```
numFailing = 0

# Inizializza le variabili usate per calcolare la media.
total = 0
count = 0

# Inizializza le variabili per il voto minimo e massimo.
minGrade = 100.0  # Ipotizzando che 100 sia il massimo voto possibile.
maxGrade = 0.0

# Legge con un ciclo controllato da evento, con predisposizione.
grade = float(input("Enter a grade or -1 to finish: "))
while grade >= 0.0 :
   # Incrementa il contatore di voti sufficienti o insufficienti.
   if grade >= 60.0 :
      numPassing = numPassing + 1
   else :
      numFailing = numFailing + 1

   # Determina se il voto è il nuovo massimo o minimo.
   if grade < minGrade :
      minGrade = grade
   if grade > maxGrade :
      maxGrade = grade

   # Aggiunge il voto alla somma totale.
   total = total + grade
   count = count + 1

   # Legge il voto successivo.
   grade = float(input("Enter a grade or -1 to finish: "))

# Visualizza i risultati.
if count > 0 :
   average = total / count
   print("The average grade is %.2f" % average)
   print("Number of passing grades is", numPassing)
   print("Number of failing grades is", numFailing)
   print("The maximum grade is %.2f" % maxGrade)
   print("The minimum grade is %.2f" % minGrade)
```

4.6 Il ciclo `for`

Capita spesso di voler esaminare tutti i caratteri di una stringa, uno dopo l'altro ordinatamente. Il ciclo `for` (descritto nella sezione Sintassi 4.2) rende particolarmente semplice la realizzazione di questa procedura. Supponiamo, ad esempio, di voler visualizzare una stringa, scrivendo, però, un solo carattere per riga. Non possiamo banalmente visualizzare la stringa con la funzione `print`, ma dobbiamo eseguire un ciclo che scandisca i singoli caratteri della stringa, uno dopo l'altro, e li visualizzi singolarmente. Ecco come si può usare un ciclo `for` per raggiungere questo obiettivo:

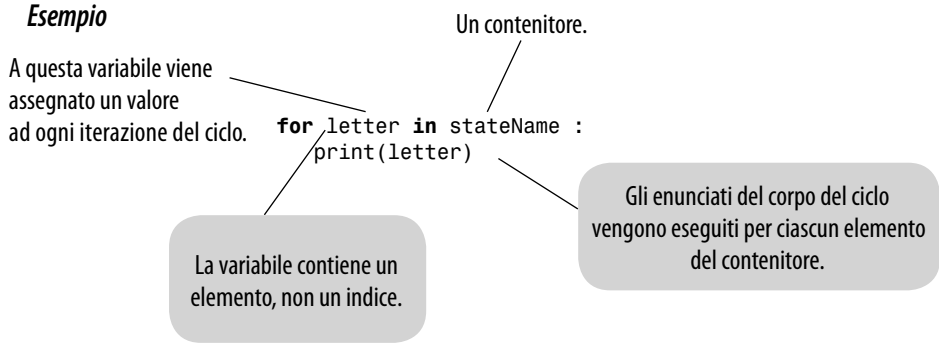
```
stateName = "Virginia"
for letter in stateName :
   print(letter)
```

Sintassi 4.2 Enunciato `for`

Sintassi

```
for variabile in contenitore :
    enunciati
```

Esempio



e questo è ciò che viene visualizzato:

```
V
i
r
g
i
n
i
a
```

Il corpo del ciclo viene eseguito per ciascun singolo carattere della stringa `stateName`, a partire dal primo. All'inizio di ciascuna iterazione del ciclo, il carattere successivo viene assegnato alla variabile `letter`, quindi viene eseguito il corpo del ciclo. Questo ciclo dovrebbe leggersi come "`for each letter in stateName`", cioè "per ciascuna lettera in `stateName`", ed è equivalente al seguente ciclo `while` che usa una variabile esplicita come indice:

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

Si noti una differenza importante tra il ciclo `for` e il ciclo `while`: nel ciclo `for` alla variabile di controllo `letter` vengono assegnati i valori `stateName[0]`, `stateName[1]`, e così via (e per questo viene detta *variabile elemento*, perché contiene sempre un elemento della stringa); nel ciclo `while`, invece, alla variabile di controllo `i` vengono assegnati i valori `0`, `1`, e così via (e per questo viene detta *variabile indice*, perché contiene sempre l'indice di un elemento della stringa).

