



UNIVERSITÀ DI PISA

Corso di Laurea in Informatica Umanistica

RELAZIONE

Pensiero Computazionale

**Sue applicazioni nei principali concetti algebrico-matematici
nella scuola media inferiore e biennio superiori**

Candidato: *Elisa Iacopini*

Relatore: *Prof.ssa Linda Pagli*

Correlatore: *Prof. Alessandro Lenci*

Anno Accademico 2016-2017

*Ai miei genitori, grazie al loro sostegno ho potuto raggiungere questo traguardo.
Alla nonna Anna che mi manca tanto.*

Indice generale

1. Introduzione	4
2. Il pensiero computazionale nella scuola	7
2.1 Come nasce l'espressione pensiero computazionale	7
2.2 Cosa si intende per pensiero computazionale	12
2.3 Imparare il pensiero computazionale	14
3. Introduzione al linguaggio di programmazione	17
3.1 Dall'algoritmo al programma	18
3.2 Strutture dati: variabili e liste	25
3.2.1 Liste	37
3.3 Input e output	38
3.4 Funzioni	40
3.5 Cicli iterativi. Ciclo while	43
3.5.1 Ciclo for	48
3.6 Comando condizionale if-else	50
3.7 Esempi di applicazione del pensiero computazionale	58
4. Conclusioni	77
5. Bibliografia	78

1.Introduzione

Argomento di questo studio è il pensiero computazionale. Questo studio prende in considerazione diversi aspetti: la nascita del termine, il concetto racchiuso in esso, la sua applicazione pratica e la sua importanza come competenza trasversale da aggiungere a tutte le discipline didattiche.

Il pensiero computazionale è un termine entrato ormai a far parte del nostro linguaggio quotidiano, indica le abilità necessarie allo studio e alla pratica dell'informatica, infatti con esso si intende ragionare come un informatico.

Questo lavoro vuole dimostrare come il pensiero computazionale possa essere utilizzato per realizzare in modo semplice calcoli algebrici relativi ad argomenti matematici tipici della scuola media inferiore e del biennio della secondaria. Il linguaggio di programmazione scelto è il Python, che si presta più di altri linguaggi all'implementazione di soluzioni del genere sopra detto. Inoltre questo lavoro si pone come scopo quello di fornire agli studenti della scuola secondaria di primo grado e biennio di secondo grado, uno strumento per comprendere gli aspetti fondamentali del pensiero computazionale e apprenderne le abilità.

L'idea di questa tesi nasce dall'esperienza vissuta frequentando il corso di laurea in Informatica Umanistica che ha stimolato la voglia di capire il mondo che si cela dietro le pagine digitali e interattive. Grazie al percorso di studi seguito ho sviluppato una maggiore capacità di valutare i diversi contesti guardandoli da più prospettive, apprendendo in tal modo l'essenza del pensiero computazionale.

I ragazzi oggi sono costantemente connessi con un mondo digitale in continua evoluzione, ma i concetti fondamentali dell'informatica non cambiano. È importante che gli alunni capiscano che i mondi virtuali in cui sono abituati a muoversi sono progettati prima di tutto su carta.

Una delle finalità del pensiero computazionale è quella di sviluppare la capacità intellettuale necessaria per individuare la procedura risolutiva di semplici problemi matematici esclusivamente con la logica e successivamente di concretizzarla usando i fondamenti dei linguaggi di programmazione. Sono disponibili diverse applicazioni che in teoria supportano il *pensiero computazionale* ma in realtà una volta appreso il loro meccanismo di funzionamento il processo di apprendimento diventa meccanico e gli stimoli logici tendono a scomparire.

Nel 2015 è entrata in vigore la legge n.107, nella quale, tra gli obiettivi formativi prioritari, viene considerato lo «sviluppo delle competenze digitali degli studenti, con particolare riguardo al pensiero computazionale, all'utilizzo critico e consapevole dei social network e dei media» (art.1, c.7 lettera h). La legge introduce altresì il Piano Nazionale Scuola Digitale mediante il quale si ribadisce il concetto che un piano per l'educazione nell'era digitale non è un semplice dispiegamento di tecnologia. Il Piano prevede la formazione dei docenti per l'innovazione didattica, il

potenziamento degli strumenti tecnologici e laboratoriali a disposizione delle scuole e la realizzazione di attività finalizzate a sviluppare le competenze digitali degli alunni. La formazione che gli insegnanti devono maturare non riguarda solo l'alfabetizzazione digitale, ma è un tipo di formazione che mette in campo abilità tecniche, cognitive e culturali legate alla competenza digitale richiesta ad un docente.

La tesi è così composta: un capitolo introduttivo, cui segue un capitolo che illustra la nascita dell'espressione pensiero computazionale, il suo significato e la sua applicazione. Come sarà più dettagliatamente detto nel capitolo dedicato, l'espressione pensiero computazionale nasce nel momento in cui si comprende l'importanza di insegnare l'informatica ai bambini e ai ragazzi, e ci si chiede come ciò possa essere realizzato. Precursori sono Seymour Papert, il quale ritiene che l'apprendimento avviene soprattutto attraverso l'esperienza: costruire, maneggiare oggetti reali, procedere per prove ed errori. Il computer quindi diviene una macchina per simulare esperimenti, infatti Papert vede l'uso del computer come supporto all'istruzione e come ambiente d'apprendimento che favorisce lo sviluppo di nuove idee.

È nel 2006 che il termine diviene noto per mezzo dell'articolo *Computational Thinking*, di Jeannette Wing, la quale ritiene che il pensiero computazionale sia un'«abilità fondamentale» da aggiungere «alla lettura, alla scrittura, all'aritmetica» e quindi valida per chiunque, non solo per gli informatici. Il pensiero computazionale deriva dai concetti fondamentali dell'informatica, metodi rigorosi che affrontano e risolvono in modo preciso un determinato problema.

Nel 2012 la Royal Society, considerando altamente insoddisfacente l'istruzione informatica delle scuole del Regno Unito, fornisce delle raccomandazioni nel rapporto *“Shut down or restart? The way forward for computing in UK schools”*. In Francia, è stata l'Académie des Sciences¹ nel 2013 a sollevare la questione con il rapporto *L'enseignement de l'informatique en France. Il est urgent de ne plus attendre*. Negli Stati Uniti, nel 2015, è stato approvato lo *“Every Student Succeed Act”* (ESSA), un atto legislativo che riconosce l'informatica come un'abilità da inserire tra le *“well rounded educational subject”*, ossia l'insieme delle materie che vengono insegnate a scuola, come, per esempio, la matematica, la madrelingua e le scienze. In Italia il Consorzio Interuniversitario Nazionale per l'Informatica (CINI) ha avviato, a partire dall'anno scolastico 2014-2015, il progetto Programma il Futuro, in collaborazione con il Ministero dell'Istruzione e Università e Ricerca (MIUR). L'obiettivo è fornire alle scuole una serie di strumenti semplici, divertenti e facilmente accessibili, per formare gli studenti ai concetti di base dell'informatica. In una società costantemente interconnessa è essenziale lavorare non solo

¹L'Accademia Nazionale degli Scienziati della Francia, fondata nel 1666 da Luigi XIV per incoraggiare e proteggere lo spirito della ricerca scientifica francese. Fu in prima linea negli sviluppi scientifici in Europa nei secoli XVII e XVIII.

sull'alfabetizzazione digitale, ma anche sullo sviluppo di abilità tecniche per l'utilizzo degli strumenti digitali in modo da favorire l'introduzione al pensiero logico-computazionale e la familiarizzazione con gli aspetti operativi delle tecnologie informatiche. «Il pensiero computazionale è l'insieme dei processi mentali usati per modellare una situazione e specificare i modi mediante i quali un agente elaboratore di informazioni può operare in modo effettivo all'interno della situazione stessa per raggiungere uno o più obiettivi forniti dall'esterno»[1].

La definizione data mette in evidenza gli aspetti fondamentali legati al concetto di pensiero computazionale:

processo mentale, si riferisce alle attività mentali utilizzate per affrontare e risolvere problemi, indipendentemente dall'applicazione di tecnologie o strumenti.

agente in grado di elaborare le informazioni si riferisce ad un esecutore (persona o macchina) che sia in grado di comprenderle ed eseguirle in autonomia. Parlare di *agente* significa voler distinguere il processo mentale da quello esecutivo, per ciò le istruzioni individuate devono essere acquisite dall'esecutore per ottenere i risultati auspicati.

raggiungimento degli obiettivi, fa riferimento ad una precisa situazione in cui l'esecutore opera per raggiungere degli obiettivi che non si assegna autonomamente, ma sono forniti dal contesto.

Spesso si tende a confondere il pensiero computazionale con le tecniche di *problem solving*. Il termine pensiero computazionale definisce la capacità di individuare anche la procedura da seguire per raggiungere l'obiettivo non solo la capacità di scomporre il problema in sotto-problemi e risolverli. A questi aspetti fondamentali è opportuno aggiungerne uno di cui non si fa menzione nelle definizioni di cui sopra, ma che costituisce una prassi nell'implementazione di soluzioni informatiche, così come nella vita quotidiana: verificare il funzionamento delle soluzioni attraverso la prova concreta. Uno dei punti di forza del pensiero computazionale è quello di abituare a vedere e verificare un ragionamento mediante una rappresentazione rigorosa, ciò aiuta a individuare un contesto extra-verbale entro cui inserire il proprio ragionamento. Il capitolo 3 è dedicato alla presentazione e alla discussione delle principali strutture dati e dei costrutti sintattici necessari per trasformare il pensiero computazionale in procedure da usare per raggiungere obiettivi prefissati. Inoltre il capitolo è corredato con diversi esempi che illustrano in modo dettagliato l'uso degli strumenti linguistici di Python che, si ricorda, è il linguaggio scelto per implementare le soluzioni ai problemi matematici presi in esame.

2. Il pensiero computazionale nella scuola

2.1 Come nasce l'espressione "pensiero computazionale"

L'espressione pensiero computazionale diviene celebre nel momento in cui si comincia a interrogarsi su l'importanza di insegnare l'informatica nella scuola primaria e secondaria e su come debba essere fatto.

Il termine viene introdotto da Seymour Papert, matematico, informatico e pedagogista che negli anni '60 va al MIT² a lavorare con il gruppo di quella università che si occupa di Intelligenza Artificiale. Egli ritiene che l'apprendimento avvenga soprattutto attraverso l'esperienza: costruire, maneggiare oggetti reali, procedere per prove ed errori. In tale contesto, il computer diviene una macchina per simulare esperimenti, Papert sostiene l'uso del computer come supporto all'istruzione e ambiente d'apprendimento che favorisce lo sviluppo di nuove idee. L'attenzione è rivolta in particolare ai bambini, ma egli ritiene che ciò valga per qualunque età, adolescenti e adulti. Secondo Papert la lentezza di elaborazione di un determinato concetto da parte del bambino non è dovuta alla maggiore complessità o formalità, ma alla «povertà della conoscenza di quei materiali che renderebbero il concetto semplice e concreto».[2]

Analogamente, sono da ricordare come precursori Donald Knuth e George Forsythe, non tanto per aver parlato esplicitamente di pensiero computazionale, ma per aver sollevato l'attenzione su questo nuovo aspetto.

Donald Knuth rinomato informatico statunitense e studioso di matematica³, professore emerito alla Stanford University, nel 1974 ha scritto «In realtà una persona non ha davvero capito qualcosa fino a che non è in grado di insegnarla ad un computer». [3]

George Forsythe, informatico e analista numerico, ha contribuito a istituire l'informatica come disciplina accademica, è infatti tra i fondatori del Dipartimento di Computer Science della Stanford University, uno dei primi a nascere e tuttora tra i migliori al mondo. Nel 1968 ha scritto: «Le acquisizioni più valide nell'educazione scientifica e tecnologica sono queglii strumenti mentali di tipo generale che rimangono utili per tutta la vita. Ritengo che il linguaggio naturale e la matematica siano i due strumenti più importanti in questo senso, e l'informatica sia il terzo». [4]

Ma è nel 2006 che il termine diviene noto per mezzo dell'articolo *Computational Thinking*, di Jeannette Wing, direttrice dell'Istituto di Data Sciences della Columbia University, in cui insegna informatica.

2 Il Massachusetts Institute of Technology (MIT) è la più importante università di ricerca del mondo, con sede a Cambridge, nel Massachusetts. Il MIT gode ormai di una notevole reputazione per la qualità dell'insegnamento e della ricerca. Al MIT si trovano alcuni dei complessi tecnologici più all'avanguardia del mondo, tra cui un laboratorio di intelligenza artificiale.

3 Creatore del linguaggio Latex e autore di *The Art of Computer Programming* i sulla programmazione di algoritmi e la relativa analisi formale degli stessi (4 volumi).

Anche la Wing non fornisce una definizione completa di pensiero computazionale, mette in evidenza come sia necessario comprendere le potenzialità ed i limiti di un processo informatico, indipendentemente dal fatto che venga eseguito da un umano o una macchina. Ritiene che il pensiero computazionale sia un'«abilità fondamentale» da aggiungere «alla lettura, alla scrittura, all'aritmetica» e quindi valida per chiunque, non solo per gli informatici. Il pensiero computazionale deriva dai concetti fondamentali dell'informatica, metodi rigorosi che affrontano e risolvono in modo preciso un determinato problema. Con il pensiero computazionale vengono acquisiti strumenti mentali che permettono di riformulare il problema. Pensare come un informatico non significa solo essere in grado di programmare, va oltre. Non si tratta di un'abilità meccanica, ripetitiva come, per esempio, imparare a memoria, significa essere capace di concettualizzare e riuscire pensare a più livelli di astrazione.

La Wing prosegue puntualizzando che pensiero computazionale non significa pensare come un computer, «è un modo con cui gli esseri umani possono risolvere un problema». Gli esseri umani sono dotati di intelligenza e fantasia, a differenza dei computer che sono noiosi e monotoni, ma che ci permettono di affrontare problemi che, prima dell'era dei computer, non avremmo mai osato prendere in esame. Pertanto la Wing ritiene opportuno «diffondere la gioia, i timori e la potenza dell'informatica, con l'obiettivo di rendere popolare il pensiero computazionale.»[5]

Nel 2012 la Royal Society⁴, considerando altamente insoddisfacente l'istruzione informatica delle scuole del Regno Unito, fornisce delle raccomandazioni nel rapporto “*Shut down or restart? The way forward for computing in UK schools*”⁵, il risultato di un progetto avviato nel 2010. In questo rapporto, il pensiero computazionale viene considerato come un modo per visualizzare e comprendere il mondo in cui viviamo. Si considera come aspetto essenziale la possibilità che tutti gli alunni acquisiscano familiarità con i concetti ed i principi dell'informatica in modo da offrire loro l'opportunità di sviluppare le attitudini necessarie al loro beneficio individuale e alla futura prosperità della nazione. Quindi comprendere il mondo digitale per avere la possibilità di assumere un ruolo attivo anziché passivo, e apprendere le basi scientifiche della tecnologia informatica per poter partecipare in modo più intelligente, concreto e critico ai problemi principali della società contemporanea.

Il Ministero dell'Istruzione nel 2013 ha pubblicato il *National curriculum in England: computing programs of study* ⁶ divenuto obbligatorio per tutte le scuole a partire dal 2014. Il documento si apre asserendo che «Un'istruzione di elevata qualità nell'informatica fornisce agli allievi la capacità di usare il pensiero computazionale e la creatività per comprendere e cambiare il mondo». [6]

4 Un'associazione scientifica britannica, fondata nel 1660 per iniziativa di accademici allo scopo di promuovere l'eccellenza scientifica come viatico per il benessere della società.

5 Chiudi o riavvia? La via da seguire per l'informatica nelle scuole del Regno Unito.

6 Programma scolastico nazionale relativo all'insegnamento dell'informatica.

Attraverso l'informatica agli alunni vengono insegnati i principi dell'informazione e del calcolo. L'uso del computer ha legami profondi con la matematica, la scienza, il design e la tecnologia e garantisce inoltre che gli alunni diventino alfabetizzati digitalmente, siano in grado di usare, esprimersi e sviluppare le loro idee attraverso le tecnologie dell'informazione e della comunicazione, raggiungendo un livello di consapevolezza adatto sia per un futuro lavoro sia come partecipanti attivi in un mondo digitale.

In Francia, è stata la Académie des Sciences⁷ nel 2013 a sollevare la questione con il rapporto *L'enseignement de l'informatique en France. Il est urgent de ne plus attendre*⁸. Il rapporto tratta della collocazione dell'informatica nell'istruzione primaria e secondaria, e mette altresì in evidenza la necessità di un'adeguata formazione degli insegnanti. L'insegnamento dell'informatica dovrebbe consentire a tutti gli studenti, compresi quelli che non diventeranno informatici, di comprendere il mondo digitale che li circonda, di padroneggiarlo e di accedere a nuove forme di pensiero che accompagnano lo sviluppo del mondo dell'informatica. L'informatica ha un'importanza sempre maggiore in termini di creazione di ricchezza e di posti di lavoro nel mondo, sia direttamente nell'industria informatica sia in aree importanti come l'aeronautica, l'industria automobilistica e le telecomunicazioni. L'Académie sottolinea come l'Europa e la Francia, in particolare, presentano un significativo ritardo concettuale e industriale nel settore rispetto ai paesi più dinamici, come gli Stati Uniti e alcuni paesi asiatici. Un ritardo parzialmente correlato alle carenze dell'educazione informatica, che porta a consumare e subire ciò che viene deciso altrove, impedendo di partecipare allo sviluppo in modo attivo. L'insegnamento deve essere rivolto a tutti i cittadini per comprendere i meccanismi e i modi di pensare del mondo digitale che li circonda e da cui dipendono.

Tuttavia, considerato quanto sia indispensabile l'educazione nelle pratiche digitali il semplice uso di software, computer e reti non dà un reale contributo in termini di educazione informatica. Lo sviluppo del mondo digitale è strettamente legato al progresso dell'informatica, che è diventata una scienza autonoma con le sue forme specifiche di pensiero. Se gli oggetti e le applicazioni digitali si stanno evolvendo a un ritmo costante, l'informatica rimane basata su un insieme stabile e omogeneo di concetti e conoscenze.

La decisione essenziale da prendere è quella di istituire un'educazione informatica dalla scuola elementare alla scuola superiore, orientata alla comprensione e all'alfabetizzazione informatica, e quindi ben oltre il semplice uso di hardware e software, perciò «imparare la programmazione permette di scoprire i rudimenti del pensiero computazionale».

