Laurea in Ingegneria Gestionale

Corso di Fondamenti di Informatica
A.A. 2017/2018

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Istruzioni Condizionali

Slide tratte dal lavoro di Giorgio Fumera (Univ. degli Studi di Cagliari) e riadattate da Raffele Nicolussi (Sapienza) e Domenico Lembo (Sapienza)

Istruzioni condizionali

Istruzioni condizionali

Nella formulazione di un algoritmo si ha spesso la necessità di esprimere la scelta tra due o più sequenze di istruzioni, in base al verificarsi o meno di una certa condizione durante l'**esecuzione** dello stesso algoritmo.

Esempi:

- nel calcolo del valore assoluto di un numero, il risultato dipenderà dal segno di tale numero
- la radice di un'equazione di primo grado $ax + b = 0$ è data da –b/a, **se a !=0**, **altriment**i non è definita oppure non è unica

Istruzioni condizionali

In tutti i linguaggi di alto livello sono disponibili per questo scopo:

- le espressioni condizionali (anche dette espressioni booleane), che consentono di esprimere condizioni che possono essere **vere** o **false**
- l'istruzione condizionale, che consente di eseguire due diverse sequenze d'istruzioni alternative tra loro, in base al valore di un'espressione condizionale

Espressioni condizionali: sintassi

Il tipo più semplice di espressione condizionale consiste nel confronto tra due espressioni.

Sintassi: espressione$_1$ operatore espressione$_2$

- espressione$_1$ e espressione$_2$ sono due espressioni Python **qualsiasi**, definite come si è visto in precedenza (possono quindi contenere nomi di variabili, purché a tali variabili sia stato già assegnato un valore)
- operatore è un simbolo che indica il tipo di confronto tra le due espressioni

Espressioni condizionali: operatori di confronto

Nel linguaggio Python sono disponibili i seguenti operatori di confronto, che possono essere usati sia su espressioni aritmetiche che su espressioni composte da stringhe:

simbolo	significato
==	"uguale a" (notare il **doppio** simbolo =, per evitare ambiguità con l'istruzione di assegnamento)
!=	"diverso da"
<	"minore di"
<=	"minore o uguale a"
>	"maggiore di"
>=	"maggiore o uguale a"

Espressioni condizionali: esempi

Si assume che a ciascuna variabile che compare negli esempi seguenti sia già stato assegnato un valore.

- `1 < 2`
- `a + 1 != 5`
- `x >= y`
- `"macchina" < "casa"`
- `"mappa" + "mondo" == "mappamondo"`

Espressioni condizionali: semantica

Analogamente alle espressioni aritmetiche, il cui valore è un numero, e alle espressioni che coinvolgono stringhe, il cui valore è una stringa, un'espressione condizionale assume il valore logico "**vero**" oppure "**falso**", **in base all'esito del confronto.**

I due valori logici vengono indicati in Python con i simboli:

- `True` (vero)
- `False` (falso)

Si noti che questi simboli sono essi stessi **espressioni** lecite del linguaggio Python, proprio come i numeri e le stringhe.

In particolare, nel caso di un confronto tra due valori di tipo stringa, gli operatori `<`, `<=`, `>=` e `>` si riferiscono all'ordinamento lessicografico.

Ordinamento Lessicografico

Una stringa *s precede una stringa t* in ordine lessicografico se
- *s* è un prefisso di *t*, oppure
- se *c* e *d* sono il primo carattere rispettivamente di *s* e *t* in cui *s* e *t* differiscono, e c precede d nell'ordinamento dei caratteri.

Ordinamento dei caratteri: per i caratteri che sono lettere alfabetiche, l'ordinamento è quello alfabetico. Le cifre precedono le lettere, e le lettere maiuscole precedono quelle minuscole.

Esempi:
- `'auto'` precede `'automatico'`
- `'Automatico'` precede `'auto'`
- `'albero'` precede `'alto'`
- `'H2O'` precede `'HOTEL'`

Nota: l'ordinamento (totale) è definito per tutti i caratteri possibili, anche se non sono cifre o numeri (ad es., `':'` `';'` `'@'`). La funzione Python `ord(c)` restituisce la posizione del carattere `c` nell'ordinamento (più precisamente restituisce un intero che corrisponde al *codice Unicode* associato a `c`). Viceversa, la funzione `chr(i)` restituisce il carattere in posizione `i` (cioè il carattere corrispondente al codice Unicode `i`)

Uso delle espressioni condizionali

Oltre che all'interno delle istruzioni condizionali (e **iterative**, come si vedrà più avanti), le **espressioni condizionali possono essere usate all'interno di un programma Python come tutte le altre espressioni** (numeri e stringhe):

- il loro valore **può essere assegnato a una variabile**; per esempio, assumendo che alla variabile `x` sia già stato assegnato un valore numerico:

```
risultato = x > 0
```

