frase: "Non usare il tuo telefono solo per giocare. Programmalo" in occasione della Hour of Code è significativa. Perché tanto interesse? Cercheremo di cogliere i legami che ci sono fra: programmazione (coding), pensiero computazionale, informatica e la teoria dell'apprendimento nota come costruttivismo o meglio costruzionismo. Prima di tutto un po' di storia. Il computational thinking fu introdotto da Seymor Papert nel 1996 in relazione all'ambiente LOGO (temine che deriva dal greco: λόγος, nel significato di "parola" o "pensiero"), primo linguaggio di programmazione pensato per i bambini, in cui una tartaruga si muove sullo schermo e disegna forme geometriche. I principi dell'idea di Papert si trovano già nei suoi studi sulle teorie di Jean Piaget e nel lavoro con Marvin Minsky, presso i laboratori di intelligenza artificiale del MIT. La base è la teoria del costruzionismo secondo cui il soggetto che apprende si crea modelli mentali cui attinge per comprendere le cose e che l'apprendimento avviene in maniera più efficiente ed efficace se il soggetto è coinvolto nella costruzione di oggetti tangibili (artefatti cognitivi). Papert, [15], definisce il costruzionismo come: "Una parola che indica due aspetti della teoria della didattica delle scienze alla base di questo progetto. Dalle teorie costruttiviste in psicologia prendiamo la visione dell'apprendimento come una ricostruzione piuttosto che come una trasmissione di conoscenze. In seguito estendiamo il concetto dei materiali manipolativi nell'idea che l'apprendimento è più efficiente quando è parte di un'attività come la costruzione di un prodotto significativo". E' secondo le idee di Papert che Mitchel Resnick ha sviluppato l'ambiente di programmazione visuale Scratch che dà la possibilità di creare storie e videogiochi già dalla scuola primaria.

Chi però ha teorizzato il Computational Thinking, CT, è stata Jaannette Wing, che dà questa definizione: " Il CT è il processo mentale che sta alla base della formulazione dei problemi e delle loro soluzioni così che le soluzioni siano rappresentate in una forma che può essere implementata in maniera efficace da un elaboratore di informazioni sia esso umano o artificiale". Il CT si può definire un metodo di risoluzione di problemi che utilizza la tecnologia informatica. Rappresenta l'impegno che un individuo deve impiegare per far eseguire a un altro (uomo o macchina) qualcosa dando "comandi, istruzioni" necessari all'esecuzione del compito. L'aggettivo "computazionale" non deve però essere fuorviante: non riguarda solo coloro che scelgono di affrontare studi informatici, il pensiero computazionale è una skill trasversale che tutti dovrebbero possedere. Naturalmente è necessario creare le condizioni affinché gli studenti siano in grado di "imparare a imparare", impadronirsi, cioè, di una mentalità da mettere in campo sempre in ambito lavorativo o di studio. Ormai il pensiero computazionale è importante come leggere, scrivere e far di conto. I giovani, attraverso gli studi, si preparano per professioni che in parte oggi non esistono, di conseguenza impadronirsi di una tecnica non serve, ma padroneggiare una attitudine si.

Quando si parla di utilizzo dello strumento informatico, però, bisogna tener presente quelli che sono i passaggi preliminari che portano alla soluzione del problema:

la costruzione dell'algoritmo. Ecco il legame forte con l'informatica. Ragionare in termini di pensiero computazionale vuol dire impostare algoritmi risolutivi che poi saranno implementati in ambienti specifici. Un errore è confondere l'informatica con la sola programmazione, il termine informatica indica qualcosa di più ampio e, se vogliamo, più nobile. E' l'approccio mentale che poi condiziona tutto l'apprendimento a scuola e fuori. L'informatica, nel senso del pensiero computazionale, dovrebbe rientrare tra le discipline di qualsiasi scuola secondaria di secondo grado.

Spesso il CT viene "liquidato" come attività di Problem Solving (PS), ma non è proprio così: vediamo i passi di una attività secondo il CT e quelli del PS per evidenziare le differenze.

