

Iterazioni su sequenze

Informatica@SEFA 2018/2019 - Lezione 9

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

Mercoledì, 17 Ottobre 2018

Ancora su indentazione e Geany

Problema di Tab vs Spazi

Diverse persone hanno problemi dovuti al mischiare

- ▶ Tabulazioni (TAB)
- ▶ Spazi

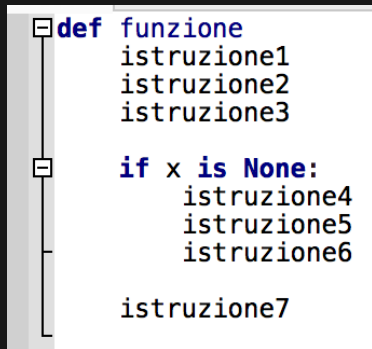
nel codice.

Questo causa

- ▶ `IndentationError` anche se il codice sembra ben indentato
- ▶ `TabError`

Tab vs Space nell'editor Geany

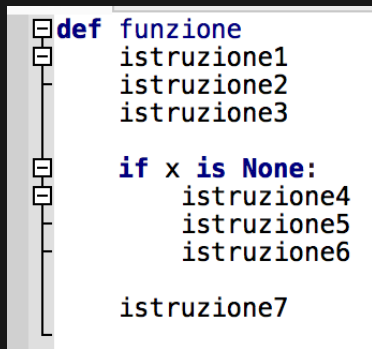
Geany evidenzia i livelli di indentazione. Linee spurie possono indicare che Tab e Spazi si sono mischiati.



```
def funzione
    istruzione1
    istruzione2
    istruzione3

    if x is None:
        istruzione4
        istruzione5
        istruzione6

    istruzione7
```



```
def funzione
    istruzione1
    istruzione2
    istruzione3

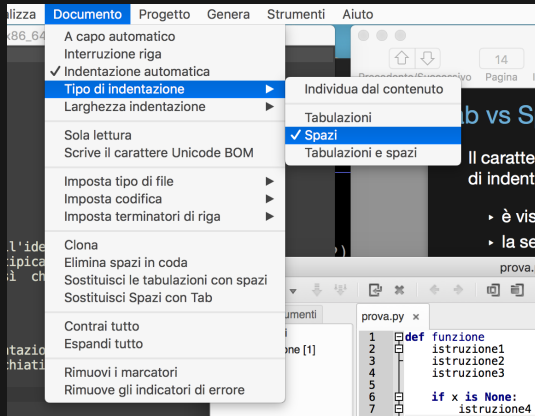
    if x is None:
        istruzione4
        istruzione5
        istruzione6

    istruzione7
```

Come si risolve questa situazione?

1. Impostate l'editor per far inserire 4 spazi invece di un TAB.
2. Re-indentate tutto il codice
3. Eseguire il file per vedere se l'avete pulito bene
4. Ripetete fino a che non ci sono più tabulazioni

Impostare l'editor Geany



Ancora sulle sequenze

Ancora sulle sequenze

- ▶ operatori di appartenenza
- ▶ operatori di confronto
- ▶ metodi
- ▶ funzioni di “analisi dei dati”

Operatore di appartenenza

Una sequenza è una collezione di valori. Una delle operazioni più utili è rispondere alla domanda

la sequenza S contiene il valore x ?

Su tuple e liste ci sono gli operatori

```
expr in seq  
expr not in seq
```

1
2

Esempio (I)

```
seq = [1, 2, 3, 5, 8]           1
print(5 in seq)                 2
print(4 in seq)                 3
print(4 not in seq)            4
print('mela' in ('noce', 4, 'mela', 'banana', 19.6) ) 5
```

```
True
False
True
True
```

Esempio (II)

```
def check_date(mese, giorno):           1
    if giorno < 1:                       2
        return False                    3
    if mese == 'feb':                   4
        return giorno <= 28             5
    elif mese in ['apr', 'giu', 'set', 'nov']: 6
        return giorno <= 30             7
    elif mese in ['gen', 'mar', 'mag', 'lug', 'ago', 'ott', 'dic']: 8
        return giorno <= 31             9
    else:                               10
        return False                    11
                                         12
print(check_date('gen', 31) )           13
print(check_date('feb', 29) )           14
print(check_date('dic', 0 ) )           15
print(check_date('mar', 1 ) )           16
```

```
True
False
False
False
```

Un'importante differenza (I)