- il loro valore **può essere stampato nella finestra della *shell*** con l'istruzione `print`, per esempio:

```
print (x > 0)
```

- **possono essere scritte nella *shell***, nella quale verrà mostrato il loro valore (`True` oppure `False`)
- i valori `True` e `False` **possono essere assegnati a variabili**, per es.:

```
trovato = True
```

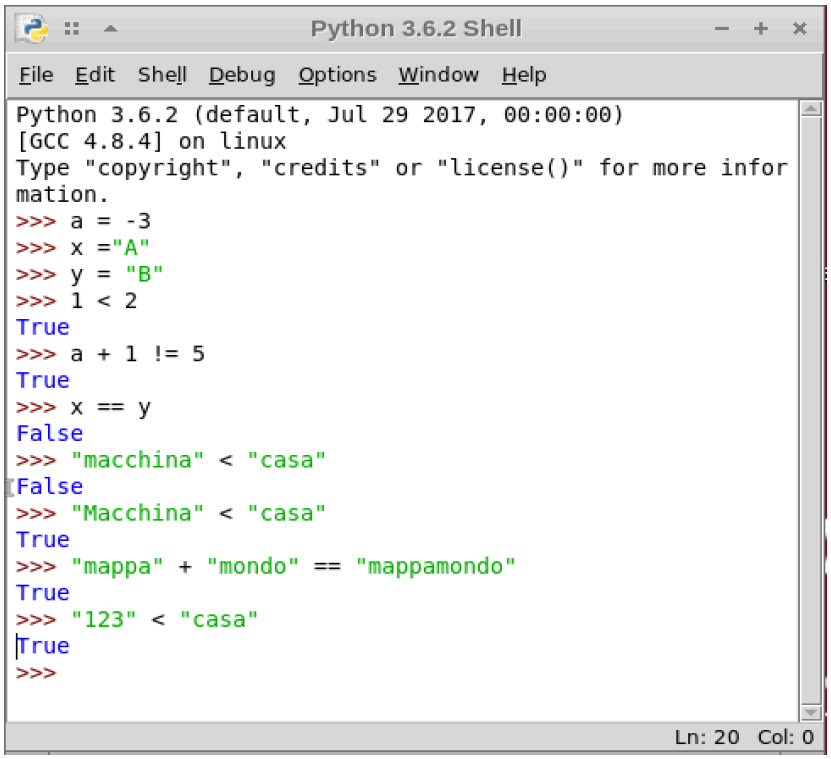
Espressioni condizionali: esempi

Espressioni condizionali: esempi

Si assume che a tutte le variabili che compaiono negli esempi seguenti sia già stato assegnato un valore.

- `1 < 2` produce `True`
- `a + 1 != 5` produce `False` se il valore associato alla variabile `a` è **4** (oppure 4.0), altrimenti produce `True`
- `x == y` produce `True` se le due variabili hanno lo stesso valore, altrimenti produce `False`
- `"macchina" < "casa"` produce `False`
- `"Macchina" < "casa"` produce `True`
- `"mappa" + "mondo" == "mappamondo"` produce `True`
- `"123" < "Casa"` produce `True`

Espressioni condizionali: esempi



Espressioni condizionali composte

Le espressioni condizionali viste finora si dicono semplici, poiché consistono in **un singolo confronto tra due valori**.

Così come nel linguaggio naturale, nei linguaggi di programmazione è possibile costruire espressioni condizionali composte, ottenute combinando espressioni condizionali **qualsiasi** per mezzo di connettivi logici come i seguenti:

- la congiunzione "e"
- la congiunzione "o", "oppure"
- l'avverbio "non"

Espressioni condizionali composte: sintassi

Sintassi:

- espr-cond$_1$ `and` espr-cond$_2$
- espr-cond$_1$ `or` espr-cond$_2$
- `not` espr-cond

- espr-cond$_1$, espr-cond $_2$ e espr-cond sono espressioni condizionali **qualsiasi** (quindi possono essere a loro volta espressioni composte)
- i simboli `and`, `or` e `not` sono detti operatori logici (o booleani), e corrispondono rispettivamente ai connettivi logici del linguaggio naturale "e", "oppure", "non"
- è possibile usare le parentesi tonde per definire l'ordine degli operatori logici

Espressioni condizionali composte: semantica

Anche le espressioni condizionali composte hanno come possibili valori i valori logici `True` e `False`.

