NUMERO: **2543A** ANNO: 2022

# A P P U N T I

STUDENTE: Di Noto Giulia

MATERIA: Informatica - Prof. Corno

# Flow chart

10/03/2022

Un diagramma di flusso è composto da una combinazione di blocchi che corrispondono alle singole operazioni di calcolo con un quadrato, se si tratta di operazioni che coinvolgono l'input o output si tende a utilizzare un parallelogramma, questi blocchi possono essere collegati con delle frecce per indicare quali sono le operazioni da compiere, l'ordine è definito proprio dalle frecce, il terzo tipo di blocco è a forma di rombo che rappresenta un punto di scelta ed è l'unico che ha un ingresso e due uscite che corrispondono al caso in cui la condizione si avvera o al caso in cui la condizione sia falsa, in funzione a quel valore di falsità o meno della condizione, la elaborazione proseguirà da un lato diverso.

Come si disegna un flow chart?
Intanto è necessario avere in mente qual è la strategia esecutiva. Poi si disegnano i blocchi collegandoli nella sequenza opportuna. Il rombo si può utilizzare sia per avere delle alternative sia per avere delle ripetizioni.

Quindi abbiamo detto che per le alternative usiamo il blocco rombo, possibile che le cose da fare siano più di due quindi avrò una serie di scelte in cascata.

Proviamo ad applicare questo formalismo a dei problemi:

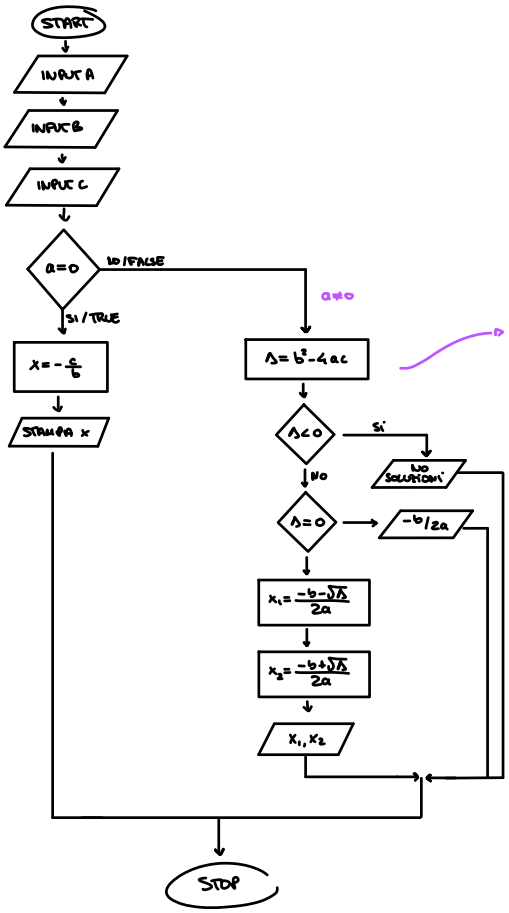
Equazioni di secondo grado, risolvi l'equazione.

OBIETTIVO
$ax^2+bx+c=0$

NODO SIMBOLICO CHE IDENTIFICA IL PUNTO DI PARTENZA DELL'ALGORITMO

BISOGNA ACQUISIRE I DATI D'INGRESSO

a=0 non significa "rendi a uguale a 0" perché è un = di confronto, non è = di assegnazione, sto dicendo "se a è uguale a 0", allora ho due possibilità, o sì o no, quindi posso proseguire in due modi diversi:




PIÙ SOLUZIONI

QUI AD ESEMPIO CI SONO TRE POSSIBILITÀ, MA IL ROMBO HA SOLO DUE USCITE, PER CUI USIAMO DEI ROMBI A CASCATA PER RAPPRESENTARE PIÙ DI 2 SOLUZIONI

Non vanno MAI incrociati i rami. Ad esempio non ha senso che un ramo che parte da un true, poi ritorni a un ramo interno false.

