Progetto Nazionale "Informatica" del Piano Lauree Scientifiche

Bari, 13-15 ottobre 2023

CONVEGNO ITALIANO SULLA DIDATTICA DELL'INFORMATICA

[ITADINFO]

Metodi e Pratiche Didattiche Innovative
Ricerca Scientifica
Esperienze
Laboratori Formativi

SPONSOR

CON IL PATROCINIO DI

Gentile E., Monga M. (a cura di).
[**ITADINFO**] *Convegno Italiano sulla Didattica dell'Informatica*. Bari, 13-15 ottobre 2023.

Prima edizione: ottobre 2023

Edito da: Università degli Studi di Bari Aldo Moro
www.uniba.it

ISBN: 978-88-6629-075-9

Gli articoli qui presentati sono stati tutti oggetto di valutazione interna.

SESSIONE PLENARIA

INFANZIA

PRIMARIA

SECONDARIA DI PRIMO GRADO

SECONDARIA DI SECONDO GRADO

UNIVERSITÀ

ALTRI CONTRIBUTI

Sperimentazione del metodo PRIMM per l'insegnamento della programmazione*

Giulia Peserico[1], Maria Serafini[1], Francesca Voltolini[1], Federica Picasso[2], Daniele Agostini[2], Francesca Fiore[2], Anna Serbati[2], e Alberto Montresor[2]

[1] Liceo Scientifico "Leonardo da Vinci", Trento, Italy
{nome.cognome}@liceodavincitn.it
[2] Università di Trento, Trento, Italy
{nome.cognome}@unitn.it

1 Introduzione

Il progetto "Algoritmicamente: dal problem solving all'informatica" ha svolto un'attività di ricerca-azione nell'a.s. 2022-23 con l'obiettivo di sperimentare la metodologia PRIMM [6, 7] per l'insegnamento della programmazione nella disciplina Informatica dell'opzione "Scienze Applicate" del Liceo Scientifico. Il progetto è stato proposto dal Liceo "Leonardo da Vinci" di Trento, in collaborazione con l'Università di Trento e l'Associazione Culturale Glow, ed è stato finanziato dalla Fondazione Caritro.

L'idea di base della metodologia PRIMM (Predict, Run, Investigate, Modify, Make) parte dall'osservazione che l'insegnamento "tradizionale" della programmazione richiede fin da subito la scrittura di codice da parte dello studente. Questo differisce dalle metodologie utilizzate nell'apprendimento delle lingue (compresa la madrelingua), dove l'attività di produzione è preceduta dalla lettura e dalla comprensione del testo.

Partendo dall'osservazione che il carico cognitivo necessario per arrivare alla scrittura del codice è molto elevato, la metodologia PRIMM inverte la sequenza tradizionale, partendo prima dalla comprensione del testo e solo successivamente arrivando alla scrittura di codice vero e proprio. In particolare, l'approccio proposto è suddiviso in cinque attività:

- **Predict**: gli studenti leggono un segmento di codice, scritto in Python, e fanno previsioni su cosa farà il codice quando viene eseguito;
- **Run**: gli studenti eseguono il codice proposto nell'attività precedente, confrontando il comportamento effettivo rispetto alla previsione;
- **Investigate**: agli studenti è richiesto di analizzare più in profondità il codice o sue varianti, utilizzando varie tipologie di esercizi, come per esempio la correzioni di bug, l'annotazione del codice, l'uso di Parson Puzzle, domande esplorative, etc;
- **Modify**: agli studenti è chiesto di modificare il codice, partendo da variazioni molto semplici e poi con modifiche sempre più complesse;
- **Make**: infine, gli studenti creano un programma totalmente nuovo, ispirandosi al codice visto in precedenza, ma realizzando una nuova funzionalità o risolvendo un problema diverso.

