Laurea in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Iterazioni 1

Le seguenti slide sono tratte principalmente dal lavoro di Giorgio Fumera (Univ. degli Studi di Cagliari) e riadattate da Raffele Nicolussi (Sapienza) e Domenico Lembo (Sapienza). Alcune sono invece una rielaborazione di slide analoghe di Diego Calvanese (Univ. di Bolzano).

L'istruzione iterativa while

Ripetere una stessa sequenza di operazioni: esempio

Esempio: Leggi 5 interi, calcolane la somma e stampala.
Prima variante (non ottimale): 5 variabili, 5 istruzioni di lettura, 5 . . .

```
i1 = int(input("immetti un intero: "))
i2 = int(input("immetti un intero: "))
i3 = int(input("immetti un intero: "))
i4 = int(input("immetti un intero: "))
i5 = int(input("immetti un intero: "))
print(i1+i2+i3+i4+i5)
```

Variante migliore che utilizza solo 2 variabili:

```
somma = 0
i = int(input("immetti un intero: "))
somma = somma + i;
...                              # ripeti per 5 volte
i = int(input("immetti un intero: "))
somma = somma + i;
print(somma)
```

⇒ conviene però usare un'**istruzione iterativa**

Istruzione iterativa

Le **istruzioni iterative** permettono di ripetere determinate azioni più volte:

- un numero di volte fissato (il numero di iterazioni è noto prima di iniziare l'esecuzione del ciclo) ⇒ **iterazione (o ciclo) definita**

  Esempio:
  per 10 volte fai un giro del campo di corsa

- finchè una condizione rimane vera (il numero di iterazioni non è noto prima di iniziare l'esecuzione del ciclo, ma è legato al verificarsi di un evento che dipende a sua volta dalle operazioni eseguite nel corpo del ciclo.)
  ⇒ **iterazione (o ciclo) indefinita**

  Esempio:
  finchè non sei ancora sazio prendi una ciliegia dal piatto e mangiala

In Python, come in altri linguaggi, entrambe le forme di iterazione possono essere espressi attraverso l'istruzione **while**

Istruzione while: sintassi

La sintassi è simile a quella dell'istruzione condizionale:

```
while espr-cond :
    sequenza di istruzioni
```

- la parola-chiave `while` deve essere scritta **senza rientri**
- espr-cond è un'espressione condizionale **qualsiasi**
- sequenza di istruzioni consiste in una o più istruzioni **qualsiasi**, Tale sequenza è chiamata *corpo* del ciclo
- ciascuna di tali istruzioni deve essere scritta in una riga **distinta**, con un **rientro** di almeno un carattere; il rientro deve essere identico per **tutte** le istruzioni della sequenza

L'istruzione iterativa: semantica

Un'istruzione iterativa viene eseguita **ripetendo** ciclicamente i seguenti passi:

1. viene valutata espr-cond
2. Se espr-cond è vera, si esegue la sequenza di istruzioni, e si **ritorna** al punto1;
   se invece espr-cond è falsa l'esecuzione dell'istruzione iterativa termina, e si passa all'eventuale istruzione **successiva**

L'istruzione iterativa: esempi

Come per l'istruzione condizionale, si consiglia di scrivere i programmi mostrati negli esempi seguenti in una finestra dell'*editor* invece che nella *shell*.

L'istruzione iterativa mostrata nell'esempio in basso richiede la ripetizione di due istruzioni finché la condizione 1> 2 risulta vera; le due istruzioni consistono nello stampare sulla *shell* la stringa `Python` e nell'assegnare alla variabile `x` il valore 7.

```
while 1 > 2 :
   print("Python")
   x = 7
```

Dato che la condizione 1 > 2 è falsa, l'esecuzione dell'istruzione `while` termina subito dopo la verifica di tale condizione, **senza che le due istruzioni al suo interno vengano mai eseguite.**

L'istruzione iterativa: esempi

Un esempio simile al precedente, con una diversa espressione condizionale:

