

La ricorsione con **Haskell**

Luca Barra

Anno accademico 2023/2024

INDICE

CAPITOLO 1	INTRODUZIONE	PAGINA 2
1.1	Livello di scuola, classe e indirizzo	2
1.2	Motivazioni e finalità	3
1.3	Prerequisiti	3
1.4	Contenuti	4
1.5	Traguardi e obiettivi Obiettivi di apprendimento — 5 • Indicazioni nazionali — 5	5
1.6	Materiali e strumenti necessari	6
1.7	Linguaggio	6
CAPITOLO 2	SVILUPPO DEI CONTENUTI	PAGINA 8
2.1	Attività 1: Introduzione alla ricorsione	8
2.2	Attività 2: Le Matrioske (unplugged) Da Unplugged a Programmazione (Fase 2 de "Le Matrioske") — 10	9
2.3	Attività 3: Programmazione in Haskell	12
2.4	Esercizi di programmazione Esempio — 13 • Esercizi per prendere confidenza — 13 • Esercizi sulle guardie — 14 • Esercizi sulle liste — 15 • Esercizi sulla ricorsione esplicita — 17 • Parte avanzata: esercizi sugli alberi e sulle funzioni anonime — 18	13
CAPITOLO 3	GUIDA PER GLI INSEGNANTI	PAGINA 21
3.1	Snodi e indicatori per fasi Attività 1 — 21 • Attività 2 — 21 • Attività 3 — 22	21
CAPITOLO 4	INDICAZIONI PER LA VALUTAZIONE	PAGINA 24
4.1	Valutazione durante le lezioni/attività	24
4.2	Riguardo gli esercizi di programmazione... Block Model — 24 • Misconceptions — 25	24
4.3	Rubrica valutativa	25

Capitolo 1

Introduzione

Note:-

Quest'attività, essendo pesante e impegnativa, richiede 2-3 lezioni (da un'ora) per essere svolta (volendo si può pensare a una quarta lezione per approfondire meglio).

1.1 Livello di scuola, classe e indirizzo

Domanda 1

A chi è rivolta questa attività?

Risposta: A studenti e studentesse del quarto/quinto anno di una scuola secondaria di secondo grado di un indirizzo **scientifico**¹.

Domanda 2

Può essere adattata/rivolta a studenti e studentesse di diverse età e indirizzi?

Risposta: Quest'attività può essere somministrata a studenti e studentesse al quinto anno di superiori senza alcuna modifica. Può, altresì, essere eseguita da studenti di età inferiore con alcune correzioni (che indicherò nel documento).

Domanda 3

In quale specifica disciplina scolastica si colloca l'attività?

Risposta: Quest'attività ha carattere *principalmente informatico* trattando un argomento prevalentemente computazionale (la ricorsione). Tuttavia si può inquadrare quest'unità didattica in un contesto più ampio trattando anche Haskell che, come linguaggio puramente funzionale, si può associare a un ambito *matematico* e *logico*. Il ragionamento ricorsivo ha anche un'utilità trasversale dato che aiuta a comprendere in maniera più profonda alcuni problemi e la loro scomposizione in sottoproblemi, aiutando a sviluppare *soft-skill* e problem solving.

¹Liceo Scientifico Op. Scienze Applicate

1.2 Motivazioni e finalità

Domanda 4

Perché si è scelta proprio quest'attività?

Risposta: L'attività nasce dalla necessità di insegnare ai ragazzi e alle ragazze delle superiori il concetto di *programma ricorsivo*. Spesso quest'argomento viene trattato in modo superficiale o non viene trattato proprio, ma è utile avere un *modello mentale* di ricorsione per poter vedere i problemi sotto un'altra luce. Infatti lo scopo di questa unità didattica non è solamente quello di insegnare un argomento, ma punta a mostrare *soluzioni alternative* ad alcuni problemi.

- ⇒ **Scrivere codice in modo elegante:** quando si insegna a programmare, almeno all'inizio, viene trascurato il fatto che il codice debba essere letto e capito da altre persone per cui ci si focalizza più sull'aspetto "*funziona*" rispetto alla "*veste grafica*". Questo può anche portare a errori di programmazione dato che si avranno difficoltà a leggere anche i propri programmi. Il presente documento non si vuole semplicemente limitare a insegnare il concetto, molto utile, della ricorsione, ma vuole anche far fronte al problema dello "*spaghetti code*" (programmi mal strutturati) fornendo una valida alternativa.
- ⇒ **Pensiero computazionale e algoritmico:** una caratteristica fondamentale nella società odierna è il saper far fronte a problemi spesso imprevisti. Non c'è solo un modo per risolvere un quesito e le soluzioni possono essere *molteplici*. Infatti, per alcuni problemi, può risultare ostico usare un approccio iterativo per cui è preferibile utilizzare la *ricorsione*.
- ⇒ **Codice efficiente:** nella parte avanza di quest'unità didattica viene portato un esempio di struttura dati che si presta a un'efficiente rappresentazione ricorsiva, ossia gli alberi (e in particolare gli alberi binari di ricerca). Questo per combattere lo stigma relativo al fatto che i programmi ricorsivi, oltre a essere più difficili da scrivere rispetto ai programmi iterativi, siano sempre inefficienti rispetto a eventuali controparti che utilizzano un approccio "classico".

1.3 Prerequisiti

Verranno dati per scontati i seguenti prerequisiti in quanto fondamentali per la comprensione dell'unità didattica.

- ⇒ Utilizzo di base del computer;
- ⇒ Concetto di algoritmo;
- ⇒ Concetto di variabile;
- ⇒ Tipi di variabili (Int, Float, etc.);
- ⇒ Logica booleana;
- ⇒ Propensione al ragionamento astratto.

Note:-

È propedeutica la conoscenza di Haskell che verrà comunque ripreso, sviluppato e approfondito nel corso dell'unità didattica.

1.4 Contenuti

I contenuti che presentano un asterisco blu (*) sono adatti anche a studenti e studentesse di altri indirizzi o di età inferiore. I contenuti che presentano un asterisco rosso (*) sono opzionali per via della maggiore difficoltà, ma se gli alunni reagiscono bene al resto dell'attività si potrebbero considerare come "bonus".