Il valore di un'espressione condizionale composta è definito come segue:

- espr-cond$_1$ `and` espr-cond$_2$ — produce `True` se **entrambe** le espressioni hanno valore `True`, altrimenti produce `False`
- espr-cond$_1$ `or` espr-cond$_2$ — produce `True` se **almeno una** delle espressioni ha valore `True`, altrimenti produce `False`
- `not` espr-cond — produce `True` se l'espressione ha valore `False`, e viceversa

Comporre espressioni booleane: `and`

Se `x` e `y` sono booleani, possiamo completamente determinare i valori possibili di `x and y` usando la seguente «tavola di verità»:

x	y	x and y
False	False	False
False	True	False
True	False	False
True	True	True

Quindi: **`x and y` è `False` a meno che `x` e `y` non siano entrambe `True`.**

Analogie fra `and` e minimo

Se immaginiamo di rappresentare `False` con lo `0` e `True` con `1`, `x and y` è il minimo fra `x` e `y`

x	y	x and y
0	0	0
0	1	0
1	0	0
1	1	1

Comporre espressioni booleane: `or`

«tavola di verità» per l'operatore `or`

x	y	X or y
False	False	False
False	True	True
True	False	True
True	True	True

Quindi: **`x or y` è `True` a meno che `x` e `y` non siano entrambe `False`.**

Analogie fra `or` e massimo

`x or y` è il **massimo** fra `x` e `y`

x	y	x or y
0	0	0
0	1	1
1	0	1
1	1	1

Comporre espressioni booleane: `not`

«tavola di verità» per l'operatore `not`

x	not x
False	True
True	False

Quindi: **`not x` è `True` se `x` è `False` ed è `False` se `x` è `True`.**

Per questo motivo l'operatore `not` si chiama **negazione.**

Negazione ed operatori di confronto

Valgono le seguenti equivalenze:

`not x == y`	equivale a	`x != y`
`not x != y`	equivale a	`x == y`
`not x < y`	equivale a	`x >= y`
`not x <= y`	equivale a	`x > y`
`not x > y`	equivale a	`x <= y`
`not x >= y`	equivale a	`x < y`

Negazione di congiunzioni e disgiunzioni

Le tre operazioni booleane sono legate dalle relazioni seguenti che sono utili quando si deve negare una congiunzione x and y o una disgiunzione x or y:

`not ( x and y )` equivale a `not x or not y`

`not ( x or y )` equivale a `not x and not y`

Queste regole si chiamano «leggi di De Morgan» in onore al matematico Augustus De Morgan (1806-1871).

Espressioni condizionali composte: esempi

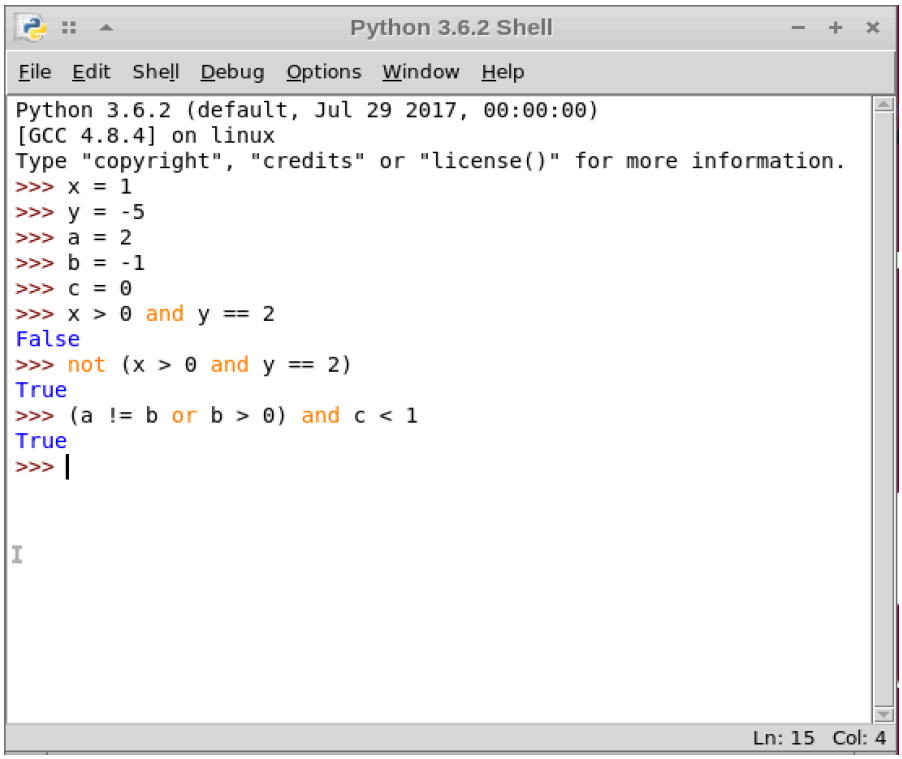
Come al solito, si assume che a tutte le variabili che compaiono negli esempi seguenti sia già stato assegnato un valore.

