

Iterazioni e Cicli While

Informatica@DSS 2023/2024

Massimo Lauria <massimo.lauria@uniroma1.it>
<https://massimolauria.net/informatica2023/>

La lunghezza del codice è fissa

Un programma ha lunghezza **fissa**.

- ▶ scritto dal programmatore
- ▶ salvato su file

L'input ha lunghezza **variabile**

- ▶ e.g. sommare 1000 numeri
- ▶ e.g. sommare 1000000 numeri

Ripetizione di linee di codice

L'esecuzione di alcune righe di codice deve essere **ripetuta**.

In effetti negli esempi che abbiamo visto il corpo di una funzione viene ripetuto. *Non basta! Avveniva solo per un numero di volte fissato dal testo del programma.*

Cicli e Iterazioni

La ripetizione di righe di codice tale che ogni esecuzione

varia con lo stato del programma

e il numero di ripetizioni

dipende dallo stato per programma

Problema:

Voglio stampare i numeri da 1 a 5

```
print(1)      1
print(2)      2
print(3)      3
print(4)      4
print(5)      5
```

Ma se voglio stampare i numeri da 1 a 100? O 1000?

ciclo while

Sintassi e significato

```
while condizione:           1
    istruzione1             2
    istruzione2             3
    istruzione3             4
    istruzione4             5
    ...                     6
                            7
```

condizione è un'espressione booleana.

1. Se `condizione` è vera vai al punto 2, altrimenti al 3.
2. Esegui il blocco di istruzioni e dopo torna al punto 1.
3. Prosegui con le istruzioni successive al blocco `while`

Esempio: stampa i numeri da 1 a 10

```
i = 1                                1
while i <= 10:                       2
    print(i)                          3
    i = i + 1                         4
```

```
1
2
3
4
5
6
7
8
9
10
```


Modifica di una variabile

Il valore associato ad una variabile cambia **nel tempo**.

L'istruzione

```
x = x + 1
```

```
1
```

porta 'x' a valere 1 in più del valore precedente. Ma l'istruzione dà **errore** se 'x' non ha già un valore definito inizialmente.

Valore iniziale

x ha un valore iniziale

y no

```
x = 23 1
x = x + 1 2
print(x) 3
y = y + 1 4
        5
```

```
24
Traceback (most recent call last):
  File "<stdin>", line 5, in <module>
NameError: name 'y' is not defined
```

Esecuzione di un assegnamento

Un assegnamento ha la struttura `var = expr` dove

1. viene valutata `expr`
2. il valore viene assegnato a `var`

se `expr` essere valutata senza errori, allora l'assegnamento va a buon fine. `expr` può tranquillamente contenere `var`, se questa è stata definita in precedenza.

```
x = 3 1
x = 'ciao'+"k"*x 2
print(x) 3
```

```
ciaokkk
```

Operazioni tipiche (con operatori alternativi)

`x = x + k`

`x = x * k`

`x = x - k`

`x = x / k`

`x = x // k`

`x = x % k`

...

`x += k`

`x *= k`

`x -= k`

`x /= k`

`x //= k`

`x %= k`

...

Realizzazione di un contatore

Nell'esempio vedere che la variabile `i` viene usata come un contatore.

- ▶ il contatore viene **inizializzato** fuori dal ciclo
- ▶ il contatore viene **incrementato** ad ogni iterazione

```
i = 1 1
while i <= 10: 2
    print(i) 3
    i = i + 1 4
```

Ogni iterazione è diversa, perché dipende anche dal valore che ha la variabile `i`.

Programma di durata non prevedibile

Vediamo il primo programma la cui durata non è prevedibile a priori.

```
def contatore(N): 1
    i = 1 2
    while i <= N: 3
        print(i) 4
        i = i + 1 5
    6
N = int(input("Quante ripetizioni? ")) 7
contatore(N) 8
```

Attenzione ai cicli infiniti

Se la condizione nel `while` non viene mai falsificata, non si esce mai dal ciclo.

```
while True: 1  
    print('Aiuto! Non riesco ad uscire da qui.') 2
```

```
Aiuto! Non riesco ad uscire da qui.  
Aiuto! Non riesco ad uscire da qui.  
Aiuto! Non riesco ad uscire da qui.  
Aiuto! Non riesco ad uscire da qui.  
[...]
```

Attenzione ai cicli infiniti (2)

Non è così ovvio capire quando si ha a che fare con un ciclo infinito. Proviamo a modificare il contatore.

```
def contatore2(N):
    i = 1
    while i != N+1:
        print(i)
        i = i + 1
contatore2(5)
```

1
2
3
4
5
6
7

1
2
3
4
5

Domanda: che succede se eseguo `contatore(-3)` e `contatore2(-3)`?

