Esercitazioni di Metodologie e Tecnologie per la Didattica dell'Informatica

Laura Ventrice e Edorardo Rolfo

Anno Accademico 2022/2023

Indice

1 Parte introduttiva: Cos'è l'Informatica?

- E' importante insegnare informatica come materia scolastica?

  Si, secondo l'articolo "Informatica e competenze digitali: cosa insegnare?"[2], l'informatica è importante che venga insegnata nelle scuole, in quanto competenza scientifica che permette di approcciare l'elaborazione dei dati in modo automatico e non.
  Come suggerito nell'articolo precedentemente menzionato: a seguito della rivoluzione industriale si è reso necessario modificare ciò che veniva insegnato ai cittadini e, citando l'articolo, "...sono state inserite nelle scuole le discipline scientifiche che ne spiegavano i princìpi scientifici alla base...". Quindi, in maniera analoga per poter interpretare ciò che ci circonda nella quotidianità ed avere una migliore comprensione della società digitale si rende necessario formare i cittadini sull'Informatica, con l'obiettivo di comprenderla meglio e permetterne un uso più consapevole. Nell'articolo inoltre si sottolinea la tendenza nella formazione italiana di confondere l'informatica con l'uso di tecnologie che manipolano dati in forma numerica, che sono in realtà competenze digitali e non scientifiche, anch'esse importanti per imparare ad utilizzare gli strumenti digitali nel mondo odierno.

- L'informatica è una scienza?

  L'informatica è una scienza, ci sono diversi articoli che trattano dell'argomento. Le discussioni a riguardo si basano sul dimostrare che l'informatica non sia una scienza, con la tesi che sia una disciplina che non ha come oggetto fenomeni naturali ma antropici, quindi artificiali e creati dall'uomo. Questa argomentazione veniva specialmente portata avanti durante un periodo in cui effettivamente si era perso il focus scientifico dell'informatica e ci si era concentrati più sull'aspetto ingegneristico.
  Come menzionato nell'articolo "Is Computer Science a Science?" [1] i giudizi riguardanti questa tematica sembra dipendano dalla tradizione in cui gli scienziati crescono, tuttavia diversi scienziati di altri settori invece supportato l'idea che l'informatica sia una scienza, facendo notare come anche le loro discipline includano la gestione di informazioni, successivamente definite come informazioni naturali. In tal senso l'informatica si concentra effettivamente su processi informativi riguardanti sia informazioni naturali che artificiali, in quanto studia come poter analizzare le informazioni e solo eventualmente digitalizzarle ed automatizzare tali processi. Altro argomento a supporto della tesi secondo la quale l'Informatica sia una scienza, è il fatto che questa disciplina segua il paradigma scientifico definito da Bacon,questo paradigma scientifico si basa sul processo di generazione di ipotesi e di validazione tramite esperimenti e in Informatica questa metodologia viene applicata allo studio dei processi informativi. Alcuni esempi menzionati in [1], riguardano i ricercatori di ingegneria del software che ipotizzano nei loro studi modelli che si concentrano

sulla programmazione e sui difetti che possono insorgere. L'efficacia di questi modelli viene testata tramite fasi di validazione, utili anche per comprenderne meglio l'utilizzo e sfruttarli in modo più efficace nella creazione di programmi con sempre meno difetti.

- Qui sotto trovate una lista di criteri estratti dall'articolo *"The Science in Computer Science"*. Sono usati dall'autore per definire la credibilità di un settore come "scienza". Siete d'accordo con la scelta di questi criteri? Motivare le risposte:

  – Organized to understand, exploit, and cope with a pervasive phenomenon

    La scienza, intesa come disciplina generale, nasce per provare a spiegare la realtà che ci circonda. Per poter sfruttare al meglio la conoscenza relativa ai fenomeni studiati è necessario organizzare le informazioni ottenute dal loro studio. Una buona organizzazione delle conoscenze si traduce in un accesso facilitato ai loro contenuti, permettendo uno studio da parte di una platea più ampia e di conseguenza ad una comprensione più profonda degli argomenti trattati. Soltanto tramite una profonda comprensione delle conoscenze è possibile imparare a sfruttarne le dinamiche che li governano ed elaborare metodologie per interagire efficacemente con essi.