- `x > 0 and y == 2`
  è **vera** (produce True) se la variabile `x` contiene un numero positivo, **e** se la variabile `y` contiene il numero 2; altrimenti è **falsa** (produce False)

- `not (x > 0 or y == 2)`
  è **vera** se l'espressione tra parentesi (la stessa dell'esempio precedente) è **falsa**, e viceversa

- `(a != b or b >= 0) and c < 1`
  è **vera** se le variabili `a` e `b` contengono valori diversi **oppure** se b contiene un numero non negativo, **e contemporaneamente** la variabile `c` contiene un numero minore di 1; altrimenti è **falsa**

Espressioni condizionali: esempi



Esercizi

Determinare, per tutte le combinazioni possibili valori di valori booleani assegnabili a x, y e z, il valore delle seguenti espressioni: prima fatelo «a mano», poi usate Python per verificare la correttezza delle vostre risposte.

```
not not (x and x)
not( x and not x) or y
not (x or not x) and y
x and (y or z)
(x and y) or (x and z)
x or (y and z)
(x or y) and (x or z)
```

L'istruzione condizionale

Come si è detto in precedenza, le espressioni condizionali sono uno dei componenti delle **istruzioni** condizionali, che consentono di esprimere in un programma la scelta tra **due** diverse sequenze di istruzioni in base al verificarsi o meno di una certa **condizione** durante l'**esecuzione** dello stesso programma.

In linguaggio naturale, un'istruzione condizionale esprime la seguente richiesta all'esecutore di un algoritmo:

> **se** una data **condizione** è vera,
> **allora** esegui una certa sequenza d'istruzioni,
> **altrimenti** esegui un'altra sequenza d'istruzioni.

L'istruzione condizionale: sintassi

```
if espr-cond :
    sequenza di istruzioni 1
else :
    sequenza di istruzioni 2
```

- le parole-chiave `if` e `else` devono essere scritte **senza rientri**
- espr-cond è un'espressione condizionale
- sequenza di istruzioni 1 e sequenza di istruzioni 2 sono due sequenze di una o più istruzioni **qualsiasi**
- ciascuna di tali istruzioni deve essere scritta in una riga **distinta**, con un **rientro** di almeno un carattere; il rientro deve essere identico per **tutte** le istruzioni

L'istruzione condizionale: semantica

Se espressione condizionale è vera (cioè, se il suo valore è `True`), viene eseguita la sequenza di istruzioni 1 (le sue istruzioni vengono eseguite nello stesso ordine nel quale sono scritte).

Se invece espressione condizionale è falsa, viene eseguita la sequenza di istruzioni 2.

Si noti che **solo una** delle due sequenze di istruzioni viene eseguita.

Diagramma di flusso

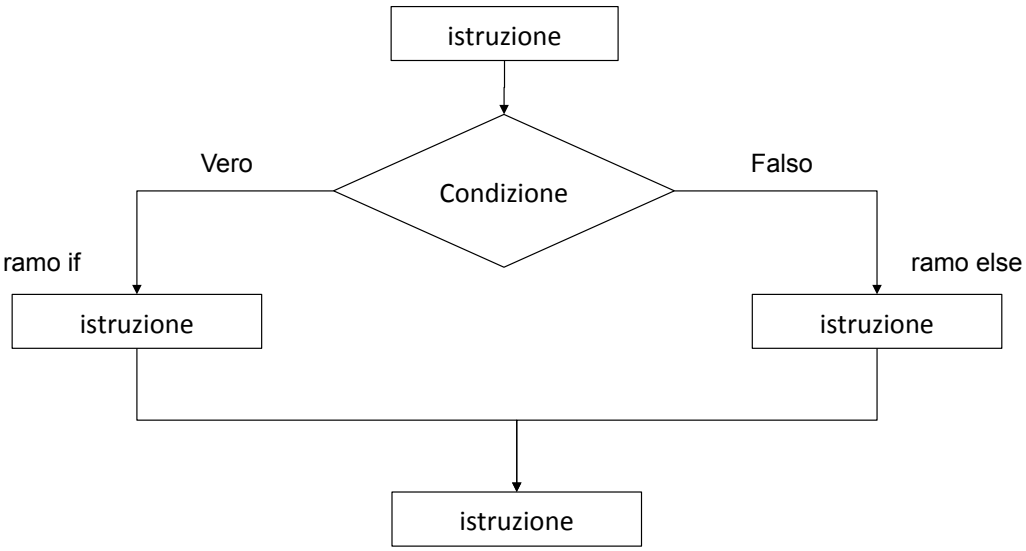
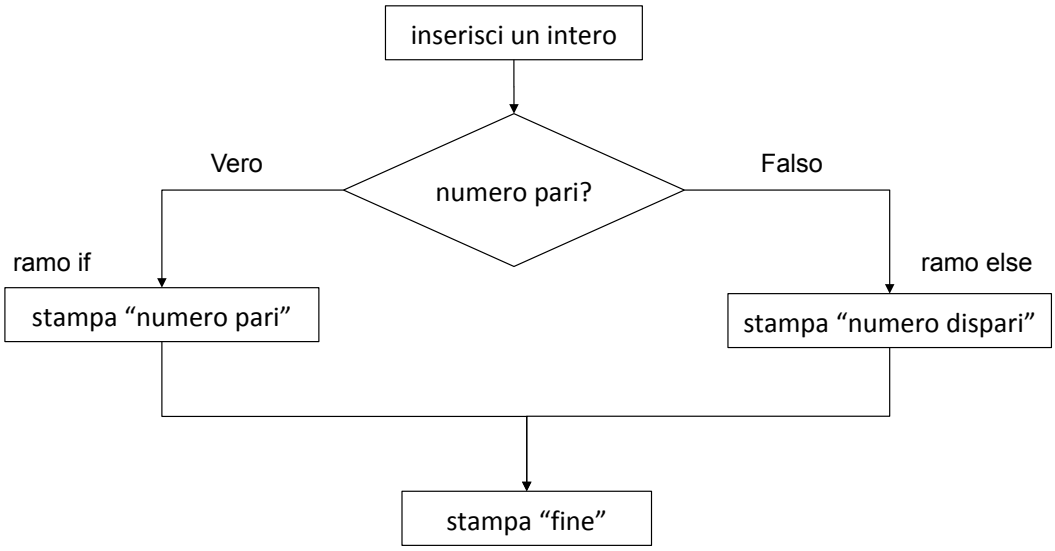