Il ciclo `for` viene usato per eseguire iterativamente istruzioni sugli elementi di un contenitore.

Il ciclo `for` può essere usato per eseguire iterativamente istruzioni sugli elementi di qualunque **contenitore**, che è un oggetto che contiene o memorizza una raccolta di elementi. Secondo questa definizione, quindi, una stringa è un contenitore che memorizza la raccolta di caratteri che la caratterizza. Nei capitoli successivi vedremo, poi, altri tipi di contenitore disponibili in Python.

Il ciclo `for` può essere usato con la funzione `range` per usare come valori una sequenza di numeri interi.

Come avete visto nei paragrafi precedenti, l'uso di cicli a contatore che iterano su un intervallo di numeri interi è molto comune e, per semplificare la creazione di tali cicli, Python mette a disposizione la funzione `range`, che genera una sequenza di numeri interi utilizzabile come "contenitore" in un ciclo `for`.

Il ciclo seguente:

```
for i in range(1, 10) :  # i = 1, 2, 3, ..., 9
   print(i)
```

visualizza, in sequenza, i valori da 1 a 9. La funzione `range`, come si può vedere, genera una sequenza di valori basata sui propri argomenti: i valori vengono inseriti nella sequenza a partire dal primo argomento, fintanto che sono inferiori al secondo argomento. Il ciclo appena visto è equivalente al seguente ciclo `while`:

```
i = 1
while i < 10 :
   print(i)
   i = i + 1
```

Va ricordato che il valore finale (cioè il secondo argomento della funzione `range`) non viene incluso nella sequenza, per cui anche il ciclo `while` equivalente si ferma prima di raggiungere tale valore.

Sintassi 4.3 Enunciato `for` con la funzione `range`

Sintassi

```
for variabile in range(...) :
    enunciati
```

Esempi

All'inizio di ciascuna iterazione, a questa variabile viene assegnato il valore successivo della sequenza generata dalla funzione `range`.

La funzione `range` genera una sequenza di numeri interi che controlla le iterazioni del ciclo.

Con un solo argomento, la sequenza inizia da 0, con incrementi unitari, e l'argomento è il primo valore che NON appartiene alla sequenza.

```
for i in range(5) :
   print(i)  # Visualizza 0, 1, 2, 3, 4
```

Con due argomenti, la sequenza inizia dal primo argomento.

```
for i in range(1, 5) :
   print(i)  # Visualizza 1, 2, 3, 4
```

Con tre argomenti, il terzo argomento è il valore dell'incremento.

```
for i in range(1, 11, 2) :
   print(i)  # Visualizza 1, 3, 5, 7, 9
```

Se non vengono specificati altri parametri, la funzione `range` genera una sequenza con incrementi unitari tra un elemento e l'altro, ma questo comportamento standard può essere modificato aggiungendo, come terzo argomento della funzione, un valore di incremento diverso da 1:

```
for i in range(1, 10, 2) :  # i = 1, 3, 5, ..., 9
   print(i)
```

In questo caso verranno visualizzati soltanto i numeri dispari compresi tra 1 e 9. Possiamo anche contare all'indietro, usando un incremento negativo:

```
for i in range(10, 0, -1) :  # i = 10, 9, 8, ..., 1
   print(i)
```

Infine, si può usare la funzione `range` con un solo argomento: in tal caso i valori partono da zero.

```
for i in range(10) :  # i = 0, 1, 2, ..., 9
   print("Hello")  # Visualizza dieci volte Hello
```

In questa forma, la sequenza di valori va da 0 a un'unità in meno dell'argomento, con incrementi unitari: è molto utile quando dobbiamo semplicemente eseguire il corpo di un ciclo un determinato numero di volte, come nell'esempio precedente. Nella Tabella 2 si possono trovare altri esempi.

Tabella 2
Esempi di cicli for

Ciclo	Valori di i	Commento
`for i in range(6) :`	0, 1, 2, 3, 4, 5	Il ciclo viene eseguito 6 volte.
`for i in range(10, 16) :`	10, 11, 12, 13, 14, 15	Il valore finale non appartiene alla sequenza.
`for i in range(0, 9, 2) :`	0, 2, 4, 6, 8	Il terzo argomento è l'incremento tra un valore e il successivo.
`for i in range(5, 0, -1) :`	5, 4, 3, 2, 1	Per contare a ritroso si usa un incremento negativo.

