Lorenzo Falchi
Jacopo Terenzi

Consegna 1

1)È importante insegnare informatica come materia scolastica?

1)In "*The Science in Computer Science"* si mette l'accento su quelle che sono le criticità dell'insegnamento dell'informatica come materia nelle scuole superiori americane.
Nelle letture fornite, non siamo riusciti a trovare riferimenti espliciti sul perché sia importante insegnare questa materia, tuttavia possiamo affermare che è fondamentale per diversi motivi. Anche se non vanno confuse competenze digitali ed informatica, uno dei motivi potrebbe essere la conoscenza di determinati meccanismi che permetterebbero un utilizzo più efficiente di software anche non prettamente "per informatici" (pensiamo ad Excel). Più in generale, è fondamentale che gli studenti abbiano almeno consapevolezza che non ci sia magia dietro un computer, che è ormai uno degli strumenti maggiormente utilizzati in qualsiasi ambito. Si può inoltre citare il mondo del lavoro che richiede di anno in anno figure sempre più specifiche e che da per scontate certe competenze in ambito informatico. L'informatica aiuta gli studenti a sviluppare competenze trasversali come, ad esempio, il problem solving, la creatività e il pensiero critico, che sono utili in qualsiasi campo.

2)L'informatica è una scienza?

2)Secondo l'articolo *"Is Computer Science Science?"* di Peter J. Denning l'informatica è la scienza dei processi informativi e delle loro interazioni con il mondo. D. afferma che l'informatica applica il metodo scientifico (ipotesi, test ed esperimenti che poi conducono a modelli) per lo studio di tali processi. Altro riferimento al metodo scientifico viene fatto nella lettura *"Qual è la natura dell'informatica?"* al paragrafo 7. Dalla metodologia utilizzata per i nostri studi possiamo dire che siamo in accordo con quanto affermato dall'autore dell'articolo. Tornando a Denning, egli asserisce che l'informatica ha caratteristiche sia di scienza pura sia applicata, sia di scienza quantitativa sia qualitativa. Infatti, gli informatici lavorano seguendo principi accettati e sistematizzati. Molta informatica viene applicata e altrettanta viene utilizzata per previsione e verifica. Anche in questo caso ci troviamo in accordo con l'autore poiché nel nostro percorso di studi universitario sono presenti sia insegnamenti prettamente teorici sia prettamente pratici.

3)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:

1. **Organized to understand, exploit, and cope with a pervasive phenomenon.**
2. **Encompasses natural and artificial processes of the phenomenon.**
3. **Codified structured body of knowledge.**
4. **Commitment to experimental methods for discovery and validation.**
5. **Reproducibility of results.**
6. **Falsifiability of hypotheses and models.**
7. **Ability to make reliable predictions, some of which are surprising.**

4)L'informatica soddisfa qualcuno dei criteri sopra elencati? Se si quali? e perchè?

3)/4) Nel seguito rispondiamo ad entrambe le domande.

1. sicuramente avere come fine ultimo la comprensione di fenomeni è un criterio valido per definire una disciplina scienza; siamo d'accordo con il primo criterio perchè l'informatica viene utilizzata soprattutto per risolvere problemi frequenti e di difficile gestione, anche perchè molto spesso bisogna dare una soluzione immediata e veloce.
2. anche il secondo criterio definisce una scienza ma non sempre l'informatica tratta fenomeni naturali. Può capitare che siano concepiti dei modelli informatici per cercare di comprendere e capire fenomeni naturali ma tratta per lo più fenomeni artificiali.
3. sicuramente il fatto di avere un corpo di conoscenza strutturato e codificato è un criterio valido per definire la scienza. In questo caso l'informatica si sposa abbastanza bene con il criterio poiché anche se molti linguaggi sono nati e si sono sviluppati su internet, perciò senza seguire una reale strada comune da parte di altri programmatori, ci sono invece degli altri linguaggi nati proprio "in laboratorio". Inoltre, è comune in programmazione utilizzare dei pattern per la scrittura di codice o anche in fase progettazione di un software.
4. è forse il criterio per eccellenza che definisce una scienza, l'informatica ci rientra assolutamente poiché utilizza questa metodologia di lavoro.
5. anche in questo caso il criterio è valido per classificare una scienza, ma per l'informatica il discorso è leggermente diverso, infatti bisogna tener conto della variabilità del risultato dovuta a tanti fattori come, ad esempio, l'hardware con cui si sta lavorando. In generale, infatti, si potrebbe pensare che lo stesso codice si comporti e ottenga gli stessi risultati su tutte le macchine ma nella realtà non è così.
6. anche questo è un criterio valido ma di nuovo per l'informatica è un discorso un po' particolare poiché è una disciplina relativamente giovane quindi magari non c'è stato nemmeno il tempo di falsificare alcuni ipotesi (basti pensare che i

primi cenni all'intelligenza artificiale risalgono alla fine degli anni 50). In generale comunque potremmo dire che l'informatica utilizza questa metodologia di lavoro.