Diagramma di flusso: esempio



Livelli di rientro

```
numero = int(input("Inserisci un intero: "))
if (numero % 2 == 0):
      print("Il numero",numero," è pari")
else:
      print("Il numero",numero," è dispari")
print("fine")
```

0 1

Attenzione ai livelli di rientro

NOTA: dato che le istruzioni condizionali sono composte da più righe, anche quando non le si scrive all'interno di un programma si consiglia di scriverle in una finestra dell'*editor* piuttosto che nella *shell*.

L'istruzione condizionale: una variante

L'istruzione condizionale può essere scritta anche senza indicare una sequenza di istruzioni (introdotta dalla parola-chiave `else`) da eseguire nel caso in cui la condizione sia **falsa.**

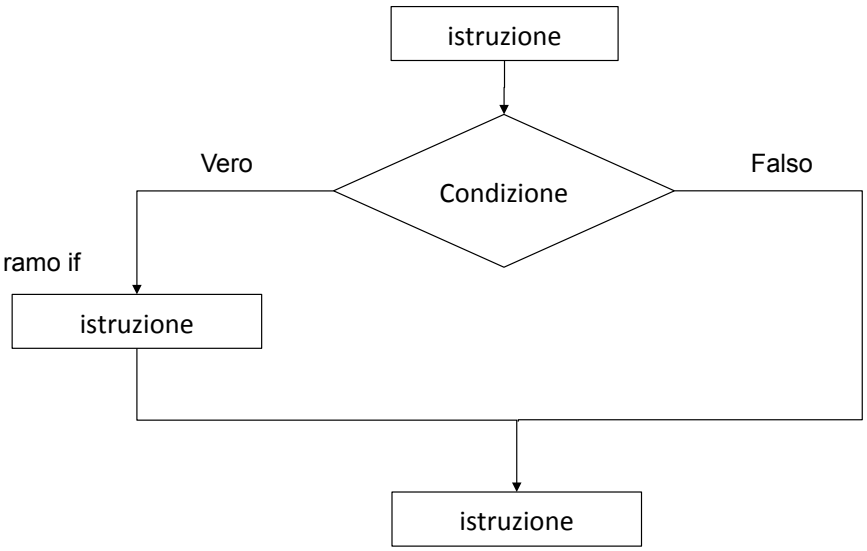
Questo è utile nei casi in cui un algoritmo preveda l'esecuzione di una certa sequenza di istruzioni solo nel caso in cui una data condizione sia vera.

Sintassi:

`if` espr-cond `:`
    sequenza di istruzioni

Semantica: se espr-cond è vera, allora viene eseguita la sequenza di istruzioni; in caso contrario, non viene eseguita nessuna istruzione.

Diagramma di flusso



L'istruzione condizionale: esempio

```
temperatura=float(input("inserisci la temperatura "))
if temperatura >= 25.0 :
     print("Fa caldo")
print("fine")
```

Se il valore della variabile `temperatura` (nel momento in cui l'istruzione condizionale viene **eseguita**) è un numero maggiore o uguale a 25, viene prima mostrato (nella *shell*) il messaggio

```
Fa caldo
```

e nella riga seguente il messaggio

```
fine
```

se `temperatura` è minore di 25 viene stampato solo

```
fine
```

Nota sulle istruzioni condizionali

I **rientri** sono l'**unico** elemento sintattico che indica quali istruzioni fanno parte di un'istruzione condizionale. Le istruzioni che **seguono** un'istruzione condizionale (senza farne parte) devono quindi essere scritte **senza rientri**.