  – Encompasses natural and artificial processes of the phenomenon

    Lo studio delle scienze nasce per poter comprendere a pieno i processi di fenomeni pervasivi, anche per poter spiegare processi ancora ignoti. Inoltre la tecnologia e le conoscenze sviluppate durante lo studio, possono essere successivamente usate per applicazioni artificiali o anche in altri campi di studio.

  – Codified structured body of knowledge

    Come menzionato nel primo punto, la conoscenza derivante da una scienza deve essere organizzata e strutturata.

  – Commitment to experimental methods for discovery and validation

    Per regolarizzare lo studio e le scoperte in ambito scientifico è stato introdotto il metodo sperimentale. Si tratta di un procedimento per formare e controllare la conoscenza scientifica che si basa su esperimenti, in particolare l'idea è che gli scienziati non siano passivi in questo processo ma che contribuiscano alla conoscenza comune in maniera attiva. Questo tipo di metodologia oltre che essere utile per provare che le scoperte abbiano

un valore dal punto di vista scientifico, fa aumentare la credibilità della disciplina, in quanto ciò che viene definita come conoscenza potrà essere osservata in certe condizioni e sarà riproducibile.

- Reproducibility of results
  La non riproducibilità dei risultati porta all'inaffidabilità di quest'ultimi, è quindi necessario che questi siano riproducibili per essere ritenuti accettabili.

- Falsifiability of hypotheses and models
  Una teoria è scientifica solo se è in grado di suggerire quali esperimenti e osservazioni potrebbero dimostrarla falsa. Dal punto di vista logico, questo tipo di procedimento segue la regola di inferenza del *modus tollens*. Inoltre questo principio nasce per separare l'ambito delle teorie controllabili, che appartiene alla scienza, da quello delle teorie non controllabili, da Popper stesso identificato con la metafisica. Se una teoria non soddisfa questo principio, non è possibile controllare la validità del suo contenuto informativo relativamente alla realtà che presume di descrivere.

- Ability to make reliable predictions, some of which are surprising
  Lo studio di un fenomeno porta alla comprensione dei suoi comportamenti ed alla capacità fare delle previsioni riguardanti essi.

- L'informatica soddisfa qualcuno dei criteri sopra elencati? Se si quali? E perchè?

  L'informatica soddisfa i criteri precedentemente elencati, in particolare:

  - Organizza per comprendere, sfruttare e affrontare un fenomeno pervasivo:
    Alcuni esempi in ambito informatico possono essere la diffusione di conoscenza tramite video, o rendere facile l'accesso e l'utilizzo di conoscenza condivisa come ad esempio enciclopedie. In altre parole i fenomeni pervasivi vengono individuati nell'output dei sistemi informatici e in tutto ciò che concerne l'automatizzazione della risoluzione di problemi.
  - Comprensione di processi naturali e artificiali di fenomeni:
    L'informatica è una disciplina che costantemente crea legami con altre discipline, ognuno dei quali crea una nuova disciplina. Alcuni esempi sono la bioinformatica e le scienze cognitive. La comprensione di processi avviene spesso tramite questa multidisciplinarità, alcuni esempi sono le reti neurali, in cui proprio progettando le reti in un certo modo si è scoperto che gli esseri umani hanno struttura neurale differente, o le scienze

cognitive, in cui si cerca di produrre sistemi che abbiano anche uno scopo esplicativo rispetto al modello reale di riferimento.

- Codifica strutturata per il corpo della conoscenza: In informatica vengono utilizzate le strutture dati per poter strutturare la conoscenza.
- Impegno nei metodi sperimentali per la scoperta e la validazione:

  Alcuni esempi tratti dall'articolo [1] sono:

  * i ricercatori cognitivi che ipotizzano che un comportamento intelligente è il risultato di processi informativi nel cervello e nel sistema nervoso. A partire da questa ipotesi costruiscono sistemi e confrontano i processi con quelli reali;
  * gli algoritmici sperimentali studiano le performance di algoritmi reali su dataset reali e formulano modelli per predirre i loro requisiti in termini di memoria e di tempo per produrre teorie sempre più accurate sulla complessità.
- Riproducibilità di risultati: un algoritmo deve produrre output sempre identici se le condizioni di input non vengono modificate. Ad esempio i software sono riproducibili.
- Falsificabilità di ipotesi e modelli:

  Nonostante in passato si sia notato che circa la metà degli articoli scientifici di informatica non avessero una parte di verifica dei modelli proposti [1], si può menzionare che ci sono parti di informatica dedicate alla verifica dei programmi e alla loro correttezza.
- Capacità di fare previsioni affidabili, alcune delle quali sono sorprendenti:

  Ci sono diversi esempi in molti ambiti, alcuni che si possono menzionare sono in ambito di computazione come ad esempio algoritmi ottimi per risolvere problemi comuni e l'identificazione di problemi comuni che sono intrattabili.

2 I numeri binari

1. Qual è il tema/concetto informatico oggetto dell'attività?

   Codifica e decodifica dell'informazione, nello specifico la rappresentazione dei numeri in binario.

2. Quali sono gli obiettivi formativi? Per definirli usare la *Proposta di Indicazioni Nazionali per l'insegnamento dell'Informatica nella Scuola.*

   Gli obiettivi formativi di questa attività sono:

   - **Conoscenze**: imparare a riconoscere la numerazione binaria (O-P5-D-1) [1]
   - **Abilità**: utilizzare la rappresentazione binaria come rappresentazione alternativa a quella decimale (O-M-P-2) [2]
   - **Competenze**: saper utilizzare le conoscenze acquisite riguardanti la rappresentazione binaria dei numeri anche per altre applicazioni, come la codifica delle lettere o per l'ottenimento di numeri più grandi (O-M-D-1, O-P3-A-2) [3]

[1] O-P5-D-1 - Utilizzare combinazioni di simboli per rappresentare informazioni familiari complesse

[2] O-M-D-2 - Riconoscere se due rappresentazioni alternative semplici della stessa informazione sono intercambiabili per i propri scopi

[3] O-M-P-1 - Sperimentare piccoli cambiamenti in un programma per capirne il comportamento, identificarne gli eventuali difetti, modificarlo O-P3-A-2 - Comprendere che problemi possono essere risolti mediante la loro scomposizione in parti più piccole.

3. Suddividere l'attività in fasi e per ogni fase individuare snodi e indicatori

Fase	Snodi	Indicatori
Introduzione al gioco: spiegazione delle regole base del gioco ed assegnazione di numeri casuali (piccoli) da provare a convertire	1 Comprensione della corrispondenza tra il numero di punti con il corrispondente decimale. 2 Comprensione della relazione tra la posizione della carta ed il suo valore. 3 Comprensione del funzionamento dei lati delle carte (scoperto/fronte on, coperto/retro off).	1 L'allievo riconosce il valore indicato su ciascuna carta (singolarmente). 2 L'allievo comprende che ogni carta ha valore doppio di quella precedente, ovvero che il valore della carta successiva è pari al doppio della precedente. I numeri dispari hanno sempre la prima carta scoperta, mentre i pari coperta. 3 L'allievo riesce a coprire/scoprire le carte in base ad un numero binario fornitogli.
Imparare a contare: con un numero limitato di carte chiedere di contare fino al massimo numero ottenibile, descrivendo i passaggi eseguiti	1 Capire il range di numeri ottenibili con un determinato numero di carte. 2 Conteggio: partire da 0 ed ottenere tutti i numeri ottenibili.	1 Date diverse quantità di carte, gli alunni riescono a capire il numero massimo ottenibile con esse. 2 Lo studente è in grado di descrivere il procedimento di conteggio con poche carte. 3 L'allievo riesce ad applicare il procedimento descritto per il conteggio.
Numeri grandi (>31?): come si possono ottenere numeri grandi	1 Capire come l'aggiunta di nuove carte possa aumentare la quantità di numeri rappresentabili. 2 Capire di quanto il valore massimo viene aumentato per ogni carta aggiunta. 3 Capacità di ottenimento di numeri arbitrariamente grandi.	1 Lo studente è in grado di riconoscere se un valore è rappresentabile a partire da un numero di carte prefissato. 2 Lo studente è in grado di riconoscere se un valore è rappresentabile tramite un numero di carte prefissato e, nel caso così non fosse, quante carte aggiungere per renderlo possibile. 3 Riuscire a scrivere numeri arbitrariamente grandi da binario a decimale e viceversa.

