Università degli studi di Torino

Dipartimento di informatica

Metodologie e tecnologie didattiche per l'informatica

Consegne svolte

Autore:
Leonardo Magliolo

Anno Accademico 2023/2024

Consegna 1:

3 punti/spunti/idee sulla quali siete d'accordo e perchè (riportate la frase e l'articolo da cui proviene):

1)Uno degli spunti, a mio parere, più articolati e complessi rilevati all'interno degli articoli letti consiste nell'idea presentata da Enrico Nardelli riguardante la produzione di artefatti cognitivi dinamici:
<<È cambiata la natura degli artefatti, delle macchine, che produciamo. Non sono più artefatti fisici, sono "*artefatti cognitivi dinamici*" (o *macchine della conoscenza* o *macchine cognitive*), azione congelata che viene sbloccata dalla sua esecuzione in un computer e genera conoscenza come risultato di tale esecuzione. La conoscenza statica dei libri diventa conoscenza dinamica nei programmi. Conoscenza in grado di produrre automaticamente, senza l'intervento umano, nuova conoscenza.>>
Sebbene di primo acchito non mi è parsa una presentazione particolarmente chiara del fenomeno che cercava di descrivere, dopo la visione suggerita della presentazione del professor Enrico Nardelli in collaborazione con il dipartimento di Torino e dopo l'attenta lettura del testo "Di cosa parliamo quando parliamo di 'programmi" il contenuto dell'articolo mi è sembrato decisamente più chiaro:
Il programma non è solo considerabile come dato, ma è la rappresentazione linguistico-notazionale che descrive una conoscenza o un ragionamento rappresentato mediante uno specifico linguaggio, questa rappresentazione che per l'uomo ha un valore semantico viene interpretata da un artefatto (spesso identificato come computer) che risponde meccanicamente osservando la sequenza di simboli, producendo in output ulteriori simboli considerabili come dati ma che racchiudono anch'essi una semantica interpretabile dall'uomo e che quindi possono essere considerati come forma di conoscenza ricavata a partire da procedure automatizzabili.

2) <<Da un punto di vista metodologico, si può dire che i programmi sono congetture, mentre le esecuzioni sono tentativi di refutazioni — che hanno fin troppo spesso successo>>
La citazione del filosofo James Fetzer, presente nel testo "Di cosa parliamo quando parliamo di 'programmi'", esprime un concetto interessante che può risultare banale solo a posteriori. Fetzer suggerisce di considerare un programma informatico come un'ipotesi e la sua esecuzione come un tentativo di confutarla. Questa prospettiva alternativa mi ha aiutato a comprendere meglio il motivo per cui l'attività di programmazione può essere considerata scientifica.

3)Dal manifesto di Vienna per l'umanesimo digitale:
<<Come tutte le tecnologie, le tecnologie digitali non emergono dal nulla. Sono modellate da scelte implicite ed esplicite e, quindi, incorporano un insieme di valori, norme, interessi economici e ipotesi su come il mondo che ci circonda sia o dovrebbe essere. Molte di queste scelte rimangono nascoste in programmi software che implementano algoritmi invisibili.>>
Il paragrafo riportato mi ha fatto riflettere sull'importanza dell'applicazione di un dato algoritmo in un determinato contesto e su come questo possa rappresentare, spesso in modo implicito e senza che chi prende le decisioni se ne renda necessariamente conto, una decisione politica.
Considerare le procedure automatiche che guidano il comportamento delle tecnologie digitali come lo "specchio" dei valori della società che le ha prodotte mi ha permesso di cogliere l'essenza del manifesto per l'umanesimo digitale, giustificando la necessità di una visione che consideri nuovi programmi di studio che combinino conoscenze delle scienze umane e sociali assieme a studi di tipo scientifico-ingegneristico.