7. Questo è sicuramente un aspetto molto utile di una disciplina ma non pensiamo che sia fondamentale per classificare una disciplina come scienza, crediamo infatti che gli aspetti fondamentali siano altri (metodo scientifico, per esempio). Per quanto riguarda l'informatica, comunque, ci rientrerebbe poiché tramite modelli matematici convertiti in informatici è possibile fare previsioni anche con un certo grado di attendibilità.

Consegna 2

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

1) Il tema/ concetto informatico oggetto dell'attività sono i numeri binari, operazioni, conversioni, imparare a leggerli e applicazioni al mondo dell'informatica.

2)Quali sono gli obiettivi formativi?

2) Gli obbiettivi formativi individuati sono:

Ambito algoritmi

O-P3-A-1. riconoscere gli elementi algoritmici in operazioni abituali della vita quotidiana (p.es.: lavarsi i denti, vestirsi, uscire dall'aula...);

O-P3-A-2. comprendere che problemi possono essere risolti mediante la loro scomposizione in parti più piccole.

Ambito programmazione

O-P3-P-2. ordinare correttamente la sequenza di istruzioni;

O-P3-P-3. utilizzare i cicli per esprimere sinteticamente la ripetizione di una stessa azione un numero prefissato di volte;

Ambito dati e informazione

O-P3-D-1. scegliere ed utilizzare oggetti per rappresentare informazioni familiari semplici (es. colori, parole, ...);

O-P3-D-2. definire l'interpretazione degli oggetti utilizzati per rappresentare l'informazione (legenda).

Ambito consapevolezza digitale

O-P3-N-1. riconoscere usi dell'informatica e delle sue tecnologie nella vita comune;

3)Suddividete l'attività in fasi e per ogni fase individuate snodi e indicatori

3)Suddividiamo l'attività nelle 4 fasi

1)CONSEGNA:

In questa prima fase viene introdotta l'attività, vengono spiegate infatti le regole. Nasce quindi una discussione sul regolamento e di conseguenza domande e dubbi.

2)SVOLGIMENTO:

In questa fase viene svolta l'attività, gli studenti hanno a disposizione per capire meglio l'esercizio 6 facce di dado, ognuna delle quali ha il doppio del valore alla sua sinistra, quindi in ordine: 1,2,4,8,16,32. E questo lo definirei come snodo dell'attività, cioè assimilare la figura grafica del dado ad un numero reale.

3) DISCUSSIONE:

In questa fase l'insegnante risponde alle domande verificando i risultati degli studenti.

4)CONCLUSIONE:

anche se non è presente esplicitamente in questo capitolo, la conclusione sta nel fatto di verificare che le competenze acquisite dagli studenti siano uniformi e corrette, prima di passare ad un argomento successivo.

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

4) in questa attività abbiamo avuto modo di trovare dei concetti visti precedentemente nelle lezioni. Ad esempio come citato prima, la suddivisione in fasi dell'attività, per avere una visione generica più organizzata e mirata nel caso ci fossero delle problematiche da correggere. Inoltre sono presenti degli snodi, sotto forma di esempio grafico, come punto chiave di apprendimento dell'attività. Quest'ultima osservazione si potrebbe leggere in chiave di Costruttivismo cognitivo, ovvero "active learning", cioè il fatto di far fare pratica agli studenti per favorire l'apprendimento.

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?

5) Il testo dell'attività appare abbastanza chiaro e semplice, soprattutto considerando il target (bambini dai 6 anni in su), pertanto non siamo riusciti ad individuare alcun suggerimento.

Consegna 3

Siamo in generale ovviamente d'accordo con la definizione di algoritmo data a lezione, tuttavia vorremmo aggiungere alcuni dettagli: un algoritmo è una sequenza di passaggi chiari e ben definiti, organizzati in modo logico e ordinato, che permettono di risolvere un problema o eseguire una serie di operazioni. Gli algoritmi sono delle istruzioni dettagliate che una persona o un computer possono seguire per ottenere un risultato desiderato.