- ⇒ * **Ricorsione:** il cuore di quest'attività è insegnare la ricorsione e il ragionamento ricorsivo, aiutando a riconoscere quando esso sia preferibile a un ragionamento iterativo;
- ⇒ * **Funzioni:** ci si occuperà sia di funzioni in generale (principalmente per fare pratica con Haskell) che di funzioni ricorsive;
- ⇒ * **Tipi di Haskell:** Num, Int, Float, String;
- ⇒ **Inferenza di tipo:** mostrare come in Haskell si deve essere precisi con i tipi e di come ciò sia un aiuto alla programmazione;
- ⇒ **Funzioni a più argomenti:** come si possono usare funzioni che richiedono più di un parametro in Haskell (curryficate);
- ⇒ **Funzioni con guardie:** una rappresentazione per casi alternativa all'utilizzo di "IF" che risulta molto utile anche nella definizione di funzioni ricorsive;
- ⇒ **Liste:** usate come "scusa" per introdurre il pattern matching;
- ⇒ **Pattern matching:** come il pattern matching delle liste sia utile per definire funzioni ricorsive;
- ⇒ * **Funzioni anonime:** funzioni che usano lambda expression o funzioni parzialmente applicate;
- ⇒ * **Alberi:** visione di un albero con l'utilizzo della ricorsione.

Le attività in cui è strutturata l'unità:

- **Introduzione teorica:** si introdurranno i concetti fondamentali di quest'attività, a partire dalla ricorsione (supportandosi con esempi presi dall'esperienza comune e dal mondo dell'arte). Verrà posto l'accento sulla divisione in "passo base" e "passo induttivo" (fondamentale per evitare programmi che non funzionano o che non terminano). In questa parte si inizierà anche a esplorare il mondo di Haskell con la sua sintassi e i comandi di GHCi, il suo interprete, discutendo un esempio noto, ossia la successione di Fibonacci;
- **Attività ludica:** sarà proposta un'attività, inizialmente unplugged, utilizzando un oggetto, magari non molto comune, ma che la maggior parte delle persone ha presente nel proprio immaginario, ossia le Matrioske (o bambole russe). Le matrioske hanno una natura intrinsecamente ricorsiva contenendo al loro interno una copia identica, ma di dimensioni minori (infatti la matrioska più interna è la più piccola). Nella fase successiva dell'attività, tramite un programma Haskell (hs) fornito nell'apposita sezione, sarà possibile vedere in modo esplicito la ricorsività trattata nella fase unplugged;
- **Programmazione:** l'attività finale di quest'unità consisterà nel definire semplici funzioni in Haskell (di cui è presente, per l'insegnante, una soluzione) per testare quanto imparato nelle due parti precedenti. Ciò comprende anche un'eventuale parte avanzata che può essere liberamente modificata e adattata dall'insegnante a seconda delle proprie esigenze.

1.5 Traguardi e obiettivi

1.5.1 Obiettivi di apprendimento

Note:-

I seguenti obiettivi di apprendimento sono stati scritti usando la tassonomia di Bloom rivisitata.

Tassonomia	Obiettivi
CREATE	Lo/La studente/ssa, al termine dell'attività, sarà in grado di sviluppare semplici programmi ricorsivi in Haskell.
EVALUATE	Lo/La studente/ssa, al termine dell'attività, saprà valutare la convenienza di un approccio ricorsivo rispetto a un approccio iterativo.
ANALYZE	Lo/La studente/ssa, al termine dell'attività, comprenderà i concetti di passo base e passo induttivo/ricorsivo e saprà simulare l'esecuzione di programmi che usano esplicitamente la ricorsione.
APPLY	Lo/La studente/ssa, al termine dell'attività, riuscirà a usare in modo efficace la ricorsione per risolvere problemi.
UNDERSTAND	Lo/La studente/ssa, al termine dell'attività, saprà identificare, classificare e descrivere programmi ricorsivi.
REMEMBER	Lo/La studente/ssa, al termine dell'attività, ricorderà le componenti fondamentali di un programma ricorsivo (passo base e passo induttivo).

1.5.2 Indicazioni nazionali

Traguardi:

- Comprendere i principali fondamenti teorici delle scienze dell'informazione;
- Acquisire la padronanza di strumenti dell'informatica, utilizzare tali strumenti per la soluzione di problemi significativi in generale;
- L'uso di strumenti e la creazione di applicazioni deve essere accompagnata non solo da una conoscenza adeguata delle funzioni e della sintassi, ma da un sistematico collegamento con i concetti teorici ad essi sottostanti;
- Il rapporto fra teoria e pratica va mantenuto su di un piano paritario e i due aspetti vanno strettamente integrati evitando sviluppi paralleli incompatibili con i limiti del tempo a disposizione.

Obiettivi specifici:

- ⇒ **Ambito algoritmico:** implementazione di un linguaggio di programmazione, metodologie di programmazione;
- ⇒ **Ambito calcolo numerico e simulazioni:** semplici simulazioni.

Fonte: INDICAZIONI NAZIONALI

1.6 Materiali e strumenti necessari

Materiale fisico:

- ⇒ Matrioske (bambole russe);
- ⇒ Adesivi (in alternativa si possono utilizzare dei pezzi di carta e del nastro adesivo);
- ⇒ Fogli di carta;
- ⇒ Penne;
- ⇒ Lavagna multimediale;
- ⇒ Computers.

Materiale software:

- ⇒ GHCi (interprete per Haskell);
- ⇒ Editor di testo o IDE (Integrated Development Enviroment);
- ⇒ Terminale o Powershell.

Note:-

GHCi può essere reperito al seguente link: <https://www.haskell.org/downloads/>. Si raccomanda di utilizzarlo su un computer con Linux, anche se rimane comunque installabile su Windows. Come Editor/IDE si possono usare:

- ⇒ Gedit, Nano, Vim: installati di default sulla maggior parte delle distro linux;
- ⇒ Notepad: per chi decide di utilizzare Windows;
- ⇒ Notepad++;
- ⇒ VsCode: <https://code.visualstudio.com/download> o su linux si può utilizzare il proprio package manager (apt, pacman, yay, etc.). Es. `sudo apt install code`;
- ⇒ IntelliJ: <https://www.jetbrains.com/toolbox-app/>. Dopo aver installato il toolbox si può installare la versione community di IntelliJ (più che sufficiente).

