Istruzioni Condizionali in Python

Istruzioni di Controllo di Flusso

Finora ci siamo focalizzati su:

- Come rappresentare informazioni
  - In particolare: tipi di dato semplici e collezioni
- Come effettuare calcoli
  - In particolare: espressioni semplici e composte

In ogni caso, abbiamo assunto che le istruzioni siano eseguite una dopo l'altra

Vedremo ore come alterare il flusso di controllo

...I.e. come eseguire istruzioni non necessariamente in sequenza

- In Python, questo è possibile mediante un insieme di istruzioni
- ...Note come istruzioni di controllo di flusso

Ve ne sono due categorie principali, i.e. istruzioni di selezione e di iterazione

Istruzioni Condizionali

Iniziamo considerando le istruzioni condizionali (o di selezione)

Le istruzioni di selezione:

- ...Permettono di scegliere quale tra un insieme di istruzioni eseguire
- ...Sulla base del valore di una espressione

In Python ce ne sono due:

- Istruzione `if`
- Istruzione di pattern matching (da Python 3.10)

In questo corso vedremo solo l'istruzione `if`


Istruzione `if`

L'istruzione "`if`"

- Esegue una o più istruzioni facenti parte di un blocco
- ...Solo se una determinata condizione è verificata

La sintassi è:

```
if <espr. condizione>:
    <blocco>
```

- `<espr. condizione>` denota tipicamente un valore logico
- `<blocco>` consiste di una o più istruzioni
- ...Indentificabili perché devono avere lo stesso livello di indentazione

Indentazione = numeri di spazi (o tab) prima dell'istruzione


Istruzione `if`

Vediamo un esempio

In [1]:
```
a = 2
if a < 4:
    print('La variabile "a" ha un valore...')
    print('...Inferiore a 4')
```

```
La variabile "a" ha un valore...
...Inferiore a 4
```

Le due istruzioni di stampa:

- Seguono il simbolo " **:** "
- Sono preceduti dallo stesso numeri di spazi/tab
  - Non importa quanti, purché l'indentazione sia la stessa

Si dice che formano un blocco


Istruzione `if`

Vediamo un esempio

In [2]:
```
a = 2
if a < 4:
    print('La variabile "a" ha un valore...')
      print('...Inferiore a 4')
```

```
  Cell In[2], line 4
    print('...Inferiore a 4')
    ^
IndentationError: unexpected indent
```

- Se l'indentazione è inconsistente
- ...Python segnala un errore di sintassi


Istruzione `if`

Vediamo un esempio

In [3]:
```
a = 2
if a != 0:
    b = 2 / a
    print(f'b = {b}')
```

```
b = 1.0
```

- La condizione `a != 0` in questo caso è vera
- ...Quindi il blocco viene eseguito (ed otteniamo le due stampe)

Istruzione `if`

Vediamo un esempio

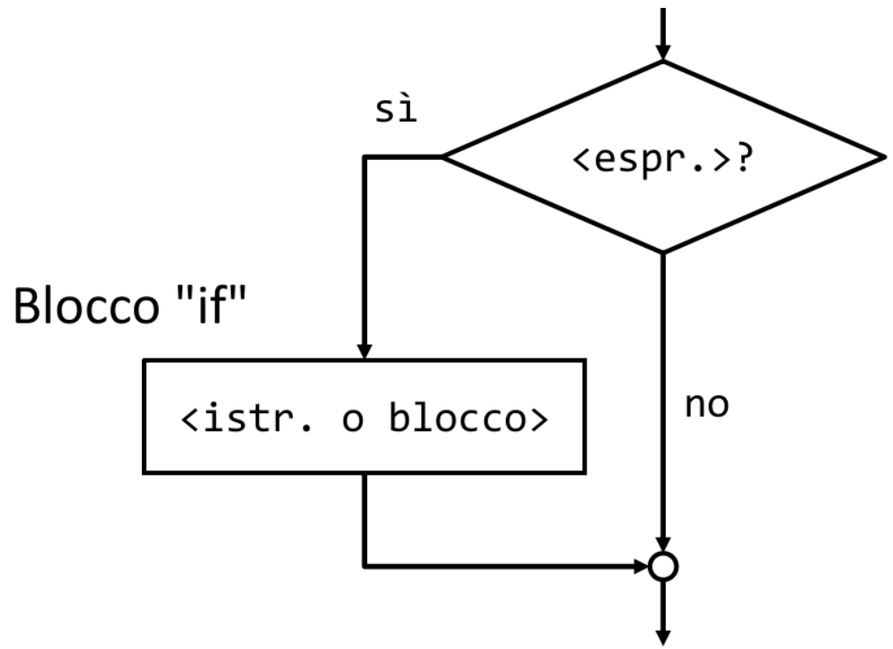
In [4]:
```
a = 0 # Il valore di a è stato modificato
if a != 0:
    b = 2 / a
    print(f'b = {b}')
```