*I materiali sono liberamente scaricabili (previa registrazione email) a questo indirizzo: https://fablab.unitn.it/sperimentazione-primm/

Dal punto di vista teorico, PRIMM si colloca nell'intersezione fra un approccio più strutturato basato sulla scoperta guidata e sull'istruzione diretta, e un approccio più costruzionista [5] basato sulla scoperta pura e su problemi aperti. Seguendo i suggerimenti di Grover et al., le cinque fasi partono da attività fortemente guidate per arrivare ad attività completamente libere, riducendo mano a mano il livello di scaffolding [3].

Per quanto possibile, molte delle attività si svolgono favorendo il lavoro di gruppo e lo scambio fra pari, seguendo l'idea che l'articolazione verbale delle soluzioni favorisce in generale l'apprendimento, in particolar modo nell'ambito della programmazione dove la componente linguistica è fondamentale.

Nel resto dell'articolo descriveremo il contesto in cui abbiamo operato, le attività svolte e i risultati della sperimentazione. Concluderemo con una riflessione sulla validità dell'approccio e su possibili miglioramenti da applicare negli anni successivi.

2 Descrizione del contesto

Il Liceo Scientifico "Leonardo Da Vinci" è composto da 75 classi per un totale di 1.530 alunni. Ci sono 42 classi per l'indirizzo scienze applicate e 33 per l'ordinamentale, con una media di 20 studenti per classe. L'informatica viene insegnata nell'opzione Scienze Applicate per due ore a settimana, dalla prima alla quinta classe; tuttavia, è dalla classe seconda che si inizia ad affrontare la programmazione utilizzando il linguaggio Python.

La sperimentazione didattica è stata applicata in cinque classi seconde del liceo (102 studenti), su un totale di nove, sotto la guida delle professoresse Giulia Peserico, Maria Serafini e Francesca Voltolini. Nelle altre quattro classi seconde (86 studenti), che fungono da classi di controllo, sono state adottate le metodologie didattiche tradizionali. La Tabella 1 contiene la lista delle classi coinvolte e di controllo, l'insegnante responsabile, il numero di studenti con la suddivisione di genere.

Per quattro delle classi coinvolte ciascuna attività aveva una durata di 100 minuti , mentre per la quinta classe le attività si svolgevano in due sessioni di 50 minuti ciascuna.

Nome classe	Docente	Orario	N° alunni	M	F
Primm1	Docente1	100'	23	13	10
Primm2	Docente1	100'	20	8	12
Primm3	Docente1	100'	25	14	11
Primm4	Docente2	100'	19	15	4
Primm5	Docente3	50'+50'	18	13	5
Controllo1	Docente4	50'+50'	21	16	5
Controllo2	Docente4	50'+50'	20	11	9
Controllo3	Docente5	50'+50'	21	13	8
Controllo4	Docente6	50'+50'	22	13	9

Tabella 1: Lista delle classi coinvolte, con docenti, orario e dimensione.

3 Descrizione delle attività

Per le cinque classi coinvolte nella sperimentazione sono state realizzate delle schede didattiche per introdurre i concetti di base della programmazione in linguaggio Python (variabili, tipi, operatori booleani, selezione, iterazione, liste, stringhe, uso della libreria Turtle).

La maggior parte delle attività proposte era strutturata in lavoro a coppie, spesso organizzate per gruppi di livello, e comprendeva:

- 3-4 proposte di *Predict*, che proponevano un nuovo concetto del quale si chiedeva di prevedere il funzionamento, seguite ognuna da una fase di *Run*, nella quale gli studenti copiavano ed eseguivano il codice per verificare la propria predizione;
- alcune domande esplorative, nelle quali si chiedeva di spiegare a parole quanto capito del nuovo concetto, confrontandolo anche con concetti precedentemente appresi;
- un codice all'interno del quale trovare alcuni errori (sintassi, esecuzione, semantica);
- un Parson's Puzzle;
- alcuni esercizi di modifica del codice (*Modify*);
- alcuni esercizi di scrittura del codice (*Make*).