1.7 Linguaggio

Linguaggio scelto: *Haskell*.

Domanda 5

Perché si è scelto questo linguaggio?

Risposta: Haskell è ideale per insegnare il concetto della ricorsione perchè:

- ⇒ ha una sintassi compatta (a differenza di C, Java, etc.): questa non distrae gli studenti e le studentesse dal concetto che si vuole insegnare;
- ⇒ la sua natura puramente funzionale rende quasi impossibile la codifica di programmi che non siano ricorsivi (si possono scrivere con le funzioni di libreria, ma in realtà si sta usando implicitamente la ricorsione);
- ⇒ è strongly-typed, il che può sembrare faticoso inizialmente, ma è molto utile dato che costringe gli alunni e le alunne a utilizzare i tipi in modo corretto (rispetto ad altri linguaggi che supportano operazioni "sporche" come il cast);
- ⇒ migliora la propria abilità nel programmare e nello scrivere codice in generale, spingendo a cercare soluzioni innovative e favorendo una visione dei problemi complessi come entità composte da più problemi semplici.

Capitolo 2

Sviluppo dei contenuti

2.1 Attività 1: Introduzione alla ricorsione

Per gli insegnanti 1

Bisogna inizialmente introdurre in linea generale il contenuto dell'attività agli studenti e alle studentesse, facendo attenzione a non perdersi nei dettagli che verranno approfonditi durante le fasi successive. Questo aiuta ad accendere la curiosità degli alunni e delle alunne e a farli/e partecipare attivamente all'attività.

Note:-

In questo documento ho inserito alcune note per gli/le insegnanti (come quella sopra). Esse fungono da indicazioni o suggerimenti di supporto al/alla docente.

Più avanti in questo capitolo sono presenti gli esercizi che andranno somministrati nel corso dell'attività.

Materiale utilizzato in questa fase:

- ⇒ Lavagna multimediale;
- ⇒ Computers.

Per gli insegnanti 2

In questa fase i computers sono opzionali. Il docente può, a discrezione personale, decidere di condividere sui computers degli studenti e delle studentesse il codice di esempio (per una maggiore interazione) oppure mostrarlo sulla lavagna multimediale.

Fase 1:

- ⇒ **Consegna:** come prima cosa, per introdurre quest'attività si consiglia di ripassare brevemente i tipi di dati (Int, float, etc.). Successivamente il docente introduce la ricorsione spiegando i concetti di passo base e passo ricorsivo. Per supportare quest'attività si può fare riferimento al primo esercizio di programmazione proposto (Fibonacci ricorsivo). Al termine di questa fase si dovranno anche introdurre gli elementi sintattici basilari di Haskell.
- ⇒ **Svolgimento:** inizialmente si presenterà la ricorsione in maniera discorsiva anche facendo esempi legati alla quotidianità (per esempio un **treno**: se ha 1 solo vagone, passo base, allora è semplicemente un vagone. Se ha $n > 1$ vagoni allora è un treno composto da una "testa" e da $n - 1$ vagoni). Eventualmente si possono anche mostrare, su una lavagna multimediale, immagini che presentano la ricorsione (per esempio alcuni dei quadri di Escher come "Mani che disegnano" o "Cascata"). Dopo di ch  gli studenti e le studentesse verranno incoraggiati a partecipare portando ulteriori esempi di ricorsivit  sulla base delle loro esperienze personali. Infine si mostra l'esempio riguardante Fibonacci, scritto in Haskell e si approfitter  per dare un'idea concreta di utilizzo di ricorsione.

- ⇒ **Discussione:** il/la docente dovrà gestire una discussione sulla base degli esempi portati dagli studenti e correggere eventuali errori concettuali tramite il dialogo.
- ⇒ **Conclusione:** Per concludere quest'attività il/la docente introdurrà la sintassi base di Haskell (insieme all'utilizzo di GHCi) e si assicurerà che essa venga compresa dagli studenti.

Per gli insegnanti 3

Può essere utile, prima di proseguire con l'attività unplugged, porre alcune domande agli studenti e alle studentesse per assicurarsi che abbiano compreso le basi su cui si sviluppa la ricorsione. Si può prendere spunto dalla sezione riguardante gli "Indicatori" presente nel capitolo successivo.

Note:-

Comandi utili di GHCi:

- `:load` o `:l`, serve per compilare un file `hs` (Haskell);
- `:type` o `:t`, mostra il tipo di una funzione;
- `:reload` o `:r`, ricompila tutti i file precedentemente compilati;
- `:!clear`, pulisce lo schermo del terminale di GHCi (solo linux).

2.2 Attività 2: Le Matrioske (unplugged)

Materiale utilizzato in questa fase:

- ⇒ Matrioske;
- ⇒ Adesivi;
- ⇒ Fogli di carta;
- ⇒ Penne.

Per gli insegnanti 4

- L'attività è fruibile anche da studenti e studentesse del primo biennio;
- In alternativa agli adesivi si possono usare cartoncini e nastro adesivo;
- Se non ci sono abbastanza matrioske per tutti si possono formare dei gruppetti;
- Il valore che viene utilizzato non è importante, si è scelto di utilizzare "1" a titolo indicativo;
- Se si decide di usare un numero diverso da "1" si deve tenere presente che i passi ricorsivi, ossia le matrioske non più interne, devono avere tutte lo stesso valore.

Fase 1:

- ⇒ **Consegna:** si utilizzano le matrioske per spiegare il concetto di funzione ricorsiva. Come fase preliminare, il docente deve aver attaccato a ogni matrioska un adesivo con scritto "Aprimi e aggiungi 1 al valore al mio interno", con l'eccezione delle matrioske più interne che avranno un adesivo con la scritta "1".
- ⇒ **Svolgimento:**
1. Viene fornita una matrioska (preparata precedentemente dal docente secondo la consegna) a ciascun studente e ciascuna studentessa (o gruppo di studenti);
 2. Vengono fornite penne e fogli di carta a ogni studente e a ogni studentessa;

3. Ogni studente e ogni studentessa legge l'adesivo attaccato alla matrioska e ne riporta la scritta su un foglio;
4. Ogni studente e ogni studentessa apre, se possibile¹, la matrioska;
5. Si ripetono i punti 3 e 4 finché possibile. Se non è possibile si passa al punto 6;
6. Ogni studente e ogni studentessa esegue le istruzioni che ha trascritto sul foglio cominciando dalla penultima (in quanto l'ultima è semplicemente "1") e proseguendo a ritroso.

