# Iterazioni su intervalli o su condizioni booleane

## Informatica@SEFA 2018/2019 - Lezione 10

Massimo Lauria <massimo.lauria@uniroma1.it>
http://massimolauria.net/courses/infosefa2018/

Venerdì, 19 Ottobre 2018

# Iterazioni su sequenze

# Iterazioni su sequenze: ciclo `for`

```
for variabile in sequenza:
    istruzione1
    istruzione2
    ...
```

Il blocco di istruzioni viene ripetuta per ogni elemento nella sequenza. Di volta in volta `variabile` assume il valore dell'elemento visitato in quel momento.

# Esempio

```
colori = ['blu', 'rosso',
          'verde', 'giallo']

testo  = 'ammaccabanane'

for x in colori:
    print(x)

for x in testo:
    print(x)
```

```
blu
rosso
verde
giallo
a
m
m
a
c
c
a
b
a
n
a
n
e
```

# Esempio: somma di numeri

```
def somma_numeri(seq):                                          1
    accumulatore = 0      # inizializzare a 0                  2
    for v in seq:                                               3
        accumulatore = accumulatore + v                         4
                                                                5
    return accumulatore                                         6
                                                                7
print( somma_numeri([3,1,5,-2]) )                               8
print( somma_numeri( [-3, -10, 0, 7, 10, -5, 1, -10,            9
                      6, -9, 4, -7, -10, 1, 6, 8, 7,            10
                      9, -8, 3, 6, -2, -7, 0, -2, 10,           11
                      -5, -8, 9, -2]))                          12
```

```
7
-1
```

# Esempio: produrre una nuova lista

```
def maiuscole(seq):                                          1
    accumulatore = []                                        2
    for s in seq:                                            3
        accumulatore.append( s.upper() )                     4
    return accumulatore                                      5
                                                             6
print( maiuscole(['verde','azzurro','giallo']) )             7
```

```
['VERDE', 'AZZURRO', 'GIALLO']
```

# Esercizio: calcolare il minimo (I)

Calcolare il minimo di una lista.

- in una lista vuota il minimo non è definito
- in python esiste `min`
- realizziamola noi per esercizio

# Esercizio: calcolare il minimo (II)

```
def minimo(seq):                                                    1
    if len(seq)==0:                                                 2
        raise ValueError('Minimo non definito')                     3
                                                                    4
    temp_min = seq[0]                                               5
    for v in seq:                                                   6
        if temp_min > v:                                            7
            temp_min = v                                            8
                                                                    9
    return temp_min                                                 10
                                                                    11
print( minimo( [3,1,5,-2] ))                                        12
print( minimo( ('verde','azzurro','giallo')  ))                     13
```

```
-2
azzurro
```

# Esercizio: concatena

Concatenare due liste: ad esempio come

```
[1,2,3] + [4,5,6,7]
```

```
def concatena(lista1,lista2):

    risultato = []
    for x in lista1:
        risultato.append(x)

    for x in lista2:
        risultato.append(x)

    return risultato

print( concatena([1,2,3],[4,5,6,7]))
```

```
[1, 2, 3, 4, 5, 6, 7]
```

# Sequenze di interi e intervalli

# Oggetti di tipo `range`

```
print(type(range(3)))

for i in range(3):
    print(i)

print(type(range(2,5)))

for i in range(2,5):
    print(i)
```

```
<class 'range'>
0
1
2
<class 'range'>
2
3
4
```

# Intervalli di interi

`range(n)`

`range(L,H)`

producono rispettivamente

- la sequenza di numeri da $0$ a $n-1$.
- la sequenza di numeri da `L` a `H`-1.

(essenzialmente `range(n)` e uguale a `range(0,n)`)

# Intervalli di interi (II)

```
X = range(5)

print(len(X))
print(X)
print(list(X))

print(X[0])
print(X[1])
print(X[3])
print(X[4])
```

```
5
range(0, 5)
[0, 1, 2, 3, 4]
0
1
3
4
```

```
Y = range(3,12)

print(len(Y))
print(Y)
print(list(Y))

print(Y[0])
print(Y[1])
print(Y[7])
print(Y[8])
```

```
9
range(3, 12)
[3, 4, 5, 6, 7, 8, 9, 10, 11]
3
4
10
11
```

# Intervalli di interi (III)

```
print( list(range(-5,3)) )
print( list(range(1,-1)) )
print( list(range(1,1)) )
```

```
[-5, -4, -3, -2, -1, 0, 1, 2]
[]
[]
```

# Scansione di una sequenza con `range`

```
coordinate = [(0,2), (-2,4), (3,5), (-1,0) ]
print("Ciclo sui valori")
for punto in coordinate:
    print( punto )

print("Ciclo sugli indici")
for i in range(len(coordinate)):
    print( coordinate[i] )
```

```
Ciclo sui valori
(0, 2)
(-2, 4)
(3, 5)
(-1, 0)
Ciclo sugli indici
(0, 2)
(-2, 4)
(3, 5)
(-1, 0)
```

# I primi N cubi

```
def primi_cubi(N):                                    1
    L=[]                                              2
    for i in range(1,N+1):                            3
        L.append(i**3)                                4
    return L                                          5
                                                      6
print( primi_cubi(12) )                               7
```

```
[1, 8, 27, 64, 125, 216, 343, 512, 729, 1000, 1331, 1728]
```