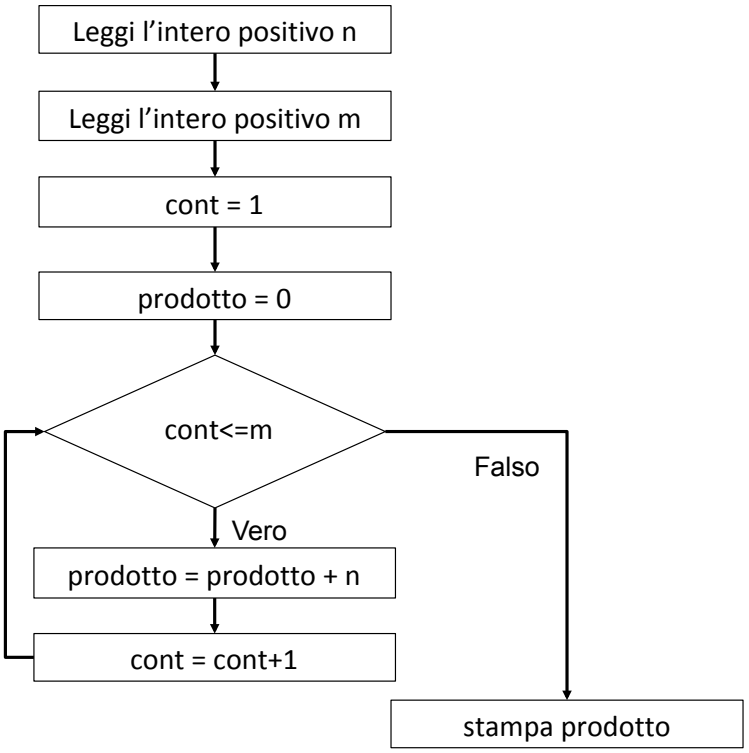
```
while 1 < 2 :
    print("Python")
    x = 7
```

Dato che la condizione risulta **sempre vera**, le due istruzioni all'interno dell'istruzione `while` vengono ripetute (in teoria) **infinite** volte. L'effetto visibile sarà la stampa continua della stringa `"Python"` sulla *shell*.

In casi come questo è possibile interrompere l'esecuzione di un programma scegliendo la voce `Interrupt Execution` dal *menu* `Shell` dell'ambiente `IDLE`, oppure premendo i tasti `CTRL + C`

Diagramma di flusso: esempio

Dati due interi non negativi *n* ed *m*, calcola *n*m* senza usare l'operatore * (moltiplicazione) e stampa il risultato



Moltiplicazione senza operatore *

```
# File prodottoInteri.py

n = int(input("Introduci un intero non negativo: "))
m = int(input("Introduci un intero non negativo: "))
cont = 1
prodotto = 0
while (cont <= m):
  prodotto = prodotto + n
  cont = cont + 1
print("La moltiplicazione di", n, "e", m, "è", prodotto)
```

Elementi caratteristici nella progettazione di un ciclo

iniziliazzazione
while (*condizione*) :
  operazione
  passo successivo

1. inizializzazione: definizione del valore delle variabili utilizzate nel ciclo prima dell'inizio dell'esecuzione del ciclo (cioè prima dell'istruzione di ciclo)
   Es. `prodotto = 0`
2. condizione: espressione valutata all'inizio di ogni iterazione, il cui valore di verità determina l'esecuzione del corpo del ciclo o la fine del ciclo
   Es. `cont <= m`
3. operazione del ciclo: calcolo del risultato parziale ad ogni iterazione del ciclo (nel corpo del ciclo)
   Es. `prodotto = prodotto + n`
4. passo successivo: operazione di incremento/decremento della variabile che controlla le ripetizioni del ciclo (nel corpo del ciclo)
   Es. `cont = cont + 1`

Errori comuni nella scrittura di cicli while

- Dimenticarsi di inizializzare una variabile che viene utilizzata nella condizione del ciclo (cioè la variabile è usata per la prima volta nell'espressione condizionale del ciclo, il che causa un errore durante l'esecuzione).
- Dimenticarsi di aggiornare le variabili che compaiono nella condizione del ciclo. Il ciclo non terminerà mai.
- Sbagliare di 1 il numero di iterazioni. Tipicamente è dovuto ad un errore nella condizione del ciclo o nell'inizializzazione di variabili usate nella condizione.

  Esempio:

```
i = 0
while i <= 10 :  #dovrebbe essere (i < 10)
  print("*")
  i = i + 1
```

L'istruzione iterativa: esempi

Per capire meglio il modo in cui il calcolatore esegue un'istruzione iterativa si suggerisce di provare a eseguirla passo dopo passo con carta e matita, tenendo traccia in ogni istante di:

- quale delle istruzioni della sequenza da ripetere sia in esecuzione
- quali siano i valori delle variabili
- che cosa viene stampato nella *shell*

Questo metodo può essere usato più in generale per comprendere l'esecuzione di un programma qualsiasi.

