UNIVERSITÀ DEGLI STUDI DI PADOVA

DEPARTMENT OF CHEMICAL SCIENCES
DEPARTMENT OF MOLECULAR MEDICINE

DiSC

Metodi di Calcolo per la Chimica

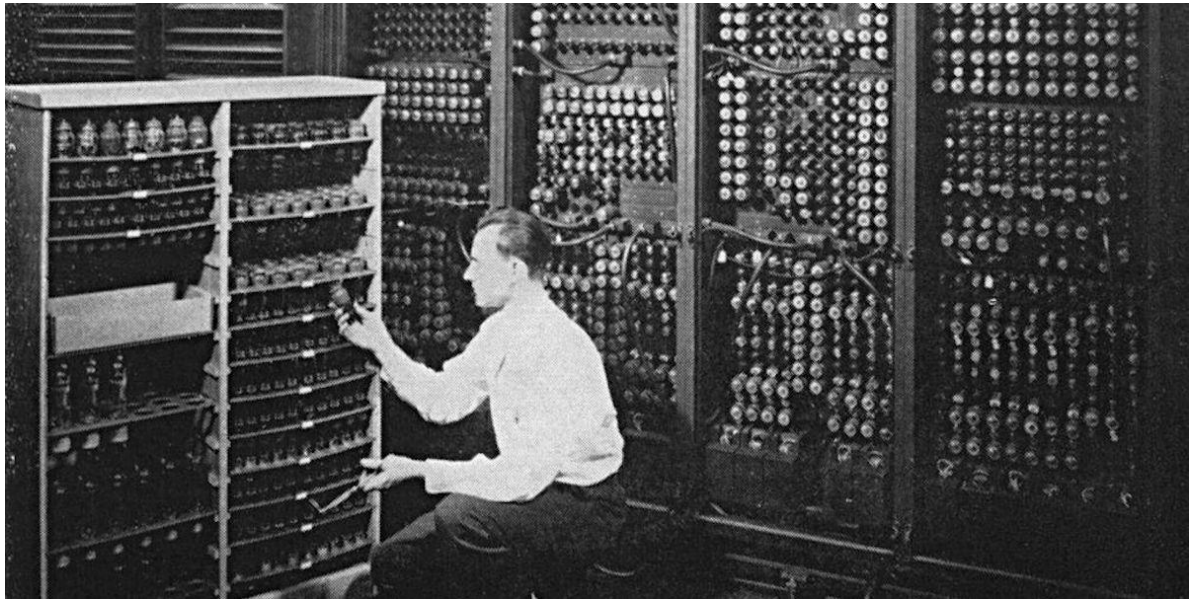
Concetti Base della Programmazione

Agostino Migliore

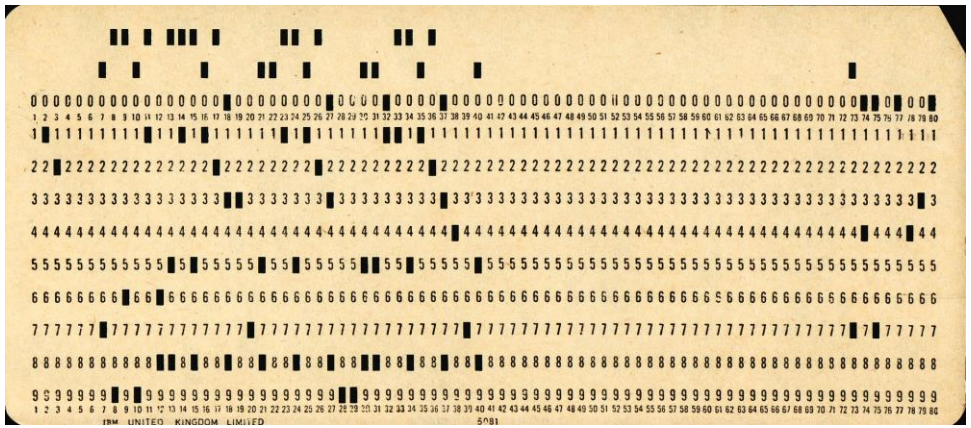
Introduzione alla programmazione

Prima di analizzare gli strumenti che ci servono per programmare un computer (cioè per fare in modo che il computer elabori opportunamente le informazioni che gli forniamo e produca l'output desiderato) facciamo pochi cenni utili alla storia della programmazione.

Nei dintorni del 1940 iniziarono i primissimi dispositivi che possono chiamarsi computers. Si programmavano direttamente in linguaggio macchina. Il programmatore doveva settare ogni singolo bit a 1 o 0 su un computer enorme che occupava un intero locale. Non vi erano monitors, si usavano schede perforate per fornire i dati al computer, che inviava i risultati su telescriventi.