Come esempio, si considerino le due sequenze di istruzioni:

```
if x > 0 :
   print ("A")
   print ("B")
```

```
if x > 0 :
   print "A"
print "B"
```

Nella sequenza a sinistra le due istruzioni `print` sono scritte con un rientro rispetto a `if`: questo significa che fanno **entrambe** parte dell'istruzione condizionale, e quindi verranno eseguite solo se la condizione `x > 0` sarà vera.

Nella sequenza a destra **solo la prima** istruzione `print` dopo `if` è scritta con un rientro, e quindi solo essa fa parte dell'istruzione condizionale; la seconda istruzione `print` verrà invece eseguita **dopo** l'istruzione condizionale, indipendentemente dal valore di verità della condizione `x > 0`.

Istruzioni condizionali: esercizio

Determinare che cosa viene stampato nella *shell* dalla sequenza di istruzioni mostrata in basso, nel caso in cui **prima** della loro esecuzione il valore della variabile `x` sia 1, e nel caso in cui il valore di `x` sia −1.

```
if x > 0 :
   print ("A")
   y = 2
if y <= 2 :
   print ("B")
   print ("C")
else :
   print ("D")
print ("E")
```

Soluzione

x=1

A
B
C
E

x=-1

Traceback (most recent call last):
File "C:/Users/rnico/Desktop/if.py", line 6, in <module>
if y <= 2 :
NameError: name 'y' is not defined
>>>

Istruzioni condizionali *nidificate*

Un'istruzione condizionale può contenere al suo interno istruzioni qualsiasi, e quindi anche altre istruzioni condizionali. Si parla in questo caso di istruzioni *nidificate* (o *annidate*).

L'uso di istruzioni condizionali nidificate consente di esprimere la scelta tra **più di due** sequenze di istruzioni alternative.

Un'istruzione condizionale nidificata all'interno di un'altra si scrive con la stessa sintassi mostrata sopra; questo implica che:

- le parole-chiave `if` e (se presente) `else` devono essere scritte con un **rientro** rispetto a quelle dell'istruzione condizionale che le contiene
- le sequenze di istruzioni che seguono `if` e `else` devono essere scritte con un ulteriore rientro

Livelli di rientro

```
voto=int(input("Immetti il voto di un esame superato: "))
if voto>=18 and voto<=30 :
      if voto>=24 :
            print("Il risultato e' buono")
      else :
            print("Il risultato e' sufficiente")
else :
      print("voto non corretto")
```

Attenzione ai livelli di rientro

Esercizio: modificare il programma in modo che sia ammesso anche il valore `31` (che rappresenta la lode), ed il programma stampi `"Il risultato e' eccellente"` nel caso in cui voto sia uguale a `31`.

Istruzioni condizionali nidificate: esempio

Assumendo che alle variabili `x` e `y` sia già stato assegnato un valore, quello che segue è un esempio di istruzione condizionale (`if y == 1 : ...`) nidificata all'interno di un'altra (`if x == 0 : ...`).
Si noti che entrambe contengono anche la parte `else`.

```
if x > 0 :
   print ("A")
   if y == 1 :
      print ("B")
   else :
      print ("C")
   print ("D")
else :
   print ("E")
```

Istruzioni condizionali nidificate: esempio

Per capire meglio una sequenza di istruzioni come quella mostrata in precedenza è utile **eseguirla** passo dopo passo, così come verrebbe effettivamente eseguita dal calcolatore.

A questo scopo bisogna individuare prima di tutto le due sequenze di istruzioni che vengono eseguite nel caso in cui l'espressione condizionale della **prima** istruzione `if` (quella "esterna") sia vera, e nel caso in cui tale espressione sia falsa.

Questa informazione è sempre fornita **esclusivamente** dai **rientri** all'inizio di ciascuna riga, secondo la sintassi già descritta.

Istruzioni condizionali nidificate: esempio

Le due sequenze di istruzioni contenute nella prima istruzione `if` sono evidenziate in basso in magenta (per il caso in cui `x > 0` è vera) e in blu (per il caso in cui `x > 0` è falsa).

```
if x > 0 :
    print ("A")
    if y == 1 :
        print ("B")
    else :
        print ("C")
    print ("D")
else :
    print ("E")
```

Istruzioni condizionali nidificate: esempio

In particolare, la sequenza corrispondente al caso in cui `x > 0` sia vera è composta da **tre** istruzioni:

- l'istruzione `print ("A")`
- l'istruzione condizionale (nidificata) `if y == 1 : ...`
- l'istruzione `print ("D")`

Si noti che l'istruzione `print ("D")` **non** fa parte dell'istruzione condizionale `if y == 1 : ...`, e quindi verrà eseguita (nel caso in cui `x > 0` sia vera) **dopo** di essa, indipendentemente dal valore di verità della condizione `y == 1`.