Vediamo un tipico utilizzo del ciclo `for`. Vogliamo visualizzare il saldo nel nostro conto bancario di risparmio per un certo numero di anni, come in questa tabella:

Anno	Saldo
1	10 500.00
2	11 025.00
3	11 576.25
4	12 155.06
5	12 762.82

Lo schema del ciclo `for` è decisamente adatto allo scopo, dal momento che la variabile `year` parte dal valore 1 e aumenta con incrementi costanti (unitari) fino al raggiungimento del suo valore finale:

```
for year in range(1, numYears + 1) :
   Aggiorna balance.
   Visualizza year e balance.
```

La Figura 4 mostra il diagramma di flusso corrispondente e, nel seguito, viene presentato il programma completo.

File ch04/sec06/investment.py

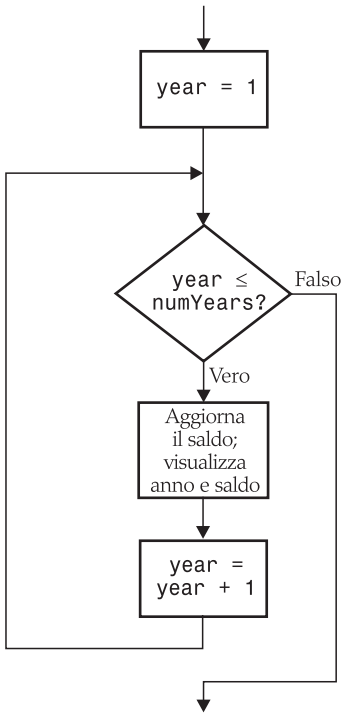
```
##
#  Questo programma visualizza una tabella che mostra
#  la crescita di un investimento.

# Definisce le costanti.
RATE = 5.0
INITIAL_BALANCE = 10000.0

# Legge il numero di anni per cui eseguire la simulazione.
numYears = int(input("Enter number of years: "))

# Visualizza la tabella con il saldo di ciascun anno.
balance = INITIAL_BALANCE
for year in range(1, numYears + 1) :
   interest = balance * RATE / 100
   balance = balance + interest
   print("%4d %10.2f" % (year, balance))
```

Figura 4
Diagramma di flusso di un ciclo for



Esecuzione del programma

```
Enter number of years: 10
   1  10500.00
   2  11025.00
   3  11576.25
   4  12155.06
   5  12762.82
   6  13400.96
   7  14071.00
   8  14774.55
   9  15513.28
  10  16288.95
```

Suggerimenti per la programmazione 4.1

Contare le iterazioni

Individuare il limite inferiore e superiore di un ciclo può non essere banale. Bisogna iniziare da zero o da uno? Come condizione di terminazione, è meglio usare `<= b` oppure `< b` ?

Per comprendere meglio un ciclo è molto utile contare il numero di iterazioni, operazione che risulta essere più agevole per i cicli con limiti asimmetrici. Questo ciclo viene eseguito `b - a` volte:

```
i = a
while i < b :
   . . .
   i = i + 1
```

e la stessa proprietà è, ovviamente, valida per il ciclo `for` equivalente:

```
for i in range (a, b) :
```

Questi limiti asimmetrici sono particolarmente utili per esaminare tutti i caratteri di una stringa. Il ciclo seguente, infatti, viene eseguito `len(string)` volte e la variabile `i` assume il valore di tutte le posizioni valide nella stringa, da `0` a `len(string) - 1` (è proprio perché questi cicli sono così comuni che viene consentito di omettere lo zero iniziale nell'invocazione della funzione `range`):

```
for i in range (0, len(string)) :
   . . . # elabora i e/o string[i]
```

Il ciclo seguente, che ha limiti simmetrici, viene eseguito `b - a + 1` volte:

```
i = a
while i <= b :
   . . .
   i = i + 1
```

Questo "+1" aggiuntivo è fonte di molti errori di programmazione. Per esempio, quando `a` vale 10 e `b` vale 20, `i` assume i valori 10, 11, 12, 13, 14, 15, 16, 17, 18, 19 e 20: sono 11 valori, cioè 20 – 10 + 1.