⁷ L'Accademia Nazionale degli Scienziati della Francia, fondata nel 1666 da Luigi XIV per incoraggiare e proteggere lo spirito della ricerca scientifica francese. Fu in prima linea negli sviluppi scientifici in Europa nei secoli XVII e XVIII.

⁸ *L'insegnamento dell'informatica in Francia. È urgente non aspettare oltre.*

Il rapporto ricorda che ormai gli strumenti digitali interagiscono nell'industria, comunicazione, ricreazione, cultura, salute, scienza e società, tuttavia l'informatica non è più un semplice strumento di calcolo ma «porta ad una nova forma di pensiero, chiamato pensiero computazionale».[7]

Negli Stati Uniti, nel 2015, è stato approvato lo “*Every Student Succeeds Act*” (ESSA), un atto legislativo che riconosce l'informatica come un'abilità da inserire tra le “*well rounded educational subject*”⁹, ossia l'insieme delle materie che vengono insegnate a scuola, come, per esempio, la matematica, la madrelingua e le scienze. ESSA rappresenta il passaggio successivo rispetto alla precedente versione, la legge *No Child Left Behind* (NCLB)¹⁰, promulgata nel 2002, attraverso la quale sono stati messi in evidenza gli aspetti nei quali gli studenti stavano facendo progressi e quelli in cui, invece, avevano bisogno di ulteriore supporto.

In ESSA, si ritiene che l'informatica sia un'abilità fondamentale per l'educazione scolastica, dalla prima elementare all'ultimo anno delle superiori. L'obiettivo è quello di fornire a tutti gli studenti un'educazione formativa ampliata e adeguata all'era in cui viviamo, per favorire più opportunità di successo in ambito scolastico e lavorativo.

In un'economia che sta rapidamente cambiando, *Computer Science For All*¹¹ è l'iniziativa lanciata nel 2016 dal Presidente Obama consapevole dell'importanza dell'informatica come abilità fondamentale per nuove opportunità economiche e mobilità sociale. L'obiettivo di questo programma è quello di insegnare l'informatica dall'asilo alle superiori, e consentire così l'apprendimento «delle abilità di pensiero computazionale necessaria per essere creatori, e non semplici consumatori, nell'economia digitale, e cittadini attivi di una società sempre più tecnologica».[8]

In Italia, il Consorzio Interuniversitario Nazionale per l'Informatica (CINI), ha avviato a partire dall'anno scolastico 2014-15 il progetto Programma il Futuro, in collaborazione con il Ministero dell'Istruzione, Università e Ricerca (MIUR).

L'obiettivo è fornire alle scuole una serie di strumenti semplici, divertenti e facilmente accessibili, per formare gli studenti ai concetti di base dell'informatica. Il progetto ritiene che ciascuno studente debba essere consapevole, e non consumatore passivo, delle tecnologie e dei servizi digitali di cui oggi disponiamo per affrontare la società moderna. L'educazione al pensiero computazionale aiuta a sviluppare competenze logiche e capacità di risolvere problemi in modo creativo ed efficiente, e va oltre all'iniziale alfabetizzazione digitale. Secondo il Programma «Il modo più semplice e divertente di sviluppare il pensiero computazionale è attraverso la programmazione (coding) in un contesto di gioco.»[9]

Nel 2015 è entrata in vigore la legge n.107, nella quale, tra gli obiettivi formativi prioritari, viene considerato lo «sviluppo delle competenze digitali degli studenti,

9 *Materie per un'istruzione a tutto tondo.*

10 Non lasciare nessun bambino indietro nel processo educativo.

11 Informatica per tutti.

con particolare riguardo al pensiero computazionale, all'utilizzo critico e consapevole dei social network e dei media» (art.1, c.7 lettera h). La legge prevede altresì l'adozione del "Piano Nazionale per la Scuola Digitale" (PNSD), che persegue la realizzazione di attività che sviluppino le competenze digitali degli studenti, potenziamento degli strumenti didattici e formazione dei docenti (art.1, c.56 e 58).

Più precisamente, nel documento viene espressa la volontà di sviluppare abilità trasversali adeguate al nostro secolo. In una società interconnessa, basata sulle conoscenze e l'informazione, è essenziale lavorare sull'alfabetizzazione informativa e digitale, considerando l'introduzione al pensiero logico e computazionale e la familiarizzazione con gli aspetti operativi delle tecnologie informatiche. È previsto inoltre, che gli studenti della scuola primaria svolgano un corpus di 10 ore annuali di educazione al pensiero logico-computazionale (Azione 17).

2.2 Cosa si in tende per pensiero computazionale

L'articolo *Computational Thinking* ha dato luogo ad un' appassionante confronto, tutt'ora in corso, su quale sia l'esatta definizione di pensiero computazionale, e nonostante i corposi ed autorevoli interventi citati sopra, manca una definizione operativa universalmente accettata.

La Wing fornisce una definizione più completa solo in successivi interventi: «Computational thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that be effectively carried out by an information processing agent.»¹²

Nel Programma il Futuro il concetto di pensiero computazionale viene definito come un insieme di «procedure che vengono poi attuate da un esecutore, che opera nell'ambito di un contesto prefissato, per raggiungere degli obiettivi assegnati».[9] Più recente e puntuale la definizione di Nardelli, professore di informatica presso l'Università di Roma "Tor Vergata" e coordinatore del progetto "Programma il futuro", «Il pensiero computazionale è l'insieme dei processi mentali usati per modellare una situazione e specificare i modi mediante i quali un agente elaboratore di informazioni può operare in modo effettivo all'interno della situazione stessa per raggiungere uno o più obiettivi forniti dall'esterno».[10]

Le definizioni citate mettono in evidenza gli aspetti fondamentali che costituiscono il concetto di pensiero computazionale:

- *processo mentale*
- *agente in grado di elaborare informazioni*
- *raggiungere degli obiettivi*

Il primo aspetto, *processo mentale*, si riferisce alle attività mentali utilizzate per affrontare e risolvere problemi, indipendentemente dall'applicazione di tecnologie o strumenti. Significa utilizzare il ragionamento per esaminare un determinato fatto, riuscire a scomporlo in sotto-problemi semplici cercando di eliminare dettagli inutili per concentrarsi su ciò che è rilevante.

Le istruzioni elementari individuate devono essere disposte nel giusto ordine, in modo che tutto funzioni correttamente. Viene, cioè, individuato l'algoritmo che permette di raggiungere l'obiettivo in una sequenza ordinata di passi nel modo più efficiente possibile.

Questo genere di processo mentale viene quotidianamente compiuto da ciascun individuo, pensiamo, per esempio, al percorso per raggiungere il luogo di lavoro in automobile:

- si pensa al percorso complessivo

¹² Il pensiero computazionale è l'insieme dei processi mentali usati per formulare i problemi e le loro soluzioni in modo tale che la descrizione delle soluzioni sia effettivamente eseguibile da un agente che elabora le informazioni. Questa descrizione si trova nella stesura di un articolo che non è mai pubblicato, dal titolo *Demystifying Computational Thinking for Non-Computer Scientists* (2010).

- viene scomposto nelle varie strade riflettendo sugli ostacoli che si possono incontrare (semafori, traffico, passaggi a livello, ztl,...)
- le strade alternative individuate saranno seguite nell'ordine pensato per arrivare il prima possibile.

La questione cambierebbe se al posto dell'automobile potessimo utilizzare la bicicletta: viene elaborata un'ulteriore astrazione perché la situazione è decisamente mutata, ma l'obiettivo rimane quello di trovare un percorso che sia il più efficiente possibile.

Il termine *agente in grado di elaborare le informazioni* si riferisce ad un esecutore (persona o macchina) che sia in grado di comprenderle ed eseguirle in autonomia. Parlare di *agente* significa voler distinguere il processo mentale da quello esecutivo, per ciò le istruzioni individuate devono essere acquisite dall'esecutore per ottenere i risultati auspicati.[]

L'esecutore non deve essere necessariamente un computer, può trattarsi di un essere umano: i passi individuati per definire il percorso per arrivare ad un certo luogo (come abbiamo visto sopra per raggiungere la sede di lavoro), possono risultare utili a qualcun altro ed essere replicati. La capacità di fornire precise istruzioni, coordinare in modo chiaro e preciso il lavoro dei collaboratori, deve essere considerata un'abilità rilevante che può essere appresa e allenata attraverso il pensiero computazionale.

L'esecutore deve eseguire istruzioni in autonomia rispetto al soggetto che ha ragionato sul modo di arrivare alla soluzione. È necessario prendere in considerazione la differenza tra un agente umano ed uno meccanico, in quanto cambiano i modi di formulare il problema. Nel primo caso, può esserci una comunicazione orale oppure scritta, quest'ultima, a sua volta, può essere elaborata a parole o schemi, ed eventuali lacune possono essere colmate dall'intelligenza della persona. Nell'altro caso, invece, è necessario un linguaggio rigoroso che renda esplicito ogni dettaglio e non lasci spazio ad ambiguità.

Il terzo aspetto, *raggiungere degli obiettivi*, fa riferimento ad una precisa situazione in cui l'esecutore opera per raggiungere degli obiettivi che non si assegna autonomamente, ma sono forniti dal contesto. Non si tratta esclusivamente di un problema che deve essere risolto. Gli errori più comuni consistono nell'equiparare il pensiero computazionale al *problem solving*, ossia la competenza a risolvere problemi, e nel ritenere che l'essenza sia quella di essere in grado di scomporre il problema in sotto-problemi. La visione corretta parte dal considerare che «i matematici hanno risolto problemi per millenni» e che «solo quando hanno iniziato a riflettere sulla possibilità e sulle implicazioni di far risolvere i problemi a qualcun altro, si è messa in moto un'evoluzione che ha portato alla nascita della nuova disciplina scientifica, l'informatica». [12] E spesso, scomporre il problema, non è

così necessario, è rilevante invece avere chiaro il quadro del problema considerato un certo contesto.

A questi aspetti fondamentali è opportuno aggiungerne uno di cui non si fa menzione nelle definizioni citate sopra, ma che costituisce una prassi nell'implementazione di soluzioni informatiche, così come nella vita quotidiana: verificare il funzionamento delle soluzioni attraverso la prova concreta. Il percorso ritenuto più idoneo per raggiungere il luogo di lavoro, preso come esempio in precedenza, viene utilizzato e valutato, e, se necessario, apportate le opportune correzioni.

2.3 Imparare il pensiero computazionale

«Il pensiero computazionale costituisce una meta-competenza», una competenza che aiuta ad acquisire altre competenze, «perché la sua applicazione sviluppa la capacità di mettersi nei panni di un esecutore che deve raggiungere un obiettivo e di formulare una procedura che lo guidi ad ottenere tale risultato dato il contesto in cui opera». [13]

Con l'espressione pensiero computazionale viene indicata l'informatica insegnata nella scuola primaria e secondaria. L'espressione viene utilizzata per una necessità di sintesi dato che il termine informatica non presenta una distinzione di aree di competenza, per cui ne fanno parte concetti che spaziano dalla teoria della computabilità, alla architettura delle reti di computer, ai sistemi *peer to peer*.

Pertanto valutare la necessità di insegnare l'informatica nella scuola, non deve essere confuso con l'intento di insegnare l'uso di una determinata tecnologia o un certo strumento, ma rappresenta la volontà di far acquisire i concetti culturali e scientifici di base che rimangono dopo aver studiato e praticato l'informatica.

La matematica e l'italiano vengono insegnati già alla scuola elementare perché considerati competenze fondamentali e trasversali, importanti sia come materie da apprendere, sia come discipline da applicare ad altri contesti.

L'italiano è alla base delle relazioni sociali, e permette altresì di esprimere concetti che riguardano qualunque oggetto, ma «possiede una sua intrinseca ambiguità»[14] che non gli consente quella precisione che è necessaria in alcuni casi.

La matematica consente di superare le lacune della lingua naturale perché il suo formalismo permette di descrivere gli eventi con precisione assoluta. Con la matematica siamo in grado di misurare con precisione e stabilire relazioni numeriche, non si possono auspicare progressi utilizzando solo concetti generici come poco e abbastanza. Tuttavia se alla teoria non seguono conseguenze pratiche, questo formalismo può risultare improduttivo, come è accaduto «nella biologia in cui diversi aspetti erano stati modellati matematicamente già molti decenni fa. Ma solo con l'applicazione dell'informatica sono stati raggiunti risultati senza precedenti»[15].

Con l'informatica è possibile trattare casi particolari ed eccezioni, definire un insieme di regole e animarle, «rappresentare qualunque mondo vogliamo esplorare e dargli senso»[16]. Si pensi ai fenomeni fisici, biologici o sociali di cui si conoscono le leggi e grazie all'informatica è possibile crearne delle simulazioni evitando quindi di costruire ogni volta una rappresentazione fisica del modello logico-matematico.

In questo modo è possibile affrontare problemi reali, perseverare nel trattare problemi difficili, imparare attraverso l'esperienza concreta e essere creativi mediante la computazione.

Come la matematica e l'italiano, l'informatica, intesa come scienza e non come tecnologia, è una competenza fondamentale e trasversale con un prezioso valore educativo che deve essere insegnata nelle scuole.

Si è così diffuso il concetto di *coding*, termine inglese per indicare la programmazione informatica. Anzitutto, condividendo a pieno il pensiero di Nardelli, trovo doveroso sottolineare che non vi è alcuna necessità intrinseca di sostituire il termine italiano con quello inglese, e che «utilizzare la nostra lingua significa sostenere la nostra identità sociale e culturale».[17]

In secondo luogo, è importante sottolineare che pensiero computazionale non significa imparare a programmare.

Un programma informatico è la rappresentazione del pensiero computazionale che dà concretezza al processo mentale retrostante, così come un'espressione linguistica è la rappresentazione del pensiero. Mediante l'elaborazione di un programma lo studente è costretto a riflettere, ad essere critico sulle istruzioni impartite: l'esecutore è un meccanismo automatico privo di qualunque intelligenza, pertanto non in grado di colmare eventuali omissioni non previste dal suo programma. Fare pratica in condizioni difficili può essere un valido allenamento per preparare gli studenti a qualunque futuro lavoro. Programmare è uno degli strumenti mediante i quali si può apprendere il pensiero computazionale.

Il pensiero computazionale, la logica con cui gli informatici vedono e rappresentano la realtà, costituisce un nuovo modo di affrontare situazioni quotidiane e professionali: valutare problemi complessi, immaginare una descrizione chiara di come e quando intervenire, ipotizzare soluzioni suddivise, saper coinvolgere e guidare altre persone.

Per insegnare il pensiero computazionale non sono indispensabili dispositivi digitali e infrastrutture tecnologiche, è molto più importante la formazione di insegnanti che siano in grado di realizzare attività didattiche stimolanti. Da tenere in considerazione altresì, senza voler demonizzare la tecnologia che ci consente di essere sempre connessi, che studi scientifici hanno messo in evidenza come le capacità cognitive vengano ostacolate da smartphone e tablet, questo disturbo è conosciuto come effetto *brain drain*. Probabilmente «non è un caso che nella Silicon

Valley¹³, da ormai molti anni, i figli della classe dirigente studino in scuole senza smartphone né tablet, ma con libri di carta e laboratori».[18]

Ormai la nostra è una società digitale in cui i ragazzi utilizzano con estrema naturalezza i dispositivi tecnologici, pertanto è opportuno fornire loro i concetti logici fondamentali dell'informatica al fine di avere consumatori consapevoli e attivi nello sviluppo del loro futuro e dell'intera società.

A conclusione di questa descrizione, può essere apprezzabile prendere in considerazione una sequenza tratta dal film Apollo 13[19], che esprime in maniera concreta e concisa l'essenza del pensiero computazionale.

Gli astronauti a bordo del modulo lunare hanno il filtro per la purificazione dell'aria danneggiato, e gli esperti del centro di controllo devono risolvere il problema costruendo un nuovo filtro con il materiale a disposizione nel modulo lunare, e successivamente fornire agli astronauti le istruzioni per assemblare il nuovo filtro.

In questo caso gli esperti del centro di controllo devono ragionare come un informatico:

- valutano ed elaborano il problema
- collaborano per raggiungere l'obiettivo
- forniscono agli astronauti le istruzioni precise, ossia l'algoritmo, per costruire il nuovo filtro
- in un linguaggio che devono essere in grado di comprendere

13 È l'area della California nella quale sono situate alcune delle maggiori aziende di alta tecnologia come Intel, IBM, Google, Facebook, Ebay.

3. Introduzione al linguaggio di programmazione

Il pensiero computazionale è il fondamento dell'informatica, la quale tuttavia necessita di strumenti capaci di realizzare le procedure individuate col pensiero computazionale, tali strumenti sono i linguaggi di programmazione che sono un altro pilastro su cui poggia l'informatizzazione. Un linguaggio di programmazione è uno strumento rigoroso, in quanto supera le ambiguità del linguaggio naturale, creato per fornire precise istruzioni ad un esecutore meccanico privo di intelligenza. Sebbene il processo evolutivo dell'informatica è in continua evoluzione, le strutture dati e i costrutti linguistici restano concettualmente gli stessi per tutti i linguaggi di programmazione, anche i più complessi.

L'Orsa del Codice, in inglese *The Hour of Code*, è un'iniziativa nata negli Stati Uniti nel 2013 per far sì che ogni studente, in ogni scuola del mondo, svolga almeno un'ora di programmazione. In Italia, a partire da settembre 2014 quest'iniziativa è stata proposta a tutte le scuole nell'ambito del progetto *Programma il Futuro*, ed è stata realizzata usando strumenti di facile utilizzo che non richiedono un'abilità avanzata nell'uso del computer. L'obiettivo non è quello di far diventare tutti dei programmatori informatici, ma quello di fornire agli studenti la possibilità di misurarsi in nuove esperienze.

Un altro metodo è rappresentato da *Scratch*, un ambiente di programmazione gratuito, utilizzabile online, basato su un linguaggio di programmazione a blocchi. *Scratch* consente di realizzare contenuti digitali interattivi come storie, animazioni, giochi. È stato sviluppato dal gruppo Lifelong Kindergarten del MIT Media Lab, guidato da Mitchel Resnick, la prima versione desktop di Scratch è del 2003.

Nei paragrafi seguenti sono presentati esempi in cui sono presi in considerazione calcoli algebrici per risolvere problemi di matematica e geometria utilizzando strutture dati e costrutti del linguaggio Python, invece di utilizzare ambienti predefiniti già disponibili come quelli sopra citati. Gli argomenti trattati negli esempi fatti hanno come finalità quella di mostrare il funzionamento della logica computazionale ed apprendere le competenze attraverso semplici esperienze di programmazione.