Cosa si intende per PS? E' sicuramente una delle funzioni cognitive più complesse e consiste nell'individuare una soluzione, possibilmente originale e creativa, a qualsiasi tipo di problema. Un bravo problem solver è colui che sa affrontare qualsiasi tipo di situazione e riesce a superare le difficoltà che incontra nel percorso di costruzione della soluzione. Molte sono le classificazioni delle varie fasi del Problem Solving, vediamone qualcuna. La prima prevede che ci siano: 1) definizione del problema; 2) analisi (comprensione della consegna, individuazione degli obiettivi, definizione delle variabili) e scomposizione del problema in sotto-problemi (più semplici che potrebbero anche essere stati risolti in precedenza); 3) formulazione delle ipotesi; 4) verifica delle ipotesi; 5) individuazione della/e strategia/e; 6) scelta della strategia; 7) risoluzione; 8) verifica della soluzione. Un altro approccio molto utilizzato è il F.A.R.E.: 1) Focalizzare: selezione e definizione del problema (circoscrizione); 2) Analizzare: definizione delle informazioni da ricavare e della loro importanza per poi raccogliere i dati e classificarli; 3) Risolvere: creazione di soluzioni alternativa e selezione della migliore. Sviluppo di un piano di attuazione; 4) Eseguire: definire l'obiettivo desiderato, esecuzione del piano e monitoraggio dei risultati. È interessante anche il metodo Lasswell o "metodo delle 5W"; metodo che prevede che il solver si ponga una serie di domande "guida" alla soluzione: 1) Who?; 2) What?; 3) Where?; 4) When?; 5) Why?. Qualunque sia l'approccio il PS è sicuramente una strategia che stimola la curiosità dello studente, la sua creatività e lo rende protagonista del proprio processo di apprendimento.

Le fasi del CT sono invece: 1) DECOMPOSIZIONE. Durante questa fase lo studente cerca di scomporre il problema in sotto-problemi, più semplici e cerca di capire se ha già incontrato in passato i singoli sotto-problemi. La scomposizione di un problema porta alla costruzione di un algoritmo risolutivo; 2) PATTERN RECOGNITION: capacità di individuare somiglianze e differenze che possono dare la possibilità di costruire modelli; 3) GENERALIZZAZIONE e ASTRAZIONE: fase dell'individuazione delle informazioni necessarie alla soluzione e generalizzazione per poter "esportare" la soluzione ad altri problemi simili; 4) ALGORITHM DESIGN: capacità di costruire una sequenza di passi per la soluzione non del problema specifico, ma di tutti i problemi di una determinata classe. Oppure possono essere elencate come: 1) formulazione di problemi ai quali un elaboratore può fornire supporto; 2) analisi e organizzazione logica dei dati (data modeling e data abstraction) e simulazioni; 3) identificazione, test e implementazione di possibili soluzioni; 4) au-

tomazione di soluzioni attraverso il pensiero algoritmico (algorithmic thinking); 5) generalizzazione ed applicazione di questo processo ad altri problemi.

La differenza fra le due attività è sostanziale e si evince nell'ultima voce di entrambe le classificazioni: "tutti i problemi di una determinata classe", "questo processo ad altri problemi". Impostando un'attività basata sul CT si crea un modello, l'algoritmo che può essere applicato a tutti i problemi simili. Vediamo un esempio: supponiamo di dover sommare 5 e 7. Posso semplicemente scrivere: X=5+7 oppure definire una variabile A=5 e una variabile B=7, e scrivere X=A+B. Sicuramente il secondo modo è più generale, si utilizzano variabili. Ma il metodo ancora più generale, che costituisce un modello, è quello in cui imposto un algoritmo:

Leggi($A$), Leggi($B$), $X=A+B$, Scrivi($X$)

Quest'ultimo metodo rappresenta un'applicazione del pensiero computazionale perché ho creato un modello, ho generalizzato; ogni volta che incontro questo tipo di problema so cosa fare e come procedere. Quanto illustrato costituisce il legame tra CT, costruzionismo e coding. L'algoritmo è l'oggetto su cui ruota tutta la programmazione, costituisce il modello di problemi che appartengono a una certa classe.