# Algoritmi

Un algoritmo è una descrizione passo passo di come risolvere un problema: una sequenza di azioni da compiere (istruzioni) per svolgere il compito dato e raggiungere un obiettivo specifico. Deve essere : non ambiguo, eseguibile e finito.

Uno dei principi della programmazione è DRY, cioè dont repeat yourself, quindi se c'è qualcosa che viene ripetuto più volte probabilmente c'è un modo più semplice di scriverlo.

È importante dare dei nomi sensati alle variabili. Alcuni linguaggi di programmazione chiedono di dichiarare le variabili all'inizio, altri chiedono di iniziarizzarle. Non deve succedere di non assegnare un valore alla variabile, darebbe errore.

L'uso del segno "=" è completamente diverso dal significato matematico, cioè un'identità sempre, in informatica non esiste nulla di questo genere, esistono due operazioni diverse che hanno a che fare con l'uguaglianza: l'assegnazione, cioè dare un valore a una variabile, l'altra cosa è il confronto di uguaglianza, questo tipo di = non cambia i valori ma li confronta, sono uguali o non sono uguali.

ESEMPIO 1

```
CALCOLA MCD(A,B)
IPOTESI: A,B >= 1

LEGGI A
LEGGI B

N = 1

RIPETI FINCHÈ N<=A E N<=B
1       SE A È DIVISIBILE PER N E B È DIVISIBILE PER N:
            MCD = N
2       N = N+1
        SE SCRIVESSI QUI "N = N+1" È COME SE FOSSERO DUE COSE CONTEMPORANEAMENTE (1, 2)
STAMPA MCD
```

→ QUI SE NON FOSSE MCD N NON INCREMENTEREBBE

↳ A QUESTA ALTEZZA

## Caratteri

**Queste tabelle di simboli (caratteri) sono etichettate con valori numerici: tabella ASCII.**

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DE |

CARATTERI DI CONTROLLO

CARATTERI STAMPABILI

**Il codice più aggiornato a tutti i caratteri esistenti e non solo quelli americani è: UNICODE.**

ord(nome[0]) «mi restituisce il valore numerico corrispondente al carattere nell'indice selezionato»

chr(74) «viceversa»

## Funtiomi

**Le funzioni sono un qualcosa che applico a un valore. Le funzioni predefinite non sono molte, ma esistono delle funzioni particolari che si applicano solo a un certo tipo di dato: ad esempio len si può applicare a una stringa o a una lista, invece altre funzioni come .upper() che converte in maiuscolo, può essere usata e ha senso solo con le stringhe.**

**Si possono inserire dei caratteri particolari all'interno della stringa stessa, quindi si antepone il carattere "\" prima del " virgolette da voler inserire all'interno della stringa come carattere. Anteporre a \ un altro \ per il backlash. Per andare a capo si utilizza \n.**

# Input

Possiamo immaginare di ricevere dati e informazioni da un utente esterno. Le due operazioni fondamentali perchè un programma possa interagire con l'utente in modalità testuale sono la visualizzazione di informazioni output gestito dalla funzione principale e da tutte le composizioni delle stringhe che ci permettono di visualizzare ciò che vogliamo, e l'altra operazione è quella di input.

Questo avviene tramite la funzione input(" "), in cui il programma aspetta che l'utente inserisca qualcosa, a quel punto ciò che l'utente ha scritto viene raccolto sotto forma di stringa e ne viene restituito il valore della funzione input stessa Cher sovente andremo a memorizzare sotto una variabile.