ENIAC, M. Weik, Public domain, via Wikimedia Commons



Pete Birkinshaw/Flickr. Licenza: CC BY 2.0

Negli anni 50, seguendo i progressi tecnici nella realizzazione dei computer, nasce un importante linguaggio di programmazione: **FORTRAN** (FORmula TRANslator = traduttore di formule), finalizzato all'esecuzione di calcoli matematici, soprattutto di interesse scientifico.

All'inizio degli anni 70 nacque il linguaggio **C**, molto versatile per strutturare e rappresentare dati. Tale linguaggio fu la base per un altro linguaggio di programmazione più rivoluzionario, **C++**, presentato da Bjarne Stroustrup nel 1983. C++ introdusse la cosiddetta programmazione orientata agli oggetti ("object oriented programming", OOP), usando una nuova struttura: la classe (in inglese: class).

La classe si può definire come una categoria, un tipo, un modello di oggetto. Essa descrive le proprietà di tutti gli oggetti che sono caratterizzati da

- una stessa struttura interna,
- uno stesso comportamento,
- uno stesso insieme di operazioni applicabili ad essi.

In altri termini, una certa classe definisce un insieme di attributi e metodi che possono essere utilizzati per creare e manipolare gli oggetti che appartengono ad essa, che sono di un certo tipo. Un oggetto è una realizzazione specifica, un esemplare, un esempio (in inglese: instance) di quella classe. Classe e tipo possono sembrare la stessa cosa, ma si può individuare una sottile differenza concettuale tra di essi. Infatti, in termini concettuali, la classe può essere vista come il modello, il "template", che definisce il tipo di oggetto, mentre il tipo raprresenta la (generica) forma degli oggetti risultanti dal quel template. Quando si parla di classe l'enfasi è sulla costruzione, sul progetto, sul template, mentre quando si parla di tipo è sul prodotto.

Per chiarire il concetto usiamo una semplice analogia.

Definiamo la classe *automobile* caratterizzata da un motore, quattro ruote, ecc. (assemblati in una certa maniera) e che si muove in un certo modo, ecc.
La *Ferrari* è un oggetto della classe *automobile*.
Alla domanda "che tipo di oggetto è la *Ferrari*?" possiamo rispondere che "è un'*automobile*".

Volendoci esprimere secondo lo stile di un linguaggio di programmazione possiamo definire la classe:

```
classe automobile:
   definizione – assegnazione di (T, M, S, m, np, C, c, ...)
      targa = T
      marchio = M
      sottomodello = S
      motore = m
      numero porte = np
      colore = C
      cric per cambiare le ruote = c
      ...
```

La targa è inclusa tra i parametri di definizione perché ogni oggetto della classe possa essere definito in

modo univoco. A questo punto, supponiamo che John possieda una Ferrari Purosangue, con targa AA000AA, ecc. Prima di tutto, per semplificare la scrittura, assegniamo un nome breve alla vettura di John:

VJ = veicolo di John

Tale oggetto della classe automobile è definito come segue:

VJ = automobile(AA000AA, Ferrari, Purosangue, ..., rosso, ...,)

A questo punto, il computer sa che *VJ* è un'*automobile* con certe caratteristiche specifiche. Infatti, se chiediamo al computer di che tipo (in inglese: type) sia tale oggetto, digitando su un terminale qualcosa come

type(VJ)

otteniamo la risposta

automobile

Inoltre, con opportuna sintassi dipendente dal linguaggio di programmazione usato, possiamo chiedere quale sia la targa, ecc. Per esempio, l'istruzione necessaria potrebbe essere del tipo

VJ.targa

e, pressando il tasto invio per attuare tale istruzione, otterremmo sul terminale il risultato

AA000AA

Si noti che una classe è caratterizzata da attributi (che definiscono le caratteristiche o proprietà degli oggetti che appartengono a tale classe) e metodi o funzionalità, cioè procedure che operano sugli attributi.

Ritorniamo adesso un attimo alla nostra storia. Abbiamo detto che C++ ha introdotto la OOP. Dopo, **JAVA** è stato il primo linguaggio progettato appositamente per la OOP (a differenza di C++ che è stato creato come adattamento alla logica OOP di un linguaggio preesistente). Nel modello (o "paradigma") di OOP, il programma viene visto come un insieme di oggetti interagenti tra di loro. Questa è anche la logica di **Python**, un linguaggio di programmazione di alto livello caratterizzato da dinamicità, semplicità e flessibilità.