⇒ **Discussione:** Il docente comincia la discussione chiedendo agli/alle alunni/e che valore è risultato seguendo questo processo. Successivamente si possono portare gli studenti e le studentesse a ragionare su cosa sia questo valore (nel caso dell'esempio proposto con "1" il risultato rappresenta il numero di matrioske di cui è composta la matrioska). Per concludere il dibattito si fa notare che, in questo caso, il risultato si comporta come una variabile "contatore" che aumenta di "1" per ogni matrioska. Questo serve per creare un collegamento cognitivo con il concetto più familiare di iterazione, ma allo stesso tempo mostra che ci sono modi alternativi di vedere un problema.

⇒ **Conclusion:** Il/La docente si assicura che ogni alunno/a abbia capito cosa è stato fatto in questa fase e quali siano le implicazioni di ciò.

2.2.1 Da Unplugged a Programmazione (Fase 2 de "Le Matrioske")

Materiale utilizzato in questa fase:

- ⇒ Fogli di carta;
- ⇒ Penne;
- ⇒ Lavagna multimediale;
- ⇒ Computers.

Per gli insegnanti 5

Non ci si deve soffermare sugli aspetti tecnici del codice in quanto molto avanzati (do, let, show, monade IO()), ma ci si concentra sulla funzione per costruire le matrioske.

La funzione `costruisciMatrioska` riceve in input un numero corrispondente a quante matrioske si vuole racchiudere e restituisce la matrioska corrispondente (ogni matrioska è indicata da un livello numerico).

Fase 2:

⇒ **Consegna:** Si continua a ragionare sulle matrioske e sulla ricorsione utilizzando il codice lasciato in calce a questa sezione. Il codice può essere mostrato sulla lavagna multimediale oppure essere condiviso con gli/le alunni/e in modo che possano vederlo sul proprio computer.

⇒ **Svolgimento:**

1. Mostrare e/o condividere il codice al fondo della sezione con gli studenti e le studentesse;
2. L'insegnante spiega brevemente che cosa fa la funzione `costruisciMatrioska`;
3. Ogni studente e studentessa scrive su un foglio (ricevuto nella fase precedente) un'ipotesi sull'output del programma mostrato;
4. A turno, chi vuole, può provare a illustrare alla classe il risultato a cui si è arrivati spiegando i ragionamenti fatti;
5. Si possono ripetere a piacere i precedenti due passaggi variando l'intero passato come parametro alla funzione `costruisciMatrioska`.

¹La matrioska non ha attaccato l'adesivo con la scritta "1".

- ⇒ **Discussione:** Si può far ragionare su come la rappresentazione di una matrioska sia resa in maniera molto elegante dall'utilizzo della ricorsione e che, utilizzando cicli (programmazione interattiva), non si riuscirebbe a dare efficacemente l'idea di "matrioska annidata dentro un'altra matrioska" che, viene fornita intuitivamente dal codice Haskell mostrato. Per incentivare la discussione si possono fare agli/alunni/e le domande presenti nell'apposita sezione "Indicatori" nel successivo capitolo.
- ⇒ **Conclusione:** L'insegnante si assicura che tutti abbiano prestato attenzione all'esercizio e riassume brevemente i punti fondamentali di questa fase soffermandosi sull'alta leggibilità di una soluzione ricorsiva e sulla sua utilità.

Semplice analisi del codice:

- `data Matrioska...`: rappresenta la dichiarazione di una struttura dati chiamata Matrioska. Questa struttura dati è ricorsiva ed è formata da una "M", una stringa e una lista di Matrioske (per questo è ricorsiva). L'ultima parte (`deriving show`) serve unicamente per la stampa;
- `costruisciMatrioska :: Int -> Matrioska`: è la funzione più importante per quest'attività. Prende in input un intero "n" e restituisce una Matrioska a "n" livelli, ossia composta da "n" matrioske. Per fare ciò se l'intero "n" è diverso da 0 o da 1 chiama ricorsivamente sé stessa passando l'intero "n" meno 1;
- `stampareMatrioske :: Matrioska -> IO()`: utilizzando la monade `IO()`, e la sintassi avanzata di Haskell, stampa una Matrioska;
- `main :: IO()`: il main del programma. Si occupa di costruire una Matrioska a 5 livelli (può essere cambiato a piacimento) e di stampare la Matrioska così creata.

```
1 data Matrioska = M String [Matrioska] deriving Show
2
3 costruisciMatrioska :: Int -> Matrioska
4 costruisciMatrioska 0 = M "Nessuna Matrioska" []
5 costruisciMatrioska 1 = M "Livello 1" []
6 costruisciMatrioska n = M ("Livello " ++ show n) [costruisciMatrioska (n-1)]
7
8 stampareMatrioske :: Matrioska -> IO ()
9 stampareMatrioske (M etichetta bambole) = do
10   putStrLn etichetta
11   mapM_ stampareMatrioske bambole
12
13 -- Esempio.
14 main :: IO ()
15 main = do
16   let catenaDiBambole = costruisciMatrioska 5
17   stampareMatrioske catenaDiBambole
```

2.3 Attività 3: Programmazione in Haskell

Materiale utilizzato in questa fase:

⇒ Lavagna multimediale;

⇒ Computers.

Per gli insegnanti 6

Per ora gli esercizi con la scritta "Parte avanzata" non sono necessari, in quanto possibile estensione dell'attività.

Fase 1:

⇒ **Consegna:** Ripassare brevemente i concetti base di Haskell. Successivamente si andranno a eseguire, individualmente o a coppie (a discrezione dell'insegnante), una serie di esercizi mirati su Haskell.