È stato scelto il linguaggio Python in quanto dispone di strumenti e funzioni avanzate che consentono di creare applicazioni complesse, nonostante ciò la sua sintassi di base e lo stile di scrittura estremamente lineare lo rendono un linguaggio semplice, pertanto facilmente accessibile a tutti. Di contro Java, che è il linguaggio ad oggetti più diffuso al mondo ed anche il più utilizzato sul lavoro, non è adatto per perseguire gli scopi di questa tesi in quanto, per poterlo utilizzare, è necessario prendere confidenza con concetti complessi come quello di *Classe*, *Metodo*, *Oggetto* e *Istanza*, pertanto non è adatto a programmatori poco esperti. Python, essendo facile

da imparare, si presta ad essere usato da chi è un principiante della programmazione, ciò rappresenta un punto di forza di questo linguaggio.

3.1 Dall' algoritmo al programma

Il concetto di pensiero computazionale che è correlato al concetto di **algoritmo**. Il termine algoritmo deriva dalla trascrizione latina del nome del matematico persiano al-Khwarizmi, vissuto nel IX secolo D.C., che è considerato uno dei primi ad aver introdotto il concetto di algoritmo scrivendo il libro *Regole di ripristino e riduzione* [20].

Per poter realizzare un algoritmo è necessario individuare il problema di cui l'algoritmo rappresenta la soluzione. Per raggiungere un qualunque obiettivo, come ad esempio la risoluzione di un problema di matematica o l'individuazione del percorso più efficiente per arrivare in palestra, è necessario raccogliere e analizzare le informazioni che servono per arrivare alla soluzione. In questa prima fase bisogna riconoscere quali dati, tra quelli disponibili, servono e quali sono ridondanti, valutare se sono sufficienti a raggiungere lo scopo e se non sono ambigui.

«Un algoritmo è un procedimento che risolve un determinato problema attraverso un numero finito di passi elementari in un tempo ragionevole» [21].

«Termine che indicò nel medioevo i procedimenti di calcolo numerico fondati sopra l'uso delle cifre arabe. Nell'uso odierno, anche con riferimento all'uso dei calcolatori, qualunque schema o procedimento matematico di calcolo; più precisamente, un procedimento di calcolo esplicito e descrivibile con un numero finito di regole che conduce al risultato dopo un numero finito di operazioni, cioè di applicazioni delle regole»[22].

« In informatica, insieme di istruzioni che deve essere applicato per eseguire un'elaborazione o risolvere un problema»[23].

La parola algoritmo può far pensare a chissà quali procedimenti complessi, in realtà eseguiamo un algoritmo molto più spesso di quanto pensiamo, ad esempio quando seguiamo i passi di una ricetta di cucina, le istruzioni di montaggio per i mattoncini Lego e risolviamo un'espressione algebrica. Solitamente una ricetta di cucina è descritta a parole, le istruzioni di montaggio sono presentate come schema e l'espressione è rappresentata mediante la notazione matematica. Dagli esempi appena citati si evidenzia che per poter parlare di algoritmo devono essere presenti le seguenti proprietà:

A

Istruzioni semplici, non ambigue, ed elaborate in sequenza

B

Raggiungere un obiettivo conosciuto

C

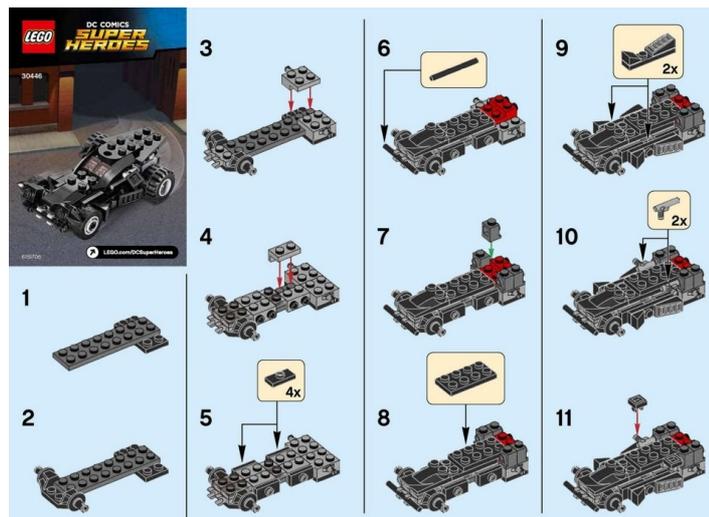
Esecuzione in un tempo finito

Fase **A**: l'algoritmo è composto da istruzioni essenziali, comprensibili da chi deve eseguire il procedimento, ed inoltre devono essere realizzate in sequenza. Osserviamo gli esempi di seguito.

Ricetta per il tiramisù:

1. *separare i tuorli dagli albumi*
2. *con le fruste elettriche montare i tuorli e metà dose di zucchero*
3. *aggiungere il mascarpone*
4. *pulire le fruste*
5. *montare a neve gli albumi versando il restante zucchero*
-
10. *livellate la superficie e lasciate rassodare per un paio d'ore in frigorifero*
11. *prima di servire il tiramisù spolverizzate la superficie con del cacao amaro*

Le istruzioni Lego:



L'espressione algebrica:

$$5+10+(32-9)*5$$

Dalle sequenze delle tre situazioni presentate viene messa in evidenza la necessità di elaborare i passaggi secondo un procedimento ordinato.

Fase **B**. Le istruzioni sono eseguite per raggiungere un determinato scopo, nel nostro caso fare il tiramisù, assemblare i mattoncini Lego e calcolare l'espressione.

Fase **C**. L'obiettivo deve essere raggiunto in un tempo limitato e conveniente: se ho degli amici a cena, il tiramisù deve essere pronto, non avrebbe senso terminarlo il giorno seguente.

Come tutte le macchine il computer non ha nessuna capacità decisionale o discrezionale ma si limita a compiere determinate azioni secondo procedure prestabilite, i programmi. Si può affermare che il computer è in grado di compiere un'unica azione: eseguire istruzioni. In ambito computazionale gli algoritmi rappresentano le sequenze di operazioni da eseguire per svolgere un determinato compito, tuttavia affinché posano essere eseguiti dal computer necessitano di essere tradotti in un linguaggio comprensibile per la macchina. Per fare ciò vengono utilizzati i **linguaggi di programmazione**.

Il linguaggio naturale rappresenta lo strumento grazie al quale gli individui riescono a comunicare tra loro. Il linguaggio può essere verbale (scritto o parlato), gestuale, visivo o musicale, in particolare, il linguaggio verbale si sviluppa seguendo delle regole precise: la sintassi e la semantica. La **sintassi** studia come le parole si combinano per formare la frase, la **semantica** si occupa del significato da attribuire alle parole e alle frasi. La sintassi e la semantica non sono uguali per tutti i linguaggi verbali.

Il linguaggio naturale è ambiguo, ovvero un'espressione può avere diversi significati, cioè può essere soggetta a più interpretazioni, pertanto non è adatto per comunicare con il computer. Un linguaggio di programmazione, invece, è una lingua formale utilizzata per descrivere in modo univoco l'insieme di azioni consecutive che un computer deve eseguire, come il linguaggio naturale, ha una sintassi e una semantica. Un **linguaggio formale** utilizza informazioni chiare e non presenta ambiguità, cioè ad un'informazione non si possono attribuire più significati. Questo tipo di linguaggio utilizza i connettivi logici grazie ai quali è possibile costruire frasi complesse formate dall'unione di frasi più semplici. Per esempio nella frase:

Ho visto Dario con il binocolo

non è chiaro se Dario è stato visto attraverso il binocolo, o se era Dario ad avere il binocolo. Se fossimo stati presenti all'evento non avremmo avuto alcun dubbio.

Esistono diversi linguaggi di programmazione (C, C++, Java, Python, ecc.) ognuno con delle sue peculiarità e la scelta dell'uno o dell'altro dipende da diversi fattori come efficienza, efficacia, adattabilità, portabilità e altro ancora.

Python è il linguaggio formale che più di altri si presta per lo scopo di questa trattazione. Python è stato sviluppato nel 1991 dall'informatico olandese Guido van Rossum, e il nome deriva dalla passione del suo ideatore per il gruppo di comici Monty Python. È un linguaggio potente, di facile apprendimento e lettura perché semplice e intuitivo. Viene impiegato per la realizzazione di desktop, siti e applicazioni web, calcolo scientifico e numerico, interfacce grafiche, giochi, database, ed altro. La NASA, Google e YouTube usano Python. Nel periodo in cui è stato scritto questo documento le versioni disponibili sono la 2.7.14 e la 3.6.4.

Per comprendere il significato di sintassi e semantica di un linguaggio analizziamo i seguenti esempi:

1) *Matteo mangiano la mela.*

“Matteo mangiano” è sintatticamente sbagliata perché, secondo la grammatica italiana, il verbo deve concordare con il soggetto nella persona e nel numero; la frase sintatticamente corretta è “Matteo mangia...”. Dal punto di vista semantico, tuttavia, è una frase comprensibile.

2) *La mela mangia Matteo.*

È una frase sintatticamente corretta, ma non semanticamente perché nella realtà è improbabile che si verifichi una tale situazione.

Per comprendere meglio la differenza tra sintassi e semantica utilizziamo il comando *if-else* di Python, il cui significato sintattico è: *se condizione vera fai questo altrimenti fai un'altra cosa*. Esempio:

a) *se Maria supera l'esame (condizione da verificare)*
può andare in vacanza (condizione verificata)
altrimenti
rimane a casa (condizione non verificata)

utilizzando l'apposito costrutto Python la frase di sopra si traduce come segue:

```
if Maria supera l'esame:  
    _può andare in vacanza ← Istruzione indentata di if  
else:  
    _rimane a casa ← Istruzione indentata di else
```

La regola sintattica del costrutto usato nell'esempio è:

```
if condizione  
    istruzione  
else  
    istruzione
```

Come si può notare sono state seguite delle regole di posizionamento che in un linguaggio di programmazione sono chiamate regole di indentazione. L'**indentazione** è un rientro rispetto al margine sinistro, una convenzione tipica dei linguaggi di programmazione per facilitare la lettura dei programmi. In Python l'indentazione ha un preciso ruolo sintattico, indica il blocco di istruzioni che formano il comando.

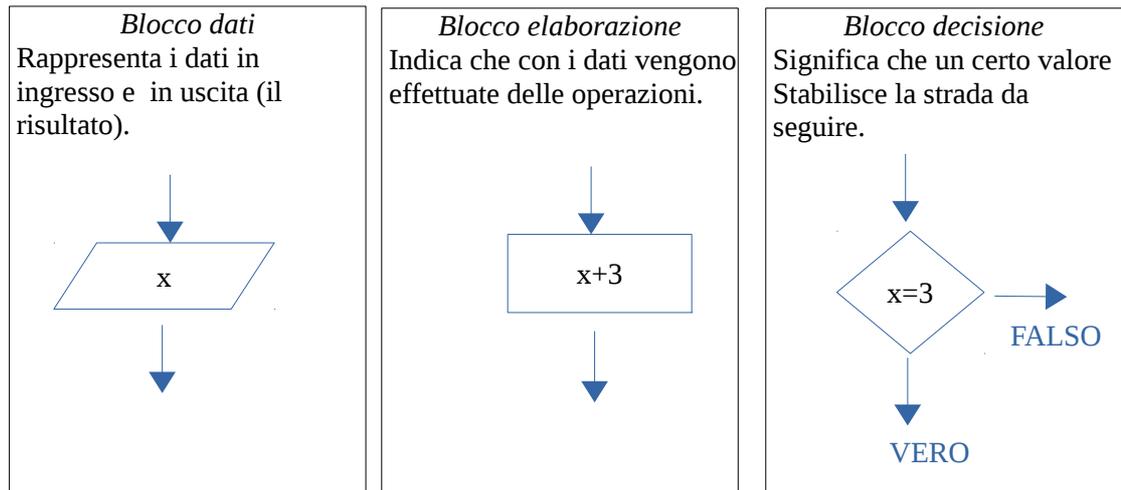
c) *if* Maria supera l'esame:
 _può andare in vacanza
else:
rimane a casa ← Istruzione non indentata

Nell'esempio di sopra la sintassi del costrutto *if-else* non è corretta perché il comando "rimane a casa" non segue le regole di indentazione.

Regola di ogni linguaggio di programmazione è quella di assegnare un nome al programma che realizza l'algoritmo. Supponiamo di dover calcolare il saldo del conto corrente, possiamo chiamare *Calcola_saldo* il programma che lo implementa. Un file è un contenitore di dati che ha un nome e un'estensione, cioè un suffisso posto dopo il nome del file e preceduto dal punto, tale suffisso varia da linguaggio a linguaggio. Il contenitore del nostro programma sarà un file a cui daremo un nome, *Calcola_saldo.py*, dove ".py" è l'estensione di Python. Se utilizziamo, per esempio, il linguaggio editore Word per scrivere una ricerca sul sistema solare, il file è *Sistema_solare.docx*, se usiamo LibreOffice abbiamo invece *Sistema_solare.odt*. Il file che contiene le foto delle vacanze al mare può essere *Mare_2017.jpg*.

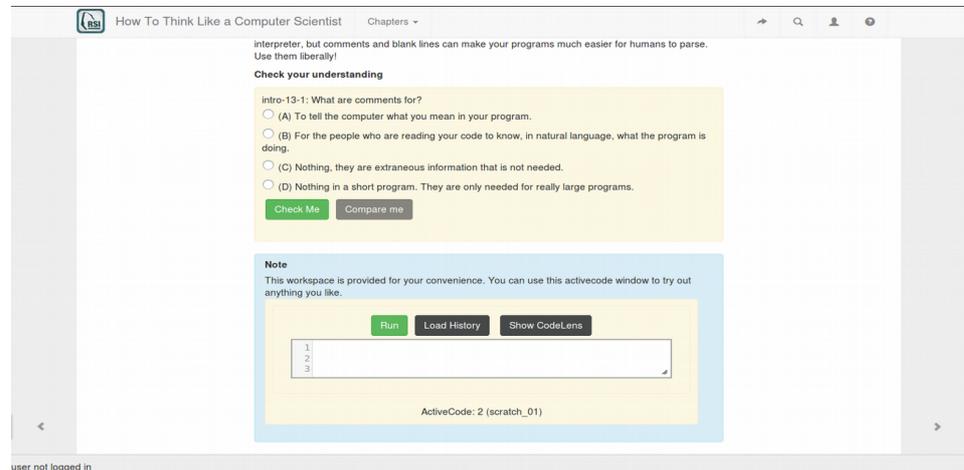
I file audio hanno attualmente l'estensione `.mp3`. Quindi, il programma visto sopra diventa `Calcola_saldo.py`, Sarebbe `Calcola_saldo.java` se il linguaggio usato fosse Java.

Nei capitoli seguenti impareremo le principali strutture di Python e analizzeremo esempi di piccoli programmi con l'aiuto di schemi, tabelle e diagrammi di flusso. Questi ultimi sono rappresentazioni grafiche utilizzate per rappresentare i passaggi informali che portano alla stesura di un programma. I principali elementi che compongono i diagrammi di flusso sono:



Per poter eseguire una eventuale verifica del programma, è possibile utilizzare un ambiente di lavoro (*workspace*) disponibile nel libro interattivo *How to Think like a Computer Scientist – Interactive Python*¹⁴. Una volta aperto il libro interattivo, digitare il capitolo *1.13 Comments* e andare in fondo alla pagina:

Figura 3.1.1



La maschera colorata di azzurro (Figura 3.1.3) mostra l'ambiente a disposizione per eseguire le verifiche. È possibile digitare i comandi nello spazio bianco, poi il tasto verde *Run* per visualizzare il risultato. Per vedere i passaggi del programma si digita il tasto *Show in Codelens*, per chiudere la finestra il tasto *Hide Codelens*.

Figura 3.1.2



¹⁴ www.interactivepython.org/runestone/static/thinkcspy/index.html

È un prodotto open source del Runestone Interactive Project (runestoneinteractive.org, runestone.academy) guidato da Brad Miller e David Ranum.

3.2 Strutture dati: variabili e liste

Le **variabili** possono essere considerate come contenitori nei quali memorizziamo informazioni utili, che verranno riutilizzate o modificate. Il termine per identificarle, successivamente viene scelto dal programmatore. Per esempio, abbiamo un salvadanaio che contiene 20 euro:

variabile `salvadanaio` = `20` valore

Questa annotazione indica che alla variabile `salvadanaio` è stato assegnato il valore 20. È necessario ricordare una regola fondamentale

VARIABLE = VALORE

a sinistra del simbolo “=” (uguale) abbiamo sempre il nome della variabile, mentre a destra c’è sempre il valore da assegnarle.

Le variabili sono identificate attraverso dei vocaboli scelti dal programmatore. Possiamo quindi avere “pluto”, “supercar”, ecc., ma è preferibile che abbiano dei nomi significativi, adeguati al contesto, per permettere una comprensione più immediata. Nell’esempio precedente, al posto di `salvadanaio` avremmo anche potuto utilizzare un altro termine:

`paperino=20`

ma si nota come non sia spontaneo pensare ad un salvadanaio contenente dei soldi.

Nei linguaggi di programmazione esistono le **keywords**, parole chiave, chiamate così perché eseguono precise funzioni e per questo non possono essere usate come nomi di variabili. In Python, e in tanti altri linguaggi di programmazione, ci sono, per esempio, le parole chiave **and**, **else**, **for**, **if**, **in**, **not**, **or**, **print**, **while**, **True**, **False**, che incontreremo più avanti. Queste sono le più comuni, ma ce ne sono altre di cui parleremo via via.

Le variabili non hanno una lunghezza prestabilita, possono essere composte da più parole, oppure da parole e cifre:

```
salvadanaiomio=15
salvadanaio2=25
```

Python prevede precise regole per assegnare i nomi delle variabili:

- il nome di una variabile non deve contenere spazi

```
salvadanaio mio=15 ERRORE!
```

```
salvadanaiomio=5
```

- possono contenere il trattino basso "_", per rendere più leggibili i nomi composti

```
salvadanaio_mio=25
```

- non possono iniziare con un numero

```
23lattine=8 ERRORE!
```

```
lattine23=8
```

- non possono contenere caratteri speciali come &, \$, /

```
& lattine=15 ERRORE!
```

```
sconto$=10 ERRORE!
```

Python fa distinzione tra caratteri minuscoli e maiuscoli, per cui la parola "cane" è diversa da "Cane", tuttavia, per semplicità ed evitare errori, i caratteri maiuscoli si usano con attenzione.

Le variabili che hanno come valore un numero sono di tipo numerico, e possono essere modificate attraverso operazioni matematiche.

