if-then-else

June 5, 2023

0.1 If ... then ... else

Per ora i nostri programmi sono stati sequenze di instruzioni eseguite una dopo l'altra. Non è certo l'unico modo possibile. L'esecuzione del programma può arrivare a dei **bivi** dove può scegliere da quale parte andare. Per fargli fare una cosa simile si devono usare istruzioni che controllano il **flusso del programma**.

Potrebbe esservi già chiaro cosa fa questo programma. Sperimentate!

[ ]:
```
key = input("Inserisci la password:")

if key == 'segretissimo':
    print("Password accettata. Benvenuto Presidente.")

print("Uscita dal programma")
```

Il programma legge in input una stringa di testo e **se** questa corrisponde alla password segreta **allora** il programma saluta con rispetto. Questo è un esempio di esecuzione **condizionale** che ruota intorno ad un'espressione il cui valore è **vero** o **falso**. In questo caso l'espressione `key == 'segretissimo'` è vera quando la variable `key` ha lo stesso valore della stringa `"segretissimo"`.

La struttura del costrutto `if` è

```
if <condizione>:
    <istruzione1>
    <istruzione2>
    <istruzione3>
```

dove al posto di `<condizione>` ci va un'espressione vero/falso, seguita dai due punti `:`. Il significato è che **se** l'espressione `condizione` è vera **allora** una certa parte di codice, detta *blocco if*, viene eseguita. Altrimenti quella parte di codice viene ignorata. Ho rappresentato il blocco if come tre righe, ma in realtà può essere lungo un numero arbitrario di righe.

Facciamo un altro esempio: il prossimo programma stampa una fila di `A` lunga tanto quanto è il valore di `x` (ma con un limite di 7). Cambiate il valore di `x` nell'esempio e osservate come si comporta il programma.

[2]:
```
x = 10

if x > 7:
    print("x è molto grande")
```

```
    x = 7

print( "A" * x  )
```

```
AAAA
```

Vedete che le istruzioni nel blocco if sono indentate di quattro spazi mentre quelle del flusso principale non lo sono. Questa indentazione fa capire a Python quali sono le istruzioni fuori e dentro il blocco.

Esercizio: cambiate l'indentazione dell'ultima riga del programma precedente. Che succede che la indentate di quattro spazi? E due? E zero?

0.1.1 Blocco `else`

Abbiamo visto come eseguire un codice quando una condizione si verifica. Ora vediamo come impostare due blocchi di codice alternativi: uno verrà eseguito se la condizione è vera, e l'altro se la condizione è falsa.

[ ]:
```
if 5 > 2:
     print('5 è infatti maggiore di 2')
else:
     print('5 non è maggiore di 2')
```

Che succede se cambiate i numeri nella prima riga? Per esempio se mettete `4 > 6`? Che succede se eliminare la terza e quarta riga? Osserviamo che il blocco `else` è **opzionale**. Nel prossimo esempio provate a sostituire la prima riga con `num=int(input())` ed eseguite il programma varie volte.

[ ]:
```
num = 8723489

if (num % 2) == 0:

    print("Il numero inserito è pari.")

else:

    print("Il numero inserito è dispari.")
```

Chiarimento 1: l'espressione `(num & 2)` è il resto di `num` diviso due. Ha valore `0` se `num` è pari, e ha valore `1` se `num` è dispari. Pertanto la condizione viene verificata da numeri pari e falsificata dai numeri dispari.

Chiarimento 2: fate attenzione! In Python `=` viene usato per l'assegnamento a variabili, mentre l'operatore `==` corrisponde alla domanda: "questo e quello sono uguali?", che può avere esito vero o falso.

0.1.2 Errori di indentazione

Finalmente i vostri programmi non sono solo una sequenza di istruzioni una dopo l'altra, ma hanno una struttura. È utile riconoscere gli errori che vi vengono segnalati nel caso ci sia qualcosa che

non va in questa struttura

[3]:
```
if 5 > 3:
```

```
  Input In [3]
    if 5 > 3:
              ^
SyntaxError: incomplete input
```

Qui sopra Python si aspetta del codice nel blocco if, ma questo codice è assente e python si lamenta che si e trovato alla fine del codice prima del previsto.

[4]:
```
if 5 > 3:

print("Siamo nel blocco if?")
```

```
  Input In [4]
    print("Siamo nel blocco if?")
    ^
IndentationError: expected an indented block after 'if' statement on line 1
```

Anche qui sopra Python si aspetta del codice nel blocco if, ma questo codice è assente e Python trova invece un'istruzione che interpreta "allo stesso livello" dell'istruzione `if` e pertanto non la considera come parte del blocco if.