## Conversione

Per convertire una espressione posso farlo in un'unica riga:

```
age = int(input("inserisci la tua età:"))
price = float(input("inserisci il prezzo:"))
```

## Output

La print ci dà poco controllo su come compare l'output, per esempio con gli spazi, o con le cifre decimali dei numeri decimali. Ci sono casi in cui ho bisogno di un controllo maggiore.
Possiamo avere delle istruzioni di formattazione delle stringhe. Anziché dire alla print di convertire come vuole un numero, posso prepararla da solo con il formato che preferisco, i modi sono:
- concatenazione di stringhe
- f-string
- un operatore di formattazione %
- il metodo .format()

Quello che vedremo è il f-string:

```
print(f'the area of a circle of radius {r} is {area}')
```

Per controllare le conversioni che fa posso specificare dei dettagli utilizzando degli specificatoci di formato:

```
f"la distanza è {dist:8.2} metri"
                    ↳ indica fino a due cifre decimali
```

- Ampiezza, cioè quanto spazio devo occupare
- Precisione che indica quante cifre decimali voglio usare: "." E "precisione" es. .2

SE uso l'operatore > e < con le stringhe, risulterebbe:
a= uno
b= due

a>b poiché la u viene dopo in ordine alfabetico la d
se per caso però scrivessimo Uno con la U maiuscola, non diventa più maggiore ma diventa minore della u minuscola secondo le tabelle ASCII e UNICODE

U < u
c= Una
c < a

Nel confronto tra numeri reali: il calcolatore quando lavora con i float non ha un numero di cifre infinito e quindi deve approssimare.
Nella libreria Python esiste una funzione che ci aiuta a capire se l'approssimazione è abbastanza piccola: isclose( ) che definisce se i due numeri reali sono abbastanza vicini o no.

# Ciclo while

**Un ciclo while più semplice è controllato da un contatore che conta il numero di ripetizioni, avrà quindi il confronto con una costante che mi dice di aver finito; Oppure una condizione generale che diventa falsa interrompe il ciclo, senza prevedere facilmente quanti giri fare.**

Possibili esempi con errori

| Ciclo | Visualizza | Spiegazione |
|---|---|---|
| `i = 0`<br>`total = 0`<br>`while total < 10 :`<br>`  i = i + 1`<br>`  total = total + i`<br>`  print(i, total)` | 1 1<br>2 3<br>3 6<br>4 10 | Quando `total` diventa uguale a 10, la condizione del ciclo diventa falsa e il ciclo termina. |
| `i = 0`<br>`total = 0`<br>`while total < 10 :`<br>`  i = i + 1`<br>`  total = total - i`<br>`  print(i, total)` | 1 -1<br>2 -3<br>3 -6<br>4 -10<br>. . . | Dato che `total` non diventa mai uguale a 10, si ha un ciclo infinito (Errori comuni 4.2) |
| `i = 0`<br>`total = 0`<br>`while total < 0 :`<br>`  i = i + 1`<br>`  total = total - i`<br>`  print(i, total)` | (Niente) | La condizione `total < 0` è già falsa alla prima verifica, per cui il ciclo non viene mai eseguito. |
| `i = 0`<br>`total = 0`<br>`while total >= 10 :`<br>`  i = i + 1`<br>`  total = total + i`<br>`  print(i, total)` | (Niente) | Probabilmente il programmatore voleva scrivere "fermati quando la somma vale almeno 10", però la condizione presente nel ciclo dice quando il corpo va eseguito, non quando il ciclo deve terminare (Errori comuni 4.1). |
| `i = 0`<br>`total = 0`<br>`while total >= 0 :`<br>`  i = i + 1`<br>`  total = total + i`<br>`print(i, total)` | (Niente, il programma non termina) | Dato che `total` è e sarà sempre maggiore o uguale a zero, il ciclo prosegue all'infinito. Non visualizza niente perché la funzione `print` viene invocata fuori dal corpo del ciclo, dato che non ne fa parte (si osservi l'incolonnamento a sinistra). |

**Un altro errore a cui fare attenzione è la variabile di controllo del ciclo, deve comparire quattro volte: inizializzazione, condizione, corpo e aggiornamento.**

# Valori sentimella

I valori così detti sentinella: immaginiamo di dover scrivere un programma che legga una serie di numeri inseriti dall'utente in un ciclo, il ciclo terminerà quando l'utente ha inserito tutti i valori che doveva inserire, dobbiamo però inventarci un meccanismo che renda questo possibile.
Questo si può fare in diversi modi, tra cui per segnalare la fine di una sequenza di valori è usare il valore sentinella cioè che di per sé non sarebbe lecito e lo usiamo come indicatore che la sequenza sia terminata. Ad esempio utilizzare un valore che non sia confondibile con i valori effettivi.