Supponiamo di avere un salvadanaio elettronico che registra i movimenti e mostra il totale aggiornato sul suo display. La nonna ci regala 10€ che li mettiamo nel salvadanaio. La variabile, a cui prima abbiamo assegnato 20, viene ora modificata con un'operazione di somma:

```
salvadanaio = salvadanaio + 10
```

È come scrivere `salvadanaio = 20+10`, ma l'operazione di addizione viene eseguita direttamente con la variabile `salvadanaio` perché ci possono essere state molte operazioni che la coinvolgono e che cambiano continuamente il suo valore. È necessario memorizzare tutte le variazioni in modo da poter effettuare i calcoli sui valori aggiornati. Se non si tiene traccia di tutte le modifiche si rischiano errori che possono avere conseguenze importanti.

Consideriamo ora lo scambio di valore tra due variabili. Supponiamo di avere due salvadanai (`s1` e `s2`), e di voler scambiare i rispettivi contenuti:

```
- s1=500.000
```

```
- s2=20
```

Il risultato dello scambio deve essere:

```
- s1=20
```

```
- s2=500.000
```

Se scriviamo `s1=s2`, che significa "s1 prende il valore di s2": `s1=20` e `s2=20`

Se scriviamo `s2=s1`, che significa "s2 prende il valore di s1": `s1=500` e `s2=500`.

Nel primo caso abbiamo perso il valore di `s1`, nel secondo il valore di `s2`.

Quando si fa l'assegnamento di un valore a una variabile si perde il dato inserito in precedenza, pertanto è necessaria una variabile di appoggio che ci permetta di memorizzare momentaneamente il valore di un conto per poi trasferirlo all'altro:

- `appoggio=salvadanaio1`

la variabile `appoggio` prende il valore di `salvadanaio1`: `appoggio=500.000`

- `salvadanaio1=salvadanaio2`

`salvadanaio1` prende il valore di `salvadanaio2`: `salvadanaio1=20`

- `salvadanaio2=appoggio`

`salvadanaio2` prende il valore di `salvadanaio1`: `salvadanaio2=500.000`

In Python esiste un comando per identificare e controllare il tipo numerico: **int()** che indica un intero, e **float()** che indica un numero con la virgola:

`a=int(12)` corrisponde a 12

`a=float(12)` inserisce la virgola: 12,0

`b=int(23,5)` elimina i valori che seguono la virgola: 23

`b=float(23,5)` corrisponde a 23,5

In Python gli operatori numerici sono:

Simboli	Operatori	Espressioni
+	addizione	$7+4=11$
-	sottrazione	$7-4=3$
*	moltiplicazione	$7*4=28$
/	divisione	$7 / 4=1,75$
//	divisione intera	$7 // 4 = 1$
%	modulo	3

L'operazione di modulo rappresenta il resto di una divisione intera.

Per esempio, “nove modulo tre” ha resto zero:

$$9 \% 3=0 \quad \text{infatti} \quad 9/3=3 \text{ con resto zero}$$

Invece

$$7 \% 4=3 \quad \text{infatti} \quad 7 // 4=1$$

Il valore di una variabile può essere anche il risultato di un’espressione, ossia di una serie finita di operazioni aritmetiche:

espressione = $5+6*2$ → è uguale a $5+(6*2)$
stampa (espressione)

17

Il risultato è 17 e non 22 perché le operazioni di moltiplicazione e divisione hanno la precedenza su addizione e sottrazione, nell’ordine in cui sono scritte da sinistra a destra. Vengono rispettate le regole matematiche di precedenza tra gli operatori. Per esempio, risolviamo l’espressione $5+6/2*4$:

Da sinistra verso destra la prima operazione da eseguire è $6/2=3$

L’espressione diventa $5+3*4$

Da sinistra verso destra la seconda operazione da eseguire è $3*4$

L’espressione diventa $5+12=17$

Per sicurezza possiamo utilizzare le parentesi tonde che ci aiutano a mettere in evidenza le sezioni che hanno la precedenza sulle altre: $5+((6/2)*4)$.

Utilizziamo Python e scriviamo il programma `Area_triangolo.py` che calcola l’area di un triangolo con base 4 e altezza 7. Intuiamo subito quali siano le variabili da usare ed i valori da assegnare:

```
#calcola l'area di un triangolo

base=4
altezza=7

#notare che nell'espressione sono utilizzate le variabili

#base e altezza
area_triangolo=int(base*altezza/2)

#il comando di stampa per visualizzare risultato
print(area_triangolo)
```

Le parti in verde sono i commenti i quali hanno funzione esplicativa e sono ignorati dal calcolatore ai fini della programmazione. Servono per facilitare la comprensione del programma. Tutti i linguaggi di programmazione prevedono la possibilità di inserire commenti, ciascuno con la propria modalità, secondo Python queste note devono essere precedute dal segno di cancelletto “#”, senza questo simbolo viene segnalato un errore di sintassi e il programma non viene eseguito.

In python il comando **print** rappresenta l’istruzione di stampa (che normalmente avviene sul display).

12

Le variabili ricoprono un ruolo decisivo poiché il loro valore può variare in modo dinamico, per chiarire meglio il concetto esaminiamo l’esempio che segue:

a) `area_triangolo=4*7/ 2`

b) `area_triangolo = base*altezza/ 2`

Nel caso a) il valore della variabile area triangolo resta immutato a meno che non si modificano direttamente i valori $4*7/2$, nel caso b) ogni volta che cambia il valore della variabile base e/o della variabile altezza cambia anche il valore della variabile area_triangolo. Pertanto nel caso a) area_triangolo ha un valore costante durante l’esecuzione del programma, mentre nel caso b) il suo valore è dinamico e dipende da altre variabili.

Le variabili possono non avere valore numerico ed essere costituite anche da una sequenza di caratteri, in questo caso si dice che sono di tipo stringa.

Un carattere può essere una lettera, un numero o un segno di punteggiatura.

Una stringa non è semplicemente una collezione, ma una sequenza di caratteri, per cui gli elementi che la compongono seguono un ordine da sinistra a destra.

Solitamente le stringhe sono riconoscibili perché vengono inserite tra le virgolette, quando è possibile, con l’istruzione **str()** è possibile convertire un valore nel tipo stringa. Di seguito sono descritti alcuni esempi per mostrare le caratteristiche delle variabili di tipo stringa:

- `stringa1 = "Ciao!"`

I caratteri sono in ordine, il primo è la “c” e l’ultimo il punto esclamativo.

`print(stringa1)`

- `stringa2 = "!oaiC"`

La stringa non ha senso, ma anche in questo caso i caratteri sono in ordine.

`print(stringa2)`

- `stringa3 = ""`

È una stringa vuota.

```
print(stringa3)
```

```
- stringa4 = "La somma di 3+3..."
```

La somma tra virgolette non viene considerata un'operazione matematica.

```
print(stringa4)
```

```
- numero = 3+3
```

Questa non è una stringa, viene eseguita l'addizione.

```
print(numero)
```

```
- num=4
```

```
nuovo=num+num
```

```
print(nuovo)
```

```
cambia_in_stringa=str(nuovo)
```

```
r=s+s
```

```
print(r)
```

In questo caso n è un numero, mentre s è una stringa perché n è stato convertito in stringa.

Visualizziamo i comandi di stampa:

```
Ciao!  
!oai ← questa è la stringa vuota  
La somma di 3+3...  
6  
8  
88
```

In Python le stringhe possono essere inserite sia tra doppi apici sia tra singoli apici, ciò risulta vantaggioso nel caso in cui la stringa contenga uno dei due segni:

```
parola_straniera = "Shopping" è un termine inglese entrato nel vocabolario italiano
```

la stringa è delimitata dagli apici singoli e contiene una parola evidenziata tra gli apici doppi.

Le stringhe possono essere concatenate, ossia combinate con altre stringhe oppure con variabili di altro tipo per dar luogo a risultati più complessi. I simboli "+" e "*" sono utilizzati per effettuare la concatenazione di stringhe, la virgola "," si usa con altri tipi di dato, vediamo alcuni esempi:

```

print ('Mi piace "Star Wars"...')

nome = "Mario" #notare lo spazio dopo il nome e prima degli apici
cognome = "Rossi"
print (nome + cognome) #sono state concatenate le variabili
print (nome + "Rossi") #sono state concatenate variabile e stringa
print ("1"+"2") #non sono stati inseriti spazi

print (nome * 3) #è una concatenazione particolare
print ((nome + cognome) * 3) #viene ripetuta la stringa
print ("ciao" * 5)
numero = 5
#per concatenare tipi diversi si usa il simbolo ","
print ("Ho letto", numero, "pagine")
#non si possono concatenare tipi diversi con il simbolo "+"
print ("Ho letto"+numero+ "pagine") #errore!

```

Visualizzazione stampa:

```

Mi piace "Star Wars"...
Mario Rossi
Mario Rossi
12
Mario Mario Mario
Mario RossiMario RossiMario Rossi
ciaociaociaociaociao
Ho letto 5 pagine
ERRORE!

```

Le variabili, infine, possono assumere il valore vero o falso, e in questo caso sono di **tipo booleano**, dal nome del matematico George Boole che nel 1847 considerato uno dei fondatori della logica matematica. Definì un'algebra (l'algebra booleana) basata su due variabili a due valori, vero o falso. Vero o falso sono detti valori di verità, in quanto affermano, oppure negano, le cose, o i fatti, che ci circondano; se un valore è vero non può essere anche falso, uno esclude l'altro.

Consideriamo, per esempio, i seguenti enunciati che esprimono un valore di verità:

Premessa	Conclusione
$2*3+2=8$	vero
I Minions sono viola	falso, infatti i Minions sono gial
$10>5$ $5>8$	vero falso

Gli esempi mostrano che gli enunciati possono essere costituiti da espressioni matematiche o proposizioni, la cui soluzione è ricavata attraverso il ragionamento logico: ci sono delle premesse per le quali si giunge ad una conclusione.

Con gli enunciati è possibile sia eseguire operazioni di confronto, mediante i relativi operatori, sia comporre espressioni booleane, o logiche, attraverso gli operatori booleani. In entrambi i casi il risultato è sempre un valore di verità.

Gli operatori di confronto sono:

Simboli	Operatori	Espressioni vere
==	uguaglianza	$2 == 2$
!=	disuguaglianza	$2 != 5$
<	minore	$2 < 5$
<=	minore o uguale	$x <= 5$
>	maggiore	$7 > 5$
>=	maggiore o uguale	$x >= 5$

In questo modo le variabili vengono confrontate e il risultato sarà dato dalla conferma (vero) o dalla negazione (falso) del confronto:

2==2 vero
2 >5 falso
x >= 5 vero o falso dipende dal valore di x
pandistelle != macine vero, infatti le due stringhe sono diverse
pandistelle == macine falso, infatti non sono uguali

Occorre fare attenzione a non confondere l'operatore di uguaglianza (==) con l'operazione di assegnamento (=), infatti

2==2 è un'operazione di confronto
numero = 2 alla variabile numero assegno il valore 2

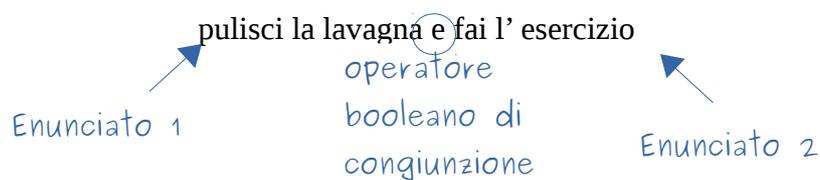
Gli operatori booleani (o logici) sono:

e	congiunzione	x1 e x2
o	disgiunzione	x1 o x2
non	negazione	non x1

x1 e x2 rappresentano un qualsiasi enunciato:

- con la *congiunzione* "e" il risultato è vero solo se *entrambi* gli enunciati sono veri;
- con la *disgiunzione* "o" il risultato è vero se *almeno uno* degli enunciati è vero;
- con la *negazione* "non" il risultato è dato dalla negazione dell'enunciato.

Supponiamo che la professoressa chieda a Dario: "pulisci la lavagna e fai l'esercizio". L'espressione "pulisci la lavagna e fai l'esercizio" rappresenta un'espressione booleana:



In questo caso l'insegnante vuole che vengano eseguite entrambe le richieste, per cui l'espressione è vera solo se risultano vere le due condizioni, è falsa in tutti gli altri casi. Per comprendere meglio la situazione, utilizziamo la tabella di verità (o

tabella logica) che mostra le soluzioni dell'espressione booleana partendo dai possibili valori degli enunciati.

I valori di verità sono due (vero o falso) e ciascun valore di verità esclude l'altro, quindi, avendo due enunciati, le combinazioni attendibili sono quattro. Per semplificare l'elaborato, gli enunciati vengono sostituiti da variabili.

Enunciati		Espressione booleana	
numero	lavagna_pulita	esercizio_svolto	lavagna_pulita e esercizio_svolto
1	V	V	V
2	V	F	F
3	F	V	F
4	F	F	F

1- "lavagna_pulita" e "esercizio_svolto" sono entrambe vere:

Dario ha eseguito gli incarichi a lui assegnati. L'espressione booleana è vera.

2- "lavagna_pulita" è vero e "esercizio_svolto" è falso:

Dario ha solo pulito la lavagna. L'espressione è falsa.

3- "lavagna_pulita" è falso e "esercizio_svolto" è vero:

Dario ha solo eseguito l'esercizio. L'espressione è falsa.

4- "lavagna_pulita" è falso e "esercizio_svolto" è falso:

Dario non ha eseguito quanto gli è stato chiesto. L'espressione è falsa.

Con l'operatore di congiunzione il risultato è sempre falso, tranne quando sono veri entrambi gli enunciati.

In Python gli operatori booleani sono **and**, **or** e **not** e i valori di verità sono **True** e **False**. Osserviamo l’algoritmo che rappresenta la terza riga della tabella di verità:

```
lavagna_pulita=False
esercizio_svolto=True

espressione=(lavagna_pulita and esercizio_eseguito)
print(espressione)
```

Visualizziamo il risultato:

False

Se usiamo l’operatore di disgiunzione, la frase diventa:

pulisci la lavagna  o fai l’esercizio

In questo caso per la professoressa è sufficiente che venga eseguita una richiesta, tuttavia possono anche essere realizzate entrambe. Vediamo la tabella di verità:

Enunciati		Espressione booleana	
numero	lavagna_pulita	esercizio_svolto	lavagna_pulita o esercizio_svolto
1	V	V	V
2	V	F	V
3	F	V	V
4	F	F	F

1- “lavagna_pulita” e “esercizio_svolto” sono vere:

Dario ha eseguito entrambi gli incarichi nonostante alla professoressa ne bastasse uno. L’espressione booleana è vera.

2- “lavagna_pulita” è vero e “esercizio_svolto” è falso: Dario ha pulito la lavagna. L’espressione è vera perché per l’insegnante era sufficiente eseguire un solo compito.

3- “lavagna_pulita” è falso e “esercizio_svolto” è vero. L’espressione è vera.

4- “lavagna_pulita” è falso e “esercizio_svolto” è falso: Dario ha fatto finta di non sentire la professoressa e non ha eseguito nemmeno uno degli incarichi richiesti. L’espressione è falsa.

Con l’operatore di disgiunzione il risultato è sempre vero, tranne quando sono falsi entrambi gli enunciati.

Utilizziamo Python e osserviamo l’algoritmo che rappresenta la seconda riga della tabella di verità:

```
lavagna_pulita=True
esercizio_svolto=False

espressione=(lavagna_pulita or esercizio_esequito)
print(espressione)
```

Risultato:

```
True
```

Con l’operazione di negazione si contraddice quanto affermato: la frase “Ho fame” diventa semplicemente “Non ho fame”.

In questo caso abbiamo un solo enunciato per cui le combinazioni attendibili sono due, vediamo la tabella di verità:

Numero	Ho fame	Non ho fame
1	V	F
2	F	V

1- “Ho fame” è vero: sento appetito, quindi l’espressione “non ho fame” è falsa.

2- “Ho fame” è falso: non sento appetito, “non ho fame” è vera.

Con l’operazione di negazione viene invertito il valore di verità.

Vediamo l’algoritmo della tabella:

```
ho_fame=True

espressione=not(ho_fame)
print(espressione)
```

Risultato:

```
False
```

Un'espressione è definita espressione booleana solo se contiene operatori logici:

`x1 and x2` è un'espressione booleana,

`x1 <= x2` non è un'espressione booleana, è un'operazione di confronto

`((x1 <= x2) or (x3 + 2))` è un'espressione booleana.

3.2.1 Liste

Una lista può essere considerata come una variabile che può contenere un numero di elementi variabile ma fissato a priori, viene creata da un'istruzione in cui i dati sono racchiusi tra le parentesi quadre:

```
lista=[elemento1, elemento2,elemento3,...]
```

```
prova_lista=["Pippo", 7, [5]]
```

Valore tipo stringa tipo intero tipo lista, è tra parentesi quadre

Si noti che i valori della lista possono essere di tipo diverso.

La lista dell'esempio, `prova_lista`, è composta da tre elementi, ciascuno dei quali occupa una posizione individuata da un indice che inizia da zero:

Lista	"Pippo"	7	[5]
Indice	0	1	2

La stringa "Pippo" è in posizione 0, il numero 7 è in posizione 1, il dato tipo lista [5] è in posizione due. Se vogliamo indicare un particolare dato della lista, possiamo selezionarlo nel modo seguente:

```
selezione=lista[indice]
```

Se vogliamo selezionare il numero 7, che corrisponde al secondo elemento di `prova_lista`:

```
selezione=prova_lista[1]  
print(selezione)
```

Per conoscere la lunghezza della lista si usa l'istruzione **len()**:

```
lunghezza_prova_lista=len(prova_lista)  
print(lunghezza_prova_lista)
```

Il comando di stampa può anche essere eseguito direttamente sul dato selezionato:

```
print("L'elemento in posizione 1 è",prova_lista[1])  
print("La lista contiene",len(prova_lista),"elementi")
```

La stampa degli ultimi tre comandi è:

```
L'elemento in posizione 1 è 7
La lista contiene 3 elementi
```

3.3 Input e output

Sono chiamate input le informazioni iniziali riguardanti i dati d'ingresso del programma. Riprendiamo l'esempio della ricetta del tiramisù già preso in esame all'inizio del capitolo, prima di iniziare la preparazione dobbiamo assicurarci di avere tutti gli ingredienti che rappresentano i dati di input del procedimento.

Output invece è il risultato finale del procedimento che, a seconda dei casi, può essere una visualizzazione a video o una stampa su carta vengono.

Riprendendo l'esempio della ricetta l'assemblaggio e la lavorazione degli ingredienti dà origine al tiramisù che rappresenta il prodotto finale della procedura ed è pertanto l'output.