4. Quali ingredienti delle varie teorie/metodologie viste nelle lezioni precedenti trovate in questa attività?

   L'attività proposta pone l'insegnante in un ruolo marginale con il compito di spiegare l'attività agli studenti ed aiutarli a raggiungere autonomamente le risposte corrette ai problemi posti. Gli studenti sono quindi incoraggiati a partecipare attivamente al lavoro e non essere più rilegati al ruolo di "ricettore di informazioni". Il ruolo passivo dell'insegnante e lo svolgimento delle attività, che sono incentrate sugli studenti, sono metodologie definite dall'**apprendimento attivo**. La suddivisione dell'attività di apprendimento in diverse fasi, con difficoltà graduali ed applicazioni leggermente differenti, rientra anche nella strategia intitolata **Learning Cycle Instructional Models**, in cui l'apprendimento di un argomento viene suddiviso in 5 fasi con difficoltà ed applicazioni differenti.

5. Riuscite a individuare nel testo dell'attività suggerimenti per il/la docente? Secondo voi quali altre indicazioni devono essere integrate volendo rendere il documento una guida "completa" rispetto a snodi e indicatori?

   Suggerimenti:
   Porre più enfasi sulle operazioni di conversione vera e propria.
   Un approccio prettamente visuale come quello del gioco è ottimo per l'introduzione al sistema binario, ma rischia di rendere gli alunni troppo dipendenti dalla presenza delle carte, facendogli fare troppo affidamento a ciò che viene rappresentato sulle carte anziché apprendere il meccanismo riguardante la generazione dei numeri ottenuti.
   Creare una fase intermedia in cui le carte scoperte non presentano più il valore decimale direttamente, ma soltanto un simbolo per indicare che quella carta deve essere utilizzata durante il conteggio, se fossero solo bianche sarebbe sufficiente. In questo modo si rende più facile il passaggio da gioco ad un utilizzo reale effettivo del meccanismo appreso, ovvero l'utilizzo di una versione "pura" della notazione binaria.

   Indicazioni:
   Nelle attività proposte avviene un passaggio diretto tra la conversione di numeri piccoli a quella dei numeri grandi, con l'aggiunta di carte ed un aumento di difficoltà, senza affrontare il conteggio vero e proprio.
   È importante far contare gli alunni fino al massimo con un numero ristretto di carte, in modo tale da mostrare il ragionamento/meccanismo dietro il conteggio, raggiungendo così un momento in cui bisogna capire come poter procedere oltre il massimo ottenibile con quelle carte.
   L'aggiunta di una nuova carta comporta il passaggio in cui tutte quelle precedenti vengono coperte, passaggio che potrebbe non essere così intuitivo ad una persona non famigliare al meccanismo di conta in binario.

3 Problem solving: design della soluzione

3.1 Definizione di Algoritmo

Definizione: elenco di passaggi da eseguire in un determinato ordine ideati per il raggiungimento di un obiettivo definito chiaramente in principio. Ciascuno di questi passi deve descrivere delle azioni in modo dettagliato, semplice da comprendere e senza ambiguità; l'insieme di queste azioni descrive in modo completo la procedura, specificando inoltre le condizioni di inizio e terminazione della stessa.
Un algoritmo deve essere caratterizzato dalle seguenti proprietà:

- *Finitezza*: l'algoritmo deve avere un inizio ed una fine ben precisi
- *Precisione*: le azioni da eseguire devono essere ben definite, precise e che non lascino spazio ad ambiguità
- *Buona definizione dei dati* inseriti in ingresso e voluti in uscita
- *Fattibilità*: la realizzazione di un algoritmo deve essere possibile