Tutte le lezioni sono state svolte in laboratorio di informatica e ciascuno studente aveva un pc (con interprete Thonny [2]) a disposizione per il lavoro. Al termine di ciascuna attività venivano assegnati dei compiti a casa per studiare e ripassare quanto appreso in classe e alcune attività di Modify e Make.

Dopo le prime lezioni, dove sono stati introdotti i concetti di base, le schede sono state strutturate riducendo la parte *Predict-Run*, togliendo i Parson's Puzzle e aumentando le attività di *Modify* e *Make* durante le attività in coppia. È stato inoltre deciso di introdurre all'inizio di ogni scheda una breve spiegazione dei nuovi concetti e una sessione collettiva per riprendere quanto introdotto, in quanto si è rilevato che il lavoro a casa non sempre era svolto con la dovuta attenzione.

Durante l'anno si sono svolte delle verifiche sia di tipo strutturato (con domande a risposta multipla e aperta, domande simili alle attività di *Predict*, rilevazione dell'errore), sia di tipo pratico che consistevano nello scrivere 3-4 programmi per la soluzione di altrettanti problemi.

In fondo all'articolo riportiamo, come esempio, la scheda utilizzata per la spiegazione del costrutto *if semplice (a una via)*. Nella prima fase della scheda sono stati proposti tre esercizi di *Predict*, chiamati *"Cosa fa il codice?"*, ognuno con le relative domande di approfondimento (Figure 1, 2, 3). É stato quindi inserito un semplice esercizio di ricerca degli errori (Figura 4) seguito da un Parson's Puzzle (Figura 5 e la sua soluzione 6). Infine, dopo aver schematizzato e rivisto con gli studenti i termini chiave (Figura 7), si è passati alle fasi *Modify* e *Make* (Figure 8 e 9).

4 Risultati

Per verificare l'efficacia del metodo PRIMM, sono stati raccolti tre tipi di dati sulle cinque classi sperimentali e le quattro classi di controllo:

- Il primo tipo di dati riguarda la motivazione dello studente a svolgere questo tipo di attività. Lo strumento utilizzato per raccogliere questo dato è stato l'Intrinsic Motivation Inventory (IMI) [4], tradotto ed adattato, utilizzando solo i moduli che si sono ritenuti utili per questa sperimentazione, ovvero: Interesse/appagamento, Sforzo/importanza, Percezione di competenza, Valore/Utilità, Pressione/Tensione.
- Il secondo tipo di dati riguarda l'apprendimento di conoscenze e abilità riguardanti la programmazione ed è stato rilevato tramite un test accuratamente preparato.

- Il terzo tipo di dato include la classe di appartenenza, il genere e i voti finali nelle varie discipline scolastiche.

I primi due tipi di dato sono stati rilevati secondo un disegno quasi-sperimentale Pre-Post, cioè sono stati rilevati prima dell'inizio della sperimentazione e dopo la fine per rilevare gli eventuali effetti e le differenze tra gruppo sperimentale e di controllo. Gli esiti dell'analisi, per la quale è stata utilizzata una regressione lineare con errori standard robusti per tenere in conto l'eterogeneità dei gruppi, evidenziano i seguenti risultati:

- Non esiste una correlazione statisticamente significativa tra l'appartenenza al gruppo sperimentale e un incremento del punteggio del test. Nonostante questo, gli incrementi dei punteggi dei test del gruppo sperimentale sono lievemente più alti di quelli di controllo.
- Esiste una correlazione statisticamente significativa tra l'appartenenza al gruppo sperimentale e un incremento del punteggio IMI Sforzo/importanza dell'attività. La metodologia PRIMM quindi potrebbe promuovere l'impegno degli studenti nelle attività e la comprensione della loro importanza.
- Esiste una correlazione fortemente significativa tra l'appartenenza al gruppo sperimentale e un incremento del punteggio IMI riguardante la Percezione di competenza. La metodologia PRIMM quindi potrebbe favorire la Percezione della propria competenza e la sicurezza degli studenti nei compiti di programmazione. Gli studenti potrebbero inoltre essere in grado di autovalutarsi in modo più preciso.
- Gli studenti con il miglioramento più marcato del punteggio del test finale rispetto a quello iniziale hanno ottenuto voti più alti in informatica.
- L'appartenenza al gruppo sperimentale e quindi la metodologia PRIMM potrebbe avere un impatto positivo sui voti di informatica; il risultato dell'analisi si avvicina molto alla significatività statistica.
- È stata inoltre esplorata la differenza di genere nei risultati. Nonostante non ci siano risultati conclusivi, l'analisi rileva, nel gruppo di controllo, uno sbilanciamento maggiore nell'interesse da parte degli studenti maschi rispetto alle studentesse, suggerendo che gli studenti maschi nel gruppo di controllo potrebbero essere stati più interessati dalle attività didattiche rispetto alle studentesse. Questo indicherebbe che la metodologia PRIMM potrebbe essere più inclusiva rispetto a quella tradizionale.

5 Riflessioni e conclusioni

La prima iterazione di questa ricerca-azione ha prodotto risultati promettenti, ma suggerisce anche la necessità di perfezionare il campione e alcune metodologie e procedure. Per esempio, alcuni item del test sembrano necessitare di una verifica perché non danno risultati consistenti con tutti gli altri indicatori. Inoltre, molte attività sono state svolte con gruppi suddivisi per livello dello studente. Questo potrebbe avere influito sul fatto che gli studenti con maggiore miglioramento nel punteggio del test abbiano ottenuto anche voti più alti in informatica. Nelle future iterazioni i gruppi verranno suddivisi in un'ottica di peer tutoring [1]. In generale, l'adozione del metodo PRIMM sembra promettente, e già da ora è chiaro che ha un notevole e positivo impatto sull'approccio degli studenti alla programmazione, in particolare sul proprio senso di competenza, sull'impegno e l'importanza attribuita alle attività proposte, ma sono necessarie ulteriori ricerche e ripetizioni per ottimizzare la sua efficacia.

Alla fine dell'anno abbiamo chiesto agli studenti dei commenti sulla modalità di lavoro PRIMM sperimentata. Un folto gruppo di studenti ha lamentato la mancanza di lezioni frontali "classiche" con spiegazione dell'insegnante prima di affrontare attività di laboratorio; possiamo ipotizzare che questo dipenda da un percorso scolastico (italiano) molto incentrato sulla lezione frontale e che gli studenti non abbiano dimestichezza con approcci innovativi. Alcuni studenti inoltre hanno affermato che il lavoro a coppie per livello non è stato produttivo e avrebbero desiderato essere a coppie con compagni più bravi, che li aiutassero maggiormente nel lavoro. Infine, un gran numero di studenti ha comunque rilevato che questa modalità di lavoro ha permesso loro di "andare al proprio passo" svolgendo le schede assegnate senza la fretta del lavoro collegiale, e potendole poi ripercorrere a casa. Tutto sommato il lavoro è stato apprezzato dagli studenti e verrà riproposto anche in futuro.

Riferimenti bibliografici

[1] F. J. Alegre Ansuategui and L. Moliner Miravet. Emotional and cognitive effects of peer tutoring among secondary school mathematics students. *International Journal of mathematical education in science and technology*, 48(8):1185–1205, 2017.

[2] Aivar Annamaa. Thonny: A Python IDE for learning programming. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE '15, page 343. ACM, 2015.

[3] S. Grover, R. Pea, and S. Cooper. Designing for deeper learning in a blended computer science course for middle school students. *Computer Science Education*, 25(2):199–237, 2015.

[4] E. McAuley, T. Duncan, and V. V. Tammen. Psychometric properties of the intrinsic motivation inventory in a competitive sport setting: A confirmatory factor analysis. *Research quarterly for exercise and sport*, 60(1):48–58, 1989.