Un metodo per scoprire l'errore dovuto a questo "+1" è quello di immaginare i pali verticali e le sezioni orizzontali di uno steccato. Ciascuna sezione ha un palo alla propria sinistra, con un ulteriore palo finale posto a destra dell'ultima sezione. Quando ci si dimentica di contare l'ultima iterazione di un ciclo a limiti simmetrici si parla di "errore di un palo dello steccato" (*nota del traduttore*: negli Stati Uniti).

In un ciclo `for` di Python, il "+1" è, invece, piuttosto evidente:

```
for year in range (1, numYears + 1) :
```

Occorre, infatti, specificare esplicitamente un limite superiore che sia di un'unità maggiore dell'ultimo valore da includere nell'intervallo.

Consigli pratici 4.1

Scrivere un ciclo

In questa sezione analizzeremo, passo dopo passo, la procedura da seguire per realizzare un ciclo.

Descrizione del problema. Acquisire dodici valori di temperatura (uno per ciascun mese dell'anno) e visualizzare il numero corrispondente al mese che ha la temperatura più alta. Ad esempio, in base ai dati riportati nel sito `http://worldclimate.com`, le medie mensili della temperatura massima giornaliera nella Death Valley (la "Valle della Morte") sono (in gradi Celsius e in ordine da gennaio a dicembre):

18.2 22.6 26.4 31.1 36.6 42.2 45.7 44.5 40.2 33.1 24.2 17.6

In questo caso, il mese con la temperatura più elevata (45.7 gradi Celsius) è luglio e il programma dovrebbe visualizzare il numero 7.

Fase 1 Decidere ciò che va fatto *all'interno* del ciclo.

Ogni ciclo viene progettato per compiere alcune azioni ripetutamente, ad esempio:

- Acquisire un altro dato.
- Aggiornare un valore (come un saldo bancario o una somma).
- Incrementare un contatore.

Se non siete in grado di capire cosa debba essere fatto all'interno del ciclo, iniziate scrivendo i compiti che dovreste svolgere se doveste risolvere il problema a mano. Ad esempio, nel caso dell'individuazione del mese avente la temperatura media più elevata, potreste scrivere:

```
Leggi il primo valore.
Imposta a 1 il mese con la temperatura massima.
Leggi il secondo valore.
Se il secondo valore è maggiore del primo
   Imposta la temperatura massima a tale secondo valore.
   Imposta a 2 il mese con la temperatura massima.
```

```
Leggi il terzo valore.
Se tale valore è maggiore del primo e del secondo
   Imposta la temperatura massima a tale terzo valore.
   Imposta a 3 il mese con la temperatura massima.
Leggi il quarto valore.
Se tale valore è maggiore della temperatura massima vista finora
   …
```

Ora esaminate attentamente ciò che avete scritto e cercate di individuare un insieme *uniforme* di azioni, che possano essere inserite nel corpo del ciclo. La prima azione è facile:

```
Leggi il primo valore.
Imposta a 1 il mese con la temperatura massima.
```

L'individuazione dell'azione successiva richiede un po' di astuzia. Nella descrizione abbiamo usato frasi come "maggiore del primo", "maggiore del primo e del secondo" e "maggiore della temperatura massima vista finora", ma dobbiamo riuscire a usare un'unica frase che funzioni in tutti i casi. È evidente che l'ultima frase, quella più generale, è quella che ci serve.

Analogamente, dobbiamo trovare un modo generale per assegnare un valore al "mese con la temperatura massima". Ci serve una variabile che tenga traccia del mese attualmente in esame, che assuma i valori da 1 a 12 ordinatamente. Possiamo così esprimere la seconda azione del ciclo:

```
Se tale valore è maggiore della temperatura massima vista finora
   Imposta la temperatura massima a tale valore.
   Imposta al mese attuale il mese con la temperatura massima.
```

Mettendo tutto insieme, il nostro ciclo diventa:

```
Ripeti
   Leggi il valore successivo.
   Se tale valore è maggiore della temperatura massima vista finora
      Imposta la temperatura massima a tale valore.
      Imposta al mese attuale il mese con la temperatura massima.
   Incrementa di un'unità il mese attuale.
```

Fase 2 Specificare la condizione che controlla il ciclo.