## 4 L'Ambiente di Calcolo Evoluto Maple

Un ACE è un sistema in grado di compiere calcolo numerico, calcolo simbolico e visualizzazione grafica. Gli ACE sono l'evoluzione degli Computational Algebra Systems, CAS, inventati da matematici e informatici alla fine degli anno Ottanta. I principali ambienti di calcolo evoluto utilizzati oggi nella ricerca, nel mondo del lavoro e nella didattica sono Maple e Mathematica. I motori matematici di entrambi sono simili, ma il primo dispone anche di strumenti per la valutazione automatica e per la simulazione di modelli fisici rendendolo più adatto nell'apprendimento e insegnamento. Per questo motivo nel progetto ministeriale PP&S è stata data la possibilità ai docenti di matematica, di informatica, di fisica di impararlo e utilizzarlo con i loro studenti, [2], [6].

Maple può essere considerato un ambiente di apprendimento perché dà la possibilità agli studenti di organizzare autonomamente il proprio percorso. Perché, in sintesi è "un sistema dinamico, aperto, forse caotico, in cui le persone che apprendono hanno la possibilità di vivere una vera e propria "esperienza di apprendimento"; esso è ricco e ridondante di risorse per poter essere funzionale alle differenti situazioni reali in cui si svilupperà il processo formativo, determinato dai sistemi personali di conoscenza che caratterizzano ciascun allievo. Gli "obiettivi di apprendimento" rappresentano, in questa prospettiva, più la direzione del percorso che la meta da raggiungere. I "contenuti" non sono pre-strutturati e sono presentati da una pluralità di prospettive; non tutti devono essere appresi ma rappresentano una "banca dati" cui attingere al bisogno [13]. Nel prossimo paragrafo viene illustrato concretamente non solo come con l'utilizzo di Maple è possibile impostare una lezione di matematica in modo veramente diverso, ma anche di introdurre il coding (la programmazione) ad un livello più adatto a studenti della secondaria di secondo grado e a sviluppare CT.

## 5 Una metodologia integrata: ACE, Coding e CT

Partiamo dal presupposto di aver introdotto Maple come ACE nelle lezioni di matematica, ad esempio in una prima liceo, e di voler proseguire con il coding. Perché introdurre la programmazione con Maple piuttosto che con Scratch, ambiente semplice e, se vogliamo, più accattivante o con Pyton? Perché oltre che avere a disposizione un interprete in Maple c'è la possibilità di usare all'interno del codice una serie lunghissima di comandi e di poter sfruttare le enormi potenzialità per quanto riguarda la grafica. L'introduzione alla programmazione in una classe del primo anno di scuola di secondo grado riguarda lo studio dei costrutti fondamentali: a) sequenziale; b) selezione (binaria e multipla); c) iterazione (definita e indefinita). Si inizia con quello più semplice: la sequenza. Vediamo come si implementano tali strutture. Una struttura è sequenziale quando contiene solo istruzioni di: lettura, scrittura e assegnazione. In Maple non ci sono istruzioni di lettura, la scrittura, cioè l'output, è automatica. Vediamo un primo esempio: calcolo dell'area di un quadrato, la codifica in Maple (naturalmente si lavora in modalità MATH) è:

```
L:=4; AREA:=L*L;
```

```
4
16
```

Ovviamente cambiando il valore di *L* automaticamente si aggiorna il valore dell'area. La struttura generale del costrutto è:

**if** condizione/espressione booleana **then**
esecuzione blocco 1;
**else**
esecuzione blocco 2
**end if**;

In "linguaggio Maple"

```
x:=-8;
if x<0 then x:=x*( -1)
end if;
```

```
x:=-8
x:=8
```

L'esempio si riferisce alla visualizzazione del modulo di un numero intero. Ma l'**if** in Maple ha sfaccettature interessanti. Analizziamo il seguente codice:

```
restart:
a:=6;
b:=5;
x:= if(a<b,NULL,b);
print("x=",x);
```

```
6
5
"x="5
```