Contatore alla rovescia

```
i = 10           1
while i > 0:     2
    print(i)    3
    i = i - 1  4
```

```
10
9
8
7
6
5
4
3
2
1
```

provate a cambiare:

- ▶ il 10 con 9 o con 11
- ▶ lo 0 con 1
- ▶ il > con >= o con <
- ▶ invertire le righe nel blocco

Altro ciclo infinito

```
1  
a = 5  
2  
while a != 0 :  
3  
    print('a ==',a)  
4  
    a = a - 1  
5
```

```
a == 5  
a == 4  
a == 3  
a == 2  
a == 1
```

```
1  
a = 5  
2  
while a != 0 :  
3  
    print('a ==', a)  
4  
    a = a - 2  
5
```

```
a == 5  
a == 3  
a == 1  
a == -1  
a == -3  
a == -5  
a == -7  
a == -9  
a == -11  
[...]
```

Esempio: quante piegature?

Un foglio è spesso 0.01mm. Quante volte va piegato a metà per raggiungere lo spessore di 500mm?

Potete risolvere il problema con un ciclo `while`

Esempio: quante piegature? (codice)

```
spessore = 0.01           1
piegature = 0             2
obiettivo = 500           3

                             4
while spessore < obiettivo: 5
    print("Lo spessore dopo "+str(piegature), end='') 6
    print(" piegature è "+str(spessore)+"mm") 7
    piegature = piegature + 1 8
    spessore = spessore * 2 9
                             10
print("Per arrivare a "+str(obiettivo)+"mm", end='') 11
print("ci sono volute "+str(piegature)+" piegature.") 12
```

Esempio: quante piegature? (output)

```
Lo spessore dopo 0 piegature è 0.01mm
Lo spessore dopo 1 piegature è 0.02mm
Lo spessore dopo 2 piegature è 0.04mm
Lo spessore dopo 3 piegature è 0.08mm
Lo spessore dopo 4 piegature è 0.16mm
Lo spessore dopo 5 piegature è 0.32mm
Lo spessore dopo 6 piegature è 0.64mm
Lo spessore dopo 7 piegature è 1.28mm
Lo spessore dopo 8 piegature è 2.56mm
Lo spessore dopo 9 piegature è 5.12mm
Lo spessore dopo 10 piegature è 10.24mm
Lo spessore dopo 11 piegature è 20.48mm
Lo spessore dopo 12 piegature è 40.96mm
Lo spessore dopo 13 piegature è 81.92mm
Lo spessore dopo 14 piegature è 163.84mm
Lo spessore dopo 15 piegature è 327.68mm
Per arrivare a 500mm ci sono volute 16 piegature.
```

E per raggiungere la distanza terra-luna, che è 380000km?

Un appunto su print

La funzione `print` porta sempre a capo dopo la stampa

```
print('cane') 1  
print('gatto') 2
```

```
cane  
gatto
```

Un appunto su print (2)

È possibile specificare cosa stampare a video alla fine della stampa (o anche non fargli scrivere nulla).

```
print('cane', end='****') 1
print('gatto',end='')      2
print('chiusura',end='\n') 3
print('nuovariga')         4
```

```
cane****gattochiusura
nuovariga
```

```
i = 0 1
while i < 10: 2
    print(i, end='') 3
    i = i + 1 4
```

```
0123456789
```

Altri esempi con il ciclo `while`

Somma dei primi N numeri

```
def somma(N): 1
    accumulatore = 0 # inizializzato 2
    i = 1 3
    while i <= N: 4
        accumulatore = accumulatore + i 5
        i = i + 1 6
    return accumulatore 7

print(somma(0)) 8
print(somma(10)) 9
print(somma(30)) 10
print(somma(50)) 11
print(somma(100)) 12
print(somma(1000)) 13
```

```
0
55
465
1275
```

Appunti sui numeri casuali

Nel modulo `random` la funzione `randint(a,b)` pesca a caso un intero x dove $a \leq x \leq b$.

- ▶ lancio di un dado: `random.randint(1,6)`
- ▶ lancio di una moneta: `random.randint(0,1)`

```
import random 1
i=1 2
while i <= 20: 3
    print(random.randint(1,6), end=' ') 4
    i = i + 1 5
```

```
5 4 3 3 4 5 3 2 3 2 2 1 4 4 1 4 6 5 1 6
```

Usate `help(random)` e `help(random.randint)`

Statistiche su lanci di moneta

Se lanciamo 1000 monete, quante teste e quante croci otteniamo?

```
import random 1
teste=0 2
lanci=1000 3
i=1 4
while i <= lanci: 5
    if random.randint(0,1) == 1: 6
        teste = teste+1 7
    i = i + 1 8
print("Teste:",teste) 9
print("Croci:",lanci-teste) 10
```

```
Teste: 507
Croci: 493
```

Calcolo della media

Media di 1000 numeri casuali tra 1 e 100

```
import random 1
somma=0 2
popolazione=100 # intervallo di scelta 3
ripetizioni=1000 # numero di esperimenti 4
i=1 5
while i <= ripetizioni: 6
    dato = random.randint(1,popolazione) 7
    somma = somma + dato 8
    i = i + 1 9
print("Media:",somma/ripetizioni) 10
```

Media: 50.534

Test di primalità

Un numero intero positivo $n > 1$ **non** è primo quando esiste un $k < n$ per cui k divide esattamente n .

Scriviamo una funzione che verifichi se un numero n è primo e in quel caso restituisca `True`.