La semplicità, l'immediatezza espressiva di Python lo rendono somigliante a uno pseudocodice (che descriveremo tra poco). Inoltre, Python, come altri linguaggi di programmazione moderni, si presta molto bene alla creazione di codici (programmi) con blocchi. Un blocco e una parte di programma che può anche essere esguita come un'unità separata. La definizione di una classe è un esempio di blocco. Vedremo altri esempi in seguito.

Programma e Algoritmo

Per svolgere un compito a un certo momento, un computer necessita di una sequenza di istruzioni che gli dicano passo passo cosa fare. Un programma è un compito, composto da una serie di istruzioni, che il computer deve svolgere. Il programma è scritto in un linguaggio di programmazione opportuno per noi piuttosto che per la macchina. Poi, nel caso di Python e di diversi altri linguaggi di programmazione di alto livello, un cosiddetto interpreter (un opportuno programma che fa il mestiere di "interprete") legge le istruzioni del programma e via via le traduce nel linguaggio macchina per la loro esecuzione.

Noi non abbiamo bisogno di conoscere come funziona l'interpreter. Invece, dato un certo problema o compito, dobbiamo analizzarlo, individuare la procedura per risolverlo e definire tale procedura in termini di un set discreto di passi eseguibili e non ambigui. In altre parole, dobbiamo costruire un agoritmo:

Algoritmo = procedimento che consiste di una successione ordinata di istruzioni atta a risolvere un problema, ottenendo un preciso risultato a partire da un certo set di dati iniziali.

Una volta definito, l'algoritmo deve essere tradotto nel linguaggio di programmazione (così formulando il programma) per poter essere eseguito. Vi sono due tipi principali di rappresentazioni che aiutano a tracciare un algoritmo in modo semplice e intuitivo, prima di esprimerlo in un linguaggio di programmazione specifico: pseudocodice e diagramama di flusso.

Pseudocodice versus Codice

Possiamo definire **pseudocodice** (o **pseudolinguaggio**) un qualsiasi linguaggio informale (ma non ambiguo) che permetta di descrivere un algoritmo usando una sintassi "naturale", "umana", senza le rigide regole di un reale linguaggio di programmazione. Nello pseudocodice ogni riga rappresenta un'istruzione e, se non altrimenti specificato, le righe vanno considerate in ordine discendente. Per fare un esempio, scriviamo, in pseudocodice, un blocco decisionale atto a verificare se un utente è maggiorenne. Tale pseudocodice è mostrato sotto a sinistra, mentre a destra potete vedere un modo di scrivere il corrispondente piccolo codice in Python.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

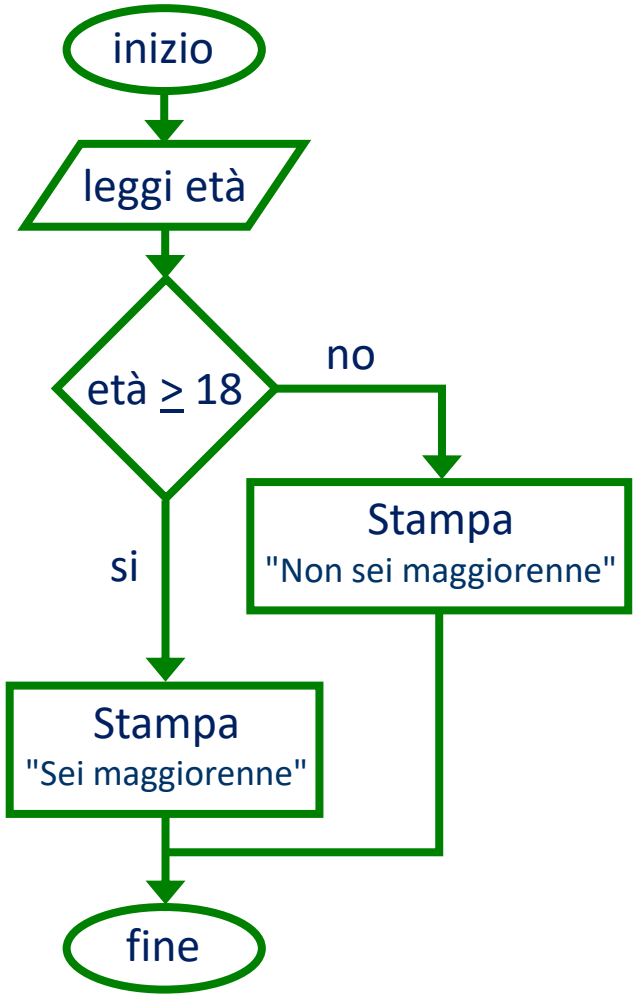
Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Considerato che, in inglese, "scrivere " nel senso di " stampare, pubblicare " si dice "print" , e che age = età, if = se, else = altrimenti, potete notare quanto lo stile di Python sia prossimo a quello di uno pseudocodice, rendendolo quindi un linguaggio molto intuitivo. Ciò, chiaramente, ne rappresenta un punto di forza.