⇒ **Svolgimento:**

1. Il/La docente introduce brevemente le guardie, le liste, gli operatori e le principali funzioni su di esse;
2. Il/La docente assegna un esercizio per tipologia ("per prendere confidenza", "sulle guardie", "sulle liste", "sulla ricorsione esplicita");
3. Gli/Le alunni/e svolgeranno, individualmente o in coppia, gli esercizi proposti;
4. Ogni studente e ogni studentessa illustra le proprie soluzioni alla classe giustificando le implementazioni scelte.

⇒ **Discussione:** Si discutono le soluzioni avanzate dagli studenti e dalle studentesse. L'insegnante ha il compito di mediare il dialogo e suggerire implementazioni alternative mostrando come, all'interno dell'ambito della ricorsione, ci siano modi differenti di vedere un problema.

⇒ **Conclusione:** Al termine della fase l'insegnante potrà fornire una valutazione degli esercizi utilizzando la rubrica valutativa in fondo al documento.

Note:-

La precedente fase è l'ultima della versione base dell'unità didattica ma è possibile, a seconda dei casi, a una fase ulteriore.

Fase 2:

⇒ **Consegna:** Si trattano concetti un po' più avanzati come le funzioni anonime e gli alberi visti come struttura dati ricorsiva.

⇒ **Svolgimento:**

1. Il/La docente introduce le funzioni anonime e gli alberi;
2. Il/La docente mostra e spiega esercizi sugli alberi tratti dalla sezione apposita (essendo complessi si punta a una spiegazione piuttosto che allo svolgimento da parte degli/delle studenti/studentesse)
3. Gli studenti e le studentesse sono incoraggiati/e a fare domande all'insegnante.

⇒ **Discussione:** Si discute sulla convenienza della rappresentazione ricorsiva di un albero e di come sia efficiente nel caso si debbano effettuare visite (in special modo negli alberi binari di ricerca).

⇒ **Conclusione:** Si riassumono brevemente le idee chiave illustrate e si chiariscono eventuali dubbi degli/delle alunni/e.

2.4 Esercizi di programmazione

2.4.1 Esempio

Fibonacci ricorsivo: viene usato nella prima parte dell'attività come esempio di utilizzo della ricorsione.

```
1 fibonacci :: Int -> Int
2 fibonacci 0 = 0
3 fibonacci 1 = 1
4 fibonacci n = fibonacci (n - 1) + fibonacci (n - 2)
```

2.4.2 Esercizi per prendere confidenza

Somma: Scrivere una funzione in Haskell (con tipo `Int -> Int`), che prenda in input un intero n e calcoli la somma dei primi n numeri naturali. Suggerimento da dare agli/altri alunni/e: usare la formula $\frac{n*(n+1)}{2}$.

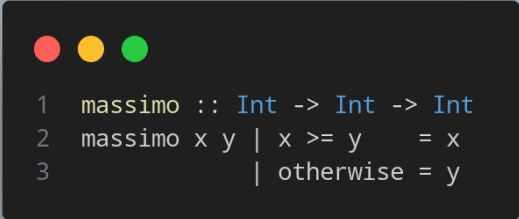
```
1 -- Un'applicazione si può scrivere in notazione infissa usando i backticks.
2 -- Per esempio la somma dei primi n numeri naturali si può scrivere come:
3
4 somma :: Int -> Int
5 somma n = n * (n + 1) `div` 2
6
7 -- Un'applicazione si può scrivere in notazione prefissa.
8 -- Per esempio la somma dei primi n numeri naturali si può scrivere come:
9
10 somma2 :: Int -> Int
11 somma2 n = div (n * (n + 1)) 2
```

Area: Scrivere una funzione in Haskell (con tipo `Float -> Float`), che calcoli l'area di un cerchio prendendo in input il suo raggio. Suggerimento: usare la costante "pi".

```
1 -- Funzione che prende in input il raggio di un cerchio e restituisce la sua area.
2
3 area :: Float -> Float
4 area n = pi * n ^ 2
```

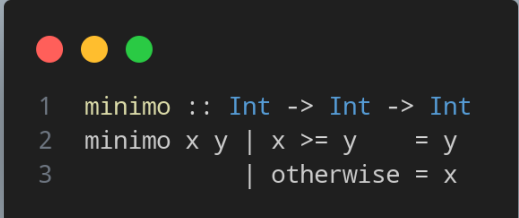

2.4.3 Esercizi sulle guardie

Massimo: Scrivere una funzione in Haskell (con tipo `Int -> Int -> Int`), usando le guardie, che prenda in input due interi e restituisca il maggiore.



```
1 massimo :: Int -> Int -> Int
2 massimo x y | x >= y    = x
3              | otherwise = y
```