Potete anche provare l'esecuzione con Python Tutor (http://pythontutor.com/live.html#mode=edit)

L'istruzione iterativa: esempi

In questo esempio (disponibile nel *file* `buongiorno.py`) , nell'espressione condizione compare una variabile alla quale viene assegnato un valore **prima** dell'istruzione iterativa; il suo valore viene poi modificato **durante** l'esecuzione di tale istruzione.

In questo modo il valore della condizione può essere vero all'inizio della **prima** iterazione (*e di alcune delle iterazioni successive*), e può poi diventare falso, consentendo la conclusione dell'istruzione iterativa dopo un numero **finito** di ripetizioni.

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

L'istruzione iterativa: esempi

Di seguito si schematizza l'esecuzione del programma seguendo il metodo suggerito sopra:

- a sinistra si riporta il programma e si evidenzia (con il colore magenta) l'istruzione in esecuzione
- al centro si mostra il valore della variabile `k` (l'unica che compare nel programma) **dopo** l'esecuzione dell'istruzione evidenziata
- a destra si mostra (in blu) ciò che compare nella *shell*

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k

L'istruzione iterativa: esempi

Il programma è composto da **due istruzioni**: un'istruzione di assegnamento e un'istruzione iterativa (che a sua volta contiene altre istruzioni).

La prima istruzione che viene eseguita è l'assegnamento `k = 1`:

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 1

L'istruzione iterativa: esempi

Inizia quindi l'esecuzione dell'istruzione iterativa. Viene prima di tutto valutata l'espressione condizionale, che risulta vera; di conseguenza, verranno successivamente eseguite le istruzioni all'interno dell'istruzione iterativa.

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 1

L'istruzione iterativa: esempi

La prima di tali istruzioni causa la stampa della stringa `Buongiorno` nella *shell* :

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 1

Buongiorno

L'istruzione iterativa: esempi

La seconda istruzione modifica il valore della variabile `k`, incrementando di una unità il valore `attuale`:

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 2

Buongiorno

L'istruzione iterativa: esempi

L'istruzione `k = k + 1` appena eseguita è l'ultima della sequenza all'interno dell'istruzione `while` (come si vede dai rientri). L'esecuzione dell'istruzione `while` procede perciò ripartendo dalla valutazione dell'espressione condizionale, che risulta di nuovo vera. Le due istruzioni al suo interno verranno quindi ripetute una seconda volta.

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 2

```
Buongiorno
```

L'istruzione iterativa: esempi

Si esegue prima l'istruzione `print`, che stampa nella *shell* per la **seconda** volta la stringa `Buongiorno`:

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 2

```
Buongiorno
Buongiorno
```

L'istruzione iterativa: esempi

Si esegue poi l'istruzione di assegnamento, che incrementa il valore della variabile `k`:

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 3

```
Buongiorno
Buongiorno
```

L'istruzione iterativa: esempi

Si riparte quindi dalla valutazione dell'espressione condizionale dell'istruzione `while`, che è ancora vera: le istruzioni al suo interno verranno quindi ripetute per la **terza** volta.

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 3

```
Buongiorno
Buongiorno
```

L'istruzione iterativa: esempi

Il messaggio `Buongiorno` viene stampato per la terza volta nella *shell*...

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 3

```
Buongiorno
Buongiorno
Buongiorno
```

L'istruzione iterativa: esempi

... e il valore della variabile `k` viene incrementato per la terza volta, diventando ora 4:

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k 4

```
Buongiorno
Buongiorno
Buongiorno
```

L'istruzione iterativa: esempi

Si valuta quindi l'espressione condizionale, che questa volta risulta **falsa**: l'esecuzione dell'istruzione `while` a questo punto termina (senza che vengano di nuovo eseguite le istruzioni al suo interno), e non essendoci nessuna istruzione **successiva** (scritta cioè con lo stesso rientro della parola-chiave `while`) termina anche l'esecuzione del programma.

```
k = 1
while k <= 3 :
    print ("Buongiorno")
    k = k + 1
```

k | 4 |

```
Buongiorno
Buongiorno
Buongiorno
```

Ciclo controllato da "Contatori"

Nell'esempio precedente si verifica la situazione molto comune in cui il ciclo fa uso di una variabile che ad ogni iterazione varia di un valore costante, ed il cui valore determina la fine del ciclo (nel programma `buongiorno.py` si tratta della variabile `k`) .