4 Progettazione soluzione/algoritmo

4.1 Commento sull'articolo [3]

L'utilizzo dei pattern, come quelli proposti dall'articolo, può essere di grande aiuto per fornire una base agli studenti ancora inesperti nella programmazione. Tuttavia l'idea è che, nonostante la validità dell'approccio proposto, fare troppo affidamento sui pattern possa poi rivelarsi controproducente.
Una delle capacità fondamentali di un buon informatico è quella del **problem solving**, se si abitua troppo uno studente a fare affidamento a pattern già prefabbricati c'è il rischio che questo, una volta presentatogli un problema fuori dalle righe o più complesso del solito, si ritrovi a non sapere come muoversi per raggiungere una possibile soluzione in quanto non in grado di astrarre il problema e individuare metodi efficaci per risolverlo senza ricorrere a conoscenze pregresse. Inoltre questa metodologia rende lo studente limitato nella fantasia per poter risolvere un problema, nonostante ci siano diverse soluzioni per uno stesso problema. Un'eventuale adozione di questa metodologia dovrebbe essere affiancata allo sviluppo da parte degli studenti di buone capacità di scomposizione dei problemi in sotto-problemi, su cui solo successivamente applicare dei pattern.

4.2 Problema analizzato per l'attività

Consegna dell'esercizio

Bob è molto allergico alle uova e ama molto andare al ristorante. Per non perdere tempo a leggere tutto il menù ti ha chiesto di realizzargli un programma che, presi in input una lista di N le pietanze così definite "nome, ingrediente, ingrediente \n", stampi i nomi delle pietanze che può ordinare e il loro numero totale.

Esistono soluzioni alternative tra di loro?

Si possono adottare almeno 2 soluzioni differenti:

1. Utilizzare una lista per tenere traccia dei cibi senza uova, stampare infine il contenuto della lista e la relativa lunghezza solo alla fine della scansione della lista di pietanze;

2. Ogni volta che una pietanza non contiene uova stamparla ed utilizzare un contatore per tenere traccia delle pietanze idonee.

Scomporre eventualmente il problema in sotto-problemi

Il problema è suddivisibile in sotto-problemi:

- Tracciamento delle pietanze idonee
- Suddivisione della stringa del menù nelle tre parti
- Conteggio delle pietanze idonee
- Stampa finale

Soluzioni in pseudocodice

Le soluzioni proposte sono presenti negli pseudocodici 1 e 2.

Algorithm 1: Prima soluzione proposta

```
Data: Lista_pietanze: contiene le pietanze del menù
Result: Stampa le pietanze che non contengono uova e il numero delle
        pietanze
risultato ← lista vuota ;
for pietanza in Lista_pietanze do
    contenuto ← dividere la stringa pietanza usando ',' come separatore;
    egg ← False ;          /* memorizza se le uova sono contenute */
    for elemento in contenuto e egg è falso do
        if elemento contiene egg then
            egg ← True
        end
    end
    if egg è falso then
        aggiungo pietanza in risultato ;
    end
end
Stampa il contenuto di risultato ;
n_pietanze ← lunghezza risultato ;
Stampa n_pietanze ;
```

Algorithm 2: Seconda soluzione proposta

```
Data: Lista_pietanze: contiene le pietanze del menù
Result: Stampa le pietanze che non contengono uova e il numero delle
        pietanze.
risultato ← 0 ;
for pietanza in Lista_pietanze do
    contenuto ← dividere la stringa pietanza usando ',' come separatore;
    egg ← false ;          /* memorizza se le uova sono contenute */
    for elemento in contenuto e egg è falso do
        if elemento contiene egg then
            egg ← True
        end
    end
    if egg è falso then
        incremento risultato di 1;
        Stampa pietanza ;
    end
end
Stampa risultato ;
```

Valutazione della soluzione

Pattern elementari

È possibile identificare 3 pattern elementari utilizzati nelle soluzioni proposte:

1. **Process-All-Item:** per la valutazione di tutte le pietanze
2. **Process-Items-Until-Done:** la valutazione degli ingredienti viene portata avanti solamente finché non viene rilevata la presenza di uova
3. **Guarded-action:** valuta la presenza delle uova alla fine della valutazione dei singoli ingredienti

Pattern algoritmici

I pattern algoritmici identificati sono i seguenti:

1. Scorrimento di tutti gli elementi contenuti in un elenco e applicazione di elaborazione sui dati presi in considerazione (scorrere tutte le pietanze ed ottenere i corrispondenti ingredienti)
2. Tracciamento di tutti gli elementi appartenenti ad un elenco e che rispettano una determinata proprietà (tracciamento delle pietanze con uova)

Variabili

- **Prima soluzione:**
  - **3 container:** uno per la lista di tutte le pietanze, uno per la lista delle pietanze senza uova, uno per gli ingredienti di una pietanza (contenuto).
  - **2 temporanee:** controllore per la presenza di uova (egg) e variabile n_pietanze per la lunghezza del risultato.
  - **2 attraversatori:** per lo scorrimento dei container.
- **Seconda soluzione:**
  - **2 container:** uno per la lista di tutte le pietanze, uno per gli ingredienti di una pietanza (contenuto).
  - **1 temporanea:** controllore per la presenza di uova (egg).
  - **1 contatore:** per contare il totale di pietanze senza uova (risultato).
  - **2 attraversatori:** 2 per lo scorrimento dei container.

5 Notional machine e Misconceptions

5.1 Classificazione misconception

Le misconception sotto elencate sono riportate come elencate e descritte nel documento d'origine [5].
Misconception di tipo **sintattico**:

1. **No. 4:** The system does not allow unreasonable operations.
2. **No. 7:** The machine understands english.
3. **No. 11:** Primitive assignment works in opposite direction.
4. **No. 14:** A variable is (merely) a pairing of a name to a changeable value (with a value). It is not stored inside a computer.
5. **No. 16:** Assignment moves a value from a variable to another.

Misconception di tipo **concettuale**:

1. **No. 2:** The computer is able to deduce the intention of the programmer.
2. **No. 5:** Difficulties understanding the lifetime of values.
3. **No. 9:** A variable can hold multiple values at a time 'remembers' old values.
4. **No. 19:** There is no size or precision limit to the values we can store in a programming variable.
5. **No. 26:** A false condition ends program if no else branch.
6. **No. 33:** while loops terminate as soon as condition changes to false.

Misconception di tipo **strategico**:

1. **No. 1:** The computer know the intention of the program or of a piece of code, and acts accordingly.
2. **No. 6:** Difficulties with telling apart the static and dynamic aspects of programs.
3. **No. 22:** Unassigned variables of primitive type (in Java) have no memory allocate
4. **No. 85:** Difficulties understanding the empty constructor.
5. **No. 131:** Methods from the simple class are not used; instead, new equivalent methods are defined and duplicated in the composed class.

5.2 Esempi reali di misconception

Esempio 1:
Un compagno di classe non capiva che "Il valore delle variabili non viene trasferito durante l'assegnazione" (Misconception **No. 16**) e che, in quello specifico caso (dato che il parallelismo non era ancora stato introdotto) le istruzioni sono eseguite una alla volta e non in contemporanea (Misconception **No. 8**). Questi problemi sono emersi durante un esercizio in cui gli veniva chiesto come sostituire il valore di 2 variabili (es. da X = 5, Y = 14 a X = 14, Y = 5).
Esempio 2:
Durante un tutorato di Informatica erano presenti alcuni studenti che all'esecuzione su carta di un algoritmo ricorsivo scrivevano solo la prima o al massimo due esecuzioni ricorsive, quindi avevano riscontrato la misconception del "modello a step: ricorsione a uno o due step, ma non di più." (Misconception **No. 56**).
Esempio 3:
Durante una esercitazione di programmazione era presente un collega che aveva difficoltà nello svolgere esercizi con due array di cui uno contenente gli indici di posizione dell'altro.
L'esercizio in questione consisteva nel partire da due vettori a e b, in cui a contiene gli indici di posizione del vettore b (non ordinati), e di modificare gli elementi in b aggiungendo la metà del valore contenuto se e solo se l'indice della cella è contenuto nel vettore a.
La misconception incontrata è stata "Difficoltà con gli array contenenti indici come dati." (Misconception **No. 153**).
Per risolvere queste misconception è stato necessario concentrarsi su specifici tipi di esercizi, in particolare nel primo caso concentrati sullo sviluppo della conoscenza sintattica dell'assegnazione, nel secondo caso di conoscenza concettuale della ricorsione e nell'ultimo caso di comprensione strategica e dello scopo degli indici.