- Se modifichiamo il valore di `a` in modo da rendere la condizione falsa
- ...Il blocco non viene più eseguito


Istruzione `if`

In generale, la semantica di una istruzione `if`

...È definita dal seguente digramma di flusso:



- Il blocco viene eseguito solo se l'espressione (condizione) è vera

Istruzione `if.. else`

Utilizzando la parola chiave `else`

...È possibile specificare un blocco da eseguire in caso la condizione sia falsa:

```
if <espr. condizione>:
    <sequenza di istruzioni>
else:
    <sequenza di istruzioni>
```

Vediamo un esempio:

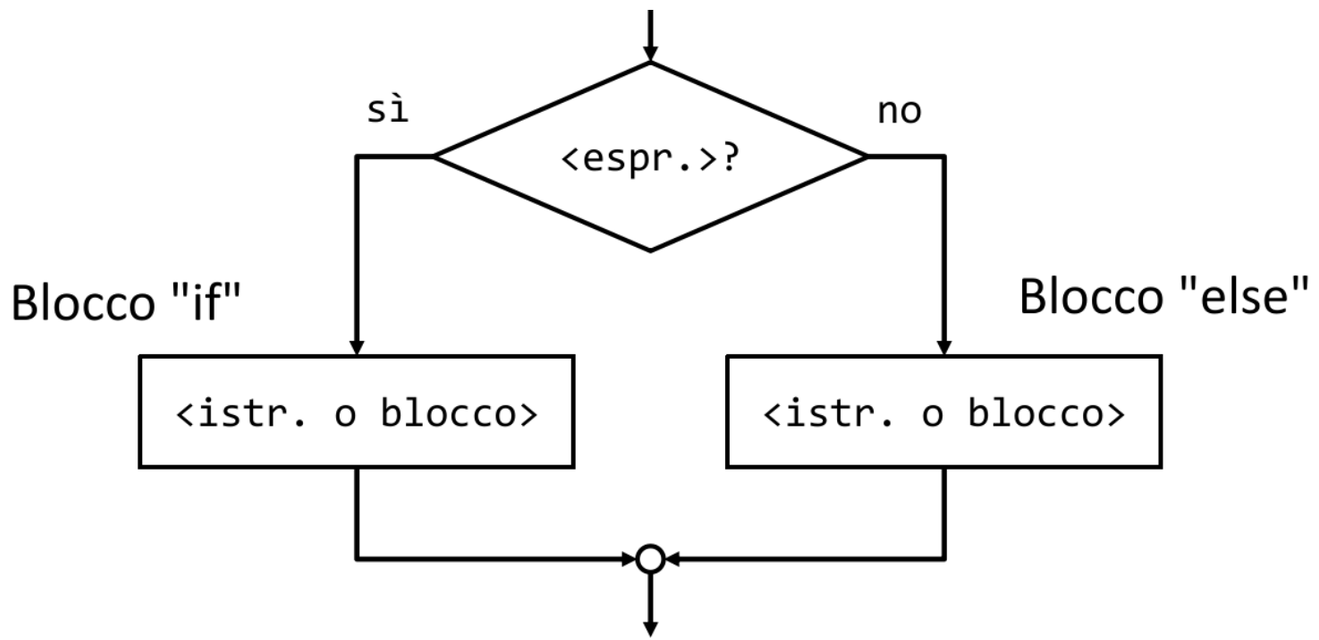
In [5]:
```
a = 0 # Il valore di a è stato modificato
if a != 0:
    b = 2 / a
    print(f'b = {b}')
else:
    print('Il valore di "a" non può essere 0')
```

```
Il valore di "a" non può essere 0
```


