INDICE

Indentazione

L'indentazione e' un elemento fondamentale del linguaggio e definisce la struttura del codice:

In [1]:
```
a=10
#la seguente istruzione non e' permessa, tutte le istruzioni/espressioni dello stesso livello
#devono essere indentate allo stesso modo
 b=20
```

```
IndentationError: unexpected indent
```

Ma quali sono le circostanze in cui si hanno *cambi di livello* nel codice?
Elenchiamo i più importanti:

I. blocchi `if`
II. cicli `for`
III. cicli `while`
IV. definizione di funzioni *()*
V. *definizione di classi ()*

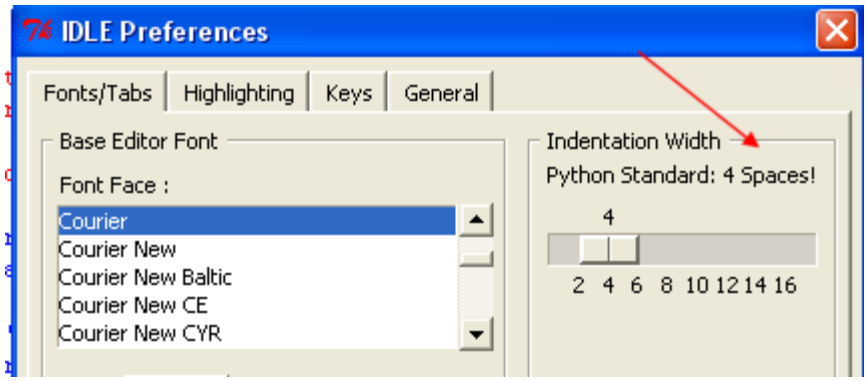
(*) questi li vedremo più avanti

Alcune note sull'indentazione in modalita' interattiva:

- Quando usiamo Python in modo interattivo in genere e' lo shell utilizzato che gestisce l'indentazione e la de-indentazione.
- in genere si scende di un livello dopo il carattere ':' (e la riga successiva ai : e' indentata rispetto alla precedete)
- si sale di un livello forzando la deindentazione (cancellando fisicamente i caratteri di indentazione con il backspace ad esempio)

note sull'indentazione in script e moduli scritti con un editor di testo.

- L'indentazione avviene con il carattere di tabulazione o con un certo numero (fissato) di spazi.
- In ogni caso e' 'fortemente consigliato' non mischiare tabulazione e spazi nello stesso file.
- Alcuni editor gestiscono l'indentazione come lo shell'interattivo (anche l'editor di IDLE, vedi figura sotto).
- Lo standard di python sono 4 spazi per ogni livello di indentazione



istruzione if

Avevamo gia' visto la sintassi del costrutto if (lezione sui numeri)

if : ... elif : ... elif : ... else: .... #la fine dell'ultima istruzione e' determinata #dal livello di indentazione del codice

Vediamo una proposizione composta da if annidati:

In [2]:
```
import random
#lista delle ore
ore=list(range(24))
#scelta casuale dalla lista delle ore
ora = random.choice(ore)
#formattazione di una stringa
str = 'sono le {0} e cioè è '.format(ora)

#
if ora >= 0 and ora <= 5 or ora > 20 and ora <=24:
    str += "notte"
    if ora <=3:
        str += " fonda"
elif ora <= 12:
    str += "mattina"
    if ora <=8:
        str += " presto"
elif ora <= 18 :
    str += "pomeriggio"
    if ora >=17:
        str += " inoltrato"
elif ora <= 20 :
    str += "sera"
else:
    str = 'ERRORE!!'

print(str)
```

```
sono le 10 e cioè è mattina
```

Truth testing in cicli if e while

Una caratteristica interessante di Python, che può essere utilizzata in particolare nei costrutti if e while, e' che qualsiasi oggetto puo' essere testato come se fosse un Bool.

I seguenti valori sono considerati falsi:

- None
- False
- zero di qualsiasi tipo numerico, ad esempio, 0, 0,0, 0j.
- qualsiasi sequenza vuota, per esempio,'', (), []
- dizionari vuoti {}
- istanze di classi definite dall'utente, se la classe definisce un metodo `bool` () o `len` () e il metodo restituisce il valore intero zero o bool False.

Tutti gli altri valori sono considerati veri.

Vedi anche truth value testing.

Sperimentiamo quanto detto sopra:

In [10]:
```
#definisco una funzione
def truthtest(a):
    if a:
        print('{0} è vero'.format(a))
    else:
        print('{0} è falso'.format(a))

truthtest([])
truthtest(0)
truthtest([2])
truthtest(0+0j)
truthtest(print)#una funzione
truthtest(math)#un modulo
```

```
[] è falso
0 è falso
[2] è vero
0j è falso
<built-in function print> è vero
<module 'math' (built-in)> è vero
```

istruzione for

Anche dell'istruzione for abbiamo gia' parlato durante la lezione sulle liste, aggiungiamo qui il costrutto completo, che comprende anche una clausola *else* che viene eseguita solo se si terminano le iterazioni senza avere incontrato alcuna istruzione `break`, che vedremo meglio dopo.

for target in iterabile: elenco istruzioni .... [else: elenco istruzioni ...]