1.Chiarezza: Un algoritmo deve essere formulato in modo chiaro e preciso, in modo che chiunque lo legga possa capire esattamente cosa deve fare senza ambiguità.

2.Definito: Ogni passaggio dell'algoritmo deve essere ben definito e non ambiguo, in modo che non ci siano dubbi su come eseguire ciascuna operazione.

3.Finitezza: L'algoritmo deve terminare dopo un numero finito di passaggi. Non deve andare avanti all'infinito, ma deve sempre raggiungere una conclusione.

4.Effettività: Ogni passaggio dell'algoritmo deve essere eseguibile in pratica, cioè dovrebbe essere possibile seguire le istruzioni utilizzando una quantità finita di risorse (come carta e penna, calcolatrice, computer, ecc.).

5.Determinismo: L'algoritmo deve essere definito in modo tale che, dato uno stesso input iniziale e le stesse condizioni, produca sempre lo stesso output o risultato.

6.Ordine: Gli step dell'algoritmo devono essere eseguiti in un ordine specifico e preciso, in quanto l'ordine può influenzare il risultato finale.

7.I/O: L'algoritmo deve prendere in input i dati necessari e produrre un output che rappresenti la soluzione al problema.

8.Limitatezza delle risorse: L'algoritmo deve utilizzare una quantità ragionevole di risorse, come memoria e tempo di calcolo, per risolvere il problema. Non dovrebbe richiedere risorse eccessive.

Consegna 4

1)Leggete l'articolo *"Programming Patterns and Design Pattern in Introductory Computer Science Course"* e scrivete un breve resoconto sull'approccio proposto: siete d'accordo con l'approccio proposto? Lo adottereste? Immaginate di proporlo in una classe in cui insegnate: quali sono le potenziali difficoltà?

1) L'articolo *"Programming Patterns and Design Patterns in the Introductory Computer Science Course"* di Viera K. Proulx esplora un approccio pedagogico per l'insegnamento di informatica introduttiva basata sulla programmazione orientata agli oggetti, utilizzando modelli di programmazione e design patterns. Di seguito è riassunto l'approccio proposto nell'articolo e alcune considerazioni sulla sua adozione in un corso.

Approccio Proposto: L'articolo propone un approccio basato su modelli di programmazione e design patterns per insegnare informatica introduttiva. L'obiettivo principale è concentrarsi sullo sviluppo delle capacità di ragionamento e progettazione degli studenti prima che la complessità dei dettagli del linguaggio di programmazione li confonda. L'autore ha sviluppato una serie di modelli che coprono argomenti tipicamente affrontati in un corso introduttivo di informatica e li presenta come tutorial. Ogni modello è strutturato in modo da guidare gli studenti attraverso le fasi di progettazione e implementazione, dal contesto generale alla realizzazione pratica. Il corso è organizzato con conferenze settimanali, discussioni di gruppo e laboratori.

Considerazioni sull'Adozione: L'approccio proposto sembra avere benefici significativi per gli studenti di programmazione. Tuttavia, ci sono alcune considerazioni da tenere in considerazione prima di adottarlo in un corso:

Livello di Conoscenza degli Studenti: L'articolo sembra adatto a studenti che sono completamente nuovi alla programmazione o hanno conoscenze molto limitate. Per studenti con esperienza precedente in programmazione, alcuni contenuti potrebbero risultare troppo elementari.

Necessità di Adattamento: È possibile che sia necessario adattare il contenuto dei tutorial e dei problemi di pratica in base alle specifiche esigenze della classe e del linguaggio di programmazione utilizzato.

Tempo Richiesto: L'insegnamento basato su modelli e tutorial può richiedere più tempo rispetto a un approccio tradizionale, quindi è importante valutare se è possibile integrare efficacemente questa metodologia nel corso.

Valutazione: La valutazione degli studenti dovrebbe essere progettata in modo da riflettere il loro apprendimento basato su modelli e design patterns.

Sviluppo delle Capacità di Problema: L'approccio sembra concentrarsi sullo sviluppo delle capacità di progettazione, ma è importante assicurarsi che gli studenti sviluppino anche le abilità di risoluzione di problemi pratici.

Disponibilità di Risorse: Sarà importante garantire che gli studenti abbiano accesso alle risorse necessarie, come i tutorial e i materiali di apprendimento.