Almeno un punto (meglio ancora 2 o 3) sui quali non siete d'accordo:
Riporto di seguito un paragrafo tratto da "Informatica: la terza rivoluzione "dei rapporti di potere"" di Enrico Nardelli:
<<Le persone sono intrinsecamente in grado di apprendere ciò che non sanno (mentre le macchine della conoscenza possono apprendere solo ciò per cui sono state progettate) ed hanno imparato, attraverso milioni di anni di evoluzione, ad adattarsi flessibilmente a cambiamenti imprevisti nell'ambiente (mentre le macchine cognitive possono – ancora una volta – adattarsi solo ai cambiamenti previsti).>>
Sebbene l'intelligenza artificiale odierna non sia in grado di concepire un agente intelligente dotato di intelligenza generale (AGI), ciò non implica che l'attuale architettura utilizzata per l'esecuzione dei programmi sia intrinsecamente incapace di replicare la flessibilità e l'adattabilità del ragionamento umano. Pertanto, ritengo che l'opinione espressa nel paragrafo, per quanto allettante, non sia scientificamente accurata se non contestualizzata nel caso specifico dello stato dell'arte odierno dell'AI, a patto che non generalizzi sulle architetture impiegate dagli odierni computer in assenza di prove strutturali.

Nonostante riletture multiple dei documenti assegnati da leggere non ho trovato ulteriori porzioni di testo per cui dopo un'attenta riflessione mi sono trovato in disaccordo.

In quali dei 3 paradigmi (matematico, ingegneristico, scientifico) situate l'informatica? perche'? fornire anche le motivazioni:
L'informatica è una disciplina che esamina, secondo prospettive diverse e complementari tra loro, lo studio e la progettazione dei processi automatici in grado di processare i dati, per tale ragione l'informatica risulta esser multidisciplinare: i paradigmi perciò sono più individuabili dipendentemente dalla branca dell'informatica che si intende trattare:

- Paradigma matematico: essendo basato su inferenze deducibili a partire da conoscenze a priori risulta particolarmente adatto per descrivere la teoria della computazione che studia le proprietà dei linguaggi formali, teorie riguardanti la verifica formale dei programmi, la teoria dei grafi o campi dell'intelligenza artificiale che sfruttino un meccanismo di Automated Theorem Proving.
  L'approccio matematico non risulta particolarmente fruttuoso qualora non sia possibile (o sia impraticabile) estrapolare delle proprietà dei processi coinvolti che possano essere considerabili assiomaticamente vere (come ad esempio nel caso dello sviluppo di una generica applicazione software).
- Paradigma scientifico: fondato sulla formulazione di ipotesi falsificabili a partire da esperimenti replicabili. Il paradigma scientifico è preferibile per lo studio di fenomeni complessi per cui è possibile ipotizzare modelli semplificati mediante astrazioni, grazie all'applicazione di sistemi di calcolo elettronici è possibile svolgere le computazioni suggerite dall'ipotesi (modello) e verificare statisticamente se per particolari situazioni il modello sia in grado o meno di predire con un certo margine d'errore una data situazione.

Un esempio tipico di branca dell'informatica studiabile a partire dal paradigma scientifico è la teoria della simulazione oppure il machine learning.

- Paradigma ingegneristico: basato su un approccio di stampo empiristico, fonda le sue radici sull'esperienza e la definizione di best practices determinate da valutazioni a posteriori su misure d'affidabilità ottenute a partire da progetti complessi.
  Risulta particolarmente idonea per la risoluzione di problemi pratici sottoposti a vincoli di affidabilità, di risorse e tempistiche impiegate. Vengono riportati di seguito alcuni esempi di problemi affrontati prevalentemente con paradigma ingegneristico: progettazione di architetture dei calcolatori, sviluppo delle applicazioni software.

In conclusione, determinare se l'informatica (nel senso più generale del termine) appartenga ad un solo paradigma tra quelli proposti risulta essere un'approssimazione piuttosto grossolana della varietà di tecniche impiegate nell'ambito per risolvere i problemi più disparati.
A seguito di riflessioni personali e vari confronti con colleghi di altri atenei, ritengo personalmente che la tendenza di vari informatici ad attribuire un paradigma piuttosto che un altro dipenda in maggior misura dalla propria formazione e dall'ambito di specializzazione intrapreso: è facile confondere il proprio curriculum informatico con la più ampia definizione di informatica, ed è ancor più facile perdersi nei dettagli delle varie sotto-tematiche piuttosto che adottare un approccio olistico che enfatizzi le similitudini e le interconnessioni sottostanti le stesse.