Quale obiettivo deve raggiungere il nostro ciclo? Ecco alcuni esempi tipici:

- C'è un contatore che ha raggiunto il proprio valore finale?
- È stato acquisito l'ultimo valore in ingresso?
- C'è un valore che ha raggiunto una determinata soglia?

Nel nostro esempio vogliamo semplicemente che il valore del mese attuale arrivi a 12.

Fase 3 Determinare di che tipo deve essere il ciclo.

Conosciamo due principali categorie di cicli. Un ciclo *controllato da contatore* viene eseguito un numero determinato di volte, mentre in un ciclo *controllato da evento* il numero di iterazioni non è noto a priori: il ciclo viene eseguito finché non accade un determinato evento.

I cicli controllati da contatore vengono solitamente realizzati con un enunciato `for`, che può ripetere il proprio corpo per ciascun elemento di un contenitore, come una stringa, oppure per ciascun numero intero appartenente a una sequenza, spesso generata con la funzione `range`.

I cicli controllati da evento, invece, vengono realizzati con enunciati `while`, con un'opportuna condizione che ne provochi la terminazione. A volte la condizione di terminazione di un ciclo cambia valore in un punto intermedio del corpo del ciclo stesso. In tal caso si può usare una variabile booleana (spesso detta **flag**, *bandiera*, perché ha solo due "posizioni" diverse, "issata oppure no") che specifichi quando è giunto il momento di uscire dal ciclo, secondo questo schema:

```
done = False
while not done :
    Fa qualcosa.
    Se tutto il lavoro è stato fatto:
        done = True
    else :
        Fa qualcos'altro.
```

Riassumendo:

- Se avete bisogno di fare un'iterazione per ciascun elemento di un contenitore, indipendentemente dalla sua posizione, pianificate di usare un ciclo `for`.
- Se dovete fare un'iterazione per ogni numero intero appartenente a un intervallo, usate un ciclo `for` con la funzione `range`.
- Altrimenti, usate un ciclo `while`.

Nel nostro esempio leggiamo 12 valori di temperatura, quindi scegliamo un ciclo `for` che usi la funzione `range` per ripetere il proprio corpo per ciascun valore appartenente a una sequenza di numeri interi, da 1 a 12, estremi inclusi.

Fase 4 Inizializzare le variabili con i valori che servono al primo ingresso nel ciclo.

Elencate tutte le variabili che dovete usare e aggiornare all'interno del ciclo e determinate come vadano inizializzate. I contatori vengono solitamente inizializzati a 0 o a 1, mentre le somme vanno inizializzate a 0.

Nel nostro esempio, le variabili sono:

```
mese attuale
temperatura massima
mese con la temperatura massima
```

Dobbiamo fare attenzione al valore iniziale per la variabile "temperatura massima": non possiamo semplicemente usare il valore zero, perché il programma deve poter funzionare

anche con i valori di temperatura tipici dell'Antartide, che possono anche essere tutti negativi.

Una buona idea: come valore iniziale della temperatura usiamo il primo valore acquisito in ingresso. Facendo così dobbiamo poi, ovviamente, ricordarci che ci rimangono da leggere soltanto 11 valori, con il "mese attuale" che inizia da 2.

Conseguentemente, dobbiamo inizializzare a 1 la variabile "mese con la temperatura massima" (in fin dei conti, in una città australiana difficilmente troveremo un mese che sia più caldo di gennaio).

Fase 5 Elaborare il risultato dopo la terminazione del ciclo.

In molti casi il risultato cercato sarà semplicemente una delle variabili che vengono aggiornate nel corpo del ciclo (ad esempio, nel nostro programma, il risultato è il valore finale della variabile "mese con la temperatura massima"), ma a volte il ciclo calcola valori che contribuiscono al calcolo del risultato. Supponiamo, ad esempio, che ci venga chiesto di calcolare il valore medio di una sequenza di valori: il ciclo calcolerà la somma, non il valore medio, ma dopo la sua terminazione avremo a disposizione tutti i dati che servono per calcolarlo, dividendo la somma dei valori per il loro numero.