Tale variabile è detta variabile di controllo o "contatore", perché ha la funzione di tenere traccia del (contare il) numero di iterazioni

E' un contatore anche la varibile `cont` dell'esempio precedente relativo alla moltiplicazione di due interi.

Prima dell'istruzione iterativa, al contatore è assegnato un valore scelto dal programmatore (di norma 1 o 0). Tale valore viene poi **incrementato** in ogni ripetizione dell'istruzione iterativa (tipicamente di **una** unità); l'espressione condizionale verificherà quindi che il valore della stessa variabile sia minore o uguale al numero desiderato di ripetizioni.

Cicli controllati in questo modo sono ciclo definiti

Ciclo controllato da contatori: esempio

Il programma seguente (disponibile nel *file* `buongiorno_2.py`) è simile a quello appena visto, ma questa volta **il numero di iterazioni dipende dal valore della variabile `n`, che viene acquisito attraverso la tastiera all'inizio dello stesso programma**. Il numero di iterazioni non è quindi noto nel momento in cui si **scrive** il programma, ma è comunque noto all'inizio del ciclo, perché dipende solo dai valori delle variabili che sono fissati prima di iniziare il ciclo. Si tratta quindi ancora di un ciclo definito, come nel caso precedente.

```
n = int(input("Inserire un numero: "))
k = 1
while k <= n :
    print ("Buongiorno")
    k = k + 1
```

Ciclo controllato da contatori: esempio

Questo programma (disponibile nel *file* `stampaInteriDecr.py`) acquisisce un numero intero e **stampa in ordine decrescente, a partire da questo, i numeri interi fino allo zero incluso**

```
n = int(input("Inserire un numero intero non negativo: "))
while n >= 0 :
    print (n)
    n = n - 1
```

`n` si comporta come un contatore che in questo caso "conta all'indietro" verso lo 0. Al tempo stesso è la variabile che usiamo per memorizzare il risultato della computazione (che in questo esempio coincide con il conteggio).

Si noti che se il valore acquisito è negativo non viene stampato nessun numero (in questo caso l'espressione condizionale del `while` risulta subito falsa).

Esercizio

Riscrivere il programma usando una variabile che funge solo da contatore

Soluzione

Riscrivere il programma usando una variabile che funge solo da contatore

```
# File stampaInteriDecr_2.py

N = int(input("Inserire un numero intero non negativo: "))
cont = 1
n = N
while cont <= N  :
  print(n)
  n = n - 1
  cont = cont + 1
```

L'istruzione iterativa: esercizi

Eseguire con Python Tutor (o con carta e matita) il programma seguente (disponibile nel *file* `prova_while_1.py`), assumendo che il valore acquisito attraverso l'espressione `input` sia 3, e determinare che cosa viene stampato nella *shell*.

```
n = int(input("Inserire un numero:"))
x = 0
k = 1
while k <= n :
    x = x + (1 / k)
    k = k + 1
print (x)
```

L'istruzione iterativa: esercizi

Eseguire con Python Tutor (o con carta e matita) il programma seguente (disponibile nel *file* `prova_while_2.py`), assumendo che il primo valore acquisito sia 3 e che i successivi siano 4, −3 e 6, e determinare che cosa viene stampato nella *shell*.

```
n = int(input("Inserire un numero:"))
a = 0
z = 1
while z <= n :
    x = int(input("Inserire un numero: "))
    a = a + x
    z = z + 1
print ("Il risultato e'", a)
```

L'istruzione iterativa: esercizi

Il programma mostrato di seguito (disponibile nel *file* `prova_while_3.py`) contiene **un'istruzione condizionale nidificata all'interno di un'istruzione iterativa**.

Osservando i rientri si deduce che l'istruzione iterativa contiene una sequenza di **tre** istruzioni: un assegnamento (`q = ...`), un'istruzione condizionale **senza** la parte `else`, e un altro assegnamento (`i = i + 1`).

L'istruzione condizionale contiene a sua volta **una sola istruzione** (l'assegnamento `b = b + 1`).

Si può infine osservare che l'istruzione `print` non fa parte dell'istruzione iterativa, e verrà quindi eseguita una sola volta **dopo** tale istruzione.