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".
L'informatica soddisfa qualcuno dei criteri sopra elencati? Se si quali? e perchè?

○ Organized to understand, exploit, and cope with a pervasive phenomenon:
L'informatica ha raggiunto un livello di diffusione così ampio che è ormai presente in quasi tutti gli aspetti della nostra vita quotidiana.
Grazie allo studio dell'elaborazione dell'informazione attraverso processi computazionali, sono stati sviluppati dispositivi come smartphone, computer e smart TV, così come reti di comunicazione sempre più avanzate.
Le scoperte dell'informatica hanno contribuito a spiegare fenomeni naturali sia di natura fisica, biologica e cognitiva, che fenomeni complessi e artificiali come l'economia.
La presenza dell'informatica si estende quindi sia al mondo naturale che in quello artificiale, offrendo nuove soluzioni e prospettive in diversi ambiti della nostra vita.

○ Encompasses natural and artificial processes of the phenomenon:
L'informatica è una disciplina che consente di analizzare e comprendere i processi che coinvolgono la codifica delle informazioni e la loro rielaborazione attraverso operazioni computazionali.
Un esempio di fenomeno naturale studiabile secondo il paradigma informatico è la replicazione delle cellule mediante *codifica* del materiale genetico tramite DNA, è possibile studiare inoltre come la cellula sia in grado di *interpretare* il codice genetico per replicarsi.

Parallelamente allo studio della vita mediante processi naturali, è possibile studiare la stessa mediante processi computazionali artificiali definiti da “automi cellulari”: “The game of life”.

○ Codified structured body of knowledge:

L'informatica possiede un “corpo contenente conoscenza strutturata e codificata” in quanto ha un insieme di conoscenze organizzate e documentate in modo strutturato. Queste conoscenze sono suddivise in campi specifici come lo studio degli algoritmi, lo studio architetturale dei calcolatori, i linguaggi di programmazione, le basi di dati e altri.
Sono disponibili libri di testo, pubblicazioni accademiche e risorse online che documentano in modo dettagliato le teorie, i concetti e le best practice dell'informatica. Inoltre, esistono standard e convenzioni che guidano la progettazione e lo sviluppo di software e sistemi informatici. L'informatica è sostenuta da una comunità scientifica che collabora per condividere conoscenze, sviluppare nuove teorie e valutare i progressi nel campo.

○ Commitment to experimental methods for discovery and validation:

Sebbene non sia necessariamente vero che l’informatica adoperi metodi sperimentali come principale strumento di risoluzione per l’ampio spettro dei fenomeni computazionali che intende studiare, esistono particolari branche dell’informatica che sfruttano maggiormente tecniche sperimentali fondate sulla scoperta e la validazione:

- Data Mining
- Simulazioni
- Validazione di modelli
- Robotica
- Computer Vision

○ Reproducibility of results:

La riproducibilità dei risultati nell'ambito informatico è interpretabile come la capacità di riprodurre gli stessi risultati di un'esecuzione o di un'elaborazione computazionale quando si dispongono delle stesse condizioni iniziali, degli stessi dati di input e dello stesso ambiente di esecuzione.
L’esecuzione delle computazioni mediante calcolatore elettronico ha reso possibile riprodurre i risultati per esecuzioni particolarmente complesse, abbattendo i costi e le tempistiche delle stesse.

○ Falsifiability of hypotheses and models:

É possibile falsificare ipotesi su modelli codificandoli algoritmicamente ed effettuandone un’analisi mediante tecniche di verifica formale degli algoritmi.
Tecniche alternative di falsificabilità delle ipotesi su processi computazionali probabilistici coinvolgono metodi statistici, ne vengono riportati alcuni di seguito:

- Cross-Validation
- A/B Testing
- Distribuition Hypothesis testing

○ Ability to make reliable predictions, some of which are surprising:

Alcune branche dell’informatica indubbiamente sono in grado di predire risultati non banali in maniera affidabile, le simulazioni computerizzate risultano essere esempi piuttosto indicativi.
Mediante tecniche di simulazione dei processi è possibile simulare sistemi complessi i cui comportamenti possono esser considerati caotici.
Alcuni esempi di sistema complesso sono:

• gli automi cellulari
• il sistema climatico
• gli ecosistemi
• gli organismi viventi
• I sistemi sociali
• I sistemi economici
• Ambiente e territorio

Consegna 2:

Rivedere l'attività sugli pseudoalgoritmi in fasi indicando:

1. **obiettivi (risultati dell'apprendimento attesi)**
2. **proposta di suddivisione in fasi**
3. **snodi e indicatori**

1)

- Conoscenze:
  - Conoscere la definizione generale di algoritmo procedurale fornita da Donald Knuth.
  - Conoscere le caratteristiche che necessariamente devono esser presenti affinché un presunto algoritmo possa esser definito tale (Finitezza, Precisione, I/O e Fattibilità).
  - Conoscere alcune caratteristiche imputabili agli algoritmi che sebbene siano rilevanti al loro studio non sono essenziali per stabilire se un presunto algoritmo possa esser definito come tale (generalità, efficienza e correttezza).
- Abilità: Saper valutare se un dato algoritmo possa esser considerato tale rispetto la definizione appresa.
- Competenze:
  - Riconoscere il ruolo centrale dell'esecutore automatico all'interno del contesto della formulazione algoritmica.
  - Riconoscere alcuni aspetti fondamentali che è necessario considerare per istruire un esecutore inconsapevole del processo in grado di raggiungere un obiettivo.

2) Prima fase: "Gli studenti, divisi a coppie, ricevono un foglio con la lista degli pseudo-algoritmi e, discutendo tra loro, devono stabilire per ciascun pseudo-algoritmo se possa essere definito effettivamente algoritmo oppure no, giustificando le loro decisioni"

   Seconda fase: "Verranno formati gruppi unendo le coppie a due a due, e gli studenti dovranno confrontare le loro decisioni con quelle dei compagni, eventualmente rivedendole per arrivare ad una nuova decisione condivisa."

   Terza fase: "Ogni gruppo deve formulare una propria definizione di algoritmo e specificare quali proprietà un algoritmo deve avere per potersi definire tale. Ogni gruppo presenterà la sua definizione al resto della classe; il conduttore metterà man mano in evidenza le somiglianze e le differenze tra le varie definizioni. L'attività si conclude con la presentazione, da parte del conduttore, di una definizione

“consolidata” del concetto di algoritmo, e la sua messa a confronto con le definizioni dei gruppi.”

3)

Snodi	Indicatori
Riconoscere che spesso la mancanza di precisione è dovuta all’assenza di una specifica descrizione dell’esecutore dell’algoritmo	Porre domande circa le caratteristiche (o capacità) dell’esecutore circa la discussione degli esercizi “le lasagne al forno” o “ricetta del cocktail cosmopolitan”
Accorgersi che per determinati problemi l’ordine proposto delle istruzioni relative al loro pseudo-algoritmo è interscambiabile senza comprometterne la correttezza o l’efficienza	Nel dare la propria definizione di algoritmo prediligono parole come “sequenza” o “ordinato” piuttosto che “insieme” o “collezione”, facendo riferimento alle istruzioni dell’algoritmo
Cogliere la sottile differenza che vi è tra alcune caratteristiche necessariamente coinvolte nella definizione di un algoritmo e proprietà desiderabili ma non obbligatorie	Aver classificato correttamente come algoritmo “operazioni matematiche” evidenziandone comunque l’inefficienza e lo scopo poco pratico in fase di discussione
Comprendere che in assenza di ulteriori specifiche circa le abilità e le conoscenze pregresse dell’esecutore automatico la scelta di classificare particolari pseudo-algoritmi come algoritmi risulta esser basata sulle proprie assunzioni personali	In fase di discussione gli alunni riescono a considerare accettabili classificazioni alternative la propria qualora le motivazioni che giustificano tale classificazione presumano abilità e conoscenze pregresse dell’esecutore differenti dalle proprie

Consegna 3:

Partendo dal materiale relativo al seminario tenuto dalla Dott.ssa Violetta Lonati *Di cosa parliamo quando parliamo di programmi* scrivete una breve riflessione su questi aspetti:

1. **nei corsi di studio che avete affrontato sono emerse tutte le sfaccettature del concetto di programma presentate nel seminario?**
2. **quale aspetto o quale “asse” vi ha creato piú difficoltá e/o pensate ne crei a chi studia informatica?**
3. **dovendo progettare un intero corso di scienze informatiche in una scuola superiore in che ordine presentereste le 6 sfaccettature?**

1) Anzitutto, per evitare possibili incomprensioni, introduco brevemente il mio percorso di studi affrontato finora:

- Scuola superiore: ITIS indirizzo automazione ed elettrotecnica
- Laurea triennale: Informatica (L-31) presso il dipartimento di Torino (percorso “Reti e sistemi informatici”)
- Laurea magistrale in corso: Informatica (LM-18) presso il dipartimento di Torino (percorso “Intelligenza artificiale e sistemi informatici”).

Per quanto riguarda gli insegnamenti in ambito informatico relativi alla programmazione svolti durante il percorso di scuola superiore, è stata posta particolare enfasi rispetto la visione secondo cui i programmi sarebbero eseguibili automaticamente e possano esser considerati oggetti fisici: la programmazione di basso livello, adoperata rispetto alle tematiche specializzanti del corso, ha reso particolarmente evidente la stretta relazione che può esservi tra i programmi e gli esecutori in grado di eseguirli automaticamente e inconsapevolmente.
Rispetto al programma affrontato all’università ritengo che siano emerse (anche se tal volta implicitamente) tutte le sfaccettature presentate nel seminario:

- In corsi come “Sviluppo delle applicazioni software” e “Basi di dati” è stato possibile constatare in maniera concreta come i programmi siano opera dell’uomo e siano concepiti per soddisfare determinate esigenze pratiche, pertanto possono essere inquadrati come strumenti utilizzabili.
  Un’altra conseguenza derivante dal considerare il software come frutto del lavoro umano è il considerare i programmi come artefatti linguistico-notazionali: al loro interno incorporano una logica formalizzata mediante le specifiche istruzioni scelte per rappresentare il ragionamento:
  aspetto di cruciale importanza da considerare qualora il software sia prodotto da più di un essere umano e pertanto debba essere comprensibile da coloro che intendono successivamente modificare o espandere il suo funzionamento.
- Durante la porzione di corso riguardante la progettazione delle memorie fisiche in “Architettura degli elaboratori” è stato possibile considerare il programma come dato rappresentato all’interno di sistemi fisici di vario genere, ciò ha permesso di considerare il programma come l’informazione rappresentata all’interno di un sistema fisico riconducendolo effettivamente ad un oggetto. E’ stato inoltre constatato come il programma possa essere eseguito per mezzo di un calcolatore elettronico (esecutore) in maniera automatica senza che vi sia consapevolezza dello stesso rispetto allo scopo dell’esecuzione.
- Infine, l’idea secondo cui i programmi possano essere considerati come entità astratte è stata particolarmente utile a mio parere per la comprensione di corsi come “Algoritmi e strutture dati” e “Linguaggi formali e traduttori”.
  Entrambi i corsi tendono a studiare le proprietà computazionali relative ai programmi (come la complessità, la correttezza o le condizioni di terminazione) senza che ci sia un diretto riferimento a “notional machines” o a elementi inerenti particolari linguaggi di programmazione.

2) Personalmente durante il mio percorso di laurea ho riscontrato poca enfasi in termini della quantità di corsi (sebbene come espresso al punto precedente ci sia stata) riguardo l’aspetto fisico caratterizzante la natura dei programmi.
   Nonostante ciò, forse, l’aspetto che mi ha recato più frustrazione durante i corsi intrapresi è la considerazione secondo cui il programma sarebbe a tutti gli effetti uno strumento: per mia naturale inclinazione tendo a prediligere una visione astratta e prevalentemente matematica dell’informatica.
   Più in generale, confrontandomi con vari colleghi di diversi dipartimenti, ho riscontrato come i laureati nel nostro specifico corso di laurea tendano a considerare il programma più

come un’entità astratta che come fisica, e al tempo stesso più come un’entità eseguibile piuttosto che come un artefatto notazionale.