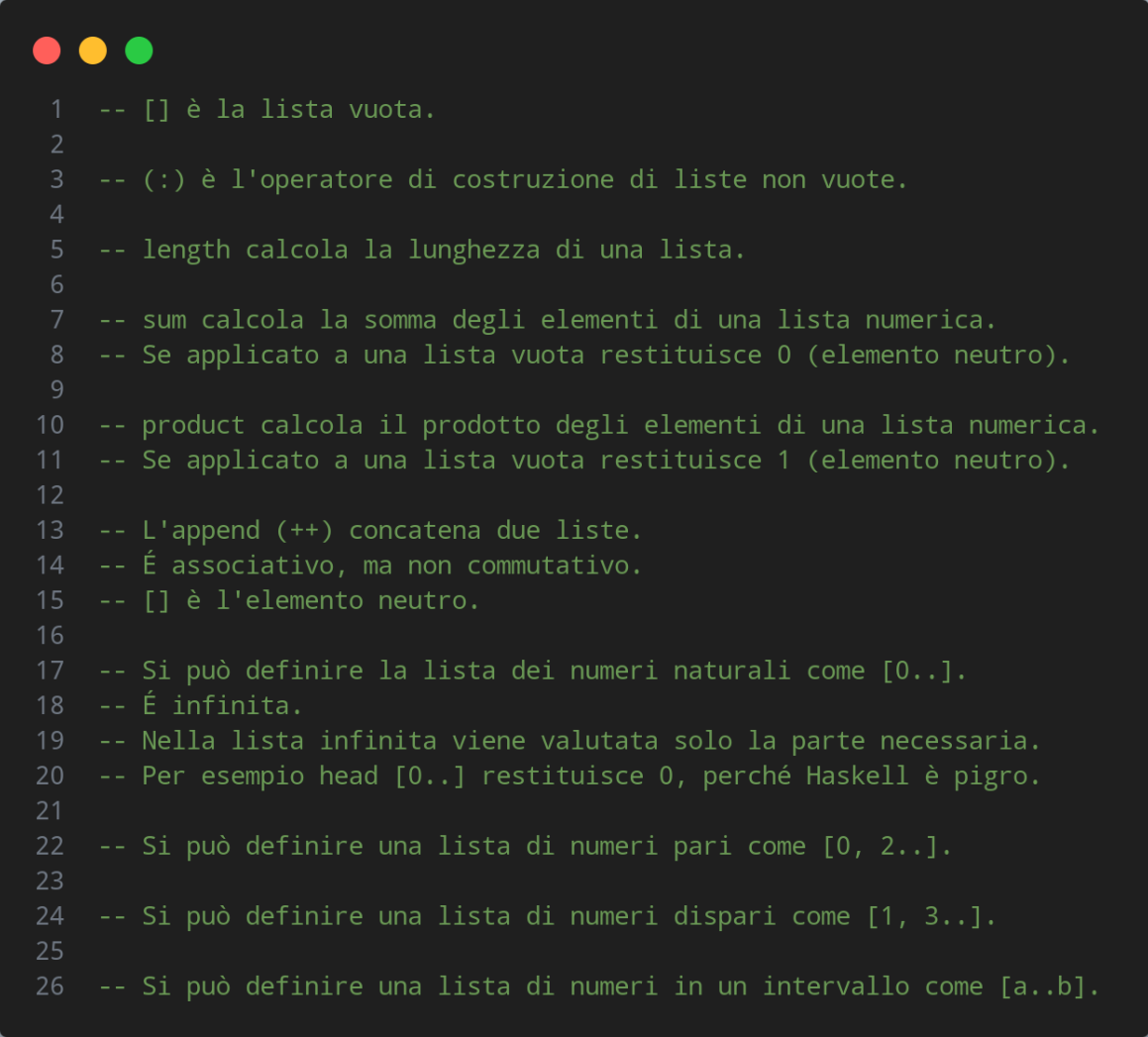
Minimo: Scrivere una funzione in Haskell (con tipo `Int -> Int -> Int`), usando le guardie, che prenda in input due interi e restituisca il minore.



```
1 minimo :: Int -> Int -> Int
2 minimo x y | x >= y    = y
3              | otherwise = x
```

2.4.4 Esercizi sulle liste

Prima di iniziare gli esercizi con le liste si deve condividere questa "dispensa" che illustra le principali caratteristiche delle liste e alcune funzioni su esse.



```
1  -- [] è la lista vuota.
2
3  -- (:) è l'operatore di costruzione di liste non vuote.
4
5  -- length calcola la lunghezza di una lista.
6
7  -- sum calcola la somma degli elementi di una lista numerica.
8  -- Se applicato a una lista vuota restituisce 0 (elemento neutro).
9
10 -- product calcola il prodotto degli elementi di una lista numerica.
11 -- Se applicato a una lista vuota restituisce 1 (elemento neutro).
12
13 -- L'append (++) concatena due liste.
14 -- É associativo, ma non commutativo.
15 -- [] è l'elemento neutro.
16
17 -- Si può definire la lista dei numeri naturali come [0..].
18 -- É infinita.
19 -- Nella lista infinita viene valutata solo la parte necessaria.
20 -- Per esempio head [0..] restituisce 0, perché Haskell è pigro.
21
22 -- Si può definire una lista di numeri pari come [0, 2..].
23
24 -- Si può definire una lista di numeri dispari come [1, 3..].
25
26 -- Si può definire una lista di numeri in un intervallo come [a..b].
```

Fattoriale: Scrivere una funzione in Haskell (con tipo `Int -> Int`), usando le liste, che prende in input un intero e restituisca il fattoriale di quel numero.

Per gli insegnanti 7

Se gli/le alunni/e si trovano in difficoltà si può suggerire loro di far uso della funzione di libreria `product` vista precedentemente.

```
1 -- La funzione fattoriale calcola il fattoriale di un numero.
2 -- product calcola il prodotto degli elementi di una lista.
3 -- [1..x] è la lista dei numeri naturali da 1 a x.
4
5 fattoriale :: Int -> Int
6 fattoriale x = product [1..x]
```

Intervallo: Scrivere una funzione in Haskell (con tipo `Int -> Int -> [Int]`), che dati in input due interi calcoli la lista di interi compresi tra quei due numeri. Suggerimento: usare la ricorsione.

```
1 -- La funzione intervallo calcola la lista dei numeri interi compresi tra due numeri.
2 -- Fa uso di guardie per gestire il caso in cui il primo numero sia maggiore del secondo.
3
4 intervallo :: Int -> Int -> [Int]
5 intervallo m n | m > n      = []
6                  | otherwise = m : intervallo (m + 1) n
```

2.4.5 Esercizi sulla ricorsione esplicita

Somma ricorsiva: Scrivere una funzione in Haskell (con tipo `Int -> Int`), che prenda in input un intero `n` e calcoli la somma dei primi `n` numeri naturali, utilizzando la ricorsione esplicita.

```
1 -- La funzione somma chiama ricorsivamente sè stessa per un numero n (Passo ricorsivo) e ha un passo base (1).
2 -- Rappresenta la somma dei primi n numeri naturali.
3
4 somma :: Int -> Int
5 somma 1 = 1
6 somma n = n + somma (n-1)
```

Fattoriale ricorsivo: Scrivere una funzione in Haskell (con tipo `Int -> Int`), che prenda in input un intero e restituisca il fattoriale di quel numero, usando la ricorsione.

```
1 -- La funzione fattoriale calcola il fattoriale di un numero ricorsivamente.
2 -- Il fattoriale di 0 è 1 (passo base), il fattoriale di n è n * (n - 1) * (n - 2) * ... * 1 (passo ricorsivo).
3
4 fattoriale :: Int -> Int
5 fattoriale n | n == 0    = 1
6               | otherwise = n * fattoriale (n - 1)
7
8 -- Si possono anche usare i pattern matching per definire la funzione fattoriale.
9 -- Il pattern matching ha utilità nelle strutture dati complesse come le liste.
10 -- Sia per le guardie che per il pattern matching l'ordine di scelta è dall'alto verso il basso.
11
12 fattoriale2 :: Int -> Int
13 fattoriale2 0 = 1
14 fattoriale2 n = n * fattoriale2(n - 1)
```