Una stringa è una sequenza, ed è essenzialmente simile ad una tupla di caratteri.

```
s = 'avvicendamento'      1
t = tuple(s)              2
print(s)                  3
print(t)                  4
print(len(s))             5
print(len(t))             6
print(s[4])               7
print(t[4])               8
```

```
avvicendamento
('a', 'v', 'v', 'i', 'c', 'e', 'n', 'd', 'a', 'm', 'e', 'n', 't', 'o')
14
14
c
c
```

Un'importante differenza (II)

Tuttavia l'operatore `in`, not `in` su stringhe funziona diversamente che su liste e tuple.

```
s = 'avvicendamento' 1
t = tuple(s)          2
print( s )            3
print( t )            4
print( 'm' in s )     5
print( 'm' in t )     6
print( 'vvic' in s )  7
print( 'vvic' in t )  8
```

```
avvicendamento
('a', 'v', 'v', 'i', 'c', 'e', 'n', 'd', 'a', 'm', 'e', 'n', 't', 'o')
True
True
True
False
```

Operatori di confronto

Possono essere usati tra sequenze. L'ordine è lessicografico.

```
lst = [1, 2, 'abc', 5] 1
print(lst == [1, 2, 'abc', 5]) 2
print(lst < [1, 2, 'abcd']) 3
```

```
True
True
```

Ma non tutte le sequenze sono confrontabili.

```
lst = [12 , 1 ] < ['abc', 23 ] 1
print(lst) 2
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    lst = [12 , 1 ] < ['abc', 23 ]
TypeError: '<' not supported between instances of 'int' and 'str'
```

Il tipo di sequenza conta

Due sequenze con gli stessi valori non sono uguali, se il loro tipo è distinto.

```
print( (1,2) == [1,2] )
```

1

```
False
```

Metodi e sequenze

Metodi

Metodo: funzione specifica per un tipo di dato.

```
s = "aMmOnimenTo" 1
print( s.lower() )   # tutto minuscolo 2
print( s.upper() )   # tutto maiuscolo 3
print(s) 4
5
l = ["blu","rosso"] 6
print( l.append("giallo") ) # non restituisce nulla 7
print(l) # modifica la lista 8
9
print( l.index("rosso") ) # la posizione di 'rosso' 10
print(l) # non modifica la lista 11
```

```
ammonimento
AMMONIMENTO
aMmOnimenTo
None
['blu', 'rosso', 'giallo']
1
['blu', 'rosso', 'giallo']
```

Sintassi per l'uso dei metodi

```
expr.metodo(arg1,arg2,arg3)
```

1. `expr` viene valutata con valore V
2. il valore V avrà un certo tipo T
3. python esegue una funzione `metodo` che
 - dipende dal tipo T
 - è eseguita su parametri $V, arg1, arg2, arg3$

Come se si eseguisse

$$\text{metodo}_T(V, arg_1, arg_2, arg_3)$$

Esempio: `expr.count(x)`

- ▶ `count` non è definito su tutti i tipi (e.g. `float`)
- ▶ su una lista `L` conta quante volte `x` è nella lista `L`
- ▶ su stringa `s` conta quante volte `x` è una sotto-stringa in `s`

```
print( [1,3,'ciao',3,'2'].count(3) )      1
print( 'abcadabjkaab'.count('ab') )      2
print( (3.5).count('a') )                 3
```

```
2
3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/tmp/babel-0myZMc/python-4ciMfF", line 3, in <module>
    print( (3.5).count('a') )
AttributeError: 'float' object has no attribute 'count'
```

Documentazione di un metodo

```
help(str.count)
```

1

```
help(list.count)
```

2

3

Help on method_descriptor:

```
count(...)
```

```
    S.count(sub[, start[, end]]) -> int
```

Return the number of non-overlapping occurrences of substring sub in string S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Help on method_descriptor:

```
count(...)
```

```
    L.count(value) -> integer -- return number of occurrences of value
```

Una serie di metodi utili

```
L=['a','b','c'] 1
L.append('d')    # aggiunge UN elemento alla lista 2
print(L)        3

L.extend(['e','f','g']) # concatena una sequenza alla lista 4
print(L)        5

L.extend( ('h','i','j','k') ) # l'argomento non deve essere 6
    una lista 7
print(L)        8

L.insert( 4, 'intruso') 9
print(L)             10
                    11
                    12
```

```
['a', 'b', 'c', 'd']
['a', 'b', 'c', 'd', 'e', 'f', 'g']
['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k']
['a', 'b', 'c', 'd', 'intruso', 'e', 'f', 'g', 'h', 'i', 'j', 'k']
```