Esempio

```
Salary = 0.0
while salary >= 0
    salary = float(input())
    if salary >= 0.0 :
        Total = Total + salary
        count = count + 1
```

Quindi nel caso in cui un dato inserito sia negativo come -1, l'if sarà una condizione falsa quindi non proseguo con le operazioni, torno su al while, non sarà maggiore o uguale a 0 allora esco dal ciclo while. In questo caso se il dato inserito dall'utente non è un dato normale, ma è particolare e concordato prima, allora esco dal ciclo.

## Booleame

Un'altra possibilità fa uso di variabili logiche o dette variabili booleane, cioè variabili che contengono i valori vero o falso, sono valori tipi di dato supportati dal linguaggio e possiamo memorizzarli nelle variabili e utilizzarli per i nostri controlli. Esistono due valori nuovi che si chiamano false e true.
Il valore di tale variabile può apparire ovunque nel mio programma sia necessaria una condizione, sia di un if che di un while. Posso avere delle variabili che si ricordano una condizione logica, cioè se una cosa è successa o non è successa.
Questa variabile viene detta Flag, bandierina alzata o abbassata, mi ricordo se ho visto fare questa cosa o no, se l'ho vista allora metto vero, se non l'ho vista allora metto falso.

## Break

Esiste un'altra istruzione che si può usare che è l'istruzione break, quando viene incontrata interrompe prematuramente il ciclo, come se la condizione fosse falsa ma a differenza che mi fa uscire anche senza completare le istruzioni correnti e va avanti a seguire le istruzioni successive al ciclo.

# Il ciclo for

Un caso particolare di ciclo con sintassi semplificata è il ciclo che viene introdotto da for.. in, è un ciclo che itera su un contenitore, su ogni suo singolo elemento, ad ogni nuova iterazione passa dal primo elemento, al secondo, al terzo ecc.. così no dopo l'altro vengono presi in considerazione ogni singolo elemento.
Immaginiamo di voler considerare un ciclo in cui stampo una per una le lettere di un nome:

```
State_meme = "Virginia"
i=0
while i< len(state_meme):
    letter=state_meme[i]
    print(letter)
    i=i+1
```

```
State_meme = "Virginia"
for letter in state_meme:
    print(letter)
```

## Range

Un altro contenitore è generato da una funzione che si chiama range, la funzione range(n) genera un contenitore di numeri interi, crea una sequenza di numeri interi che vanno da 0 a n-1. Quindi usando questa funzione range posso con un for emulare un ciclo while controllato da un contatore.
In questo caso non serve l'inizializzazione di i=1.

La funzione range può essere chiamata in tre modi diversi, con un argomento, con due argomenti o con tre argomenti. Con un argomento mi genero un contenitore che vanno da 0 a n-1, ad esempio range(5) mi genera una sequenza di cinque numeri: 0,1,2,3,4; se io specifico due argomenti, il primo argomento indica i valore di partenza compreso e il secondo il valore di arrivo escluso, quindi range(1,5)=1,2,3,4; con tre argomenti quando vogliamo che l'incremento non sia un incremento di 1, ad esempio range(1, 11, 2)=1,3,5,7,9, il passo d'incremento può anche essere negativo.
Non è possibile usare le booleane nel ciclo for.