In generale, l'approccio proposto sembra interessante e può essere efficace per insegnare le basi della programmazione e del design orientato agli oggetti. Tuttavia, è importante valutare attentamente le esigenze specifiche della classe e adattare l'approccio di conseguenza.

2)riportate il testo del problema analizzato per l'attività 2) Progettazione soluzione/algoritmo e la soluzione che avete progettato: suddivisione in sotto problemi, pattern algoritmici, elementari e ruoli delle variabili.

2)Il problema scelto è il seguente

Swap tra due valori

Realizzare un programma che acquisisca da tastiera (standard input) due valori interi e li memorizzi nelle variabili varA e varB. In seguito, scambiare il valore di tali variabili e stamparne a schermo il nuovo contenuto. Attenzione: gestire il caso in cui non si riceve un valore intero, il programma non deve terminare.

Soluzione proposta:

```
import java.util.Scanner;

public class SwapValori {
    Scanner scanner = new Scanner(System.in);

    int varA = 0;
    int varB = 0;
    boolean inputValido = false;

    while (!inputValido) {
       System.out.print("Inserisci il valore di varA (deve essere un intero): ");
       if (scanner.hasNextInt()) {
          varA = scanner.nextInt();
          inputValido = true;
       } else {
          System.out.println("Input non valido. Inserisci un valore intero valido.");
          scanner.next(); // Pulisce l'input errato
       }
    }

    inputValido = false;

    while (!inputValido) {
       System.out.print("Inserisci il valore di varB (deve essere un intero): ");
       if (scanner.hasNextInt()) {
          varB = scanner.nextInt();
          inputValido = true;
       } else {
          System.out.println("Input non valido. Inserisci un valore intero valido.");
          scanner.next(); // Pulisce l'input errato
```

```
            }
        }

        System.out.println("Valore di varA prima dello scambio: " + varA);
        System.out.println("Valore di varB prima dello scambio: " + varB);

        int temp = varA;
        varA = varB;
        varB = temp;

        System.out.println("Valore di varA dopo lo scambio: " + varA);
        System.out.println("Valore di varB dopo lo scambio: " + varB);

        scanner.close();
    }
```

Suddivisione in sotto problemi
Abbiamo individuato diversi sotto problemi principali: il primo è la definizione e l'utilizzo dello scanner per prendere l'input da tastiera (potrebbe essere a sua volta scomposto in due sotto problemi); il secondo è il controllo dell'input; il terzo è lo swap vero e proprio delle variabili.

Ruoli delle variabili
Valore più recente: Le variabili varA e varB conservano i valori più recenti immessi dall'utente.
Flag unidirezionale: La variabile inputValido funge da flag unidirezionale, passando da false a true quando l'utente fornisce un input valido.
Temporanea: Le variabili temp e scanner possono essere considerate temporanee. temp è utilizzata temporaneamente per effettuare lo scambio tra varA e varB, mentre scanner è utilizzata temporaneamente per acquisire input dall'utente.

Pattern elementari
Lettura in input finchè non si riceve il valore desiderato: è quello che chiede l'esercizio, continuare a chiedere finchè non si riceve un intero.

Pattern algoritmici
1)Pattern name: read from input
Pattern start point: every exercise that needs values from input
Pattern objective: return values taken from input
define scanner
if(value is valid) var=scanner

2)Pattern name: variables exchange
Pattern start point: 2 variables
Pattern objective: returns 2 variables whose values have been changed
temp=var1
var1=var2
var2=temp

3)Descrivete anche eventuali critiche (es. sulla formulazione del problema), problemi e considerazioni emerse durante l'attività
Un punto critico è il fatto di dover trasformare un problema troppo generico in codice, nel senso che molto spesso ci troviamo di fronte algoritmi resi obsoleti, come ad esempio la somma in colonna, utile forse solamente per un allenamento logico/matematico, ma a lato pratico ci sono migliaia di strumenti che riescono ad arrivare allo stesso risultato in un tempo enormemente minore.
Inoltre svolgendo l'attività ci si è resti conto che la formulazione del problema è molto importante non solo per la comprensione da parte degli studenti ma anche per l'analisi da parte del docente. Infatti si può dire che parti come la suddivisione in sotto problemi sono state problematiche da sviluppare proprio a causa della formulazione del problema.