Differenza tra append e extend

L1=['a','b','c']	1
L1.append(['e','f','g'])	2
print(L1)	3
	4
L2=['a','b','c']	5
L2.extend(['e','f','g'])	6
print(L2)	7

```
['a', 'b', 'c', ['e', 'f', 'g']]  
['a', 'b', 'c', 'e', 'f', 'g']
```

Analisi dei dati

Alcune funzioni utili: sum, min, max, sorted

```
numeri = ( 3, 12, 6, -2 , -1 )      1
parole = ( 'libro', 'abaco', 'calamaio', 'pressa' )      2
                                     3
print ( sum(numeri) )                # somma una sequenza di numeri      4
                                     5
print ( min(parole) )                # min in una sequenza ordinabile    6
print ( max(numeri) )                # max in una sequenza ordinabile    7
                                     8
# ordina                             9
print ( sorted(numeri))              # restituisce sempre una lista      10
print ( sorted(parole))              11
```

```
18
abaco
12
[-2, -1, 3, 6, 12]
['abaco', 'calamaio', 'libro', 'pressa']
```

Questionario

bit.ly/INFO2018-09a

Iterazioni su sequenze

Ripetizione di istruzioni

Nei programmi che abbiamo visto fino ad oggi ogni istruzione del programma viene eseguita un numero limitato di volte.

Per elaborare grandi quantità di dati dobbiamo riutilizzare le istruzioni.

Iterazioni su sequenze: ciclo `for`

```
for variabile in sequenza:      1
    istruzione1                 2
    istruzione2                 3
    ...                         4
```

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: stampa tutti i valori

```
colori = ['blu', 'rosso', 'verde', 'giallo']      1
                                                    2
for x in colori:                                  3
    print(x)                                       4
```

```
blu
rosso
verde
giallo
```

La stampa viene ripetuta per quattro volte. La prima volta `x='blu'`, la seconda con `x='rosso'`, ...

Esempio: somma di numeri (I)

Scriviamo noi una versione della funzione `sum`

```
def somma_numeri(seq):           1
    ?????                       2
    for v in seq:               3
        ?????                   4
        ?????                   5
    ?????                       6

print( somma_numeri([3,1,5,-2]) ) 7
                                   8
```

Come la completiamo?

Esempio: somma di numeri (II)

[-3, -10, 0, 7, 10, -5, 1, -10, 6, -9, 4, -7, -10,	1
1, 6, 8, 7, 9, -8, 3, 6, -2, -7, 0, -2, 10, -5, -8, 9, -2]	2

quanto vale questa somma?

Esempio: somma di numeri (III)

Scriviamo noi una versione della funzione `sum`

```
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: somma di numeri (IV)

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

```
Aggiungo 3 all'accumulatore 0
Aggiungo 1 all'accumulatore 3
Aggiungo 5 all'accumulatore 4
Aggiungo -2 all'accumulatore 9
7
```


Esempio: lunghezza di una sequenza

```
def lunghezza(seq):           1
    accumulatore = 0         2
    for v in seq:            3
        accumulatore = accumulatore + 1  4

    return accumulatore       5

print( lunghezza([3,1,5,-2]) ) 6
print( lunghezza(['verde','azzurro','giallo']) ) 7
```

4

3

Esempio: produrre una nuova lista

```
def quadrati(seq):  
    accumulatore = []  
    for v in seq:  
        accumulatore.append(v*v)  
    return accumulatore  
  
print( quadrati([3,1,5,-2]) )
```

1
2
3
4
5
6
7

[9, 1, 25, 4]

Esempio: produrre una nuova lista (II)

```
def maiuscole(seq):           1
    accumulatore = []        2
    for s in seq:            3
        accumulatore.append( s.upper() ) 4
    return accumulatore       5

print( maiuscole(['verde','azzurro','giallo']) ) 6
                                                                    7
```

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 su sequenza 3
        vuota') 4

    temp_min = seq[0] 5
    for v in seq: 6
        if temp_min > v: 7
            temp_min = v 8

    return temp_min 9

print( minimo([3,1,5,-2]) ) 10
print( minimo(['verde','azzurro','giallo']) ) 11
12
13
```

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

Lecture

- ▶ Cap 7.1, 7.3, 8.4, 8.5. Libro di Python.