Istruzioni condizionali nidificate: esempio

In conclusione:

- se `x > 0` è vera:
  - prima viene stampato `A`
  - poi, se `y == 1` è vera viene stampato `B`, altrimenti viene stampato `C`
  - poi viene stampato `D`
- se invece `x > 0` è falsa, viene solo stampato `E`

Istruzioni condizionali nidificate: esercizio

Determinare ciò che viene stampato nella *shell* dall'esecuzione delle istruzioni dell'esempio precedente, per ciascuno dei seguenti valori delle variabili `x` e `y`:

- `x = -1, y = -1`
- `x = 2, y = 1`
- `x = -2, y = 1`
- `x = 3, y = 2`

Soluzione

- `x = -1, y = -1` → `E`
- `x = 2, y = 1` → `A B D`
- `x = -2, y = 1` → `E`
- `x = 3, y = 2` → `A C D`

L'istruzione condizionale: la variante elif

Un'altra variante dell'istruzione if è data dalla presenza della parola chiave `elif`.

Si tratta di un'abbreviazione di `else if` e significa:

esegui il blocco di istruzioni tabulato sotto elif se la condizione necessaria al primo if è falsa e la condizione elif è vera.

Sintassi:

```
if  espr-cond1 :
    sequenza di istruzioni1
elif espr-cond2 :
    sequenza di istruzioni2
```

Semantica:
se espr-cond1 è vera, allora viene eseguita la sequenza di istruzioni1; in caso contrario, se espr-cond2 è vera viene eseguita la sequenza di istruzioni2

L'istruzione condizionale: la variante elif - esempio

```
a = int(input("Valore?"))
if a > 5:
  print (a," > ",5)
elif a <= 2:
  print (a," <= ",2)
else:
  print ("Compreso tra 2 e 5")
```

Si noti che un programma che fa uso di elif può sempre essere riscritto usando solo if-else, ma è necessario annidare le condizioni

```
a = int(input('Valore?'))
if a > 5:
  print (a," > ",5)
else :
  if a <= 2:
     print (a," <= ",2)
  else:
     print ("Compreso tra 2 e 5")
```

Uso di elif per condizioni mutuamente esclusive

Esempio: in base al valore della temperatura (intero) stampare un messaggio secondo la seguente tabella:

temperatura t	messaggio
$30 < t$	molto caldo
$20 < t \leq 30$	caldo
$10 < t \leq 20$	gradevole
$t \leq 10$	freddo

```
temperatura=float(input("inserisci la temperatura "))
if temperatura > 30 :
     print("molto caldo")
elif temperatura <= 30 and temperatura > 20 :
     print("caldo")
elif temperatura <= 20 and temperatura > 10 :
     print("gradevole")
else :
     print("freddo”)
```

Esempi di programmi contenenti istruzioni condizionali

Un programma che calcola il valore assoluto di un numero acquisito attraverso la tastiera.
File: `valore_assoluto_1.py`

```
n = int(input("Inserire un numero: "))
if n > 0 :
    print ("Valore assoluto:", n)
else:
    print ("Valore assoluto:", -n)
```

Esempi di programmi contenenti istruzioni condizionali

Una versione alternativa, in cui si usa un'istruzione condizionale senza la parte `else`.
File: `valore_assoluto_2.py`

```
n = int(input("Inserire un numero: "))
if n < 0 :
      n = - n
print ("Valore assoluto:", n)
```

Esempi di programmi contenenti istruzioni condizionali

Un programma che acquisice i coefficienti di un'equazione di primo grado, $ax + b = 0$, e ne calcola la soluzione se questa esiste ed è unica, altrimenti stampa un messaggio opportuno.

File: `eq_primo_grado.py`

```
a = int(input("Inserire il coefficiente a: "))
b = int(input("Inserire il coefficiente b: "))
if a != 0 :
    print ("La soluzione e'", - b / a)
else :
    print ("La soluzione non esiste o non e' unica.")
```

Esempi di programmi contenenti istruzioni condizionali

Un programma che acquisice i coefficienti di un'equazione di secondo grado, $ax^2 + bx + c = 0$, e **determina se le radici siano reali oppure complesse**, stampando un opportuno messaggio.

File: `eq_secondo_grado_1.py`

```
a = int(input("Inserire il coefficiente a: "))
b = int(input("Inserire il coefficiente b: "))
c = int(input("Inserire il coefficiente c: "))
delta = (b ** 2) - (4 * a * c)
if delta >= 0 :
    print ("Le radici sono reali.")
else :
    print ("Le radici sono complesse.")
```

Esempi di programmi contenenti istruzioni condizionali

Una variante dello stesso programma, che **stampa anche i valori delle radici**.