Consegna 5

1)Partendo dall'appendice A della tesi Visual Program Simulation in Introductory Programming Education provare a classificare le misconceptions in base al tipo di difficoltà (sintattica, concettuale, strategica), scrivetene alemno 5 per categoria nella consegna

Misconceptions sintattiche:

1. La macchina comprende l'intenzione del programma.
2. La macchina deduce automaticamente l'intenzione del programmatore.
3. La macchina comprende l'inglese come un umano.
4. Difficoltà nella comprensione delle espressioni che mancano del concetto di valutazione.
5. Difficoltà nella comprensione dell'esecuzione sequenziale.

Misconceptions concettuali:

1. Difficoltà nel comprendere gli aspetti temporali delle variabili.
2. Parallelismo implicito: diverse righe di un programma semplice sono attive simultaneamente.
3. Difficoltà nella comprensione del ciclo di vita dei valori.
4. L'assegnazione primitiva funziona in entrambe le direzioni (scambia).
5. Le variabili ricevono sempre un particolare valore predefinito alla creazione.
6. Una variabile può contenere più valori contemporaneamente.
7. Difficoltà nel distinguere gli aspetti statici e dinamici dei programmi.
8. Il sistema previene operazioni irragionevoli.
9. Le variabili si aggiornano automaticamente in base al contesto logico.
10. Problemi nella comprensione di astrazione e riferimento.
11. Mancanza di comprensione delle variabili e dei loro valori.
12. Confusione tra assegnazione e aggiornamento delle variabili.
13. Difficoltà nella valutazione delle espressioni e nell'uso delle variabili.
14. Mancanza di comprensione dei concetti di variabile e assegnazione.

Misconceptions strategiche

1. Difficoltà nell'utilizzare correttamente le variabili.
2. Difficoltà nell'assimilare l'esecuzione parallela.
3. Difficoltà nell'allocazione di memoria e nella gestione dei valori.
4. Problemi nell'uso appropriato di espressioni e operatori.
5. Difficoltà nell'identificare quando utilizzare le variabili.
6. Problemi nell'uso di variabili e assegnazioni.
7. Problemi nell'utilizzo corretto delle variabili e delle espressioni.

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

2)Non avendo noi mai fatto ripetizioni o da tutor a qualche altro studente, ci affidiamo alla nostra esperienza personale. Personalmente le misconception più dure a morire sono state sicuramente la difficoltà nel distinguere gli aspetti statici e dinamici dei programmi (maggiormente riguardo il linguaggio java) e la gestione della memoria (più legata al linguaggio C).

3)come le avete risolte?

3)L’unico modo che abbiamo reputato valido per risolvere le *misconceptions* è stato studiare ma soprattutto capire a fondo il funzionamento dei meccanismi che ci davano problemi. Generalmente il nostro approccio implica tornare alle basi degli argomenti e vedere in quale punto non ci si ritrova più con quanto spiegato. A volte può essere utile cercare di capire la teoria attraverso gli esercizi ma ci è capitato poche volte di riuscire, normalmente si fa meno fatica a studiare prima la teoria. Un’altra valida soluzione è quella di studiare in gruppo poiché è molto probabile che, nel gruppo, ognuno abbia capito qualcosa e quindi sia più facile riuscire a, per esempio, risolvere un esercizio ma anche capire la teoria in sé. Nello specifico delle misconception che abbiamo menzionato, particolarmente utili sono stati strumenti che mostrassero l’utilizzo della memoria passo passo, avendo da una parte il codice e dall’altra il comportamento della memoria.

Consegna 6

1)nei corsi di studio che avete affrontato sono emerse tutte le sfaccettature del concetto di programma presentate nel seminario?

1)In questo documento i programmi sono intesi come parte integrante degli strumenti che utilizziamo quotidianamente.
Questa nozione non è nuova per noi, perchè durante gli studi di informatica, abbiamo sempre frequentato corsi/svolto progetti che proiettavano soluzioni per problemi della vita reale.
Ad esempio, progettare un programma web mail per quanto riguarda Programmazione 3, oppure un sistema di calcolo di distanza tra città per algoritmi e strutture dati.
Altri corsi invece, ci hanno portato a progettare dei software “ausiliari” per la risoluzione di problemi/ algoritmi di livello superiore.
Ad esempio durante il corso di linguaggi formali e traduttori, abbiamo progettato un "traduttore” per linguaggio macchina, affinché potessimo capire come funzionasse il compilatore di java.
Questo tipo di progetti/corsi, infatti, non è finalizzato al puro scopo di realizzare qualcosa che possa essere immediatamente utile per la società, ma piuttosto una serie di concetti chiave per migliorare il nostro pensiero logico e informatico.