Potenze di 2: Scrivere una funzione in Haskell (con tipo `Int -> Int`), che prenda in input un intero e restituisca 2 elevato a quell'intero, usando la ricorsione esplicita.

```
1 -- La funzione pow2 chiama ricorsivamente sè stessa per un numero n (Passo ricorsivo) e ha un passo base (0).
2 -- Rappresenta la potenza di 2 di un numero n.
3
4 pow2 :: Int -> Int
5 pow2 0 = 1
6 pow2 n = 2 * pow2(n - 1)
```

2.4.6 Parte avanzata: esercizi sugli alberi e sulle funzioni anonime

```
1  -- Il tipo di dato Tree è un tipo polimorfo ricorsivo che può essere definito come segue:
2
3  data Tree a = Leaf | Branch a (Tree a) (Tree a)
4      deriving Show
```

```
1
2  -- La funzione tmax prende un albero binario di ricerca e restituisce il valore massimo contenuto nell'albero.
3
4  tmax :: Tree a -> a
5  tmax (Branch x _ Leaf) = x
6  tmax (Branch _ _ r)    = tmax r
7
8  -- La funzione insert prende un valore di tipo a e un albero binario di ricerca e restituisce un albero binario di ricerca
9  -- che contiene il valore inserito nell'albero.
10
11 insert :: Ord a => a -> Tree a -> Tree a
12 insert x Leaf = Branch x Leaf Leaf
13 insert x t@(Branch y l r) | x == y = t
14                           | x < y  = Branch y (insert x l) r
15                           | x > y  = Branch y l (insert x r)
16
17 -- La funzione delete prende un valore di tipo a e un albero binario di ricerca e restituisce un albero binario di ricerca
18 -- che non contiene il valore eliminato dall'albero.
19
20 delete :: Ord a => a -> Tree a -> Tree a
21 delete _ Leaf = Leaf
22 delete x (Branch y t1 t2) | x < y = Branch y (delete x t1) t2
23                           | x > y = Branch y t1 (delete x t2)
24 delete x (Branch _ t Leaf) = t
25 delete x (Branch _ Leaf t) = t
26 delete x (Branch _ t1 t2) = Branch y (delete y t1) t2
27   where
28     y = tmax t1
29
30 -- La funzione empty determina se un albero è vuoto.
31
32 empty :: Tree a -> Bool
33 empty Leaf = True
34 empty _    = False
35
36 -- La funzione depht calcola la profondità di un albero.
37
38 depth :: Tree a -> Int
39 depth Leaf = 0
40 depth (Branch _ t1 t2) = 1 + max (depth t1) (depth t2)
41
```

```

1
2  -- La funzione elements effettua una visita in ordine infisso e restituisce la lista corrispodente.
3
4  elements :: Tree a -> [a]
5  elements Leaf = []
6  elements (Branch x t1 t2) = elements t1 ++ [x] ++ elements t2
7
8  -- La funzione tmin prende un albero binario di ricerca e restituisce il valore massimo contenuto nell'albero.
9
10 tmin :: Tree a -> a
11 tmin (Branch x Leaf _) = x
12 tmin (Branch _ t _) = tmin t
13
14 -- La funzione tmin prende un albero binario di ricerca e restituisce il valore massimo contenuto nell'albero.
15 -- Questa versione funziona anche sugli alberi vuoti.
16
17 tmint :: Tree a -> Maybe a
18 tmint Leaf = Nothing
19 tmint (Branch x t _) = case tmint t of
20     Nothing -> Just x
21     Just y -> Just y
22
23 treeSort :: Ord a => [a] -> [a]
24 treeSort = elements . foldr insert Leaf
25
26 -- La funzione bst restituisce True se un albero è un albero binario di ricerca, false altrimenti.
27
28 bst :: Ord a => Tree a -> Bool
29 bst Leaf = True
30 bst (Branch x t1 t2) = bst t1 && bst t2 &&
31     (empty t1 || tmax t1 < x) &&
32     (empty t2 || x < tmin t2)

```

```

1  -- Una funzione anonima (anche detta lambda espressione) è una funzione senza nome.
2  -- Una funzione anonima può essere applicata direttamente ad un argomento.
3  -- Il corpo di una funzione anonima si estende il più possibile a destra.
4
5  successore = \x -> x + 1

```


Capitolo 3

Guida per gli insegnanti

3.1 Snodi e indicatori per fasi

Per gli insegnanti 8

Come specificato più volte nel documento gli "Indicatori" sono strutturati in modo da essere posti direttamente, agli studenti e alle studentesse, sotto forma di quesiti.

3.1.1 Attività 1

Fase 1:

- Snodi:
 - ⇒ Capire la struttura di una funzione ricorsiva.
 - ⇒ Analizzare un programma ricorsivo.
 - ⇒ Comprensione della sintassi di Haskell.
- Indicatori:
 - ⇒ Quali sono le caratteristiche di una funzione ricorsiva?
 - ⇒ Identificare il passo base e il passo ricorsivo nel programma "Fibonacci ricorsivo".
 - ⇒ Elencare i principali tipi di Haskell.

3.1.2 Attività 2

Fase 1:

- Snodi:
 - ⇒ Capire cosa simboleggia il risultato ottenuto eseguendo le istruzioni sulle matrioske.
 - ⇒ Mettere in corrispondenza il concetto iterativo di "contatore" con il risultato ricorsivo ottenuto durante l'attività.
 - ⇒ Capire l'importanza del passo base.
 - ⇒ Comprensione dell'esistenza di soluzioni più "appropriate" per la risoluzione di problemi.
- Indicatori:
 - ⇒ Che risultato si è ottenuto? Cosa rappresenta? Soluzione: si è ottenuto il numero di matrioske "annidate" l'una dentro l'altra.
 - ⇒ Era possibile raggiungere un simile risultato in un altro modo? Se sì, quale? Soluzione: si potevano aprire le matrioske e contarle una a una.