[5] Seymour Papert. *Mindstorms: Children, Computers, and Powerful Ideas*. Basic Books, Inc., USA, 1980.

[6] Sue Sentance, Jane Waite, and Maria Kallia. Teachers' experiences of using primm to teach programming in school. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 476–482, Minneapolis, USA, 2019. ACM.

[7] Sue Sentance, Jane Waite, and Maria Kallia. Teaching computer programming with primm: a sociocultural perspective. *Computer Science Education*, 29(2-3):136–176, 2019.

Lavoro in coppia

Di seguito c'è del codice scritto in Python:

```
print("Benvenuto nel mio programma di conversazione")
print()
print("Ti piace andare in bicicletta? Rispondi si o no")
risposta = input()
if risposta == "si":
    print("Molto bene! Ti terrai in forma.")
print("Ciao ciao.")
```

Secondo voi, che cosa appare sul monitor all'esecuzione?
Ora scrivete il codice in Thonny e controllate. Fa quello che vi aspettavate? Se non lo fa, descrivete nel dettaglio cosa accade in modo diverso.

Rispondete alle seguenti domande:

1. Cosa succede se si digita "Si" invece di "si" quando si esegue il programma? (provate e verificate)
2. Qual è la differenza tra = e == ? In questo programma vengono utilizzati entrambi?

Figura 1: Predict - 1

Lavoro in coppia

Di seguito c'è del codice scritto in Python:

```
print("Come è andata la verifica?")
voto = float(input("Scrivi il voto che hai preso: "))
if voto > 6:
    print("Bene. Hai preso di più della sufficienza!")

print("Alla prossima!")
```

Secondo voi, che cosa appare sul monitor all'esecuzione?
Ora scrivete il codice in Thonny e controllate. Fa quello che ti aspettavate? Se non lo fa, descrivete nel dettaglio cosa accade in modo diverso.

Rispondete alle seguenti domande:

3. Cosa accadrebbe se usassimo il controllo `if voto >= 6` ?
4. Cosa accadrebbe se usassimo il controllo `if voto < 6` ?
5. Cosa accadrebbe se usassimo il controllo `if voto != 6` ?

Figura 2: Predict - 2

Lavoro in coppia

Di seguito c'è del codice scritto in Python:

```
print("Come è andata la verifica?")
voto = float(input("Scrivi il voto che hai preso: "))
if voto > 6:
    print("Bene. Hai preso più della sufficienza!")
    risposta = input("Sei contento del voto? Rispondi si o no")
    if risposta=="si":
        print("Ottimo! Anch'io sono contento per te!")
```

Secondo voi, che cosa appare sul monitor all'esecuzione?
Ora scrivete il codice in Thonny e controllate. Fa quello che ti aspettavate? Se non lo fa, descrivete nel dettaglio cosa accade in modo diverso.

Rispondete alle seguenti domande

6. Avete notato che il secondo `if` è spostato più internamente? Cosa vuol dire?
7. Se noi avessimo messo il secondo `if` allineato al primo, quale sarebbe stato l'output? (prova a modificare il codice e vedere su Thonny il risultato.)

Figura 3: Predict - 3

Lavoro in coppia

In questo programma ci sono **2 errori**, riesci ad individuarli? Evidenziali con un colore. *(Se non riesci a trovare alcuni errori puoi aiutarti copiando il codice in Thonny e verificandone il funzionamento)*
print("Ciao! sono un po' curioso.") risposta = input("Ti piace la pizza?") if risposta > "si"   print("Bene. Allora possiamo fare una serata pizza e cinema.") print("A presto.")