Ecco, infine, il ciclo completo:

```
Leggi il primo valore e memorizzalo come temperatura massima.
mese con la temperatura massima = 1
Per mese attuale che va da 2 a 12
   Leggi il valore successivo.
   Se tale valore è maggiore della temperatura massima vista finora
      Imposta la temperatura massima a tale valore.
      Imposta al mese attuale il mese con la temperatura massima.
```

Fase 6 Eseguire il ciclo a mano in alcuni casi tipici.

Eseguite a mano il vostro codice, passo dopo passo, come visto nel Paragrafo 4.2. Scegliete alcuni valori di prova in modo che la loro elaborazione non sia troppo complessa: l'esecuzione del corpo del ciclo quattro o cinque volte è solitamente sufficiente per individuare la maggior parte degli errori più frequenti. Fate attenzione, in modo particolare, al momento in cui si entra nel ciclo per la prima e per l'ultima volta.

A volte può essere ragionevole fare una piccola modifica per agevolare l'esecuzione manuale. Ad esempio, seguendo l'esecuzione del codice del programma che raddoppia l'investimento, può essere più comodo usare un tasso di interesse del 20%, invece del 5% previsto nel codice: il ciclo sarà più breve. Allo stesso modo, nel seguire a mano l'esecuzione del codice che analizza le temperature, usate 4 valori, invece di 12.

Supponiamo che i dati siano 22.6, 36.6, 44.5 e 24.2. La traccia dell'esecuzione, in questo caso, è la seguente:

mese attuale	valore acquisito	mese con temperatura massima	temperatura massima
		~~1~~	~~22.6~~
~~2~~	~~36.6~~	~~2~~	~~36.6~~
~~3~~	~~44.5~~	3	44.5
4	24.2		

Come si può vedere, al termine dell'esecuzione le variabili "mese con temperatura massima" e "temperatura massima" hanno i valori corretti.

Fase 7 Realizzare il ciclo in Python.

Riportiamo qui il ciclo relativo al nostro esempio, scritto in Python. L'Esercizio P4.4, al termine del capitolo, vi chiederà di completare il programma.

```
highestValue = float(input("Enter a value: "))
highestMonth = 1
for currentMonth in range (2, 13) :
   nextValue = float(input("Enter a value: "))
   if nextValue > highestValue :
      highestValue = nextValue
      highestMonth = currentMonth

print(highestMonth)
```

4.7 Cicli annidati

Quando il corpo di un ciclo contiene un altro ciclo, i due cicli si dicono annidati. La visualizzazione di una tabella, con righe e colonne, è un tipico esempio di utilizzo di cicli annidati.

Nel Paragrafo 3.3 avete visto come si possano annidare due enunciati `if`, uno dentro l'altro. In modo del tutto analogo, a volte le iterazioni più complesse richiedono un **ciclo annidato** (*nested loop*). Ad esempio, nell'elaborazione di una tabella l'esigenza di cicli annidati sorge quasi spontanea: un ciclo esterno esamina tutte le righe della tabella, mentre un ciclo interno elabora le colonne della riga in esame.

In questo paragrafo vedrete come visualizzare una tabella. Per semplicità, visualizzeremo le potenze di x, in questo modo:

x^1	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
...	...	...	...
10	100	1000	10 000

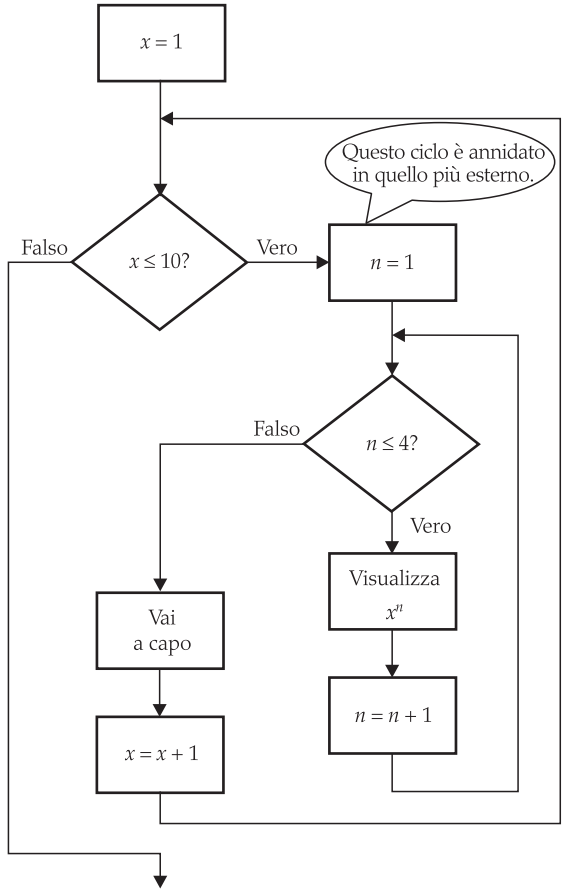
usando l'algoritmo descritto da questo pseudocodice:

```
Visualizza l'intestazione della tabella.
Per x che va da 1 a 10
   Visualizza una riga della tabella.
   Vai a capo.
```