2)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?

2)le 6 facce dei programmi sono contenute esplicitamente, tranne l’ultima perchè è un discorso molto ampio e complesso, e ci si aspetta che venga assimilato durante i corsi e non propriamente insegnato dai professori.

3)dovendo progettare un intero corso di scienze informatiche in una scuola superiore in che ordine presentereste le 6 sfaccettature?

3)Come prima fase metteremmo la 3.2, ovvero programmi come opera dell’uomo, per spiegare l’origine e come si inizia a progettare un programma. Successivamente spiegheremmo la fase 3.1, quindi un’introduzione a cosa sono i programmi fine a se stessi e come vengono utilizzati nel mondo reale. La fase 3.5 la verrebbe spiegata dopo l’introduzione perchè serve unicamente a capire che il software, dopo la eventuale pubblicazione, viene eseguito solamente da chi ne deve fare uso. Insieme invece, spiegheremmo le fasi 3.3, 3.4, perchè entrambe spiegano un concetto simile, ovvero la natura del programma di essere legata ad un hardware, e al senso opposto di essere vista solamente come un oggetto software astratto. Ad esempio, i software POS sono legati obbligatoriamente ad un hardware dedicato, come anche i sistemi cluster di mining.

Mentre altri applicativi, come ad esempio programmi di streaming dati online sono visti quasi esclusivamente come oggetti software astratti. Come ultima fase, invece abbiamo la 3.6, che a parer nostro è un concetto molto avanzato, che possibilmente si dovrebbe apprendere al di fuori di un corso di studi, magari in ambiente lavorativo, quando si ha a che fare con una realtà in cui si osservano molti programmi non propedeutici diversi fra loro, che potrebbero arricchire il nostro vocabolario di conoscenza.

Consegna 7

Introduzione funzione *size*

problema: dato un array, contare il numero degli elementi che contiene.

conoscenze pregresse degli studenti:

Livello sintattico e modello concettuale:

-importare funzioni dalle librerie;
-accesso agli array tramite indici;
-for loop;

Cosa gli studenti sanno già fare:

-usare variabili come contatori;
-iterare su una collezione di una dimensione ben definita per compiere delle azioni;

Target concept:

-funzione *size*;
-uso della funzione size per scorrere l'array fino all'ultimo elemento;

La funzione *size* è una funzione già implementata, gli studenti non hanno bisogno di implementarne una loro versione.

Introduzione funzione *free*

problema: in linguaggio C, come eseguire correttamente un ciclo in cui si alloca dinamicamente memoria?

conoscenze pregresse degli studenti:

Livello sintattico e modello concettuale:

-importare funzioni dalle librerie;
-variabili ed assegnamenti
-for e while loop;

Cosa gli studenti sanno già fare:

-usare variabili come contatori;
-uso delle funzioni *malloc* e *realloc*;
-iterare su una collezione per compiere delle azioni;

Target concept:

-funzione *free*;
-uso della funzione size per liberare la memoria allocata dinamicamente;

La funzione *free* è fondamentale in linguaggio C dove la gestione della memoria è lasciata al programmatore ma è utile anche per far capire allo studente l'utilità di strumenti come il garbage collector nel linguaggio Java.

Introduzione costrutto *switch*

problema: in linguaggio C, come eseguire correttamente un ciclo in cui si alloca dinamicamente memoria?

conoscenze pregresse degli studenti:

Livello sintattico e modello concettuale:

-variabili ed assegnamenti

-costrutto if

Cosa gli studenti sanno già fare:

-usare variabili come contatori;

Target concept:

-costrutto *switch*;

-uso del costrutto *switch* per una migliore gestione dei casi con possibilità più numerose rispetto al solo if;

Il problema sarebbe potuto essere risolto anche con l'utilizzo del solo if, viene introdotto il concetto di switch per gestire meglio determinate situazione.

Nel nostro percorso universitario abbiamo percepito che il NLD fosse utilizzato un po in tutti i corsi, particolarmente nel corso di *algoritmi e strutture dati*, in cui non c'era un solo particolare argomento che fosse introdotto così ma sembrava proprio essere il modus operandi dei professori durante tutto il corso. Personalmente abbiamo trovato l'approccio inizialmente un po' più ostico, rispetto ad altri corsi in cui veniva prima classicamente spiegata la teoria utili agli esercizi, ma che alla lunga ha sicuramente pagato.