Diagramma di flusso (flowchart)

Possiamo alternativamente rappresentare tale codice usando anche un **diagramma di flusso** come segue.



Il diagramma di flusso parte sempre da un blocco iniziale, che ne costituisce il punto di ingresso ed è rappresentato da un ovale.

Segue un blocco di input/output (I/O), che nel caso specifico descrive un'azione di lettura (e presuppone una immissione del dato, effettuata come attività di ingresso all'inizio). I blocchi di I/O sono rappresentati tramite parallelogrammi.

Si arriva quindi a un blocco decisionale o test a forma di rombo. Esso rappresenta una condizione, che è il fulcro di un costrutto o struttura di controllo (in particolare, in questo caso, una struttura condizionale): l'azione successiva dipendera dal fatto che la condizione età > 18 sia soddisfatta o meno dal dato in ingresso. Tale blocco ha quindi una freccia entrante e due uscenti.

Segue una di due possibili azioni, rappresentata da un blocco di elaborazione o di azione a forma di rettangolo. I blocchi di lettura/scrittura e di azione hanno una sola freccia entrante e una uscente.

Il blocco finale chiude la procedura.

Concludiamo che, nel diagramma di flusso, ci sono tre tipi di operazioni:

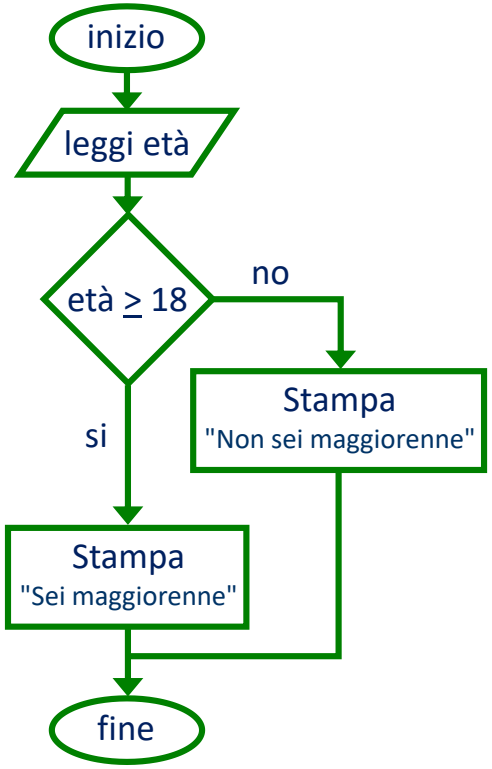
- Azioni, che consistono in un'attività o un'elaborazione da svolgere, le quali generalmente richiederanno l'uso di funzioni nel programma.
- Condizioni, o test, che determinano due o più direzioni in cui la procedura può continuare a seconda che la condizione desiderata sia soddisfatta o meno. Il costrutto if ... else nel programma vero e proprio contiene le condizioni e le azioni conseguenti.
- Ingresso/uscita, ovvero input/output, che riguardano l'immissione di informazioni dallo esterno e l'invio di informazioni verso l'esterno.

I diagrammi di flusso sono soggetti a condizioni di validità:

- condizioni sui blocchi: quelle viste prima, riguardanti i numeri di frecce in contatto con i blocchi di azione e di lettura /scrittura;
- condizioni sulle frecce: ogni freccia deve entrare in un blocco;
- condizioni sui percorsi: dal blocco iniziale deve sempre essere possibile raggiungere ogni altro blocco e da ogni blocco deve essere possibile raggiungere il blocco finale.

Componenti di un programma

Diagramma di flusso



Pseudocodice

1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3. scrivi "Sei maggiorenne"
4. altrimenti:
5. scrivi "Non sei maggiorenne"

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Diagramma di flusso, pseudo codice e codice vero e proprio sono messi a confronto sopra.

Dallo pseudocodice e, ovviamente, dal piccolo codice (in inglese: code snippet), emergono diversi ingredienti essenziali di un programma, che discuteremo in termini di pseudolinguaggio e, sfruttando la semplicità del codice Python, anche in termini più precisi interenti al linguaggio Python.