L'istruzione iterativa: esercizi

```
n = int(input("Inserire un numero:"))
b = 0
i = 1
while i <= n :
    q = int(input("Inserire un numero: "))
    if q > 0 :
        b = b + 1
    i = i + 1
print ("Risultato:", b)
```

Eseguire questo programma con carta e penna, assumendo che il primo valore acquisito sia 4 e che i successivi siano 2, −8, −3 e 5, e determinare che cosa viene stampato nella *shell*.

Soluzione

Inserire un numero:4
Inserire un numero: 2
Inserire un numero: -8
Inserire un numero: -3
Inserire un numero: 5
Risultato: 2

L'istruzione iterativa: esercizi

Quest'ultimo esercizio (il programma, mostrato di seguito, è disponibile nel *file* `prova_while_4.py`) contiene un **esempio di istruzione iterativa nidificata all'interno di un'altra istruzione iterativa**.

Si osservi che la prima istruzione iterativa (`while a < 4` ...) contiene una sequenza di **tre** istruzioni: un assegnamento (`b = 1`), un'istruzione iterativa nidificata (`while b < 3` ...) e un altro assegnamento (`a = a + 1`).

L'istruzione iterativa nidificata contiene a sua volta una sequenza composta da **due** istruzioni: un'istruzione `print` e un assegnamento (`b = b + 1`).

L'istruzione iterativa: esercizi

```
a = 1
while a < 4 :
    b = 1
    while b < 3 :
        print (a, b)
        b = b + 1
    a = a + 1
```

Eseguire questo programma con carta e penna, e determinare che cosa viene stampato nella *shell*.

Soluzione

1 1
1 2
2 1
2 2
3 1
3 2

Ciclo controllato da "sentinella"

- Ci sono cicli che processano iterativamente valori inseriti in input dall'utente
- In questi casi, si chiede di inserire in input un primo valore prima di entrare nel ciclo, e successivamente di richiede un inserimento ad ogni iterazione, in modo da processare una sequenza di valori
- L'uscita dal ciclo avviene all'inserimento di un particolare valore dato in input. Tale valore viene anche chiamato *sentinella*. La sentinella deve essere un valore non significativo per il problema che si sta risolvendo. Quindi NON deve essere un valore appartenente alla sequenza di valori che si sta processando.
- Cicli controllati in questo modo sono indefiniti

Ciclo controllato da "sentinella": esempio

Questo programma (disponibile nel *file* `stampaQuadrato.py`) **acquisisce una sequenza di numeri e stampa il quadrato di ciascuno di essi**, arrestandosi quando incontra un numero pari a zero; al termine viene stampato il messaggio `Fine`.

Lo zero gioca il ruolo di "sentinella", e si parla in questo caso di ciclo controllato da sentinella (la sentinella non deve essere un dato di input significativo)

Si noti che l'istruzione `print "Fine."` si trova **al di fuori** dell'istruzione `while` (poiché è scritta **senza rientri**), e quindi viene eseguita **una sola volta, dopo** che l'esecuzione dell'istruzione `while` è terminata.

```
n = int(input("Inserire un numero intero (0 per terminare): "))
while n != 0 :
    print ("Il suo quadrato e'", n ** 2)
    n = int(input("Inserire un altro numero intero: "))
print("Fine.")
```

Ciclo controllato da "sentinella": esempi

Questo programma (disponibile nel *file* `contaParole.py`) **conta le parole inserite in input**, arrestandosi quando l'utente quindi digita 'invio' senza inserire alcun carattere (inserisce una stringa vuota).

```
s = input("Inserire una parola (solo invio per terminare): ")
cont = 0
while s != "" :
    cont = cont+1
    s = input("Inserire un'altra parola: ")
print("il numero di parole inserite è", cont)
```

Schemi di ciclo

Esistono alcune operazioni di base molto comuni che richiedono l'utilizzo di cicli:

- conteggio: obiettivo del ciclo è contare il numero di valori in un insieme (da non confondersi con i cicli controllati da contatori, in cui il conteggio non è il fine ultimo del ciclo). L'esempio precedente è di questo tipo (anche se non è controllato da un contatore).
- accumulatore: accumula i valori di un insieme. Alla fine del ciclo la variabile usata come accumulatore contiene il valore finale che si intendeva calcolare attraverso il ciclo.
- valori caratteristici di un insieme: determina un valore caratteristico tra i valori in un insieme (ad esempio, il massimo, quando sui valori dell'insieme è definito un ordinamento)

Accumulatore: esempio

Questo programma (disponibile nel *file* `somma_1.py`) acquisisce attraverso la tastiera una sequenza di interi di lunghezza *qualsiasi* e ne calcola la **somma**. Il risultato viene calcolato memorizzando nella variabile `somma` il valore della somma **parziale** dei numeri della sequenza, durante la loro acquisizione. Per questo motivo il valore iniziale di tale variabile deve essere zero.