# Variante del calcolo del minimo

Nel caso precedente analizzavamo due volte la posizione 0.

```
def minimo(seq):                                                    1
    if len(seq)==0:                                                 2
        raise ValueError('Minimo non definito')                     3
                                                                    4
    temp_min = seq[0]                                               5
    for i in range(1,len(seq)):                                     6
        if temp_min > seq[i]:                                       7
            temp_min = seq[i]                                       8
                                                                    9
    return temp_min                                                 10
                                                                    11
print( minimo( [3,1,5,-2] ))                                        12
print( minimo( ('verde','azzurro','giallo')  ))                     13
```

```
-2
azzurro
```

# Esercizio: mix di liste

Scrivere una funzione `mix` che mischi due liste di lunghezza uguale alternandone gli elementi. Per esempio

```
mix(['a','b','c'],[-3,17,12])
```

deve restituire

```
['a', -3, 'b', 17, 'c', 12]
```

# Fattoriale di $n$

Denotato come $n!$, è il prodotto nei numeri da 1 a $n$.

```
def fatt(N):
    if N<0:
        raise ValueError("Il fattoriale è definito solo su
    numeri non negativi")

    R=1
    for i in range(1,N+1):
       R = R * i
    return R

print(fatt(0))
print(fatt(3))
print(fatt(5))
```

```
1
6
120
```

# Cicli annidati

È possibile annidare cicli `for` naturalmente. (È possibile annidare qualunque tipo di blocchi di istruzioni)

```
for i in range(10):                                                    1
    for j in range(10):                                                2
                                                                       3
        print('*',end='')     # end='' non fa andare a capo          4
                                                                       5
    print('')                 # non stampa nulla ma va a capo        6
```

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

# Cicli annidati (variazione)

```
for i in range(10):
    for j in range(i+1):

        print('*',end='')

    print('')
```

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

# Cicli annidati (variazione 2)

```
for i in range(10):
    for j in range(10-i-1):
        print(' ',end='')

    for j in range(10-i-1,10):
        print('*',end='')

    print('')
```

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

# Cicli annidati (variazione 3)

```
for i in range(10):
    for j in range(10):

        if (i+j)%2 == 0:
            print(' ',end='')
        else:
            print('*',end='')

    print('')
```

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

# Interruzione anticipata di un ciclo

I cicli `for` fanno tante iterazioni quanti sono i valori nella sequenza. Tuttavia…

- `return` interrompe la funzione
- `break` interrompe il ciclo
- `continue` interrompe l'iterazione

# Sequenza ordinata?

Una funzione che determina se una sequenza è ordinata o meno. Appena si trova un elemento non ordinato, **non serve** andare oltre nella lista.

```
def è_ordinata(seq):
    N=len(seq)

    if N==0:
        return True

    for i in range(1,N):
        if seq[i-1] > seq[i]:
            return False

    return True

print( è_ordinata([-1,3,5,12]) )
print( è_ordinata(['cane','mucca','gatto','topo']) )
```

```
True
False
```

# Interruzione con `continue`

- La singola iterazione si interrompe
- Il ciclo continua con l'iterazione seguente

Somma degli elementi in `lista` meno quelli in `vietati`.

```
def somma_con_divieti(lista,vietati):
    S = 0
    for x in lista:
        if x in vietati:
            continue
        S = S + x

    return S

print( somma_con_divieti( [2,5,-3] ,[5,7]) )
print( somma_con_divieti( [2,5,-3] , []) )
```

```
-1
4
```

# Interruzione con `break`

- Il ciclo si interrompe
- Si prosegue con la prima istruzione dopo il ciclo

Scriviamo `indice` che restituisce la posizione di un elemento nella lista.

```
def indice(elemento,lista):                                    1
                                                               2
    posizione = None                                           3
    for i in range(len(lista)):                                4
        if lista[i]==elemento:                                 5
            posizione = i                                      6
            break                                              7
                                                               8
    return posizione                                           9
                                                               10
print( indice( 4, [2,5,-3]) )                                  11
print( indice(-3, [2,5,-3]) )                                  12
```

```
None
2
```

# ciclo `while`

# Sintassi e significato

```
while condizione:                                    1
    istruzione1                                      2
    istruzione2                                      3
    ...                                              4
```

`condizione` è un'espressione booleana.

1. Se `condizione` allora vai al punto 2, altrimenti al punto 3.
2. Si esegue il blocco di istruzioni e dopo si torna al punto 1.
3. Si prosegue con le istruzioni successive al blocco `while`

# Esempio: numero primo

Ritorna `True` se il numero è primo

```
def primo(n):                                    1
    k = 2                                        2
    while k < n and (n % k) != 0:                3
        k = k + 1                                4
    return k == n                                5
                                                 6
print( primo(1) )                                7
print( primo(10) )                               8
print( primo(13) )                               9
print( primo(15) )                               10
```

```
False
False
True
False
```

# Simulare `range(a,b)`

```
a = -2                          1
b = 7                           2
                                3
i = a                           4
while i < b:                    5
                                6
    print(i)                    7
                                8
    i = i + 1                   9
```

```
-2
-1
0
1
2
3
4
5
6
```

# Ciclo infinito

Con il ciclo `for` la lunghezza della sequenza su cui si itera è sempre un limite superiore al numero di iterazioni.

```
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.
[...]
```

# Ciclo infinito (2)

```
                                    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
[...]
```

# Letture

Capitolo 6 e 7

# Esercizio per casa

Scoprire come si comporta la funzione

```
range(a,b,c)
```

per `a`,`b`,`c` interi.

- provare anche `a`, `b`, `c` zeri e negativi.
- leggere la documentazione `help(range)`