Pseudocodice

1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3. scrivi "Sei maggiorenne"
4. altrimenti:
5. scrivi "Non sei maggiorenne"

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

```
[*]: age = input("Inserisci la tua età: ")
     if int(age) >= 18:
         print("Sei maggiorenne")
         print(age)
     else:
         print("Non sei maggiorenne")
Inserisci la tua età:
```

Dalla prima riga di codice e pseudo codice emerge una variabile, *age*. La prima riga rappresenta una istruzione di assegnazione o assegnamento, atta ad attribuire alla variabile *age* un dato valore. Come vedremo in seguito, quando lanciamo il codice, esso scrive in un'apposita finestra la frase (stringa di caratteri) Inserisci la tua età: e aspetta che noi inseriamo un valore numerico da assegnare alla variabile *age* (vedi anche sopra) e pressiamo il tasto invio. Dopo, di ciò, il programma fornisce il risultato della sua elaborazione.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Variabili

Una variabile è una sorta di contenitore per un valore, che può essere un numero, una stringa e altri oggetti. Detto in termini ancora più precisi, una variabile è un riferimento, un nome che punta alla specifica posizione in memoria di un valore (che è l'oggetto). In breve, si può definire una variabile come un nome che fa riferimento a un valore. Il valore assegnato inizialmente può generalmente cambiare durante l'esecuzione dell'algoritmo.

In Python, la stessa istruzione di assegnazione crea la variabile, specificandone il nome e assegnandole un valore. Dopo di ciò, ogni volta che chiamiamo il nome della variabile usiamo il valore assegnatogli. È un po' come quando memorizziamo il numero di un amico nella rubrica telefonica: per chiamare il nostro amico, inseriamo il suo nome (la variabile) e compare il suo numero di telefono (cioè il valore della variabile, che è quello che interessa allo strumento telefono per poterci mettere in contatto con lui).

Notiamo pure che nel codice Python l'assegnazione è fatta col segno =, che quindi non rappresenta una semplice uguaglianza come negli pseudocodici o nei diagrammi di flusso.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Costanti ("literals")

Nello pseudocodice e nel codice di sopra, il numero esprimente l'età che inseriamo noi e il numero 18 sono dei valori numerici, delle costanti, che in Pyhon sono chiamate, in inglese, literals (più precisamente, in questo caso, numeric literals). Una valore o costante, o literal, è una quantità fissa, che quindi non dipende dai dati di input e non cambia durante l'esecuzione di un algoritmo. Detto in altri termi, il literal è il valore in sé, espresso come se stesso (nello stesso senso in cui diciamo "letterale", "alla lettera", "letteralmente") piuttosto che come valore assegnato ad una variabile. Si noti che anche la stringa (= sequenza di caratteri) in virgolette è una costante.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Funzioni

L'azione dello scrivere la frase "Inserisci la tua età: " e ricevere il valore in ingresso è enunciata nello pseudocodice ed effettuata dalla funzione input() nel codice Pyton, che è in grado di prendere il valore di ingresso fornito dallo utilizzatore e di darlo in pasto al programma.

In generale, una funzione è un blocco di codice, o modulo, a se stante che prende in entrata dei valori e restituisce uno o più risultati. La funzione è identificata da un nome. Una volta creata una funzione, ogni volta che serve fare le operazioni svolte da essa, non è necessario riscriverle: basta invocare (chiamare con nome) la funzione. Inoltre, se vogliamo, possiamo modificare la funzione (per esempio, per ottimizzarla o arricchirla) senza dover intaccare il codice che la invoca. Nel caso presente, semplicemente invochiamo la funzione *input()* ed essa svolge la sua azione. Tale funzione è stata scritta prima, altrove. In realtà è una delle cosiddette funzioni incorporate (built-in functions) nel linguaggio Python, messe a disposizione degli utenti.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Metodi

Le funzioni definite all'interno di una classe, che consentono di manipolare gli oggetti pertinenti, sono i cosiddetti metodi, a cui abbiamo accennato prima.