Istruzione `if.. else`

La semantica di una istruzione `if... else`

...È definita dal seguente digramma di flusso:



- Il blocco `if` viene eseguito se l'espressione (condizione) è vera
- Il blocco `else` viene eseguito se l'espressione (condizione) è falsa

Istruzione `if... elif`

Utilizzando la parola chiave `elif`

- ...È possibile specificare un blocco da eseguire in caso la condizione sia falsa
- ...Ma un'ulteriore condizione sia vera

La sintassi è la seguente:

```
if <espr. condizione>:
    <sequenza di istruzioni>
elif <espr. condizione>:
    <sequenza di istruzioni>
```

Il blocco "condizionato" dalla parola chiave `elif` viene eseguito:

- Se l'espressione che segue `if` denota falso
- ...E l'espressione che segue `elif` denota vero

Istruzione `if... elif`

Può essere presente più di un `elif` (più eventualmente un `else`)

```
if <espr. condizione>:
    <sequenza di istruzioni>
elif <espr. condizione>:
    <sequenza di istruzioni>
elif <espr. condizione>:
    <sequenza di istruzioni>
...
else:
    <sequenza di istruzioni>
```

Il blocco condizionato da ogni `elif` è eseguito:

- Se le espressioni dell'`if` e degli `elif` precedenti sono false
- ...E l'espressione dell'`elif` è vera

Il blocco else è eseguito nel caso tutte le espressioni siano false

Istruzione `if... elif`

Vediamo un esempio:

In [6]:
```
mese = -1
giorni = None
if mese == 2:
    giorni = 28
elif mese in (4, 6, 9, 11):
    giorni = 30
elif mese in (1, 3, 5, 7, 8, 10, 12):
    giorni = 31
else:
    print(f'"{mese}" non è un valore di mese valido"')
print(f'giorni: {giorni}')
```

```
"-1" non è un valore di mese valido"
giorni: None
```

- L'operatore `in` è usato per verificare se il valore di `mese`
- ...È tra quelli contenuti in determinate tuple

Istruzioni `if` Annidate

Le istruzioni in un blocco possono essere altre istruzioni `if`

Vediamo come esempio un confronto di date

- Alla fine la variabile `res` deve essere negativo se `d1` precede `d2`
- ...Deve essere positivo se `d1` segue `d2` e 0 se sono uguali

In [7]:
```
d1, d2 = (2022, 10, 4), (2022, 10, 7) # formato: (anno, mese, giorno)
# Confronto
res = d1[0] - d2[0] # confrontiamo gli anni
if res == 0: # vero se gli anni sono uguali
    res = d1[1] - d2[1]
    if res == 0: # vero se i mesi sono uguali
        res = d1[2] - d2[2]
# Stampa
if res < 0:
    print(f'{d1} precede {d2}')
elif res > 0:
    print(f'{d1} segue {d2}')
else:
    print(f'{d1} e {d2} sono uguali')
```

```
(2022, 10, 4) precede (2022, 10, 7)
```

Istruzione `if` e variabili

Eventuali variabili definite in blocco `if` (o `elif` o `else`)

...Hanno lo stesso tempo di vita che avrebbero se fossere definite fuori

In [8]:
```
mese = 2
if mese == 2:
    giorni = 28
print(f'giorni: {giorni}')
```

```
giorni: 28
```

In questo caso:

- La variabile giorni è creata nel blocco `if`
- ...Ma è comunque una variabile globale

Per questa ragione può essere stampata al di fuori del blocco


Istruzione `if` e variabili

Eventuali variabili definite in blocco `if` (o `elif` o `else`)

...Hanno lo stesso tempo di vita che avrebbero se fossere definite fuori

In [9]:
```
mese = 2
if mese == 2:
    giorni = 28
print(f'giorni: {giorni}')
```

```
giorni: 28
```

Si dice che le variabili nel blocco sono definite nello stesso ambiente dell'istruzione `if`

Istruzione `if` e variabili

Se una variabili viene definita nel blocco...