6 La natura dei programmi

6.1 Esperienze del nostro percorso di studi

Edoardo

Seppure tutti i 6 punti siano stati trattati almeno una volta durante i miei percorsi di studi (sia superiori che universitari) ritengo che venga posta differente attenzione su alcuni di essi a seconda del livello di studi. L'insegnamento dell'informatica nell'istituto tecnico da me frequentato era incentrato soprattutto sull'aspetto più concreto di quelli elencati: l'essere **strumenti** creati per lo svolgimento di task. A partire dal terzo anno infatti molti dei programmi creati avevano applicazioni molto pratiche come la creazione di programmi gestionali o di consultazione di un database, trascurando spesso tutta la parte di pensiero antecedente alla codifica e buona gestione delle risorse a disposizione (il concetto di complessità computazionale non è mai stato introdotto chiaramente). Gli altri aspetti come la natura di entità astratte ed eseguibili o manufatti linguistico-notazionali venivano date per scontato e non considerate durante l'insegnamento o la valutazione.

Lungo il percorso universitario ho invece notato come i 6 aspetti vengano tutti trattati in modo più equo, pur mantenendo un focus sul programma come strumento.

Laura

Durante il mio percorso di studi superiore, al Liceo Scientifico Scienze Applicate, le uniche sfaccettature emerse sono state la visione del programma come entità astratta e come strumento. Il motivo principale è stata la scelta degli argomenti trattati, focalizzati sull'informatica teorica, in particolare alcuni concetti di Teoria della computazione e della complessità. Inoltre diverse lezioni sono state focalizzate su alcuni concetti matematici per utilizzare l'applicazione software Octave, che mi ha dato l'opportunità di vedere l'informatica da un punto di vista multidisciplinare. Ci sono stati pochi momenti dedicati alla scrittura di codice con qualche linguaggio di programmazione e per l'implementazione di un'applicazione reale, per questo l'idea di programma nelle altre sfaccettature è andato in secondo piano.

6.2 Domanda 1

Guardando le indicazioni nazionali del laboratorio Informatica e Scuola del CINI e quelle degli istituti superiori vi sembra che includano tutte le 6 facce dei programmi?

Gli obiettivi indicati dal CINI per la **scuola primaria** sembrano mirare più ad una conoscenza di tipo pratico dei programmi, dando meno importanza alla visione dei programmi come **creazioni dell'uomo** costruite grazie ad un'analisi dei requisiti richiesti o delle intenzioni del programmatore. Inoltre, nonostante ci sia menzione di dispositivi elettronici (es. O-P5-N-1), gli obiettivi che li riguardano sono slegati

dall'influenza che i programmi esercitano su di essi mancando quindi interamente della sfaccettatura di **"oggetti fisici"**.
Queste due mancanze riguardano solamente la scuola primaria, invece negli ordinamenti successivi, pur mantenendo dei traguardi più improntati su un lato pratico, si va comunque ad affrontare le altre sfaccettature precedentemente menzionate.

6.3 Domanda 2

Dovendo progettare un intero corso di scienze informatiche in una scuola superiore in che ordine presentereste le 6 sfaccettature?
Premettendo che tutte quante le seguenti caratteristiche sono pressoché di egual importanza, riteniamo che il miglior ordine adottabile per l'insegnamento sia il seguente:

1. **Opere dell'uomo**
2. **Entità astratte**
3. **Artefatti linguistico-notazionali**
4. **Entità eseguibili**
5. **Strumenti**
6. **Oggetti Fisici**