Operatori

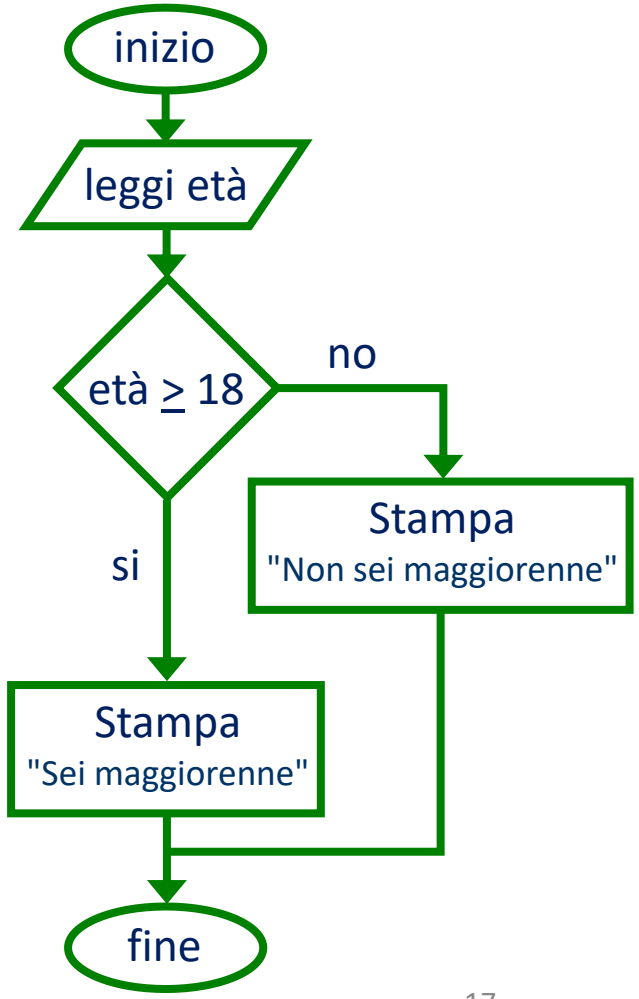
Nel diagramma di flusso, nello pseudocodice e nel codice compare anche un stesso operatore, anche se indicato col simbolo ≥ nei primi due e con >= nel linguaggio Python. Si tratta di un *operatore di confronto* (come =, ≠, < , ≤ e >). Vi sono pure *operatori aritmetici* (+, –, ...) ed *operatori logici*. Un operatore è una operazione, una legge che combina due valori per fornire un valore risultante. In questo caso, i due valori sono numeri (il valore di *age* e 18) e il riultato è un valore di tipo diverso, cioè uno dei due valori logici, o valori di verità (in inglese, logical values or truth values) vero e falso. Gli operatori servono a costruire espressioni o a controllare il flusso di un programma.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```



Espressioni

L'operatore visto prima costruisce un espressione: età ≥ 18 (pseudocodice) o analogamente int(age) >= 18 (codice). In generale, un'espressione è una sequenza di variabili, costanti o espressioni più piccole combinate tra loro mediante operatori, come siete abituati a vederle in matematica.

Si noti che il fatto che l'output della nostra espressione è un valore logico si evince chiaramente dal diagramma di flusso, che si ramifica e contiene le parole si / no sui due percorsi alternativi successivi: avremmo potuto sostituire tali parole con vero / falso.

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Costrutti di controllo

L'espressione analizzata prima appare all'interno di un costrutto (o struttura) di controllo. I costrutti di controllo sono delle costruzioni, dei dispositivi sintattici che consentono di combinare istruzioni elementari, o blocchi di istruzioni, per controllare e incanalare il flusso di esecuzione di un programma, sia che esso venga scritto in uno pseudocodice o in forma di codice vero e proprio.

In particolare, nel caso di cui sopra, abbiamo a che fare con un cosiddetto costrutto condizionale o costrutto di selezione, in cui vi è una istruzione condizionale, cioè un'istruzione (in inglese: statement) che espone una condizione e che detta al programma di selezionare uno di due percorsi a seconda che tale condizione sia vera (cioè sia soddisfatta dai dati) o falsa.