Tutti i programmi per produrre un output hanno bisogno di dati di input i quali, a volte, possono essere richiesti direttamente all'utente che li digita sulla tastiera.

Per esempio, consideriamo il calcolo dinamico dell'area di un triangolo, in cui il programma richiede di inserire la base e l'altezza da tastiera per poi procedere con il calcolo dell'area, così facendo al variare dei dati di input varia il valore dell'output. Per introdurre i dati da tastiera Python mette a disposizione il comando **input()**. È necessario ricordare che i dati inseriti da tastiera sono di tipo stringa, pertanto occorre convertirli in numero utilizzando le istruzioni **int()** oppure **float()**.

```
#calcola l'area di un triangolo

stringa1=input("Inserisci la base: ")
stringa2=input("Inserisci l'altezza: ")
#è necessario convertire il tipo stringa in numero
base=float(stringa1)
altezza=float(stringa2)

area_triangolo=float(base*altezza/2)

#per mostrare il valore dell'area si usa il comando di
#stampa
print("L'area del triangolo è ",area_triangolo)
```

Quando viene avviato il programma, il primo output è il seguente:

```
Inserisci la base:
```

L'utente inserisce il valore e preme invio:

```
Inserisci la base: 6
```

Stessi passaggi per l'inserimento dell'altezza, a seguire l'utente preme nuovamente invio ed ottiene il risultato:

```
Inserisci la base: 6  
Inserisci l'altezza: 7  
L'area del triangolo è 42
```

3.4 Funzioni

Una **funzione** è composta da un insieme di istruzioni, e restituisce un determinato output, e spesso si ripetono in alcune parti

I programmi solitamente sono lunghi e complessi, pertanto è consuetudine dividerli in vari sotto-programmi ciascuno dei quali fornisce una soluzione che contribuisce al raggiungimento del risultato finale. Una funzione rappresenta un programma che prende come argomento dei dati (detti parametri) e produce dei risultati. I programmi usano le funzioni come sotto programmi. Per poter essere utilizzate le funzioni devono essere definite, ossia:

- avere un nome,
- indicare i **parametri**, ossia i dati su cui la funzione esegue le operazioni,
- indicare le istruzioni che portano al risultato.

Quando la funzione è utilizzata all'interno del programma si usa l'espressione **chiamata di funzione**.

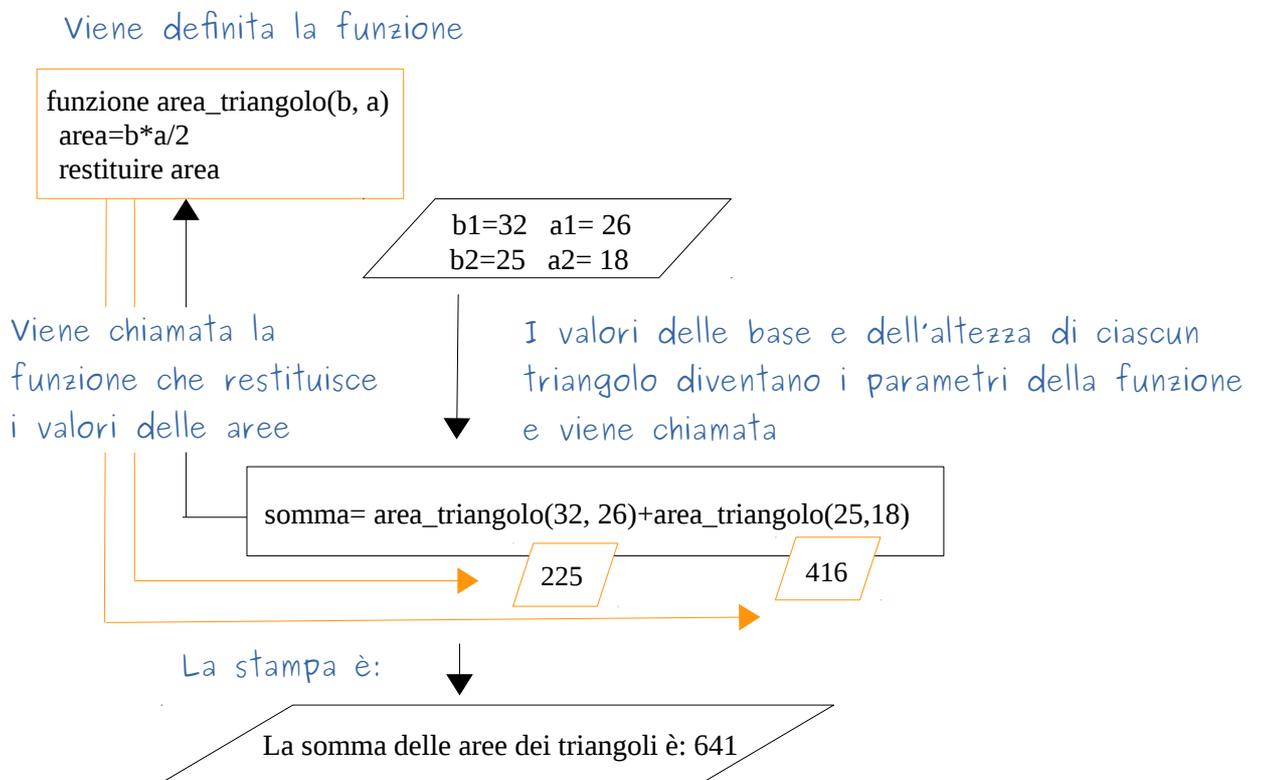
Consideriamo, per esempio, di dover calcolare la somma delle aree di due triangoli i cui dati sono:

triangolo1 base1=32 e altezza1=26

triangolo2 base2=25 e altezza2=18

Per semplificare saranno usate le variabili b1,a1 e b2,a2.

Descriviamo le fasi del programma per il calcolo dell'area del triangolo mediante un diagramma di flusso:



Senza la funzione `area_triangolo` calcoli che differiscono soltanto per i valori numerici utilizzati dovrebbero essere ripetuti più di una volta, come mostrato di seguito:

```
step 1: area1 =base1*altezza1/2
        restituire area triangolo1
step 2: area2 =base2*altezza2/2
        restituire area triangolo2
step 3: somma=area1+area2
        stampa(La somma delle aree dei triangoli è: somma)
```

Cambiano solo le variabili, la formula è la stessa

La suddivisione del programma in funzioni permette di organizzare il lavoro, ne facilita la lettura e favorisce il controllo e la gestione degli errori logici che inevitabilmente si commettono nella stesura di programmi di grandi dimensioni. Per tutti i linguaggi di programmazione sono disponibili delle funzioni predefinite, ovvero già pronte all'uso, che costituiscono quelle che si chiamano funzioni di libreria. Abbiamo già visto alcuni esempi di Python `print()`, `int()`, `float()`, `str()`, `boolean()`, `input()`.

Vediamo la sintassi di Python per definire le funzioni:

```
def nome_funzione(parametro1, parametro2,...):
    istruzione_1
    istruzione_2
    ...
    istruzione_n
    return risultato
```

Ricordare di mettere il simbolo dei due punti

Le istruzioni della funzione hanno lo stesso allineamento

Per creare una funzione è necessario utilizzare la parola chiave `def` seguita dal nome che riteniamo adeguato, e dai parametri necessari. Le istruzioni che compongono il corpo della funzione devono seguire una comune indentazione. Se la funzione deve restituire un risultato, viene chiusa dal comando `return`, altrimenti possiamo ometterlo e utilizzare `print()`.

Calcola_somma_aree_triangoli.py

```
#viene definita la funzione
def area_triangolo(b, a):
    area=b*a/2
    return area

b1=32
a1=26
b2=25
a2=18

#per fare la somma si ha la chiamata di funzione
somma= area_triangolo(b1, a1)+area_triangolo(b2, a2)
print("La somma delle aree dei triangoli è: ",somma)
```

La stampa del risultato:

```
La somma delle aree dei triangoli è: 481
```

3.5 Cicli iterativi. Ciclo while

Nella stesura di un programma si presenta molto spesso la necessità di ripetere le stesse operazioni, i metodi con cui è possibile fare ciò si chiamano cicli.

Cerchiamo di capire il concetto attraverso un esempio: supponiamo di avere una contenitore con 15 palline e di volerne spostare 5 qualsiasi in una cesta vuota.

Quali sono le azioni da compiere? Supponiamo che debbano essere eseguite da un bambino o a un robot:

1- prendi una prima pallina e spostala nella cesta

2- prendi un'altra e sposta anche questa nella cesta

Ripeti queste azioni fino a quando le cinque palline non sono spostate nella cesta.

Quindi un programma completo che chiamiamo *Pallina_nella_cesta* potrebbe essere il seguente:

```
1 prendere la pallina
  mettere nella cesta
2 prendere la pallina
  mettere nella cesta
3 prendere la pallina
  mettere nella cesta
4 prendere la pallina
  mettere nella cesta
5 prendere la pallina
  mettere nella cesta
"Bene! La cesta è piena!"
```

Nell'esempio si nota che le istruzioni vengono eseguite 5 volte, ma se le palline da spostare invece di 5 fossero state 100, quante istruzioni avrebbe avuto il nostro programma?

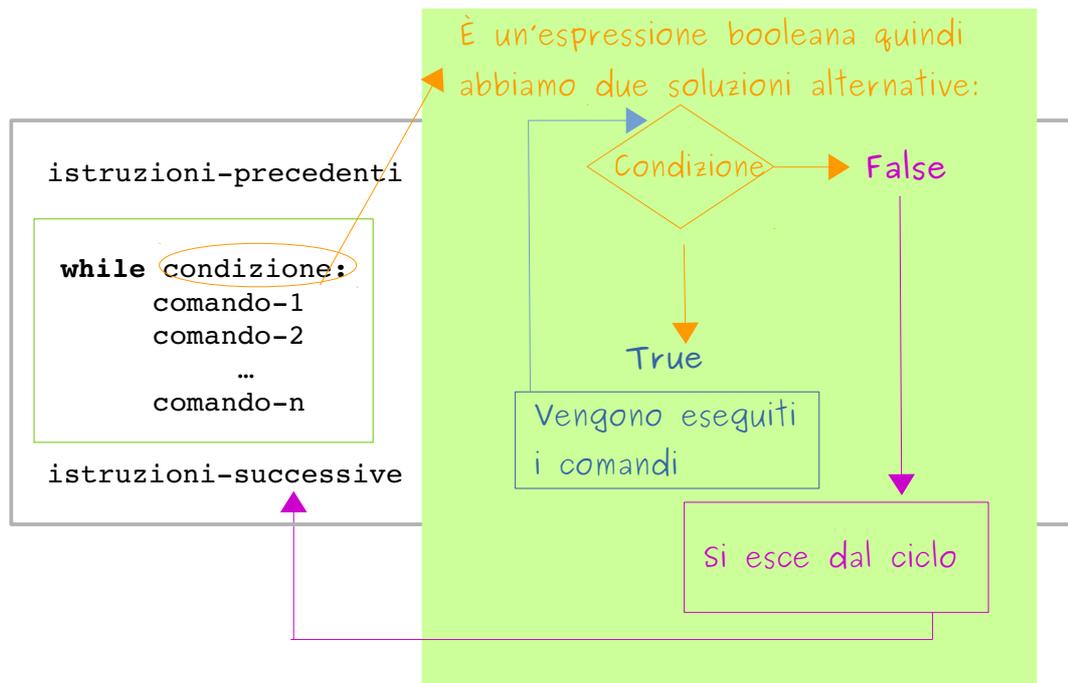
Per facilitarci il compito sono stati introdotti i cicli , con i quali si evita di ripetere la scrittura di istruzioni uguali. Per rendere più leggibile il programma bisogna eliminare le istruzioni ridondanti, e questo si realizza utilizzando i cicli.

Pallina-nella-cesta2

```
1 ripetere per 5 volte
  prendere la pallina
  mettere nella cesta
2 print("Bene! La cesta è piena!")
```

Il programma Pallina-nella-cesta2 è decisamente più semplice, inoltre per cambiare il numero di palline da 5 a 100, basta sostituire 100 a 5 nella prima istruzione “ripeti...”. La ripetizione di istruzioni uguali si chiama iterazione.

In Python ci sono due modalità iterative, il comando **while** e il comando **for** che vedremo più avanti. La figura mostra la sintassi del ciclo while.



Viene valutata la condizione, se è vera il ciclo esegue il blocco di comandi (da ricordare che è possibile avere anche un solo comando) che sarà ripetuto finché la condizione risulta vera, altrimenti (se falsa) si esce dal ciclo e vengono eseguite le istruzioni successive.

Per programmare i cicli è fondamentale individuare il numero di volte che il ciclo deve essere eseguito (5 o 100 nel caso delle palline) e le azioni che devono essere iterate, queste ultime possono essere istruzioni qualsiasi come ad esempio assegnamenti, operazioni matematiche, di confronto, comandi **if-else** (nel nostro esempio “prendere la pallina, mettere nella cesta”).

Per visualizzare la struttura del ciclo in un programma, ovvero i comandi che devono essere eseguiti nel ciclo iterativo, devono essere indentati rispetto all’istruzione **while**.

Scriviamo un frammento di programma Python per mettere le cinque palline nella cesta e lo chiamiamo `Pallina-nella-cesta.py`

```
incesta=0
pallina=1

while pallina<=5:#ricordare i due punti ":" dopo la condizione
    #i comandi interni hanno lo stesso allineamento
    incesta=pallina
    pallina=pallina+1#le istruzioni ripetute sono assegnamenti
#i comandi esterni al ciclo seguono un allineamento diverso
print("Bene! La cesta è piena!")
```

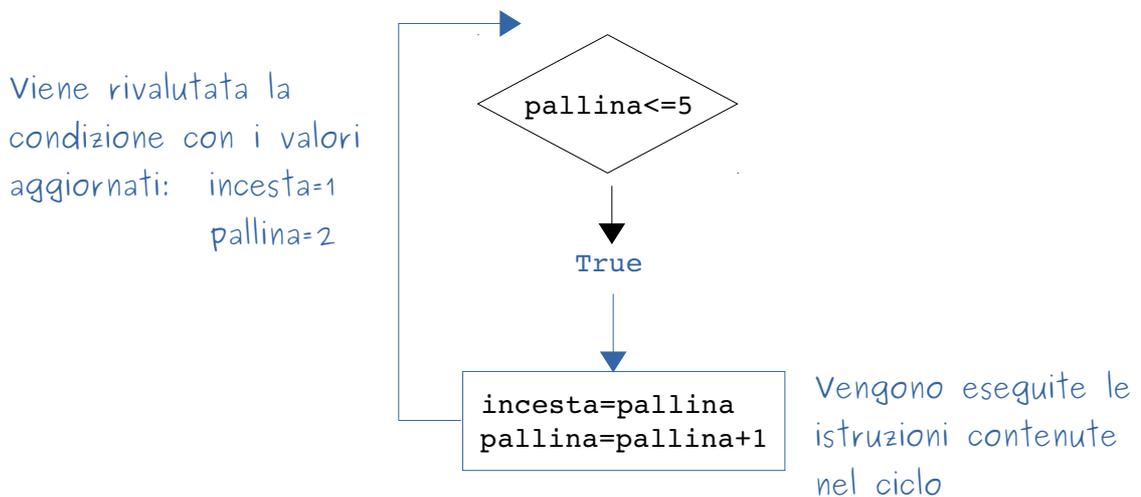
Analisi dei passaggi del ciclo.

Iterazione 1

I dati all'inizio del ciclo sono :

`incesta=0`, all'inizio la cesta è vuota,

`pallina=1`, perché prendo la prima pallina; `pallina` è minore o uguale a 1? Vero, vengono eseguite le istruzioni nel ciclo.



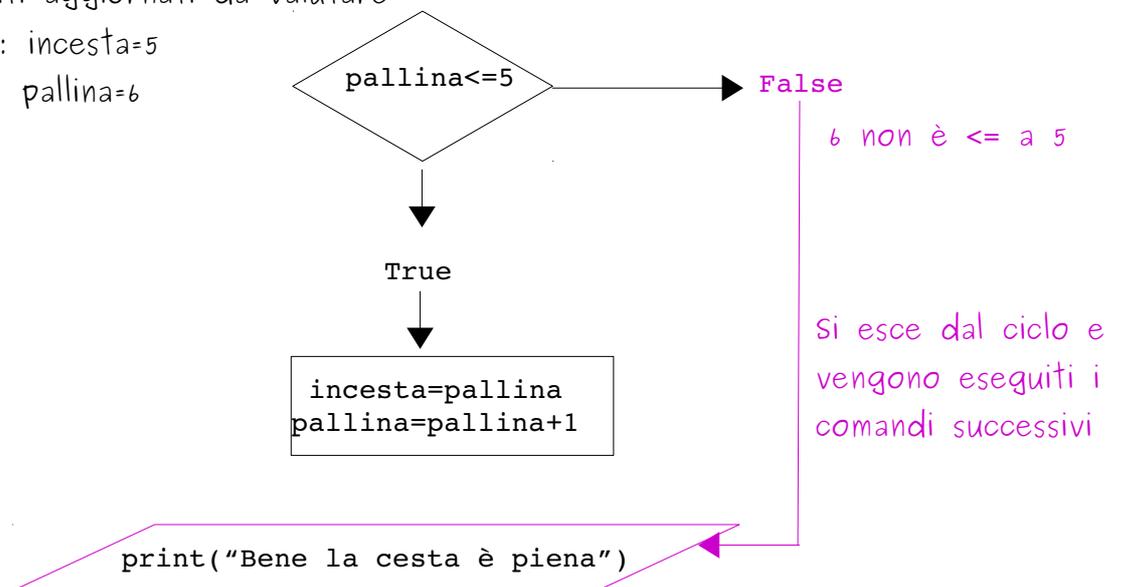
La tabella seguente illustra come variano le variabili ad ogni iterazione del ciclo:

Iterazione	Dati aggiornati	Condizione pallina <= 5	Risultato condizione	Ripeti iterazione
1	incesta=0 pallina=1	1 <= 5	True	si
2	incesta=2 pallina=3	3 <= 5	True	Si
3	incesta=3 pallina=2	4 <= 5	True	Si
4	incesta=4 pallina=5	5 <= 5	True	Si
5	incesta=5 pallina=6	6 <= 5	False	No

Come si nota nella tabella, al passaggio 5 la condizione è falsa, pertanto le istruzioni che fanno parte del ciclo vengono ignorate e si prosegue con i comandi successivi utilizzando gli ultimi dati aggiornati `incesta=5` e `pallina=6`.