Il significato è il seguente: Se $a<b$ allora $x=NULL$ altrimenti $x=b$, effettivamente nell'esempio a NON è minore di *b*, quindi *x* prende il valore di *b*, cioè 5. Cambiamo e vediamo cosa succede:

```
restart
a:=2;
b:=5;
x:= if(a<b,NULL,b);
print("x=",x);
```

```
2
5
"x="
```

In questo caso $a<b$, quindi $x$ è *NULL*!
Per la selezione multipla:

```
if n=1 then print("Lunedì")
        elif n=2 then print("Martedì")
        elif n=3 then print("Mercoledì")
        elif n=4 then print("Giovedì")
        elif n=5 then print("Venerdì")
        elif n=6 then print("Sabato")
        elif n=7 then print("Domenica")
end if;
```

Questo esempio visualizza il nome del mese corrispondente al numero.
Anche la gestione del ciclo (definito e indefinito) è supportata da Maple. Qualche esempio. La forma più semplice per quanto riguarda il ciclo definito:

```
for i from 6 by 3 to 10 do
        print(i)
end do;
```

```
6
9
```

In questo caso vengono visualizzati valori partendo da 6, sommando 3 (il passo) fino a 10, quindi arriva a 9. Un altro esempio è il seguente:

```
L:=[2,4,5];
for i in L do
    P:=i+3
end do;
```

```
[2,4,5]
5
7
8
```

Abbiamo un oggetto nuovo: la lista, all'indice *i* vengono sommati i valori contenuti nella lista, molto utile per introdurre la struttura dati vettore. Ancora esempi di utilizzo del **for**:

```
tot:=0:
for z in [1,x,y,q^2,3] do tot:=tot+z end do: tot
```

$$q^2+x+y+4$$

e

```
tot:=1:
for z in 1,x,y,q^2,3 do tot:=tot z end do: tot
```

$$3\,x\,y\,q^2$$

Anche il **while** è fra le istruzioni Maple, a desempio:

```
tot:=0;
for i from 11 by 2 while i<20 do
    print("i:=",i);
    tot:=tot+1;
end do;
```

```
tot:=0
"i:=",11
tot:=1
"i:=",13
tot:=2
"i:=",15
tot:=3
"i:=",17
tot:=4
```

*"i:=",19*
*tot:=5*

In questo caso: a *tot* viene sommato il valore di *i* (che parte da 11 e si incrementa di 2) fino a quando la condizione *i*<20 rimane falsa, esce quando diventa vera.

Maple ci dà la possibilità di strutturare "procedure" che potranno essere richiamate ed utilizzate in punti diversi e con valori diversi nel mio foglio di lavoro (worksheet). Prima di vedere come si imposta una procedura e come viene richiamata, alcune cose importanti:

a) una procedura si identifica con la parola chiave ***proc*** e con una assegnazione ad una variabile,

b) ***end proc*** chiude una procedura,

c) ***return*** mi dice cosa mi viene restituito successivamente alla chiamata,

d) possono essere definite ***variabili locali o globali***; le prime si utilizzano solo ed esclusivamente all'interno della procedura, sono variabili di "comodo", di "lavoro"; quelle globali hanno senso anche fuori dalla procedura. Per esempio:

```
Media:=proc(x,y,z)
local somma,M;
somma:=x+y+z;
M:=evalf(somma/3);
return (M);
end proc;
```

```
proc(x,y,z) local somma,M; somma:=x+y+z; M:=eval(1/3*somma); return M; end proc
```

E' stata definita una procedura *Media* che calcola la media aritmetica di tre valori; *x, y* e z assumono di volta in volta i valori che vengono "inseriti". A cosa serve la *evalf* (comando Maple)? Se non fosse stato inserito *evalf* il risultato sarebbe stato una frazione. Vediamo un esempio senza uso di evalf (in questo caso $M:=(x+y+z)/3$)

```
a1:=6
a2:=7
a3:=9
X:=Media(a1,a2,a3)
```

*6*
*7*
*9*
*22/3*

Da un punto di vista sintattico nulla da segnalare, ma il risultato è sotto forma di frazione; *evalf* serve a convertire la frazione nel numero decimale, con 9 cifre dopo la virgola, che la approssima meglio. Quindi si ottiene:

```
a:=5
b:=8
c:=6
Pippo:=Media(a,b,c)
```

*5*
*8*
*6*
*6.333333333*

dove è stato utilizzato un comando Maple all'interno di una procedura.