Come si visualizza una riga della tabella? Bisogna scrivere un valore per ciascuno degli esponenti considerati e questo richiede un secondo ciclo:

```
Per n che va da 1 a 4
   Visualizza x^n.
```

Figura 5
Diagramma di flusso di un ciclo annidato



Quest'ultimo ciclo deve essere inserito nel corpo del ciclo precedente: diciamo che questo ciclo più interno viene **annidato** (*nested*) dentro quello più esterno.

Nel ciclo esterno vengono visualizzate 10 righe e, per ciascun valore di x, il programma visualizza quattro colonne, usando il ciclo interno (come si può vedere nel diagramma di flusso della Figura 5): in totale, quindi, vengono visualizzati $10 \times 4 = 40$ valori.

In questo programma vogliamo mostrare cosa succede quando si usano più enunciati `print` per visualizzare una singola riga, risultato che si ottiene aggiungendo, come ultimo argomento di `print`, la clausola `end=""` (si veda la sezione Argomenti avanzati 4.1). Si noti che usiamo cicli anche per visualizzare l'intestazione della tabella, ma sono cicli semplici, non annidati.

File ch04/sec07/powertable.py

```
1  ##
2  #  Questo programma visualizza una tabella con le potenze di x.
3  #
4
5  # Inizializza le costanti per i valori massimi.
6  NMAX = 4
7  XMAX = 10
```

```
 8
 9  # Visualizza l'intestazione della tabella.
10  for n in range(1, NMAX + 1) :
11     print("%10d" % n, end="")
12
13  print()
14  for n in range(1, NMAX + 1) :
15     print("%10s" % "x ", end="")
16
17  print("\n", "    ", "-" * 35)
18
19  # Visualizza il corpo della tabella.
20  for x in range(1, XMAX + 1) :
21     # Visualizza la riga x della tabella.
22     for n in range(1, NMAX + 1) :
23        print("%10.0f" % x ** n, end="")
24
25     print()
```

Esecuzione del programma

```
   1         2         3         4
 x         x         x         x
-----------------------------------
   1         1         1         1
   2         4         8        16
   3         9        27        81
   4        16        64       256
   5        25       125       625
   6        36       216      1296
   7        49       343      2401
   8        64       512      4096
   9        81       729      6561
  10       100      1000     10000
```

Esempi completi 4.1

Il voto d'esame medio

Descrizione del problema. Capita spesso di dover acquisire valori ripetutamente, per poi elaborarli. Scrivete un programma che calcoli il voto d'esame medio per ciascuno studente di un corso (tutti gli studenti hanno lo stesso numero di voti).

Fase 1 Capire il problema.

Per calcolare il voto medio per uno studente dobbiamo acquisire tutti i suoi voti e sommarli. Questo si può fare con un ciclo, ma noi dobbiamo farlo per più studenti: il calcolo del voto medio per uno studente va ripetuto per ciascuno studente del corso. Ci serve, quindi, un ciclo annidato: il ciclo più interno elaborerà i voti di uno studente, mentre il ciclo più esterno ripeterà la procedura per ciascuno studente.

```
Chiedi e leggi il numero di voti.
Ripeti per ciascuno studente
   Elabora i voti dello studente.
   Visualizza il voto medio dello studente.
```

Tabella 3
Esempi di cicli annidati

Cicli annidati	Visualizzazione	Spiegazione

`for i in range(3) :`
`  for j in range(4) :`
`  print("*", end="")`
`  print()`
	`****`
	`****`
	`****`
		Visualizza 3 righe, ciascuna con 4 asterischi.