Iterazione 5

I dati aggiornati da valutare
sono: `incesta=5`
`pallina=6`



Al termine della sua esecuzione il programma `Pallina-nella-cesta.py` produce i seguenti effetti:

- la cesta contiene le cinque palline perché il robot ha eseguito i comandi
- sul display del robot appare la stampa



Particolare attenzione deve essere prestata alla definizione della condizione che deve essere controllata ad ogni iterazione e al corretto aggiornamento delle variabili usate in essa. In caso di errore il ciclo potrebbe non terminare mai. Questo errore è tipico dei programmatori alle prime armi.

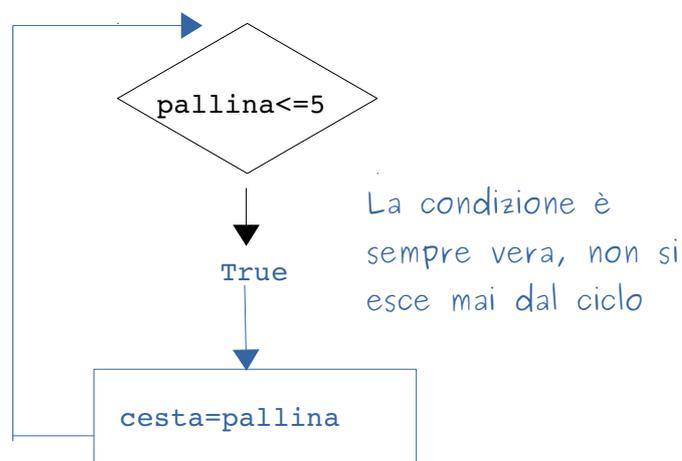
Nel nostro programma `Pallina-nella-cesta.py`

- la variabile `incesta` ci ha permesso di immagazzinare i dati necessari per conseguire l'obiettivo,
- con `pallina` abbiamo contato gli elementi da spostare e controllato il ciclo. La variabile `pallina` determina il risultato della condizione, quindi, a ogni passaggio del ciclo questo valore deve essere incrementato, altrimenti il ciclo sarebbe andato avanti all'infinito. La variabile funziona da contatore.

Nel programma `Pallina-nella-cesta.py` il contatore è la variabile `pallina`, proviamo ad omettere l'istruzione che lo aggiorna, ossia `pallina=pallina+1`.

Il programma elabora i dati iniziali `cesta=0` e `pallina=1`

- esegue l'operazione di confronto,
- il risultato è vero quindi viene eseguito l'unico comando del ciclo,
- il ciclo continua all'infinito perché il contatore non viene aggiornato e la condizione risulta sempre vera!



3.5.1 Ciclo for

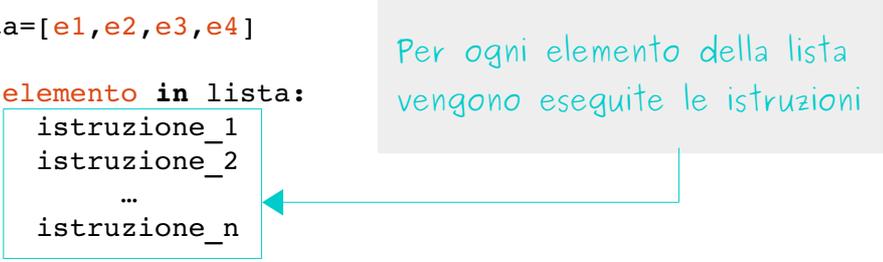
Anche il comando **for**, come il **while**, permette di iterare un gruppo di istruzioni, in questo caso il ciclo si avvale dei valori di una lista. La sintassi del ciclo **for** è mostrata di seguito:

```
istruzioni_precedenti

lista=[e1,e2,e3,e4]

for elemento in lista:
    istruzione_1
    istruzione_2
    ...
    istruzione_n

istruzioni_successive
```



Il significato del costrutto è “per ogni elemento nella lista esegui le seguenti istruzioni”. Nel ciclo le istruzioni devono avere una uguale indentazione, rientrata rispetto all’istruzione **for**. Quelle che non fanno parte del ciclo, seguono un diverso allineamento.

Supponiamo di aver invitato a casa degli amici e il nostro robot li saluta con un messaggio sul display, vediamo il programma **Saluta.py** nella Figura...

Notare che la lista si chiama **amici** e ciascun elemento che la compone è rappresentato dal termine **nome**.

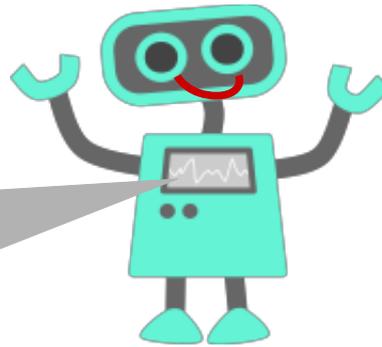
```
amici=["Alberto","Sara","Mattia","Emma","Margherita"]

for nome in amici:
    saluto="Ciao "+nome+"! Come stai?"
    print(saluto)

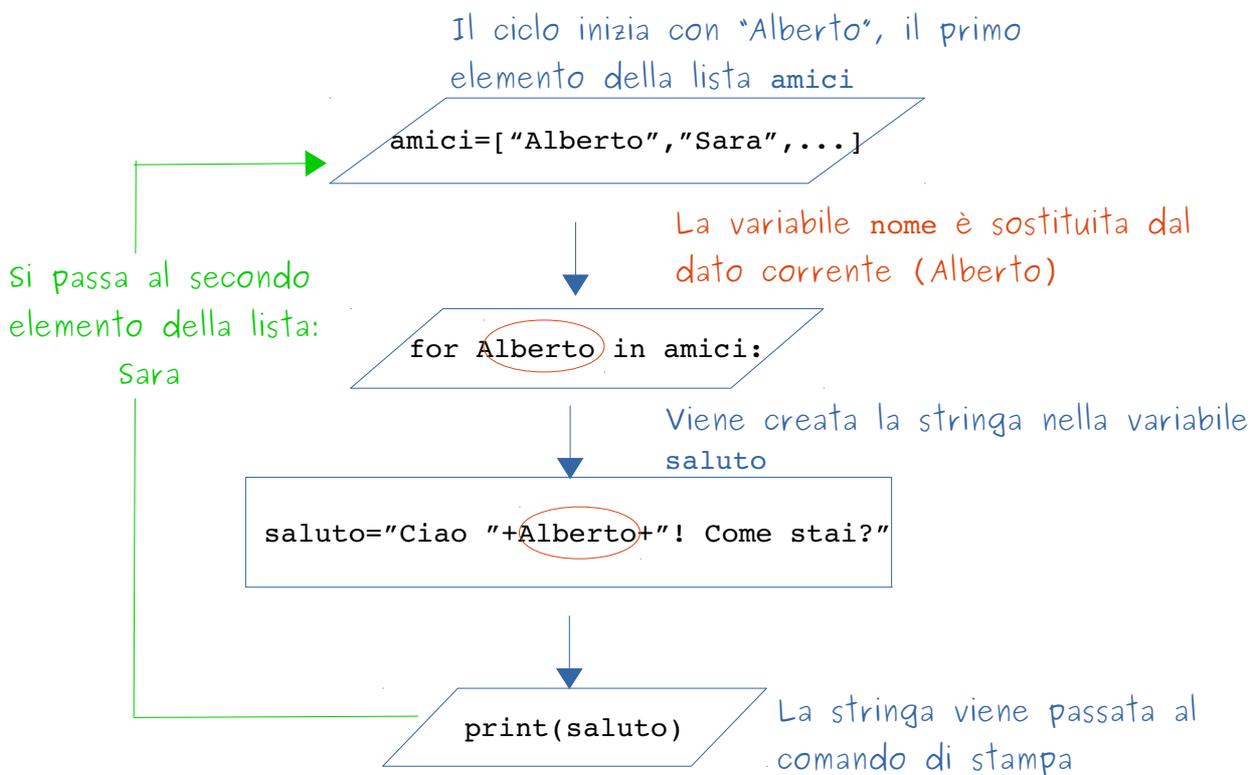
print("Ho salutato tutti?")
```

Sul display del robot appare:

```
Ciao Alberto! Come stai?  
Ciao Sara! Come stai?  
Ciao Mattia! Come stai?  
Ciao Emma! Come stai?  
Ciao Margherita! Come stai?  
Ho salutato tutti?
```



Nello schema seguente viene mostrato il funzionamento del ciclo.



Questi passaggi sono eseguiti per ciascun dato della lista, arrivati all'ultimo, nel nostro caso Margherita, si esce dal ciclo e si passa al comando successivo: `print("Ho salutato tutti?")`.

Mettendo a confronto il ciclo while con il ciclo for, si può notare che il ciclo for esegue un numero di iterazioni predeterminate, mentre el ciclo while le iterazioni si ripetono finché la condizione è vera.

3.6 Istruzione condizionale if-else

Come nella quotidianità, durante la composizione di un algoritmo, ci troviamo a dovere compiere delle scelte in base a particolari situazioni. Per esempio, a Francesca piacerebbe andare a fare un giro in bicicletta, però deve controllare il tempo, infatti:

-se piove, è meglio rimanere in casa,

-altrimenti (cioè se non piove) può uscire in bicicletta.

Francesca quindi deve scegliere in base al tempo: guarda dalla finestra, vede che sta piovendo e decide di restare a casa. Osserviamo l'algoritmo Tempo_e_scelta:

```
stampa(Francesca deve decidere cosa fare...)
```

```
se piove
```

È la condizione in base alla quale viene fatta la scelta

```
    stampa(Rimane in casa.)
```

Nel caso in cui piovesse

```
altrimenti
```

```
    stampa(Esce.)
```

Alternativa nel caso in cui non piovesse

```
stampa(...è in compagnia del suo cagnolino.)
```

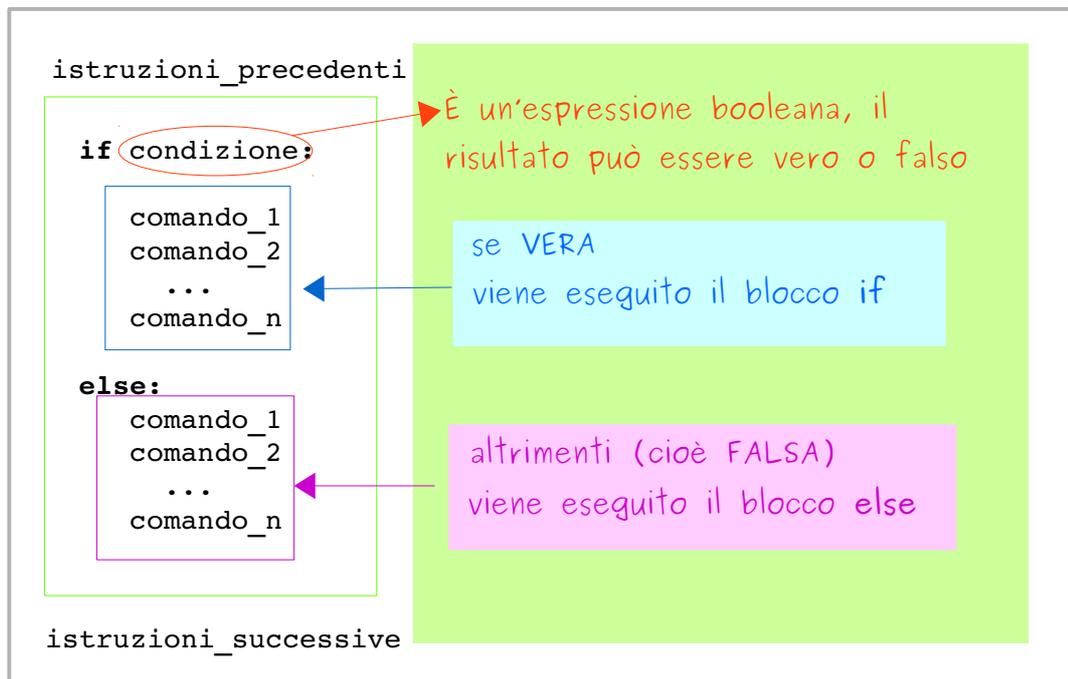
Se la condizione “piove” è vera, l'algoritmo stampa:

```
Francesca deve decidere cosa fare...  
Rimane in casa.  
...è in compagnia del suo cagnolino.
```

Se invece la condizione “piove” è falsa, si passa direttamente all'istruzione prevista per questo caso, e la stampa è la seguente:

```
Francesca deve decidere cosa fare...  
Esce.  
...è in compagnia del suo cagnolino
```

In Python abbiamo il comando **if-else**, il cui significato è analogo all'espressione "se...altrimenti", e si presenta nel seguente modo:



La condizione è rappresentata da un'espressione booleana:

- se è vera, vengono eseguite le istruzioni del blocco `if`,
- altrimenti, se è falsa, si passa direttamente ai comandi relativi a `else`.

Adesso vediamo `Tempo-e-scelta.py`, l'algoritmo in python.

`piove=False` è il dato di partenza, il valore della variabile `piove` sarà elaborato nella condizione.

```
#notare l'indentazione differente per distinguere i comandi
#questa istruzione precedente viene comunque eseguita
print("Francesca deve decidere cosa fare...")
piove=False      #non piove

if piove==True: #ricordare ":" dopo la condizione
    print("Rimane in casa.")#allineamento uguale a
else: #ricordare ":"
    print("Esce.") #allineamento uguale a blocco if

#questa istruzione successiva viene comunque eseguita
print("...è in compagnia del suo cagnolino.")
```

if è sempre seguito dalla condizione e dal simbolo dei due punti, **else** solo dai due punti.

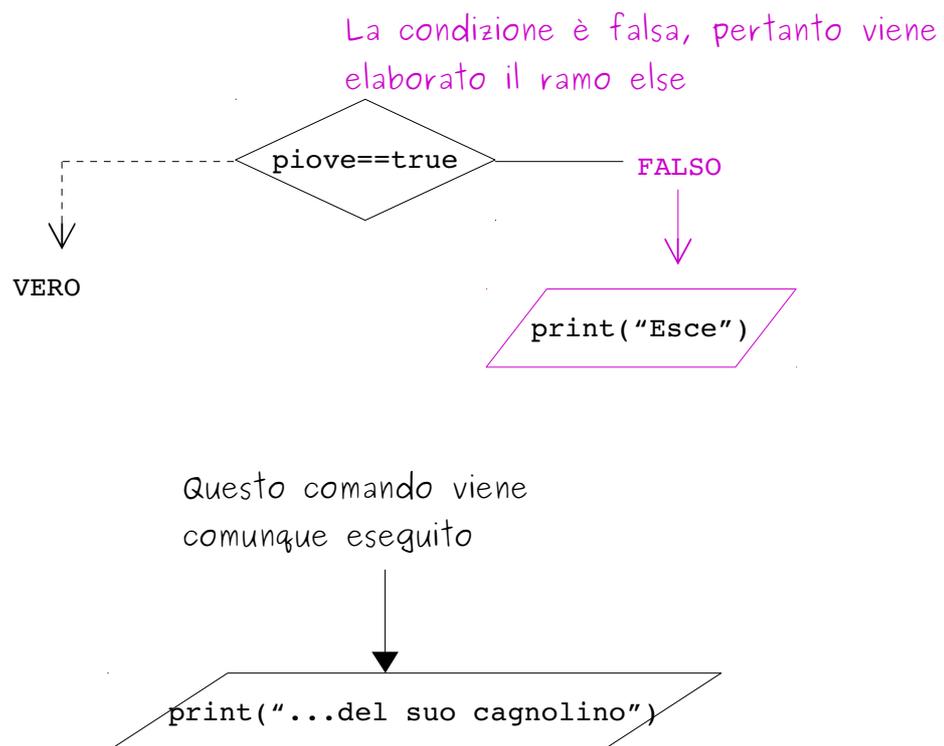
Le istruzioni del blocco if e del blocco else, si distinguono perché seguono un comune allineamento dopo la rispettiva parola chiave, infatti per uscire dal comando condizionale è sufficiente riportare l'indentazione a inizio riga come istruzioni successive.

La stampa del programma:

```
Francesca deve decidere cosa fare...
Esce.
... è in compagnia del suo cagnolino.
```

Analizziamo il programma col diagramma di flusso.

Il dato di partenza è `piove=false`, subito dopo viene valutata la condizione in base alla quale viene deciso il passaggio successivo.



Il comando condizionale si può presentare anche in modi differenti:

1. ramo if senza quello else,
2. più comandi condizionali annidati uno dentro l'altro,
3. scelte multiple con il comando elif.

1. Ramo if senza else

Abbiamo soltanto le istruzioni che dipendono da **if**.

Manca il ramo else che rappresenta un'alternativa nel caso in cui la condizione sia falsa.

```
#notare l'indentazione differente per distinguere i comandi
print("Francesca deve decidere cosa fare...")

piove=False          #non è una giornata piovosa

if piove==True:
    print("Rimane in casa.")
#è stato rimosso il blocco else

print("...è in compagnia del suo cagnolino.")
```

Stampa:

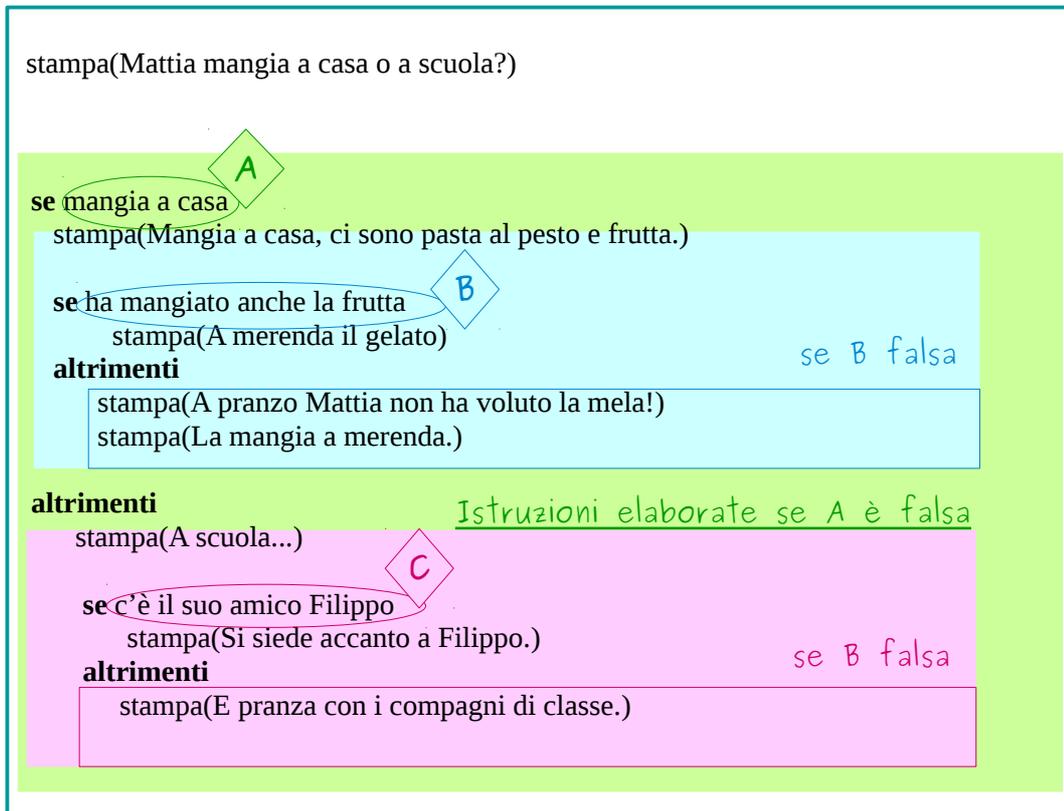
```
Francesca deve decidere cosa fare...

...è in compagnia del suo cagnolino.
```

2. if-else annidati

Ci sono situazioni in cui le scelte sono subordinate ad altre, e sono annidate una dentro l'altra. In questo caso i costrutti principali **if-else** a loro volta contengono, al loro interno, altri **if-else**.