## Cicli ammidati

funzioni utili:

SEPARATORI
- sep=' ' al posto dello spazio tra un oggetto e un altro di una stessa print mette ciò che voglio
- end=' ' al posto di andare a capo

```python
# Tracciare una tabella 4x10 con le potenze del numero inserito da 1 a 4
# Immagino un ciclo che ad ogni iterazione stampi una riga
# l'elevazione a potenza si scrive x**p dove x è la base e p l'esponente

N=10
P=4
p=1

print("Questa è la riga di intestazione")
for x in range(1,N +1):
    for p in range (1, 1+P):
        print(f'{x**p:10d}', end=' ') #fa in modo che ogni potenza prenda in totale 10 spazi bilanciando
    print(' ')
```

Non modifichiamo le variabili! E non nominiamo in ugual modo una variabile e un parametro.
Esiste una scorciatoia per inserire la documentazione all'interno della funzione stessa: una stringa tripla subito dopo la funzione stessa, come prima istruzione, e Python capisce che è la documentazione stessa, se faccio così se vado col mouse sulla funzione compare proprio la documentazione, cioè la definizione della funzione che ho creato.

Se ho la definizione di una funzione in un altro file, diverso da quello che sto usando, semplicemente per usare la funzione scrivo a inizio programma:

from nome_file import funzione

È solo necessario che i due file siano nella stessa cartella e che nel file in cui c'è la funzione, ci siano SOLO le definizioni delle funzioni, senza altre istruzioni, solo istruzione def.

## Parametri

Posso usare due meccanismi diversi per rendere i parametri opzionali: posso dare un valore di default, per cui se il parametro non viene specificato viene assunto quel valore default; oppure posso dargli un nome, nevicate che questo parametro può essere passato per nome.

```
14  def misura_ipercubo(lato, dimensioni=3):
15      return lato**dimensioni
16
```

(nome, nome, default)

## Valori restituiti (tupla)

C'è un trucco per far sì che la funzione possa restituire più di un valore: posso costruire un valore composto che al suo interno abbia più componenti, così la funzione effettivamente rispetta la sua "regola" di restituire un solo valore.
Cioè si costruisce una tupla (copia tripla, quadrupla, nupla ecc..).
Così quando chiamo la funzione posso salvare gli elementi di questa tupla in varie variabili.

Tutte le istruzioni return interrompono l'esecuzione delle funzioni.
Se la funzione non ha un'istruzione return, è come se ultima istruzione ci fosse un return senza nulla, return di none.

La concatenazione di due liste crea una nuova lista che contiene gli elementi della prima seguiti dagli elementi della seconda usando l'operatore + oppure usando il metodo .extend, in quest'ultimo caso il risultato è che modifica la lista su cui viene chiamato il metodo extend aggiungendo gli elementi dell'altra lista, che invece non viene modificata; nel primo caso invece devo memorizzare la somma su una lista, che posso scegliere qual è, posso per esempio crearne una completamente nuova.
Il metodo che si usa per ordinare una lista è il metodo sort() che modifica il contenuto della lista riordinando gli elementi in modo che alla fine saranno in ordine crescente. Quindi perdo la lista di partenza.
Un metodo analogo che non modifica la lista di partenza ma mi restituisce una nuova lista in cui gli elementi sono ordinati e la funzione è sorted().
Se volessimo ordinare in modo decrescente posso passare un secondo parametro che è reverse=True. Questo funziona su dati confrontabili.
Essendo le liste modificabili, a contrario delle stringhe, posso sostituire una porzione della mia lista esistente con un'altra lista.