Concludendo Maple, oltre che essere uno strumento potentissimo e avanzato per l'apprendimento della matematica, apprendimento inteso non "come applicazione di formule", ma come approccio problematico, è sicuramente altrettanto adatto all'introduzione alla programmazione per studenti del primo biennio della secondaria di secondo grado.

## 6 Conclusioni

Gli studenti delle classi in cui in questi anni è stata adottata la metodologia integrata sopra illustrata dalla prima autrice nell'ambito del Progetto PP&S hanno manifestato grande entusiasmo nello svolgere le attività sia in classe sia a casa e in maniera naturale hanno sviluppato pensiero computazionale e acquisito competenze utili e spendibili nel mondo del lavoro. Questa esperienza è stata condivisa nella comunità dei docenti del PP&S offrendo interessanti spunti di riflessione metodologica e sicuramente può essere estesa in futuro in tutti gli istituti secondari di secondo grado che non prevedono l'insegnamento dell'informatica per colmare quel gap che la scuola italiana di oggi presenta.

## Riferimenti bibliografici


Brancaccio, A., Demartini, C., Marchisio, M., Pardini, C., Patrucco, A., Zich, R.: Il Progetto PP&S. Informatica a Scuola. Mondo Digitale, XIII, 51(3), 565-574, (2014).

1. Brancaccio, A., Demartini, C., Marchisio, M., Pardini, C., Patrucco, A.: Interazione dinamica tra informatica e matematica nel Problem Posing and Solving. Mondo Digitale, XIII, 51(3), 787-796, (2014).
2. Brancaccio, A., Marchisio, M., Palumbo, C., Pardini, C., Patrucco, A., Zich, R.: Problem Posing and Solving: Strategic Italian Key Action to Enhance Teaching and Learning Mathematics and Informatics in the High School. In: Proceedings - International Computer Software and Applications Conference 2,7273709, pp. 845-850. IEEE, Taiwan, (2015).
3. Calvani, A.: Che cos'è la tecnologia dell'educazione, Carocci, Roma (2004).
4. Calvani, A.: Rete, comunità e conoscenza. Costruire e gestire dinamiche collaborative, Erickson, Trento (2005).
5. Demartini, C., Bizzarri, G., Cabrini, M., Di Luca, M., Franza, G., Maggi, P., Marchisio, M., Morello, L., Tani, C.: Problem Posing (& Solving) nella Scuola Secondaria Superiore di II grado. Mondo Digitale, XIV, 58(3), 1-28, (2015).
6. Ente nazionale per la digitalizzazione della Pubblica Amministrazione: http://www.digitpa.gov.it
7. Frabboni, F.: Società della conoscenza e scuola, Erickson, Trento (2005).
8. European Commission: The European eGovernment Action Plan 2011-2015. Harnessing ICT to promote smart, sustainable & innovative Government. European Commission, Bruxelles (2010).
9. Legge Gelmini 133 del 2008 – legge 169/2008.
10. Legge Piano Nazionale Scuola Digitale, PNSD, 107/2015.
11. Maplesoft: www.maplesoft.com
12. Marconato, G.: Didattica e nuove tecnologie Ambienti di apprendimento, (2015), ITALY-DOCENTI-FOLIO-2015 A02 N07 – Ambienti di apprendimento PDF.pdf, ultimo accesso 5/03/18.
13. Obama, B.: Discorso sull'Hour of code, (2013).
14. Papert, S.: Bambini e adulti a scuola con il computer. Intervista, (1997) http://www.mediamente.rai.it/home/bibliote/intervis/p/papert.htm, ultimo accesso 5/03/18.
15. Raccomandazione del Parlamento Europeo e del Consiglio del 18/12/2006 http://eur-lex.europa.eu/legal-content/IT/TXT/?uri=celex%3A32006H0962, ultimo accesso 5/03/18.
16. Wing, J.: Computational Thinking. Communications of the ACM, 49(3), 33-35 (2006).