Test di primalità (codice)

```
def primo(n): 1
    k = 2 2
    if n < 2: 3
        return False 4
    while k < n: 5
        if n % k == 0: 6
            return False 7
        k = k + 1 8
    return True 9

print("Il numero 1 è primo?" , primo(1) ) 10
print("Il numero 10 è primo?", primo(10) ) 11
print("Il numero 13 è primo?", primo(13) ) 12
print("Il numero 15 è primo?", primo(15) ) 13
print("Il numero 15 è primo?", primo(15) ) 14
```

```
Il numero 1 è primo? False
Il numero 10 è primo? False
Il numero 13 è primo? True
Il numero 15 è primo? False
```

Test di primalità (osservazioni)

Osservate che il ciclo `while` si interrompe prematuramente: una volta trovato un divisore non è necessario testare gli altri casi.

Abbiamo testato tutti i numeri da 2 a $n - 1$ per cercare un divisore. Potevamo fermarci prima? Ad esempio $n/2$? E prima?

Cicli annidati

È possibile annidare cicli `while`, ma ovviamente potrebbero servire più contatori.

```
riga = 1                                     1
while riga <= 10:                             2
    colonna = 1                               # reinizializzato ad ogni riga 3
    while colonna <= 10:                       4
        print('*',end='')                     5
        colonna = colonna + 1                 6
    riga = riga + 1                             7
    print('')                                  # non stampa nulla ma va a capo 8
```

```
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
```


Cicli annidati (variazione)

```
riga = 1 1
while riga <= 10: 2
    colonna = 1 3
    while colonna <= riga: # variazione 4
        print('*',end='') 5
        colonna = colonna + 1 6
    riga = riga + 1 7
    print('') 8
```

```
*
**
***
****
*****
*****
*****
*****
*****
*****
*****
*****
```

Cicli annidati (variazione 2)

```
riga = 1 1
while riga <= 10: 2
    colonna = 1 3
    while colonna <= 10: 4
        if (riga+colonna) % 2 ==0: 5
            print('*',end='') 6
        else: 7
            print(' ',end='') 8
        colonna = colonna + 1 9
    riga = riga + 1 10
print('') 11
```

```
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
```

Cicli annidati (esercizio)

Scrivere un programma che stampi la schermata seguente, senza utilizzare gli operatori di ripetizione di stringa (ad esempio non potete usare la moltiplicazione tra stringhe e interi).

```
      *
     **
    ***
   ****
  *****
 *****
*****
*****
*****
*****
*****
```

Tavola pitagorica

Scriviamo una funzione `tavolapitagorica(N)` che stampa una griglia di numeri, dove alla riga r e alla colonna c della griglia ci mette il valore $r \times c$. Ad esempio:

```
1  2  3  4  5
2  4  6  8 10
3  6  9 12 15
4  8 12 16 20
5 10 15 20 25
```

Un primo tentativo

```
def tavolapitagorica(N): 1
    R = 1 2
    while R <= N: 3
        C = 1 4
        while C <= N: 5
            print(R*C, end='') 6
            C = C + 1 7
        R = R + 1 8
        print() 9
    10
tavolapitagorica(5) 11
```

```
12345
246810
3691215
48121620
510152025
```

Un secondo tentativo

```
def tavolapitagorica(N):           1
    R = 1                          2
    while R <= N:                  3
        C = 1                      4
        while C <= N:             5
            print(R*C, end=' ')    6
            C = C + 1             7
        R = R + 1                 8
        print()                   9
    10
tavolapitagorica(5)              11
```

```
1 2 3 4 5
2 4 6 8 10
3 6 9 12 15
4 8 12 16 20
5 10 15 20 25
```

Un terzo tentativo

Stampiamo ogni numero incolonnando in 4 spazi, allineando a destra.

```
def allinea(v): 1
    "Allinea il numero a destra, su colonne di 4 caratteri." 2
    testo = str(v) 3
    pad = 4-len(testo) 4
    if pad>0: 5
        testo = ' '*pad + testo 6
    return testo 7
8
print(allinea(5)) 9
print(allinea(215)) 10
print(allinea(-15)) 11
print(allinea(12515)) 12
print(allinea(7515)) 13
```

```
    5
  215
  -15
12515
7515
```

Un terzo tentativo (2)

```
def tavolapitagorica(N): 1
    R = 1 2
    while R <= N: 3
        C = 1 4
        while C <= N: 5
            print(allinea(R*C), end='') 6
            C = C + 1 7
        R = R + 1 8
        print() 9
    10
tavolapitagorica(12) 11
```


Un terzo tentativo (output)

| | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|-----|-----|-----|-----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 | 22 | 24 |
| 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 | 33 | 36 |
| 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 44 | 48 |
| 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 | 55 | 60 |
| 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 | 66 | 72 |
| 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 | 77 | 84 |
| 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 | 88 | 96 |
| 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 | 99 | 108 |
| 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 | 110 | 120 |
| 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99 | 110 | 121 | 132 |
| 12 | 24 | 36 | 48 | 60 | 72 | 84 | 96 | 108 | 120 | 132 | 144 |