Prima di arrivare ad una visione più "concreta" del programma, è importante comprendere come essi siano **opere dell'uomo** ideate con determinate finalità e seguendo certi requisiti, essere in grado di individuare questi due aspetti è la base per la creazione del programma.
Apprese le capacità precedentemente descritte si può passare alla fase di astrazione, insegnando come i programmi siano **entità astratte** che lavorano e modellano una versione astratta della realtà, introducendo quindi anche la possibilità di avere diverse interpretazioni di uno stesso elemento/problema.
Posate le basi teoriche per una comprensione minimale di cosa la realizzazione di un programma comporta si può passare ad un aspetto che si avvicina un po' di più alla pratica: il loro lato **linguistico-notazionale**. L'importanza di questo aspetto risiede nel fatto che, rispettando degli standard, questi programmi siano più facilmente interpretabili e modificabili da altri programmatori.
Questi primi 3 punti elencati sono comprensibili anche senza un'applicazione pratica, ma considerando che il nostro obiettivo è quello della creazione di strumenti bisogna prima introdurre la visione dei programmi come **entità eseguibili**, tramite la creazione di programmi per la risoluzione di esercizi di allenamento, in modo da applicare i concetti precedentemente appresi ed affrontare con un approccio meno teorico i problemi posti.

Grazie a questo passaggio si può dare una dimostrazione più evidente di come ciò che viene pensato dagli studenti venga interpretato dal calcolatore.
Il trattamento degli ultimi due punti può avvenire in modo quasi parallelo: comprendere che i programmi siano **strumenti** è il fine ultimo, tuttavia l'importanza del saper sfruttare le risorse messe a disposizione per l'utilizzo di questi strumenti è probabilmente importante tanto quanto saperli creare. Comprendere quindi che l'utilizzo di un programma ha delle conseguenze e requisiti fisici in quanto **oggetto fisico** è una competenza non trascurabile durante l'apprendimento.

7 Necessity Learning Design

A partire dall'articolo [4], individuare altri momenti, oltre a quelli presentati (sezioni 4.4.1, 4.4.2, 4.4.3, 4.4.4) in cui è possibile usare il NLD per introdurre nuovi concetti.
Altri esempi di applicazione del NLD potrebbero essere:

- **Liste**: risolve la necessità di strutture dati con lunghezza non predefinita in problemi in cui i vettori comporterebbero degli sprechi o mancanze di spazio.
- **ForEach**: risolve il problema dello scorrimento di serie di un certo tipo all'interno delle strutture dati
- **Ricorsione**: risolve, in alcuni casi, problemi computazionalmente troppo onerosi se risolti con codice non ricorsivo oltre che una minore quantità di righe di codice
- **Oggetti**: risolve il problema di rappresentazione di concetti più complessi e non pienamente rappresentabili con un solo tipo di dato
- **Javascript**: permette di rendere dinamiche pagine web altrimenti statiche

Nel vostro percorso scolastico/universitario avete mai "sentito la necessità" di un meccanismo/costrutto di programmazione prima che vi venisse spiegato?
Durante il percorso universitario magistrale, mi è capitato di sentire la necessità di un meccanismo di programmazione di cui non ero a conoscenza. In particolare come introduzione ad uno dei corsi della magistrale vengono proposti 3 problemi di ottimizzazione, da risolvere con le sole conoscenze pregresse.
Questi problemi però non sono risolvibili in maniera efficiente senza l'utilizzo di particolari metodologie, per questo durante lo svolgimento del corso questi problemi vengono utilizzati per introdurre delle tecniche algoritmiche, come ad esempio il Backtracking e il Branch&Bound che permettono di ottenere delle soluzioni ottimali ai quesiti.

Riferimenti

[1] Peter J. Denning. "COMMUNICATIONS OF THE ACM". In: Association for Computing Machinery, 2005, pp. 27–31.

[2] *Informatica e competenze digitali: cosa insegnare?* 2019. URL: https://link-and-think.blogspot.com/2019/03/informatica-e-competenze-digitali-cosa.html.

[3] Viera K. Proulx. "Programming Patterns and Design Patterns in the Introductory Computer Science Course". In: *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education* (2000), pp. 80–84. DOI: 10.1145/330908.331819.

[4] Marco SBARAGLIA, Michael LODI e Simone MARTINI. "A Necessity-Driven Ride on the Abstraction Rollercoaster of CS1 Programming". In: *Informatics in Education* 20.4 (2022), pp. 641–682. ISSN: 1648-5831. DOI: 10.15388/infedu.2021.28.

[5] Juha Sorva. "Visual Program Simulation in Introductory Programming Education". Tesi di dott. Mag. 2012.