- ⇒ Che cosa succede se, quando rimonto una matrioska, non ne inserisco una (esclusa quella più interna) e riprovo a fare l'intera attività? Soluzione: il numero ottenuto questa volta sarà diminuito di "1".
- ⇒ Che cosa succede se, quando rimonto una matrioska, non inserisco quella più interna e riprovo a fare l'intera attività? Soluzione: in mancanza di un passo base il processo fallisce (dato che le matrioske sono finite), ma in alcune situazioni computazionali si potrebbe arrivare ad avere una ricorsione infinita.

Fase 2:

- Snodi:
 - ⇒ Simulare l'esecuzione di un programma ricorsivo e saperne predire l'output.
 - ⇒ Comprendere l'esistenza di strutture dati che riferiscono sé stesse (Matrioska).
- Indicatori:
 - ⇒ Qual è l'output del programma se alla funzione `costruisciMatrioska` passo 3 invece che 5? E se invece passo 7?
 - ⇒ Cosa succede se passo 0 alla funzione `costruisciMatrioska`? Soluzione: l'output è "Nessuna Matrioska".

3.1.3 Attività 3

Fase 1:

- Snodi:
 - ⇒ Saper utilizzare forma infissa e forma prefissa di un'applicazione di funzione.
 - ⇒ Saper scrivere una funzione con più argomenti.
 - ⇒ Riconoscere la maggiore leggibilità di una funzione che utilizza le guardie rispetto alla stessa funzione che utilizza un IF.
 - ⇒ Conoscere le principali liste che si possono definire (`[0..n]`, `[0, 2..]`, `[1, 3..]`, `[a..b]`).
 - ⇒ Conoscere le principali funzioni che operano sulle liste.
 - ⇒ Utilizzare correttamente la ricorsione per risolvere problemi.
- Indicatori:
 - ⇒ Quali sono i due modi di scrivere `div`? Soluzione: notazione infissa (con i backtick ```) e prefissa.
 - ⇒ Come si scrive la "firma" di una funzione che prende due interi e restituisce una stringa? Soluzione: `Int -> Int -> String`.
 - ⇒ Come si genera una lista che va da 4 a 9? Soluzione: `[4..9]`.

Fase 2:

- Snodi:
 - ⇒ Comprensione della struttura ricorsiva alla base degli alberi.
 - ⇒ Osservare che alcune funzioni possono essere riutilizzate facilmente (`tmax`, `tmin`) in altri contesti e favoriscono una soluzione elegante e leggibile.
 - ⇒ Saper riconoscere un algoritmo riguardante gli alberi in Haskell.
- Indicatori:
 - ⇒ Come viene effettuata una visita di un albero con la funzione `elements` e cosa restituisce?
 - ⇒ Quale funzione serve per capire se un albero è di ricerca e su quali altre funzioni si basa?
 - ⇒ Qual è il parametro della funzione `successore`?

Capitolo 4

Indicazioni per la valutazione

4.1 Valutazione durante le lezioni/attività

La valutazione può essere fatta sugli esercizi di programmazione presentati in questo documento oppure se ne possono creare di nuovi modificando quelli trattati a lezione. In questa sezione sono presenti:

- ⇒ una tabella che fornisce la collocazione degli esercizi nel block model;
- ⇒ una breve miscellanea di possibili misconceptions che possono avere gli/le studenti/studentesse;
- ⇒ una proposta di rubrica valutativa.

4.2 Riguardo gli esercizi di programmazione...

4.2.1 Block Model

Definizione 4.2.1: Block Model

Il Block Model è un framework usato per classificare e analizzare vari aspetti della comprensione del codice.
Legenda Comprensione: T (Text Surface), P (Program Execution), F (Function/Purpose).
Legenda Strategia: A (Atoms), B (Block/Chunks), R (Relationship), M (Macrostructure).

Esercizio	Comprensione	Strategia
Fibonacci ricorsivo	F	B
Somma	T	A
Area	P	A
Massimo	P	B
Minimo	P	B
Fattoriale	T	B
Intervallo	P	R
Somma ricorsiva	F	M
Fattoriale ricorsivo	F	M
Potenze di 2	F	M
tmax	F	R
insert	F	M
delete	P	M
empty	F	A
depth	F	R
elements	F	R
tmin	F	R

tmint	F	M
treeSort	P	N
bst	P	M
successore	T	A

4.2.2 Misconceptions

- ⇒ Gli/Le studenti/studentesse hanno difficoltà a concepire una lista infinita $[0..]$, $[0, 2..]$, etc.;
- ⇒ Mancanza del passaggio cognitivo che collega una soluzione iterativa a una soluzione ricorsivo;
- ⇒ Errata associazione tra array e liste (sono due cose diverse);
- ⇒ Problemi nella comprensione del tipo di una funzione;

4.3 Rubrica valutativa

Questa piccola rubrica valutativa può essere utilizzata nell'ambito della fase di esercitazione in Haskell come linea guida di valutazione. A discrezione del docente si possono aumentare le colonne, diversificandole, in modo da avere una valutazione più precisa.

	Assente	Parziale	Adeguate
Comprensione teorica	Conoscenza della teoria gravemente insufficiente e/o assente.	Conoscenza della teoria sufficiente.	Conoscenza della teoria completa e/o approfondita.
Sintassi di Haskell	Mancanza di comprensione della sintassi di Haskell.	Comprensione della sintassi base di Haskell (variabili, guardie, etc.), difficoltà con concetti come "Inferenza di tipo" e/o "Pattern matching".	Ottima padronanza di Haskell e di GHCI.
Esercizi	Esercizi non svolti e/o svolti in maniera scorretta.	Esercizi svolti in maniera parzialmente corretta, ma con qualche lacuna.	Esercizi svolti correttamente rispetto alle consegne.
Padronanza della terminologia tecnica	Terminologia assente e/o totalmente inadeguata.	Terminologia sostanzialmente corretta, ma con alcune imprecisioni.	Terminologia pienamente corretta.
Comprensione della ricorsione	Assenza del concetto di ricorsione e/o idea completamente sbagliata della ricorsività.	Comprensione basilare della ricorsione.	Comprensione eccellente della ricorsione e delle sue implicazioni.