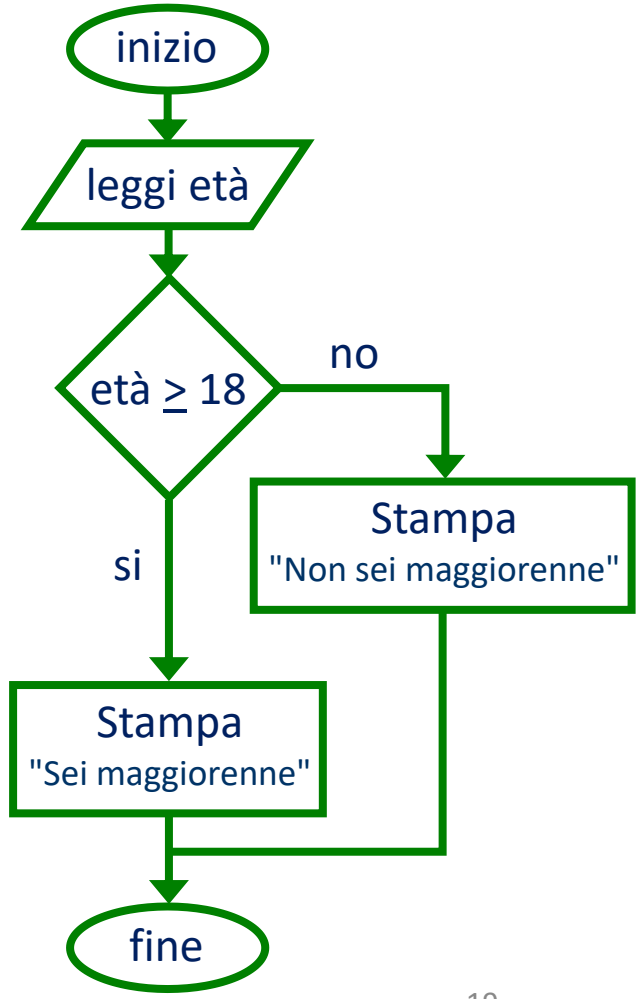
La presenza di due percorsi di esecuzione alternativi è espressa dalla parola altrimenti / else , ma si vede ancora più chiaramente usando il diagramma di flusso, che si dirama in due percorsi dopo aver stabilito la condizione.

Pseudocodice

1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3. scrivi "Sei maggiorenne"
4. altrimenti:
5. scrivi "Non sei maggiorenne"

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```



Parole chiave

Si noti che per costruire l'istruzione condizionale abbiamo usato la parola se (if), che una delle parole chiave. Le parole chiave sono parole che hanno un significato, un uso specifico all'interno di un programm e, in quanto tali, non si possono usare altrove. Per esempio, in uno programma possiamo chiamare una variabile in una miriade di modi (anche Pippo!), ma non se (nell'ambito dello pseudocodice) o if (nell'ambito del codice). Nel seguito del costrutto condizionale, per definire il percorso alternativo, compare un'altra parola chiave, altrimenti (else).

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Indentazione

Il costrutto di controllo fa uso dell'indentazione. Un set di righe consecutive con lo stesso livello di indentazione all'interno di una funzione o di una struttura di controllo (dove è preceduto da una riga con una parola chiave) viene chiamato un blocco di istruzioni (o blocco istruzioni). Per esempio, il *blocco istruzioni* che segue se / if nel codice di sopra è costituito da una sola riga, che consiste in un'istruzione di scrittura. Il *blocco istruzioni* contiene, invece, due righe nel codice modificato mostrato accanto.

```
age = input("Inserisci la tua età: ")
if int(age) >= 18:
    print("Sei maggiorenne")
    print("Puoi guidare automobili")
else:
    print("Non sei maggiorenne")
```

L'indentazione accresce la chiarezza, la leggibilità di un programma, mettendo in luce gerarchie di funzioni e cicli. L'indentazione è obbligatoria (ove previsto, come sopra) in Python, secondo una linea di pensiero che mira a conseguire la massima chiarezza possibile.

Ancora su funzioni e costrutti

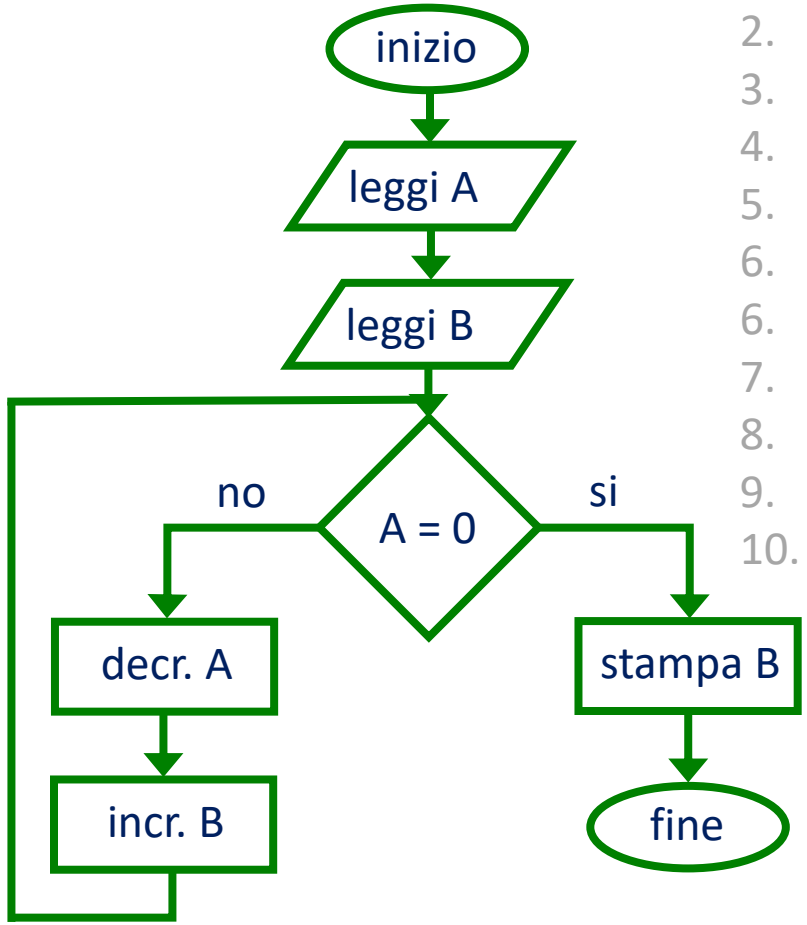
Come detto prima, una funzione può essere scritta separatamente e poi semplicemente chiamata (in gergo informatico, "invocata") tutte le volte che serve in un programma.