File: `eq_secondo_grado_2.py`

```
a = int(input("Inserire il coefficiente a: "))
b = int(input("Inserire il coefficiente b: "))
c = int(input("Inserire il coefficiente c: "))
delta = (b ** 2) - (4 * a * c)
if delta >= 0 :
    print ("Le radici sono reali.")
    print ("x1 =", (-b + (delta ** 0.5)) / (2 * a))
    print ("x2 =", (-b - (delta ** 0.5)) / (2 * a))
else :
    print ("Le radici sono complesse.")
    print ("x1 =", -b/(2.0 * a), ((-delta) ** 0.5) / (2 * a))
    print ("x2 =", -b/(2.0 * a), - ((-delta) ** 0.5) / (2 * a))
```

Esempi di programmi contenenti istruzioni condizionali

Un'altra variante, che determina se le radici siano reali e distinte, reali e coincidenti, oppure complesse coniugate. In questo esempio si sono usate tre istruzioni condizionali in sequenza (non nidificate), corrispodenti a **tre condizioni mutuamente esclusive (quindi una sola di tali istruzioni viene eseguita)**.

File: `eq_secondo_grado_3.py`

```
a = int(input("Inserire il coefficiente a: "))
b = int(input("Inserire il coefficiente b: "))
c = int(input("Inserire il coefficiente c: "))
delta = (b ** 2) - (4 * a * c)
if delta > 0 :
    print ("Le radici sono reali e distinte.")
if delta == 0 :
    print ("Le radici sono reali e coincidenti.")
if delta < 0 :
    print ("Le radici sono complesse coniugate.")
```

Esempi di programmi contenenti istruzioni condizionali

Una quarta variante, nella quale si usano **istruzioni condizionali nidificate**.

File: `eq_secondo_grado_4.py`

```
a = int(input("Inserire il coefficiente a: "))
b = int(input("Inserire il coefficiente b: "))
c = int(input("Inserire il coefficiente c: "))
delta = (b ** 2) - (4 * a * c)
if delta > 0 :
    print ("Le radici sono reali e distinte.")
else :
    if delta == 0 :
        print ("Le radici sono reali e coincidenti.")
    else:
        print ("Le radici sono complesse coniugate.")
```

Esempi di programmi contenenti istruzioni condizionali

Il programma mostrato di seguito acquisisce tre numeri, e determina se essi possano rappresentare le lunghezze dei lati di un triangolo (**cioè se siano tutti positivi, e se ciascuno sia minore della somma degli altri due**); in caso affermativo determina se si tratti di un triangolo **equilatero**, **isoscele** o **scaleno**, altrimenti stampa un messaggio opportuno.

Si noti che in linguaggio Python è possibile suddividere **una** istruzione in più righe, inserendo il carattere \ (*backslash*) nel punto in cui si interrompe una riga. Questa possibilità è stata sfruttata per suddividere in due righe l'espressione condizionale della prima istruzione `if`.

Nella prosecuzione di una riga si può inserire un rientro qualsiasi (anche nessuno). Per garantire la leggibilità del programma è però buona norma usare un rientro coerente con il contenuto della riga precedente.

Esempi di programmi contenenti istruzioni condizionali

File: `triangoli_1.py`

```
a = int(input("Inserire un numero: "))
b = int(input("Inserire un altro numero: "))
c = int(input("Inserire l'ultimo numero: "))
if (a > 0) and (b > 0) and (c > 0) and \
   (a < b + c) and (b < a + c) and (c < a + b) :
    if (a == b) and (b == c) :
        print ("Equilatero")
    else :
        if (a == b) or (b == c) or (a == c) :
            print ("Isoscele")
        else :
            print ("Scaleno")
else :
    print ("Non possono rappresentare i lati di un triangolo.")
```

Esempi di programmi contenenti istruzioni condizionali

Di seguito si mostra una seconda versione dello stesso programma, con una diversa espressione condizionale nella prima istruzione `if` (tale espressione verifica se i tre numeri **non** corrispondano alle lunghezze dei lati di un triangolo).

Questo consente di spostare nella parte `else` della stessa istruzione la determinazione del tipo di triangolo, rendendo il programma più leggibile.

Esempi di programmi contenenti istruzioni condizionali

File: `triangoli_2.py`

```
a = int(input("Inserire un numero: "))
b = int(input("Inserire un altro numero: "))
c = int(input("Inserire l'ultimo numero: "))
if not ((a > 0) and (b > 0) and (c > 0) and \
        (a < b + c) and (b < a + c) and (c < a + b)) :
    print ("Non possono rappresentare i lati di un triangolo.")
else :
    if (a == b) and (b == c) :
        print ("Equilatero")
    else :
        if (a == b) or (b == c) or (a == c) :
            print ("Isoscele")
        else :
            print ("Scaleno")
```