Mattia deve decidere dove pranzare, il seguente algoritmo prevede delle scelte. Mattia sceglie di mangiare a casa:



Come si vede dall'esempio sopra, abbiamo tre gruppi di **se** ed **altrimenti**, i più esterni contengono gli altri due.

Le scelte, in ogni caso, sono compiute in base alla valutazione delle rispettive condizioni:

- A, mangia a casa,
- B, ha mangiato anche la frutta,
- C, c'è il suo amico Filippo.

La condizione A viene analizzata per prima, è vera, pertanto vengono selezionati i comandi che seguono **se**, però fino alla condizione B, infatti a questo punto viene compiuta un'altra scelta.

Anche B è vera quindi è eseguita l'istruzione vincolata a **se**. L'algoritmo stampa:

```
Mattia mangia a casa o a scuola?  
A casa, pasta al pesto e frutta.  
A pranzo Mattia non ha voluto la mela!  
La mangia a merenda.
```

Scriviamo l'algoritmo in modo formale con Python, `Casa-o-scuola.py`.

```
#notare l'indentazione differente per distinguere i comandi  
print("Mattia mangia a casa o a scuola?")  
  
a_casa=True    #Mattia decide di mangiare a casa  
  
if a_casa==True: #la condizione è valutata ed è vera  
    print("A casa, pasta al pesto e frutta.")  
    frutta=False  
    if frutta==True:#ricordare ":"  
        print("A merenda il gelato")  
    else:  
        print("Però a pranzo Mattia non ha voluto la mela!")  
        print("La mangia a merenda.")  
  
else:#ricordare ":"  
    print("A scuola...")  
    filippo=False #Filippo è assente  
    if filippo==True:  
        print("E si siede accanto a Filippo.")  
    else:  
        print("E pranza con i suoi compagni di classe.")
```

La corretta indentazione mette in evidenza i diversi gruppi **if-else** annidati.

È opportuno fare attenzione agli **if-else** annidati in quanto potrebbero diventare troppo complessi e quindi poco leggibili. In presenza di molti **if-else** annidati è possibile che si verifichino errori nel legame tra un **if** e il suo rispettivo **else**. Pertanto è opportuno evitare di avere molti annidamenti multipli.

3- Comando elif

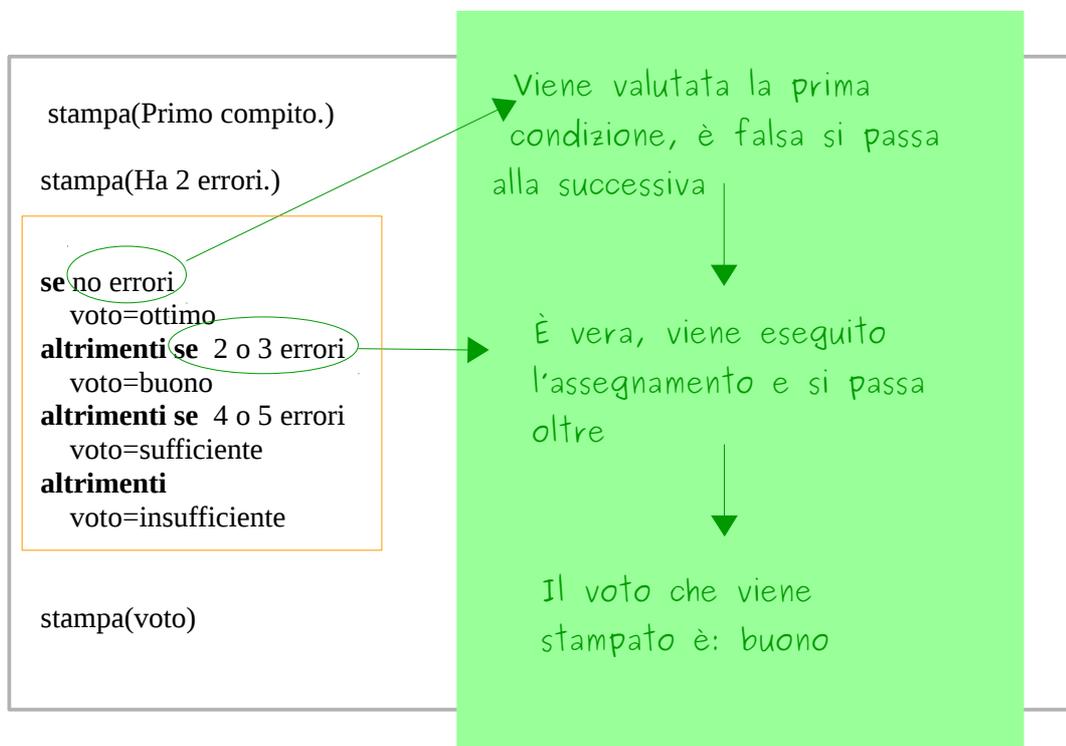
È l'abbreviazione di **else-if** ("altrimenti...se") e può essere utilizzata quando ci sono scelte multiple.

Le condizioni sono controllate in sequenza, se la prima è falsa si passa alla successiva, e così di seguito. Trovata quella vera vengono eseguite le relative istruzioni e si prosegue con i comandi esterni al costrutto **elif**. Vediamo un esempio.

La professoressa di matematica ha corretto i compiti e deve assegnare i voti:

- se non ci sono errori il voto è ottimo,
- altrimenti se ci sono 2-3 errori il voto è buono,
- altrimenti se ci sono 4-5 errori il voto è sufficiente,
- altrimenti, con più di 5 errori, è insufficiente.

Supponiamo che il primo compito corretto abbia 2 errori, vediamo l'algoritmo Voto_compito:



La quantità di **elif** che possono essere inseriti non è prestabilita, invece l'else finale deve essere unico.

Il programma `Voto_compito.py` è il seguente:

```
#notare l'indentazione differente per distinguere i comandi
print("Primo compito")
errore=2
print("Errori:",errore)#stampa la stringa seguita dal
                        #numero 2 che

if errore==0:
    voto="ottimo"
elif ((errore==2) or (errore==3)):#ricordare ":"
    voto="buono"
elif ((errore==4) or (errore==5)):
    voto="sufficiente"
else: #cioè errore>5
    voto="insufficiente"

print("Voto: "+voto)#stampa stringa + variabile
```

La stampa del risultato è:

```
Primo compito
Errori: 2
Voto: insufficiente
```

3.7 Esempi di applicazione del pensiero computazionale

Programma “Crescente_decrescente_nonOrdinato.py”

Stabilisci se i gruppi seguenti di numeri , composti da quattro numeri ciascuno, sono scritti in ordine crescente oppure in ordine decrescente[24], oppure non sono ordinati.

a) 198, 189, 99, 88

b) 209, 211, 1111, 2111

.....

g) 987, 879, 789, 778

h) 1212, 1220, 2102, 2201

I numeri devono essere confrontati fra loro e sono:

- crescenti quando $a \leq b \leq c \leq d$

- decrescenti quando $a \geq b \geq c \geq d$

Se nessuna delle due condizioni crescente o decrescente è verificata, allora i numeri sono ordinati.

Crescente_decrescente_nonOrdinato.py è il programma di tipo funzione che può essere applicato a gruppi di quattro numeri.

```
#Crescente_decrescente_nonOrdinato

def cresce_decresce_nonordinato(a,b,c,d):
    if a<=b and b<=c and c<=d:

        print("I numeri sono in ordine crescente.")

    else:
        if a>=b and b>=c and c>=d:
            print("I numeri sono in ordine decrescente.")
        else:
            print("I numeri non sono ordinati.")

#la funzione viene applicata ai numeri dell'esercizio

cresce_decresce(198,189,99,88)
cresce_decresce(34,46,25,78)#i numeri non sono in ordine
cresce_decresce(34,46,46,78)#ci sono due numeri uguali
```

Descrizione del funzionamento del programma.

In esso possiamo distinguere due sezioni di controllo alternative:

1. Se la condizione `a<=b and b<=c and c<=d` è vera si prosegue con il ramo `if` e l'istruzione di stampa.
2. Se è falsa si passa al ramo `else` che contiene un ulteriore ramo `if-else`:

<code>a>=b and b>=c and c>=d</code>	
True	False
<code>print(...decescente)</code>	<code>else</code> <code>print(...non ordinati)</code>

Consideriamo per esempio il gruppo di numeri 34 46 25 78.

Si esegue il primo controllo:

- `34<=46 and 46<=25 and 25<=78` è falso pertanto si passa al ramo `else`;

- `34>=46 and 46>=25 and 25>=78` è falso si passa al ramo `else` con

l'esecuzione della stampa `I numeri non sono ordinati.`

La figura mostra la stampa dei tre gruppi di numeri a cui è stato applicato il programma `Crescente_decescente_nonOrdinato.py`

```
I numeri sono in ordine decrescente.  
I numeri non sono ordinati.  
I numeri sono in ordine crescente.
```

Esercizio.

Sul quaderno prova ad eseguire i passaggi degli altri due gruppi di numeri (198, 189, 99, 88 e 34 46 46 78).

Programma “Proprietà_associativa.py” [25]

In base alla proprietà associativa dell’addizione in un’addizione di più addendi il risultato non cambia se a due o più addendi si sostituisce la loro somma:

$$a+(b+c) = (a+b)+c$$

Considerata la definizione, per verificare la proprietà associativa deve essere eseguito un confronto fra i due tipi di addizione:

- una nella forma $a+(b+c)$
- l’altra nella forma $(a+b)+c$

La verifica deve essere eseguita su gruppi di addendi uguali.

Il programma `Proprietà_associativa.py` crea tre funzioni, due per definire ciascuna delle forme di addizione ($a+(b+c)$ e $(a+b)+c$) e l’altra per l’esecuzione del confronto.

```
#Proprietà_associativa

def associativa1(a,b,c):
    somma1=b+c
    somma2=a+somma1
    print("La somma di",a,"+(",b,"+",c,") è",somma2, ".")

    return somma2#la funzione restituisce un risultato

def associativa2(a,b,c):
    add1=a+b
    add2=add1+c
    print("La somma di (",a,"+",b,") +",c,"è",add2, ".")
    return add2#la funzione restituisce un risultato

def confronta (s1,s2) :
    if somma2==add2:
        print("Il risultato dell’addizione non cambia.")
    else:
        print("Sono stati inseriti addendi diversi!")

#chiamate di funzione
#il risultato di associativa1 viene assegnato a s1
s1=associativa1(6,2,3)
#il risultato di associativa2 viene assegnato a s2
s2=associativa2(6,2,3)
#la funzione confronta viene applicata ai risultati
confronta(s1,s2)
```

Descrizione del funzionamento del programma.

Per osservare il funzionamento del programma sono utilizzati gli addendi 6, 2, 3.

La funzione `associativa1` elabora e visualizza il calcolo di $a+(b+c)$, e restituisce il risultato nella variabile `somma`:

- alla variabile `somma1` viene assegnato il valore di $b+c$
- alla variabile `somma2` viene assegnato il valore di $a+somma1$
- la funzione restituisce il valore di `somma2`

<code>b+c</code>	<code>somma1</code>	<code>a+somma1</code>	<code>somma2</code>	<code>return somma2</code>
2+3	5	5+6	11	11

La funzione `associativa2` elabora e visualizza il calcolo di $(a+b)+c$:

- alla variabile `add1` viene assegnato il valore di $a+b$
- alla variabile `add2` viene assegnato il valore di $add1+c$
- la funzione restituisce il valore di `add2`

<code>a+b</code>	<code>add1</code>	<code>add1+c</code>	<code>add2</code>	<code>return add2</code>
6+2	8	8+3	11	11

L'ultima funzione (`confronta`) fa il confronto tra i risultati delle due precedenti per mostrare che con la proprietà associativa il risultato non cambia:

in questo esempio la condizione `somma==add2` è vera, pertanto viene eseguita l'istruzione del comando `if`.

<code>if somma1==add2</code>	Risultato condizione
<code>11==11</code>	
<code>print("Il risultato...non cambia")</code>	<code>True</code>

La stampa del programma `Proprietà_associativa.py` è la seguente:

```
La somma di 6+(2+3) è 11.  
La somma di (6+2)+3 è 11.  
Il risultato dell'addizione non cambia.
```

Esercizio.

Sul quaderno prova ad eseguire i passaggi delle tre funzioni utilizzando parametri diversi: `associativa1(5,7,9)` e `associativa2(27,3,11)`.

I numeri di Fibonacci [26]

I numeri di Fibonacci sono associati a un modello matematico ideale legato alla crescita di una popolazione di conigli:

- una coppia di conigli produce una coppia di coniglietti ogni mese
- per il primo mese i coniglietti non si riproducono
- non muore nessun coniglio.

Ogni numero di Fibonacci è composto dalla somma dei due precedenti come mostra la Tabella 5.

- $F_1=1$
- $F_2=1$
- $F_i = F_{(i-1)}+F_{(i-2)}$ con $i>2$

Tabella 1

Numero di Fibonacci	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10
Valore	1	1	2	3	5	8	13	21	34	55

Programma “Primi_10_numeri_di_Fibonacci.py”

Un programma che crea e stampa i primi dieci numeri di Fibonacci può eseguire la somma e la stampa per ciascuno:

```
f1=1          print(f1)
f2=1          print(f2)
f3=f1+f2      print(f3)
...
f10=f9+f8     print(f10)
```

Questo procedimento tuttavia può essere migliorato calcolando i valori all'interno di un ciclo come nel programma Primi_10_Numeri di Fibonacci.py

```
#Primi_10_Numeri di Fibonacci

print("I primi dieci numeri di Fibonacci sono:")
numeri=[1,2,3,4,5,6,7,,8,9,10]
a=1
b=1
print(a)
print(b)
for i in numeri:
    c=a+b
    print(c)
    a=b #scambia a con b
    b=c #scambia b con c
```

Descrizione del funzionamento del programma.

I due primi numeri F1 e F2 sono i dati di partenza con valore 1, pertanto il ciclo viene ripetuto 8 volte.

All'interno del ciclo viene eseguita la somma e lo scambio dei valori per aggiornare i dati da sommare ad ogni passaggio.

Iterazione	c=a+b	print(c)	scambia a con b	scambia b con c
1	1+1	2	1	2
2	1+2	3	2	3
3	2+3	5	3	5
4	3+5	8	5	8
5	5+8	13	8	13
6	8+13	21	13	21
7	13+21	34	21	34
8	21+34	55	34	55

La visualizzazione del programma `Primi_10_Numeri di Fibonacci.py` appare nel modo seguente:

```
I primi dieci numeri di Fibonacci sono:  
1  
1  
2  
3  
5  
8  
13  
21  
34  
55
```

Programma “Numero_di_Fibonacci.py”

Utilizzando i numeri di Fibonacci può essere analizzato un ulteriore esempio, il programma `Numero_di_Fibonacci.py`, in cui viene restituito l'i-esimo numero di Fibonacci dove *i* è il parametro di input.

Le premesse sono le medesime dell'esempio precedente, i dati di partenza sono dati dalle variabili `a=1` e `b=1` che corrispondono a F1 e F2.

```
#Numero_di_Fibonacci

def fibonacci(n):
    if n<=2:
        #i primi due numeri di Fibonacci sono uguali a 1
        return 1

    print("Il numero Di Fibonacci corrispondente a",n,"è:")
    a=1
    b=1
    i=3#il calcolo parte dal numero 3
    while i<=n:#n è il parametro della funzione
        c=a+b
        a=b
        b=c
        i=i+1
    return b
#chiamate di funzione direttamente nel comando print
print(fibonacci(12))
print(fibonacci(36))
print(fibonacci(55))
```

Descrizione del funzionamento del programma.

I primi due numeri hanno valore 1, pertanto viene effettuato un primo controllo con il comando `if`.

Il ciclo quindi parte da un valore uguale a 3 (vedi Tabella 1) e si ferma quando supera il valore del parametro inserito nella chiamata di funzione:

<code>fibonacci(n)</code>	<code>if n<=2</code>	<code>while i<=n</code>	numero di cicli
<code>fibonacci(12)</code>	False	<code>3<=12</code>	13
<code>fibonacci(36)</code>	False	<code>3<=36</code>	37
<code>fibonacci(55)</code>	False	<code>3<=55</code>	56

Il funzionamento del ciclo è identico a quello mostrato nella Tabella 6, con la differenza che in questo caso viene restituito il valore di `b` (`return b`).

La chiamata di funzione avviene direttamente con il comando di stampa `print(fibonacci(12))`.

Il programma `Numero_di_Fibonacci.py` stampa:

```
Il numero Di Fibonacci corrispondente a 12 è:  
144  
Il numero Di Fibonacci corrispondente a 36 è:  
14930352  
Il numero Di Fibonacci corrispondente a 55 è:  
139583862445
```

I risultati della stampa mostrano come i numeri di Fibonacci crescono in modo esponenziale, cioè molto velocemente, quindi il programma `Numero_di_Fibonacci.py` non darà più risultato in tempi utile quando i diventa molto grande.

Esercizio.

Sul quaderno prova ad eseguire il funzionamento dei programmi

`Primi_10_Numeri di Fibonacci.py` e `Numero_di_Fibonacci.py`.