Come esempio, accanto mostriamo il diagramma di flusso e uno pseudocodice di una funzione che somma due numeri semplicemente sfruttando incrementi e decrementi di un'unità. La prima riga fornisce il nome della funzione (somma) e i parametri d'ingresso della funzione (A e B). Tale funzione contiene un altro tipo di costrutto: il costrutto iterativo o ciclo (in inglese: loop). In generale, si usano cicli per ripetere un dato blocco di istruzioni in modo controllato.

È bene notare che i parametri sono le variabili che compaiono nella definizione della funzione, mentre si dicono argomenti i valori passati alla funzione per tali parametri in una particolare chiamata.

Esempio

Scriviamo una funzione che somma due numeri interi utilizzando solo incrementi e decrementi di una unità.




```
1.  funzione somma(A, B)
2.      leggi A
3.      leggi B
4.      se A = 0:
5.          vai all'istruzione 9
6.      altrimenti:
6.          decrementa A
7.          incrementa B
8.          ritorna all'istruzione 4
9.      stampa B
10. fine funzione
```

Pseudocodice

```
1. scrivi: "Inserisci la tua età: " | leggi età dall'utente
2. se età ≥ 18:
3.     scrivi "Sei maggiorenne"
4. altrimenti:
5.     scrivi "Non sei maggiorenne"
```

Codice in linguaggio Python

```
1. age = input("Inserisci la tua età: ")
2. if int(age) >= 18:
3.     print("Sei maggiorenne")
4. else:
5.     print("Non sei maggiorenne")
```

Concludendo la discussione del codice di cui sopra, notiamo che esso contiene anche la funzione incorporata int(). Tale funzione di Python converte un numero decimale o una stringa in un oggetto di tipo numero intero. Vedremo nelle prossime lezioni perché in questo caso è necessario usare tale funzione.

Array

Infine, introduciamo brevemente il concetto di array. In parole semplici, una array (unidimensionale) si può definire come un elenco di elementi ai cui valori si può accedere per indice, che è il numero che rappresenta la posizione dell'elemento nella array. Si pensi per esempio a un vettore scritto come l'ennupla delle sue componenti.

Se sono necessari più indici per individuare la posizione di un elemento di una array, allora vuol dire che l'array ha più dimensioni, tante quanti sono gli indici. Nel caso bidimensionale, si pensi ad una matrice.

Letture consigliate

Alla pagina web

https://www.python.it/doc/libri/

trovate una lista di libri online e stampati suggeriti dal team di Python Italia. Tra questi consiglio innanzitutto il primo,

"Pensare in Python: come pensare da informatico"

che potete scaricare facilmente seguendo il link indicato nella pagina web.
Le note delle lezioni saranno via via scaricabili dalla pagina del corso al link

https://stem.elearning.unipd.it/disc

Seguono fonti per letture opzionali. Un utile tutorial in inglese si trova al link https://docs.python.org/3/tutorial/ . Potete inserirlo nel box del traduttore di Google per ottenerlo in italiano. Facendolo, io ottengo la pagina desiderata al link

https://docs-python-org.translate.goog/3/tutorial/?_x_tr_sl=en&_x_tr_tl=it&_x_tr_hl=it&_x_tr_pto=wapp

Altre due sorgenti di informazioni utili scritte in modo semplice e chiaro, ma in inglese, si possono trovare ai link

https://automatetheboringstuff.com/
https://python-textbok.readthedocs.io/en/1.0/index.html