3) In un contesto reale cercherei riferimenti in letteratura e nei documenti stilati da enti accreditati in modo tale da considerare approcci che risulterebbero più efficaci e consoni rispetto la scelta dell’ordinamento con cui presentare le sfaccettature.
   In assenza di esplicite indicazioni concordanti considererei anzitutto il nome del corso “Scienze informatiche” e pertanto sarei incline a presentare prima le sfaccettature che a mio parere rispecchiano maggiormente l’aspetto scientifico dell’informatica, in maniera tale da enfatizzare il loro ruolo nel corso sin da subito:
   1. I programmi sono artefatti linguistici notazionali: E’ possibile utilizzare la struttura del programma per formulare un’ipotesi precisa, pertanto è possibile considerare i programmi come assimilazione di un processo mentale.
   2. I programmi sono eseguibili automaticamente: Aspetto essenziale per considerare la validazione empirica.
   3. I programmi sono entità astratte: “Rappresentano manipolazioni di simboli, cioè di segni privi di significato intrinseco, cosicchè il significato prende forma esclusivamente nella mente di chi ne prefigura il funzionamento”. Aspetto importante per riuscire a interpretare correttamente il risultato della validazione empirica eseguita mediante l’interprete automatico.
   4. I programmi sono oggetti fisici: Utile spunto di riflessione per considerare come la disciplina informatica possa descrivere processi biologici e fisici presenti all’interno dei corsi di scienze considerati più “tradizionali” e meno trasversali.
   5. I programmi come opera dell’uomo: “Le tecnologie digitali non emergono dal nulla. Sono modellate da scelte implicite ed esplicite e, quindi, incorporano un insieme di valori, norme, interessi economici e ipotesi su come il mondo che ci circonda sia o dovrebbe essere. Molte di queste scelte rimangono nascoste in programmi software che implementano algoritmi invisibili”. Come evidenziato all’interno del manifesto per l’umanesimo digitale risulta importante riflettere sull’aspetto etico morale codificato all’interno del software prodotto, anche per uno scienziato.
   6. I programmi sono strumenti: E’ stato inserito in ultima posizione in quanto questo specifico aspetto ritengo sia più noto ed intuitivo agli studenti delle superiori, e se affrontato senza aver prima evidenziato gli aspetti cruciali dell’informatica che contribuiscono a considerarla una scienza potrebbe generare confusione.

   L’ordinamento delle prime tre sfaccettature è stato scelto per facilitare la visione dell’informatica come disciplina scientifica (e non semplicemente a supporto della scienza): formulazione dell’ipotesi -> deduzione -> validazione empirica.

   Mi rendo conto che le motivazioni che giustificano questo specifico ordinamento sono parzialmente frutto di mie considerazioni personali e non necessariamente supportate da fatti “empirici”, pertanto sarei disposto a modificare l’ordinamento qualora venissi convinto da studi o vengano rilasciate considerazioni ufficiali in merito a come debbano essere presentate le sfaccettature in rispetto al corso di scienze informatiche.

Consegna 4:

Partendo dall'appendice A della tesi *Visual Program Simulation in Introductory Programming Education*

1) **provare a classificare alcune delle misconceptions in base al tipo di difficoltà (sintattica, concettuale, strategica)**
2) **data la vostra esperienza (di studenti delle superiori, di studenti universitari, alcuni di voi come tutor, alcuni di voi come persone che fanno ripetizioni), provate ad elencare "misconception" nella programmazione in cui vi siete imbattuti, personalmente o in altri**
3) **come le avete risolte?**