Programma “Moltiplicazione.py”

In questo esempio viene mostrato come si può stampare il risultato di una moltiplicazione di due fattori utilizzando l’operazione di addizione.

La moltiplicazione viene definita in modo seguente: dati due interi positivi n e m , n viene addizionato per m volte, quindi $n*m = n+n+n+n+n$.

Per evitare di scrivere inutilmente l’addizione tante volte, è preferibile utilizzare un ciclo nel quale viene eseguita l’addizione. Devono essere considerati i due fattori (n,m) e un’altra variabile nella quale vengono raccolti i valori dell’addizione:

- il primo fattore, n , deve essere addizionato
- la variabile *prodotto* contiene la somma di n
- il secondo, m , deve contare il numero di addizioni da eseguire per questo compie un conteggio alla rovescia da m a 0 .

```
#Moltiplicazione

def moltiplica(n,m):
    prodotto=0
    while m!=0:#il ciclo conta da m a zero
        prodotto=prodotto+n
        m=m-1#il ciclo conta da m a zero
    print("Il risultato è:",prodotto)

#per inserire i valori da tastiera
n=input("Inserisci un fattore")
m=input("Inserisci un altro fattore")
#converte input da tastiera in intero
n_intero=int(n)
m_intero=int(m)
moltiplica(n_intero,m_intero)#chiamata di funzione
```

Descrizione del funzionamento del programma `Moltiplicazione.py`.

I dati di partenza (n,m) sono inseriti da tastiera tramite la funzione `input()`. È necessario notare che questi valori sono di tipo stringa, pertanto devono essere convertiti in intero per poter eseguire il calcolo. La variabile `prodotto` parte da zero. Come esempio di dati in `input` utilizziamo 3 e 5.

Il ciclo si arresta quando m è uguale a zero.

Fuori dal ciclo viene eseguita la stampa del risultato raccolto in `prodotto`.

passaggio	n	m	prodotto=prodotto+n	m=m-1
1	5	3	0=0+5	3=3-1
2	5	2	5=5+5	2=2-1
3	5	1	10=10+5	1=1-1
4	5	0		

Il programma prima di mostrare il risultato chiede l'inserimento dei dati, l'utente inserisce il primo valore e preme invio:

```
Inserisci un fattore
```

```
3
```

Dopo il secondo input (vedere il paragrafo *Input Output*), 5 in questo caso, il programma `Moltiplicazione.py` stampa:

```
Il risultato è: 15
```

Programma “Tabellina_di_n.py”

In questo esempio è mostrato un programma che stampa la Tavola Pitagorica (tabellina) di n .

La tabellina riporta i prodotti della moltiplicazione di due numeri interi (n,m) da 1 a 10, in cui n rimane costante. Consideriamo, per esempio, la tabellina del 3:

$3*1=3$ $3*2=6$ $3*3=9$ $3*4=12$ $3*9=27$ $3*10=30$

Per arrivare alla soluzione si possono eseguire tutte le moltiplicazioni, ma è preferibile utilizzare un ciclo in cui:

- n , di cui vogliamo stampare la tabellina per cui rimane costante,
- m , il moltiplicatore che deve aggiornare il valore da 1 a 10,
- *prodotto*, contiene il calcolo del prodotto tra n e m

```
#Tabellina_di_n
def tabellina(n):
    m=1
    prodotto=0
    print("La tabellina del",n,"è:")
    while m<=10:#parte da 1 e arriva a 10
        prodotto=n*m
        m=m+1
        print(prodotto)

#per inserire i valori da tastiera
n=input("Inserisci un fattore")
#converte input da tastiera in intero
m=int(n)
tabellina(m) #chiamata di funzione
```

Descrizione del funzionamento del programma.

Il dato di partenza n è inserito da tastiera tramite la funzione `input()`.

Il ciclo parte con il moltiplicatore $m=1$, ad ogni passaggio viene aggiornato per eseguire la moltiplicazione e arrivare a 10.

passaggio	n	m	prodotto=n*m	m=m+1
1	3	1	3=3*1	1=1+1
2	3	2	6=3*2	2=2+1
3	3	3	9=3*3	3=3+1
...
10	3	10	30=3*10	10=10+1

Il programma prima di mostrare il risultato chiede l'inserimento dei dati, l'utente passa il valore e preme invio:

```
Inserisci un fattore
3
```

Successivamente il programma stampa:

```
3
6
9
12
15
18
21
24
27
30
```

Programma “Moltiplicazione_egizia.py”

È un modo di moltiplicare due numeri trovato nel papiro di Ahmes, dal nome dello scriba che lo trascrisse verso il 1650 a.C. da un originale risalente al Regno Medio, e composto fra il 2000 ed il 1800 a.C.. Si tratta del documento egizio a carattere matematico più esteso: è largo circa 30 cm e lungo 5,46 m. Pare che questo documento fosse un testo di carattere didattico, orientato alle applicazioni pratiche. Secondo alcuni si tratta di un libro scolastico, secondo altri del taccuino di un allievo.

Prevede il dimezzamento progressivo di uno dei due fattori (ignorando eventuali resti) e, parallelamente, la duplicazione dell'altro fattore. Probabilmente queste operazioni si eseguivano in maniera semplice sull'abaco, strumento di calcolo dello scriba egiziano.

Consideriamo per esempio $25 \cdot 17 = 425$, il calcolo viene eseguito con un ciclo che prevede a ogni passo:

- divisione per 2 del moltiplicatore;
 - moltiplicazione per 2 del moltiplicando;
- finché il moltiplicando non si azzeri. Il prodotto viene calcolato partendo da 0 e sommando a ogni passo del ciclo il valore parziale del moltiplicatore se il moltiplicando è dispari.

Il calcolo può essere seguito con l'ausilio della Tabella 2:

- nella colonna di sinistra si mostrano i valori successivi del moltiplicando
- nella colonna centrale i valori del moltiplicatore
- nella colonna di destra i valori del prodotto.

```
#Moltiplicazione_egizia
def m_egizia(n,m):
    prodotto=0
    while n!=0:
        if n%2!=0:#se n è dispari viene eseguita la somma
            prodotto=prodotto+m
        n=n//2#viene eseguita la divisione intera per due
        m=m*2#m viene sempre moltiplicato per due
    return prodotto

risultato=m_egizia(25,17)#chiamata di funzione
print("Il risultato è:",risultato)
```

La Tabella 2 mostra l'esecuzione del programma.

Tabella 2

n//2	m*2	prodotto=prodotto+m
25	17	17=0+17
12	34	17
6	68	17
3	136	153=17+136
1	272	425=153+272
0		

Stampa del risultato:

Il risultato è: 425

Programma “MCD_Euclide.py”

Per determinare il massimo comun divisore tra due interi qualsiasi, $MCD(a,b)$, i numeri devono essere scomposti in fattori primi e il risultato è dato dal prodotto di tutti i fattori primi comuni ai due.

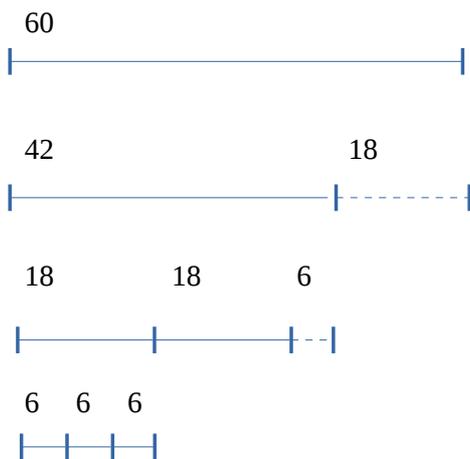
Il $MCD(60,42)$ è 6, infatti:

$60=2*2*3*5$ e $42=2*3*7$, i fattori primi in comune sono $2*3$.

Per risolvere questo problema si utilizza l’algoritmo di Euclide, un algoritmo antichissimo.

Per calcolare il $MCD(60,42)$ si costruiscono due segmenti di lunghezza 60 e 42:

- si toglie il minore dal maggiore per quante volte è possibile, in questo caso una.
- se vi è un segmento residuo (il 18, che rappresenta il resto della divisione tra 60 e 42) si toglie il residuo dal più piccolo dei segmenti precedenti (il 42) per quante volte è possibile (due volte)
- si ripete il procedimento finché non vi è più residuo.
- l’ultimo segmento costruito (il 6) dà il massimo comun divisore tra i segmenti originali.[27]



Osservando la figura, si può osservare che la divisione intera (modulo) viene ripetuta finché c’è il resto. L’ultimo costituisce il MCD. (Tabella 3)

È necessario quindi :

- fare un ciclo per ripetere la divisione
- considerare due variabili a e b che scambiano i valori all’interno del ciclo per poter eseguire le divisioni
- e una terza r che mantiene aggiornato il valore del resto

Tabella 3

a	b	a%b=r
60	42	60%42=18
42	18	42%18=6
18	42	18%6=0

```
#MCD_Euclide
def mcd_euclide(a,b):
    print("MCD tra",a,"e",b,)
    while(b != 0):
        r=a%b
        a=b #scambia a con b
        b=r
    return a

print("Il MCD è",mcd_euclide(60,42))
```

Descrizione del funzionamento del programma MCD_Euclide.

Le variabili x e y conservano i valori di a e b (60,42) per essere utilizzati nella stampa del risultato.

All'interno del ciclo viene reiterata l'operazione di modulo mediante gli scambi dei valori tra le variabili.

Il ciclo termina quando la divisione dà resto zero. La funzione restituisce a, la variabile messa in evidenza col colore rosso.

a	b	r=a%b	Scambia a con b	Scambia b con r
60	42	18	42	18
42	18	6	18	6
18	6	0	6	0

Il risultato del programma è:

```
Il MCD tra 60 e 42 è: 6
```

Programma “Inverti_lista.py”

Data una certa lista, vogliamo stampare gli elementi che la compongono in ordine inverso, e mostrarli come tipo stringa:

```
gusti_gelato=[Cioccolato, Fragola, Nocciola, Fiordilatte]
```

```
gusti_gelato_invertiti=Fiordilatte, Nocciola, Fragola, Cioccolata
```

Per fare questa inversione occorre utilizzare un ciclo in cui:

- si parte dall'ultimo elemento della lista gusti_gelato per arrivare al primo,
- gli elementi selezionati in ciascun passaggio vengono assegnati ad una nuova variabile.

```
#Inverti_lista

gusti=["Cioccolato", "Fragola", "Nocciola", "Fiordilatte"]
i=len(gusti)-1
gusti_invertiti=""#variabile tipo stringa
while i >=0:
    elemento=gusti[i]
    gusti_invertiti=gusti_invertiti+""+elemento+", "
    i=i-1
print(gusti_invertiti)
```

Descrizione del funzionamento del programma.

Nelle liste gli elementi sono numerati partendo dallo zero, come mostrato nella Tabella 4.

La funzione **len()** restituisce la lunghezza della lista, quindi, per ottenere l'ultimo elemento della lista, occorre fare `len(gusti)-1`. In questo modo il ciclo parte con l'indice `i` uguale a 3.

Tabella 4

Elemento 0	Elemento 1	Elemento 2	Elemento 3
Cioccolato	Fragola	Nocciola	Fiordilatte

`gusti[i]` sono gli elementi della lista che vengono assegnati alla variabile `elemento` (Tabella 15) ad ogni passaggio del ciclo.

Nella variabile `gusti_invertiti` (Tabella 5) vengono concatenati gli elementi in senso inverso rispetto a `gusti`.

Tabella 5

<code>i</code>	<code>elemento</code>	<code>gusti_invertiti</code>
3	Fiordilatte	Fiordilatte
2	Nocciola	Fiordilatte, Nocciola
1	Fragola	Fiordilatte, Nocciola, Fragola
0	Cioccolato	Fiordilatte, Nocciola, Fragola, Cioccolato

La stampa di `gusti_invertiti` mostra gli elementi tipo stringa.

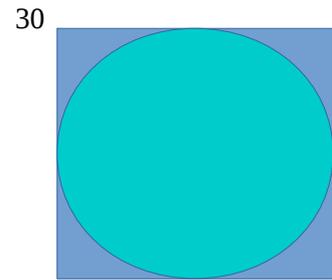
Il risultato del programma è:

```
Fiordilatte, Nocciola, Fragola, Cioccolata
```

Programma “Circonferenza.py”

Come mostrato nella figura di lato, un quadrato contiene un cerchio.

Il lato del quadrato misura 30 cm di lato, qual è la circonferenza del cerchio? [28]



Conoscendo la misura del lato del quadrato conosciamo anche il diametro del cerchio dal quale possiamo ricavare la circonferenza:

$$\text{diametro} = \text{raggio} * 2$$

$$\text{circonferenza} = \text{raggio} * 2 * \Pi$$

```
#Circonferenza

lato_quadrato=30
diametro=lato_quadrato
pi_greco=3.14
circonferenza=diametro*pi_greco
print("La circonferenza del cerchio è: ",circonferenza)
```

Descrizione del funzionamento del programma Cerchio.py.

Il dato di input è il valore del lato del quadrato, pertanto alla variabile `diametro` viene assegnato il valore di `lato_quadrato`. Avendo a disposizione i dati necessari si procede al calcolo della circonferenza. Il programma stampa:

```
La circonferenza del cerchio è: 94,2
```

4. Conclusioni

Da alcuni decenni, autorevoli studiosi hanno segnalato l'importanza di insegnare l'informatica nella scuola primaria e secondaria sia di primo che di secondo grado, ma solo nell'ultimo decennio, lavorare sull'alfabetizzazione informatica e digitale è divenuto un obiettivo da raggiungere per molte nazioni specie nel mondo occidentale. Ciò in virtù del fatto che ogni studente deve sviluppare abilità tecniche cognitive e culturali che gli consentano di non essere un consumatore passivo delle tecnologie e dei servizi digitali di cui oggi si dispone. Il raggiungimento di questo obiettivo rende gli individui cittadini consapevoli capaci di affrontare la società moderna.

Riallacciandosi al tema di questo lavoro *pensiero computazionale* non significa solo scrivere un programma utilizzando uno tra i tanti linguaggi informatici disponibili, ma significa fare esperienza con la programmazione, considerata uno degli strumenti mediante i quali si può apprendere il pensiero computazionale. Elaborare soluzioni mediante l'uso sistematico del ragionamento consente una nuova visione del mondo che ci circonda e costituisce un valido supporto per affrontare la vita quotidiana e professionale.

In questa tesi sono trattati i temi fondamentali dell'informatica, dallo studio del problema alla soluzione computazionale mediante le strutture dati e i costrutti sintattici di base di un linguaggio di programmazione. Sono mostrati i passaggi per risolvere alcuni problemi di calcolo. Prevalentemente problemi matematici classici della scuola media ai quali sono stati aggiunti problemi tipici dello studio di algoritmica, tra cui i Numeri di Fibonacci.

Per ciascun argomento trattato gli esempi forniti sono adeguati all'età degli studenti. Viene mostrato lo studio del problema, il ragionamento che porta alla soluzione e la discussione per illustrare la scelta del costrutto usato come più efficiente tra quelli possibili. Il linguaggio Python viene usato per trasformare in un programma formale l'algoritmo individuato.

Ulteriore scopo di questo studio è quello di mostrare agli insegnanti di matematica delle scuole medie inferiori e del biennio delle superiori, che molti degli argomenti disciplinari previsti, quali aritmetica, geometria, ecc. , possono essere affrontati anche dal punto di vista del pensiero computazionale e risolti con semplici programmi Python.

5. Bibliografia

- [1] Nardelli, Enrico. *Do we really need computational thinking?*, *Communications of the ACM*, in corso di pubblicazione. 2017
- [2] Wikipedia, voce Seymour, Papert. http://it.wikipedia.org/wiki/Seymour_Papert
- [3] Knuth, Donald. Computer Science and its relation to Mathematics, *The American Mathematical Monthly*, vol.81,n.4, p 323,1974.
- [4] Forsythe, George. What to do till the computer scientist comes, *The American Mathematical Monthly*, vol.75, n.5, p 456, 1968.
- [5] Wing, Jeannette M. *Computational Thinking*. *Communications of the ACM*, 49-3 pp.33-35. 2006
- [6]<https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study>
- [7] http://www.academie-sciences.fr/pdf/rapport/rads_0513.pdf
- [8] <https://obamawhitehouse.archives.gov/blog/2016/01/30/computer-science-all>
- [9] *Programma il futuro* <https://programmailfuturo.it/progetto/descrizione-del-progetto>
- [10] *Programma il futuro* <https://programmailfuturo.it/progetto/cose-il-pensiero-computazionale>
- [11] Nardelli, Enrico. *Do we really need computational thinking?*, *Communications of the ACM*, in corso di pubblicazione. 2017
- [12] Nardelli, Enrico. *Informatica nella scuola: disciplina fondamentale e trasversale, ovvero “di cosa parliamo quando parliamo di pensiero computazionale”*. pp 36-37, *Scienze e Ricerche*. 2017
- [13] Nardelli, Enrico. *Informatica, il pensiero computazionale: una competenza per il futuro*. <https://www.ilfattoquotidiano.it/2015/06/26/informatica-il-pensiero-computazionale-una-competenza-per-il-futuro/1816578/>
- [14][15] Nardelli, Enrico. *Informatica nella scuola: disciplina fondamentale e trasversale, ovvero “di cosa parliamo quando parliamo di pensiero computazionale”*.pp 38-39 *Scienze e Ricerche*. 2017
- [16][17][18] Nardelli, Enrico. *Dal coding a Borges*.<http://link-and-think.blogspot.it/2017/10/dal-coding-a-borges.html>
- [19] *Programma il futuro* <https://programmailfuturo.it/progetto/cose-il-pensiero-computazionale>
- [20] Luca, Serianni. *Grammatica Italiana*. 2005
- [21] Wikipedia, voce algoritmo. <https://it.wikipedia.org/wiki/Algoritmo>
- [22][23] Treccani, voce algoritmo. <http://www.treccani.it/vocabolario/algoritmo/>
- [24] Arpinati, Anna e Mariarosa Musiani *Matematica in azione1.Aritmetica*. p-73,n.33.

- [25] Arpinati, Anna e Mariarosa Musiani *Matematica in azione1.Aritmetica*. 63
- [26] Arpinati, Anna e Mariarosa Musiani *Matematica in azione1.Aritmetica*. p 301
- [27] Pagli, Linda e Luccio Fabrizio. *Storia matematica della rete*. 2007
- [28] Arpinati, Anna e Mariarosa Musiani *Matematica in azione1.Geometria*.p67

Ringrazio la prof.ssa Pagli per avermi seguita nella redazione della tesi, Rosanna e gli amici che mi sono stati vicini in questo percorso di studio dimostrandomi sempre un grande affetto.