esempio:

In [4]:
```
d = {'Nicola':23255656 , 'Alfredo':32447778 , 'fernando' : 34546677, 'Gino':233657898}

#funzione per ordinamento case-insensitive
def mylower(s):
    return s.lower()

for k in sorted(d.keys(),key=mylower):
    print(k,'-->',d[k])
else:
    print('---- Ho finito ----')
```

```
Alfredo --> 32447778
```

```
fernando --> 34546677
Gino --> 233657898
Nicola --> 23255656
---- Ho finito ----
```

istruzione while

Nel costrutto while si condiziona l'esecuzione di un blocco di istruzioni al valore di un'espressione:

while : elenco istruzioni eseguite fino a che l'espressione sopra restituisce True .... [else: elenco istruzioni eseguite alla fine (se non si incontra break) ...]

Anche il costrutto while comprende anche una clausola else che viene eseguita solo se si terminano le iterazioni senza avere incontrato alcuna istruzione break.

vediamo un esempio:

In [11]:
```
#partita a dadi virtuale
import random
#punteggio complessivo dei giocatori
p1 = 0
p2 = 0
win    = 5

#si inizia a giocare...si vince a 5
while p1 < win and p2 < win:
    #i giocatori tirano i dadi
    s1=random.randint(1,6)
    s2=random.randint(1,6)
    print(s1,s2)
    #valuto gli score ed eventualmente assegno un punto
    if s1>s2:
        p1 += 1
        print('un punto a 1')
    elif s1<s2:
        p2 += 1
        print('un punto a 2')
    else:
        print('patta')

#la partita e' finita, decreto il vincitore
else:
    print('-'*20)
    if p1>p2:
        print('Vince 1 !!!!')
    else:
        print('Vince 2 !!!!')
```

```
4 1
un punto a 1
5 3
un punto a 1
4 3
un punto a 1
6 6
patta
6 6
patta
4 4
```

```
patta
5 4
un punto a 1
4 5
un punto a 2
2 4
un punto a 2
2 4
un punto a 2
2 5
un punto a 2
2 1
un punto a 1
--------------------
Vince 1 !!!!
```

Uscita dai cicli for e while con break

l'interprete esce immediatamente dal ciclo while o for esterno 'più vicino', saltando la eventuale clausola else.

Vediamo un esempio con due cicli for annidati:

In [12]:
```
for i in range(5):
    print('i=',i)
    for j in range(100,104):
        print('    ',j)
        if i==2 and j >= 101:
            #esco dal ciclo for su j
            #solo quando si presentano
            #specifiche condizioni
            print('    break!!!')
            break
    else:
        print('    else')
print('fine')
```

```
i= 0
     100
     101
     102
     103
     else
i= 1
     100
     101
     102
     103
     else
i= 2
     100
     101
     break!!!
i= 3
     100
     101
     102
     103
     else
```

```
i= 4
     100
     101
     102
     103
     else
fine
```

while True:

break e' spesso usato per uscire da cicli while con espressione sempre vera.

Vediamo un esempio nel quale, all'interno di un ciclo while 'indefinito' genero un numero casuale ed esco con break solo se si manifesta un valore predefinito

In [13]:
```
import random
while True:
    a=random.randint(0,20)
    print('è uscito {0}'.format(a))
    if a == 5:
        print('finalmente, break!!!')
        break
```

```
è uscito 10
è uscito 15
è uscito 0
è uscito 19
è uscito 15
è uscito 11
è uscito 15
è uscito 3
è uscito 16
è uscito 11
è uscito 8
è uscito 5
finalmente, break!!!
```

Saltare iterazioni dei cicli for e while con continue

se incontra una istruzione `continue`l'interprete va immediatamente all'inizio del ciclo while o for 'più vicino' e riprende l'iterazione In altre parole salta tutta la restante parte dell'elenco di espressioni da eseguire ad ogni iterazione e riparte con l'iterazione successiva.

In [14]:
```
#salto i miltipli del 3
for i in range(1,20):
    if i%3==0:
        continue

    print('{0:02d} non è multiplo di 3'.format(i))
```

```
01 non è multiplo di 3
02 non è multiplo di 3
04 non è multiplo di 3
05 non è multiplo di 3
07 non è multiplo di 3
```

```
08 non è multiplo di 3
10 non è multiplo di 3
11 non è multiplo di 3
13 non è multiplo di 3
14 non è multiplo di 3
16 non è multiplo di 3
17 non è multiplo di 3
19 non è multiplo di 3
```

Vediamo cosa succede nel caso di cicli annidati:

In [15]:
```
for j in range(3):
    print('----{0}----'.format(j))
    for i in range(5):
        ##
        if i == 2:
            print('continue')
            continue
        #
        print('{0:02d}'.format(i))
```

```
----0----
00
01
continue
03
04
----1----
00
01
continue
03
04
----2----
00
01
continue
03
04
```

Gli esempi sopra sarebbero facilente implementabili anche senza continue.
In ogni caso si fa notare che l'aggiunta dei costrutti `if ... continue` è formalmente più corretta nel caso di condizioni speciali permette di non mettere mano al codice che definisce la logica successiva (neanche con un cambio di indentazione)