...È necessario che il blocco esegua perché la variabile sia creata

In [10]:
```
mese = 3 # Il valore di mese è stato cambiato
if mese == 2:
    giorni2 = 28 # una nuova variabile (perché "giorni" esiste ancora)
print(f'giorni2: {giorni2}')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[10], line 4
      2 if mese == 2:
      3     giorni2 = 28 # una nuova variabile (perché "giorni" esiste ancora)
----> 4 print(f'giorni2: {giorni2}')

NameError: name 'giorni2' is not defined
```

- Se il blocco `if` non viene eseguito
- ...La variabile non viene creata!


Istruzione `if` e variabili

Per questo, è buona norma non definire nuove variabili in un blocco `if`

In [11]:
```
mese = 3 # Il valore di mese è stato cambiato
giorni3 = None # una nuova variabile (per la stessa ragione di prima)
if mese == 2:
    giorni3 = 28
print(f'giorni3: {giorni3}')
```

```
giorni3: None
```

- Meglio definirle fuori (come `giorni3` in questo caso), con un valore di default
- ...Ed eventualmente modificarne il valore nel blocco

Attenzione: si tratta solo di una buona pratica e non di una limitazione di Python


Istruzione `if` e variabili

L'eccezione sono eventuali variabili che servono solo all'interno del blocco

In [12]:
```
mese, anno = 2, 1980
giorni3 = None # una nuova variabile (per la stessa ragione di prima)
if mese == 2:
    bisestile = (anno % 4 == 0)
    if bisestile:
        giorni3 = 29
    else:
        giorni3 = 28
print(f'giorni3: {giorni3}')
```

```
giorni3: 29
```

- In questo caso, la variabile `bisestile` è utile solo all'interno del blocco
- L'esempio mostra anche come un blocco possa contenere a sua volta un `if`
- In questo caso si parla di `if` innestati

Istruzione `if` e variabili

L'eccezione sono eventuali variabili che servono solo all'interno del blocco

In [13]:
```
mese, anno = 2, 1980
giorni3 = None # una nuova variabile (per la stessa ragione di prima)
if mese == 2:
    bisestile = (anno % 4 == 0)
    if bisestile:
        giorni3 = 29
    else:
        giorni3 = 28
print(f'giorni3: {giorni3}')
print(f'bisestile: {bisestile}')
```

```
giorni3: 29
bisestile: True
```

- Anche in questo caso, si tratta solo di una buona pratica
- ...E infatti la variabile `bisestile` è accessibile anche fuori dal blocco

Espressione Condizionale

Python fornisce anche un costrutto chiamato espressione condizionale

La sintassi è:

```
<espressione condizionale> ::= <espr. 1> if <espr. cond> else <espr. 2>
```

Non si tratta di una istruzione, ma di una espressione

- È definita mediate l'operatore ternario (i.e. con tre argomenti) `if ... else`
- Se `<espr. cond>` è vera, l'operatore denota `<espr. 1>`
- Se `<espr. cond>` è falsa, l'operatore denota `<espr. 2>`

Vediamo un esempio:

In [14]:
```
animale = 'cane'
print('bau' if animale == 'cane' else 'miao')
```

```
bau
```

Espressione Condizionale

Anche se la sintassi è simile

...L'istruzione `if ... else` e l'operatore `if ... else` sono diversi:

- L'operatore denota un valore, l'istruzione no

In [15]:
```
b = 0
a = 1/b if b != 0 else None
print(f'a: {a}')
```

```
a: None
```

- In questo caso il valore denotato viene inserito in `a`

Espressione Condizionale

Anche se la sintassi è simile

...L'istruzione `if ... else` e l'operatore `if ... else` sono diversi:

- L'istruzione può eseguire blocchi con istruzioni abritrarie
- ...Mentre l'operatore può solo valutare espressioni

In [16]:
```
b = 0
if b != 0:
    a = 1 / b
    print(f'a: {a}')
else:
    print('"b" non può avere il valore 0')
```

```
"b" non può avere il valore 0
```