[6]:
```
if 5 > 3:
    print("Siamo nel blocco if?")
  print("E qui?")
```

```
  File <tokenize>:3
    print("E qui?")
    ^
IndentationError: unindent does not match any outer indentation level
```

Nel codice precedente, Python trova l'inizio del blocco if alla seconda riga. Alla riga successiva (i.e. la terza) Python vede che l'istruzione è rientrata verso sinistra e quindi capisce che il blocco if è terminato. Tuttavia l'indentazione non corrisponde a quella dell'istruzione `if` e quindi dà errore.

0.1.3 If...then...else, annidati

È possibile annidare blocchi di codice, e in particolare è possibile annidare blocchi if...then...else. Vediamo questo esempio che dato un voto d'esame, stampa un commento.

[ ]:
```python
voto = int(input("Voto? "))

if voto < 0:
    print("Non mi prendere in giro!")
else:
    if voto > 30:
        print("Esagerato!!")
    else:
        if voto < 18:
            print("Bocciato.")
        else:
            if voto > 28:
                print("Complimenti!!")
            else:
                print("Va bene così.")
```

0.1.4 Chiarimenti sull'indentazione

Come avete visto un programma python può avere tanti blocchi di codice, anche annidati, ed è quindi lecito essere confusi e farsi delle domande: - dove inizia un blocco di codice? - dove finisce? - quali blocchi sono annidati in quali blocchi?

Per determinare l'inizio e la fine dei blocchi di codice python usa l'indentazione, ovvero la spaziatura iniziale davanti a ogni riga di codice. Il più delle volte si capisce immediatamente dove inizia e finisce un blocco, ma per precisione ecco delle indicazioni più precise.

1. Dopo ogni istruzione che termina con `:`, ad esempio `if <condizione>:` o `else:`, si deve aprire un nuovo blocco di codice. Nessun blocco di codice può essere aperto se la riga precedente non termina con in due punti.
2. La prima riga del blocco deve essere indentata più a destra della riga precedente.
3. Tutte le righe che fanno parte del blocco di codice devono avere la **stessa** indentazione, a meno che non facciano parte a loro volta di blocchi annidati in esso, e perciò ulteriormente verso destra.
4. Quando python incontra una riga che ha meno spazi a sinistra di quella precedente, allora capisce che il blocco di codice è terminato. L'indentazione di questa nuova riga deve essere tassativamente uguale a quella di uno dei blocchi più esterni.
5. Le righe costituite da spazi bianchi o contententi solo commenti, sono totalmente ignorate in questo processo.

Vediamo un diagramma di esempio:

```
<livello 1>
<livello 1>
<livello 1>

if <condizione>:
    <livello 2>

    <livello 2>
```

```
<livello 1>
<livello 1>
<livello 1>

if <condizione>:

    <livello 2>
    <livello 2>

else:
    <livello 2>
    if <condizione>:
       <livello 3>
       <livello 3>

<livello1>

<livello1>
<livello1>
```

Esercizio: determinare quali sono i blocchi if, e quali i blocchi else. Dove iniziano dove finiscono?

0.1.5 Elif

Tornate all'esempio degli if...then...else annidati. Era facile da leggere? Probabilmente no, eppure quel caso è molto tipico: una serie di if...then...else annidati, dove ogni nuova istruzione `if` è immediatamente dopo l'`else` del blocco precedente. In python esiste l'istruzione `elif` (contrazione di `else if`) che permette di scrivere questo tipo di struttura in modo più leggibile.

[ ]:
```
voto = int(input("Voto? "))

if voto < 0:
    print("Non mi prendere in giro!")
elif voto > 30:
    print("Esagerato!!")
elif voto < 18:
    print("Bocciato.")
elif voto > 28:
    print("Complimenti!!")
else:
    print("Va bene così.")
```

Non è molto più chiaro? Eppure la logica e il funzionamento sono gli stessi. Altro esempio:

[ ]:
```
volume = 57
if volume < 20:
    print("Piuttosto basso.")
elif 20 <= volume < 40:
```

```
    print("Adatto per musica di sottofondo")
elif 40 <= volume < 60:
    print("Perfetto, posso apprezzare ogni dettaglio")
elif 60 <= volume < 80:
    print("Ideale per le feste")
elif 80 <= volume < 100:
    print("Un po' altino!")
else:
    print("Oddio, le mie orecchie! :(")
```

Esercizio: rispondete alle seguenti domande. Se non conoscete le risposte potete provare a scrivere dei programmi per scoprirle. - possiamo avere un blocco `else` senza un blocco `if` corrispondente? - possiamo avere un blocco `if` senza un blocco `else` corrispondente? - possiamo avere un blocco `elif` senza un blocco `if` corrispondente?
- quanti blocchi `elif` possiamo avere?

0.1.6 Riassumendo

Abbiamo visto

- l'istruzione `if`! La prima istruzione per il controllo di flusso
- operatori `>`, `<`, `<=`, `>=` e `==` per costruire condizioni vero/falso
- il blocco `else`
- considerazioni sull'indentazione dei blocchi di codice
- uso dell'`elif`.