La sequenza di numeri termina quando l'utente digita 'invio' senza inserire alcun numero

```
somma = 0
s = input("Immetti un intero: ")
while s != "" :
   n = int(s)
   somma = somma + n
   s = input("Immetti un intero: ")
print("La somma e'", somma)
```

Accumulatore: esempio

Variante del programma precedente (disponibile nel *file* `somma_2.py`) nel quale la lunghezza della sequenza è un dato d'ingresso (quindi il ciclo è definito e si usa un contatore per controllarne la terminazione).

```
n = int(input("Quanti numeri si vogliono sommare? "))
somma = 0
k = 1
while k <= n :
    x = int(input("Immetti un intero: "))
    somma = x + somma
    k = k + 1
print("La somma e'", somma)
```

Esercizio

Prova a modificare l'esercizio precedente calcolando la somma dei soli valori pari

Soluzione

```
# File sommaPari.py

n = int(input("Quanti numeri si vogliono immettere? "))
somma = 0
k = 1
while k <= n :
    x = int(input("Prossimo valore: "))
    if (x % 2 == 0):
        somma = x + somma
    k = k + 1
print ("La somma e'", somma)
```

Esercizio

Modificare l'esercizio precedente per calcolare la media dei valori pari digitati dall'utente

Soluzione

```
# File media.py

n = int(input("Quanti numeri si vogliono immettere? "))
somma = 0
k = 1
while k <= n :
  x = int(input("Prossimo valore: "))
  somma = x + somma
  k = k + 1
print ("La media e'", somma/(k-1))
```

Accumulatore: esempio

Concatena le stringhe date in input che iniziano con ":" (solo invio per terminare).

```
# File concatenaStringhe.py

s = input("Immetti una stringa" + \
          (solo invio per terminare): ")
stot = ""
while s != "" :
  if (s[0] == ':') :
    stot = stot + s
  s = input("Immetti una stringa ")
print("stringa totale =", stot)
```

Valori caratteristici di un insieme: minimo

Il programma mostrato di seguito calcola **il valore più piccolo in una sequenza di numeri di lunghezza *qualsiasi*, acquisita attraverso la tastiera**. Anche la dimensione della sequenza è un dato d'ingresso, ed è il primo a essere acquisito. Il programma è disponibile nel *file* `minimo.py`

Il procedimento consiste nell'acquisire i valori della sequenza per mezzo di un'istruzione iterativa e nel tener traccia, istante per istante, del valore più piccolo tra quelli già acquisiti. Tale valore viene memorizzato nella variabile `minimo`. Il valore di tale variabile viene aggiornato ogniqualvolta se ne incontra uno più piccolo.

Non è difficile rendersi conto che questo algoritmo richiede che il valore iniziale della variabile `minimo` debba essere pari al **primo** valore della sequenza (oppure al valore $+\infty$, che però non può essere rappresentato in linguaggio Python).

Valori caratteristici di un insieme: minimo

```
n = int(input("Quanti sono gli elementi della sequenza? "))
x = int(input("Primo valore: "))
minimo = x
k = 2
while k <= n :
    x = int(input("Prossimo valore: "))
    if x < minimo :
        minimo = x
    k = k + 1
print ("Il valore piu' piccolo e'", minimo)
```

- Nel caso si cerchi il minimo in una sequenza di elementi ordinati di cui si conosce il valore massimo assoluto M, allora la variabile `minimo` può essere inizializzata ad M, e non è necessario trattare il primo valore in modo diverso dagli altri (ad esempio se stiamo cercando il più basso fra i voti conseguiti agli esami, possiamo porre `minimo = 30`).
- E' ovviamente possibile scrivere una variante di questo programma in cui la terminazione della sequenza viene decisa da input (ad esempio digitando solo invio), attraverso quindi un ciclo indefinito.