1) Per praticità riporto di seguito dieci delle misconceptions più interessanti lette considerando una certa varietà di tipologie tra quelle riportate nel paper, classificandole rispetto al block model proposto a lezione:
    - (1) The computer knows the intention of the program or of a piece of code, and acts accordingly: Function/Purpose, Macrostructure
    - (8) Magical parallelism: several lines of a (simple nonconcurrent) program can be simultanenously active or known: Program execution, Relationschips
    - (14) A variable is (merely) a pairing of a name to a changeable value (with a type). It is not stored inside the computer: Program execution, Atomic
    - (16) Assignment moves a value from a variable to another: Program execution, Atoms
    - (24) Code after if statement is not executed if the then clause is: Program execution, Blocks
    - (32) Difficulty in understanding automated changes to loop control variables: Program execution, Blocks
    - (46) Parameter passing requires different variable names in call and signature: Relationships, Text surface
    - (48) Cannot use globals in subprograms when they have not been passed as parameters: Program execution, Relationships
    - (54) Null model of recursion: recursion is impossible: Program execution, Macrostructure
    - (80) An object is a subset of a class. / A class is a collection of objects: Function/Purpose, Relationships
2) Una delle misconceptions che mi ha recato problemi quando da autodidatta mi sono approcciato al mondo della programmazione è relativa alla confusione che vi è sulla modifica dei valori delle variabili passate come parametro ad un metodo: se il chiamato modifica i valori delle variabili esplicitate nella sua signature, le variabili date in input dal chiamante verranno anch'esse modificate?
   Una delle misconceptions che ho riscontrato piuttosto frequentemente in individui relativamente competenti nel campo della programmazione è la scarsa comprensione riguardo le differenze che vi sono tra metodi statici e metodi di istanza.
3) Nel mio caso, per risolvere il dubbio relativo la prima misconception è stata sufficiente l'introduzione di una notational machine che esplicitasse la gestione della memoria mediante Stack e Heap: così facendo mi è parsa più chiara la differenza che vi era tra le modifiche delle variabili relative alla porzione di Stack associata alla chiamata del metodo e

la modifica di valori rappresentati all'interno della Heap a cui la variabile locale nello stack puntava.
Il formalismo mi è stato introdotto durante il corso di programmazione 1 di Unito, e dopo la visione di numerosi esempi e lo svolgimento di esercizi che imponevano di disegnare a penna lo schema della Stack e della Heap a seguito dell'esecuzione di una porzione di codice, sono riuscito a correggere le mie lacune.
Riguardo la misconception relativa ai metodi statici è bastato far riflettere gli alunni sulle relazioni che vi sono tra classe e sua istanza, in particolare evidenziare con esempi pratici situazioni in cui fosse necessario possedere una variabile condivisa tra istanze della stessa classe e presentando il concetto di variabile statica come soluzione al problema proposto.
Dopodiché sono state presentate casistiche in cui fosse necessario utilizzare metodi propri di una classe, che prescindessero da una specifica istanza in modo tale che potessero manipolare lo stato delle variabili statiche presentate precedentemente.
Per la correzione di entrambe le misconceptions ho trovato particolarmente utili strumenti di debug visuali come pythontutor.com: software in grado di tracciare il grafico Stack e Heap relativo l'esecuzione di un programma in maniera automatica e in ogni punto del codice.

Consegna 5:

Dopo aver visionato il materiale scrivete un breve commento sui pro e contro di ogni approccio:

1. **PRIMM**
2. **POGIL**
3. **NLD**

quale preferireste sperimentare e perché (4).

1) Tengo a precisare che nel caso di PRIMM il materiale fornito è stato piuttosto scarso, tutte le informazioni che riporterò sono estratte dalla sola lettura della presentazione dell'approccio contenuta su primmportal.com e sulla visione di alcune attività elencate all'interno del sito: non avendo letto pubblicazioni se esistono precisazioni riguardo che smentiscono aspetti che ho inserito nei contro sono disposto a ritrattarle dopo una lettura.

- Pro:
    - Adatto anche per situazioni in cui si insegna ad un solo allievo o si è impossibilitati a svolgere attività di gruppo
    - Contrariamente a come riportato per l'approccio NLD, è possibile utilizzare PRIMM per una moltitudine di casi che non si limitano soltanto all'introduzione di costrutti nuovi mai menzionati
    - E' possibile iterare più volte l'approccio durante lo svolgimento delle lezioni senza un abbassamento delle performance, come invece avviene in NLD
    - Risulta essere piuttosto semplice come approccio, il design delle attività non richiede necessariamente una quantità di tempo considerevole come invece accade per POGIL. Queste caratteristiche lo rendono più flessibile rispetto le esigenze richieste per lo specifico scenario educativo: se l'attività non esiste posso in poco tempo progettarne una
    - Compatibile con una vasta varietà di attività il cui design non è specificato necessariamente per PRIMM (vedi i Parson's puzzle), cosa che invece era necessaria per NLD
- Contro:

    - L’aspetto di interazione sociale, sebbene non sia in principio escluso, non sembra esser enfatizzato nella presentazione di PRIMM: la collaborazione tra gli studenti sembra un aspetto marginale
    - La sua versatilità talvolta potrebbe comportare il design di attività non particolarmente efficaci, non ci sono indicazioni chiare su come si debba comportare il facilitatore per ogni fase

2)

- Pro:
    - Adatto alla pianificazione di attività didattiche complesse e articolate
    - Conforme all’approccio di apprendimento attivo: I docenti assumono principalmente il ruolo di facilitatore piuttosto che di istruttore
    - La specifica di ruoli assegnati a ciascun allievo a rotazione permette loro di comunicare e assolvere compiti diversi implementando meccanismi della teoria che nell’articolo di Cambridge international sull’apprendimento attivo viene chiamata: Costruttivismo sociale.
- Contro:
    - Metodo ideato per le discipline STEM in generale, senza occhio di riguardo particolare per l’attività di programmazione
    - Sebbene esistano delle attività già progettate e rese disponibili, l’approccio POGIL rende impegnativo lo sviluppo di nuove attività: richiedono sforzi e tempistiche piuttosto lunghe
    - Svolgere il ruolo di facilitatore monitorando l’operato di ogni singolo studente al quale viene associato un ruolo specifico potrebbe essere impegnativo durante lo svolgimento dell’attività

3)

- Pro:
    - Adatto per classi di studenti delle superiori
    - Se effettuato nel giusto contesto educativo, introducendo un solo costrutto per volta stimola l’interesse della classe che in un secondo momento sarà più propensa a ricordarsi ed utilizzare il costrutto presentato comprendendone le necessità
    - Il metodo consente di progettare attività didattiche in modo veloce
    - Il metodo è stato concepito nativamente per l’insegnamento della programmazione
- Contro:
    - Potrebbe frustrare gli studenti che sono abituati ad ottenere ottimi risultati
    - Utile per introdurre soltanto costrutti fondamentali della programmazione
    - Non adatto all’utilizzo frequente: gli allievi scoperto il trucco non si impegnerebbero nella fase P!S, sapendo che in un secondo momento verrebbero date loro istruzioni per il costrutto mancante da parte dell’insegnante
    - Potrebbe non risultare banale da parte dell’insegnante scegliere quando passare dalla fase P!S alla fase I, una fase troppo breve P!S produrrebbe risultati riconducibili al metodo tradizionale d’insegnamento, una fase troppo lunga potrebbe contribuire ad abbassare il livello di attenzione della classe
    - Il design delle attività necessita un’attenzione particolare alle conoscenze specifiche della classe coinvolta e la sua efficacia dipende fortemente dalle attività già svolte in precedenza: la consegna del problema (così come la maggioranza degli elementi coinvolti logicamente nel problema) devono essere riconducibili a problemi già svolti in precedenza, inoltre deve essere introdotto necessariamente un costrutto per volta.
    - Mantenere segreta la presenza dei costrutti qualora studenti più esperti li conoscessero fino alla fase di Instruction non è un compito banale da attuare

4) Essendo un’insegnante alle prime armi preferirei sperimentare con il metodo PRIMM in quanto mi garantirebbe una certa libertà di manovra circa le attività che ritengo opportune svolgere in quello specifico ambito educativo: NLD richiede di conoscere un certo numero di informazioni riguardo gli studenti che sono presenti in classe e qualora ci fosse qualcuno che conosce il costrutto da presentare sarebbe per me impegnativo cercare di mantenere il segreto fino alla fase I, mentre POGIL risulta essere più sofisticato e strutturato e per tale ragione trovo che svolgere il ruolo di facilitatore in un contesto dove addirittura vengono assegnati ruoli diversi a ciascun allievo in un ambiente altamente collaborativo sia più adatto a insegnanti con esperienza.