Figura 4: Trova gli errori

```
if (somma >= 10):
print("la somma e' compresa tra 10 e 15")
addendo1 = int(input("Inserisci il primo addendo: "))
somma = addendo1 + addendo2
print("la somma e' maggiore o uguale a 10")
addendo2 = int(input("Inserisci il secondo addendo: "))
print("Ciao! Facciamo il controllo della somma.")
if (somma <= 15):
print("Ciao, alla prossima!")
```

Figura 5: Parson's Puzzle

```
print("Ciao! Facciamo il controllo della somma.")
addendo1 = int(input("Inserisci il primo addendo: "))
addendo2 = int(input("Inserisci il secondo addendo: "))
somma = addendo1 + addendo2
if (somma >= 10):
    print("la somma e' maggiore o uguale a 10")
    if (somma <= 15):
        print("la somma e' compresa tra 10 e 15")
print("Ciao, alla prossima!")
```

Figura 6: Parson's Puzzle - soluzione

TERMINE	SIGNIFICATO	PYTHON
Operatore relazionale	Un simbolo che serve per fare un confronto tra due dati.	>, >=, <, <=, ==, != sono gli operatori relazionali di Python
Condizione	Il test che inseriamo nel costrutto *if*. Si costruisce inserendo un *operatore relazionale* tra due dati. Il risultato della condizione deve essere vero (**True**) o falso (**False**)	risposta == "sì" a > 5 b != 7 10 >= x b == a
Selezione	Quando nel codice c'è un punto in cui viene effettuata una scelta e viene utilizzato il costrutto *if* per creare percorsi alternativi	`if a > 10:` `  print("ciao")`

Indentare	Scrivere una o più istruzioni "rientrate" (di 4 spazi) rispetto alle istruzioni precedenti/successive	`if a > 10:`
		`  print("ciao")`
		L'istruzione print è **indentata** rispetto a if

Annidare	Scrivere una selezione dentro un'altra selezione	`if numero > 5:`
		`  if numero < 10:`
		`  print("numero tra 5 e 10")`

Figura 7: Termini chiave

1. Cosa stampa il seguente codice?

```
numero = int(input("inserisci un numero"))
if numero%3==0:
    print("il numero va bene")
```

   Ricopia il codice per accertarti di aver risposto correttamente e modificalo in modo tale che venga controllato se un numero è divisibile per 5.

2. Modifica il codice dell'esercizio precedente per verificare se dati due numeri (interi) inseriti dall'utente, il primo è multiplo del secondo.

3. Cosa stampa il seguente codice?

```
lato1  = float(input("inserisci la misura del primo lato"))
lato2  = float(input("inserisci la misura del secondo lato"))
if lato1==lato2:
    print("il triangolo è isoscele")
```

   Ricopia il codice per accertarti di aver risposto correttamente e modificalo in modo tale che venga verificato se il triangolo è equilatero.

4. Cosa stampa il seguente codice?

```
prezzo  = float(input("inserisci il prezzo del biglietto"))
numero  = int(input("inserisci il numero di biglietti acquistati"))
if numero > 20:
    numero = numero-1
totale = numero * prezzo
print("il totale da pagare è", totale)
```

   Ricopia il codice per accertarti di aver risposto correttamente e modificalo in modo tale che venga regalato un biglietto omaggio **ogni** 20 biglietti (se si acquistano 30 biglietti se ne pagano 29, acquistandone 50 se ne pagano 48, ecc).

Figura 8: Modify

5. Calcolare il totale speso per acquistare delle pesche sapendo che se si superano i 10 euro si ha il 20% di sconto.

6. Calcolare il costo della bolletta di consumo del gas (espresso in metri cubi) ottenuto dalla somma delle seguenti voci:
   - quota fissa pari a 20€
   - 0,575€ al metro cubo per i primi 500 metri cubi
   - 0,783€ al metro cubo per ogni metro cubo eccedente i primi 500

7. Comperando un'automobile, se spendo più di 20.000 € ho uno sconto del 10%. Indipendentemente dal prezzo, se pago in contanti uno sconto di 1.000. Visualizzare quanto si spende.

Figura 9: Make