`for i in range(4) :`
`  for j in range(3) :`
`  print("*", end="")`
`  print()`
	`***`
	`***`
	`***`
	`***`
		Visualizza 4 righe, ciascuna con 3 asterischi.

`for i in range(4) :`
`  for j in range(i + 1) :`
`  print("*", end="")`
`  print()`
	`*`
	`**`
	`***`
	`****`
		Visualizza 4 righe, di lunghezza crescente: 1, 2, 3 e 4.

`for i in range(3) :`
`  for j in range(5) :`
`  if j % 2 == 1 :`
`  print("*", end="")`
`  else`
`  print("-", end="")`
`  print()`
	`-*-*-`
	`-*-*-`
	`-*-*-`
	`-*-*-`
		Visualizza trattini e asterischi alternati.

`for i in range(3) :`
`  for j in range(5) :`
`  if i % 2 == j % 2 :`
`  print("*", end="")`
`  else`
`  print(" ", end="")`
`  print()`
	`* * *`
	` * *`
	`* * *`
		Visualizza uno schema a scacchiera.

Fase 2 Calcolare il voto medio per uno studente.

Per leggere i voti e calcolarne la media si può usare l'algoritmo visto nel Paragrafo 4.5.1. In questo problema, però, vogliamo leggere un numero prefissato e costante di voti, invece di continuare a leggere fino a un valore sentinella. Dato che sappiamo quanti voti dobbiamo leggere, possiamo usare un ciclo `for` con la funzione `range`:

```
somma dei voti = 0
for i in range(1, numExams + 1) :
   Acquisisci il voto successivo.
   Aggiungi tale voto alla somma dei voti.
Calcola il voto medio.
Visualizza il voto medio.
```

Fase 3 Ripetere la procedura per ciascuno studente.

Dato che vogliamo calcolare il voto medio per più studenti, dobbiamo ripetere i passi della Fase 2 per ciascuno di loro. Non sapendo quanti sono gli studenti, useremo un ciclo `while` con un valore sentinella: che sentinella usiamo? Per semplicità, possiamo basarci su una semplice domanda con risposta sì/no. Dopo che l'utente ha inserito i voti di uno studente, chiediamo se vuole inserire quelli di un altro:

```
moreGrades = input("Enter exam grades for another student (Y/N)? ")
moreGrades = moreGrades.upper()
```

La condizione di terminazione del ciclo è una risposta negativa, mentre il ciclo viene eseguito di nuovo ogni volta che l'utente scrive `"Y"`.

Come condizione di controllo del ciclo usiamo, quindi, l'espressione `moreGrades == "Y"`, e inizializziamo la variabile di controllo `moreGrades` in modo che contenga la stringa `"Y"`. In questo modo il ciclo viene eseguito almeno una volta, così l'utente può fornire i voti del primo studente prima che gli venga chiesto se vuole proseguire oppure no.

```
moreGrades = "Y"
while moreGrades == "Y" :
   Acquisisci i voti di uno studente.
   Calcola il suo voto medio.
   moreGrades = input("Enter exam grades for another student (Y/N)? ")
   moreGrades = moreGrades.upper()
```

Fase 4 Realizzare la soluzione in Python.

Ecco il programma completo:

File ch04/worked_example_1/examaverages.py

```
##
#  Questo programma calcola il voto medio per più studenti.
#

# Leggi il numero di voti per ciascuno studente.
numExams = int(input("How many exam grades does each student have? "))

# Inizializza moreGrades.
moreGrades = "Y"

# Calcola voti medi finché l'utente non dice di smettere.
while moreGrades == "Y" :

   # Calcola il voto medio per uno studente.
   print("Enter the exam grades.")
   total = 0
   for i in range(1, numExams + 1) :
      score = int(input("Exam %d: " % i))   # Chiedi il voto di ogni esame.
      total = total + score

   average = total / numExams
   print("The average is %.2f" % average)

   # Chiedi all'utente se vuole calcolare il voto medio di un altro studente.
   moreGrades = input("Enter exam grades for another student (Y/N)? ")
   moreGrades = moreGrades.upper()
```