**Table 1 Common List Functions and Operators**

| Operation | Description |
|---|---|
| `[]`<br>`[`$elem_1$, $elem_2$, ..., $elem_n$`]` | Creates a new empty list or a list that contains the initial elements provided. |
| `len(`*l*`)` | Returns the number of elements in list *l*. |
| `list(`*sequence*`)` | Creates a new list containing all elements of the sequence. |
| `values * num` | Creates a new list by replicating the elements in the `values` list `num` times. |
| `values + moreValues` | Creates a new list by concatenating elements in both lists. |

**Table 1 Common List Functions and Operators**

| Operation | Description |
|---|---|
| *l*`[from : to]` | Creates a sublist from a subsequence of elements in list *l* starting at position `from` and going through but not including the element at position `to`. Both `from` and `to` are optional. (See Special Topic 6.2.) |
| `sum(`*l*`)` | Computes the sum of the values in list *l*. |
| `min(`*l*`)`<br>`max(`*l*`)` | Returns the minimum or maximum value in list *l*. |
| $l_1$ `==` $l_2$ | Tests whether two lists have the same elements, in the same order. |

## Ricerca lineare

Ricerca in una lista (es. elenco) :
- ricercare un valore specifico (es. 7)
  - se c'è o non c'è? 7
    if 7 in elenco:
  - dov'è?
    pos= elenco.index(7) # se sono sicuro che ci sia
  - quanti ce ne sono?
    n= elenco.count(7)
- ricerca un valore che abbia determinate proprietà (es. > 100) :
  - c'è? (almeno uno?)
  - dov'è?/dove sono?
  - quanti sono?

Per questo secondo caso mi andrò a scandire gli elementi della lista ricercando all'interno di una lista un numero che sia maggiore di 100, posso usare un ciclo:

```
pos = 0
found = False
while pos < len(values) and not found:
    if values[pos] > limit:
        found = True
    else:
        pos = pos+1
if found:
    print("Found at position:", pos)
else:
    print("Not found")
```

Se non voglio fermarmi al primo valore, ma voglio raccoglierli tutti, posso creare una nuova lista per tutti i valori e trovo utilizzando un ciclo for semplificato.

```
result[]
for element in values:
    if (element > 100)
        result.append(element)
```

Oppure se non voglio sapere gli elementi procedo con un normale ciclo for counter. Se voglio sapere dove sono uso il ciclo for semplificato, con una lista che invece degli elementi stessi ha gli indici di dove si trovano i vari elementi.

```
values[posizioni[1]]
```

Così mi crea una lista degli elementi della lista partendo dal suo indice.

## Liste nelle Funzioni

```
def numeripari(values):
    pari = []
    for val in values :
        if val%2 = 0
            pari.append(val)
    return pari
values = [3, 6, 8, 10]
valori_pari = numeripari(values)
```

Che differenza c'è tra alias e copia? Sotto quali condizioni l'alias rimane vivo? Attraverso l'alias si può modificare il valore dell'argomento, se viene ri-assegnato il valore dell'argomento si perde l'alias.

```
values[i] = values[i]*2.0
values.append(5)
values.insert(4)
values.pop()
values.extend([2,3,4])
values.sort()
```

✔

```
values = []
values = values + [3]
values = sorted(values)
```

✘

```
17  def cancella_numeri_pari(values): #values diventa sinonimo di numeri in questo caso
18      for pos in range(len(values)-1,-1,-1):
19          if values[pos]%2==0:
20              values.pop(pos)
21  numeri=[1,2,3,4,5,6,7,8,9]
22  print(cancella_numeri_pari(numeri)) #di per sè se stampo la funzione non mi restituisce nulla, "None"
23  print(numeri) #ma mi modifica la lista che poi posso stampare
```

## Tuple

Una tupla si comporta come una lista però è immutabile, quindi tutte le operazioni che uso sulle liste le posso usare sulle tuple purchè non siano operazioni di modifica della liste. La tupla si crea con le parentesi tonde () e la virgola. Posso usare quindi tutt'e le operazioni che non modificano, posso concatenare le tuple, posso iterare, posso usare in, index, ecc.. sono comode per trasportare pacchetti di valore, per restituire un valore composto.
Anche per le tuple è possibile definire le comprehension, una volta creata una tupla non posso più aggiungere altri elementi.

```
Tupla = Tuple(i*i for i in range(0, 10))
```