Istruzioni Condizionali

Mirko Cesarini - Dario Pescini
nome.cognome@unimib.it

Università di Milano Bicocca

Esercizio

- Prendiamo il seguente programma

```
1  a=int(input("Immetti il coefficiente a: "))
2  b=int(input("Immetti il coefficiente b: "))
3  c=int(input("Immetti il coefficiente c: "))
4  print("Data l'equazione algebrica ")
5  print(str(a)+"*X^2+"+str(b)+"*X+"+str(c)+"=0 ")
6  delta = b*b - 4*a*c
7  rad_delta=delta**0.5
8  x1=-(b - rad_delta)/(2*a)
9  x2=-(b + rad_delta)/(2*a)
10 print ("Le soluzioni sono")
11 print('x1: '+str(x1))
12 print('x2: '+str(x2))
```

- NB:
  - $X_{1,2} = (-b \pm \sqrt{(b^2 - 4ac)})/2a$
  - $delta^{1/2} = \sqrt{delta}$

Test

- Proviamo ad eseguire lo script, inserendo i seguenti valori
  - a=2, b=4, c=2 OK
  - a=2, b=3, c=2 errore

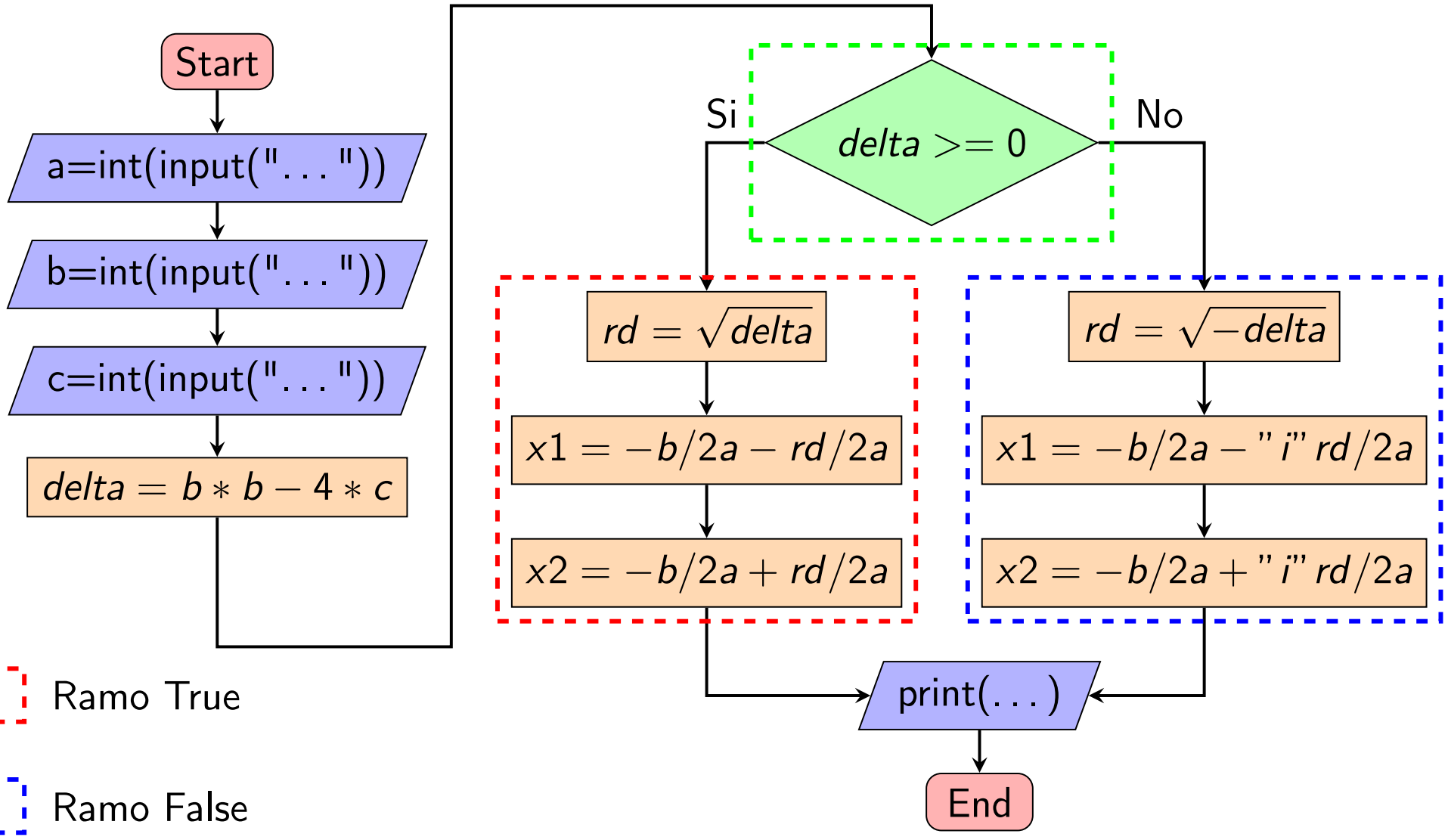
```
Traceback (most recent call last):
...
File ..., line 7, ...    rad_delta=delta**0.5
... ValueError: negative number cannot be
raised to a fractional power
```

- Da dove nasce l'errore?

```
3  ...
4  print("Data l'equazione algebrica ")
5  print(str(a)+"*X^2+"+str(b)+"*X+"+str(c)+"=0 ")
6  delta = b*b - 4*a*c
7  rad_delta=delta**0.5
8  x1=-(b - rad_delta)/(2*a)
9  x2=-(b + rad_delta)/(2*a)
10 print ("Le soluzioni sono")
11 print('x1: '+str(x1))
12 print('x2: '+str(x2))
```

- Il problema (con a=2, b=3, c=2) nasce nelle istruzioni:
  $delta = b * b - 4 * a * c$
  $rad\_delta = delta * *0.5$
- Quando delta assume un valore negativo . . .
- . . . Python non esegue la radice quadrata di un numero negativo (si potrebbero utilizzare le librerie per manipolare i numeri complessi . . . , ma per ora non le consideriamo)
- Cosa si può fare?
  - Il programma dovrebbe comportarsi in modo diverso a seconda del valore della variabile delta

Istruzioni condizionali



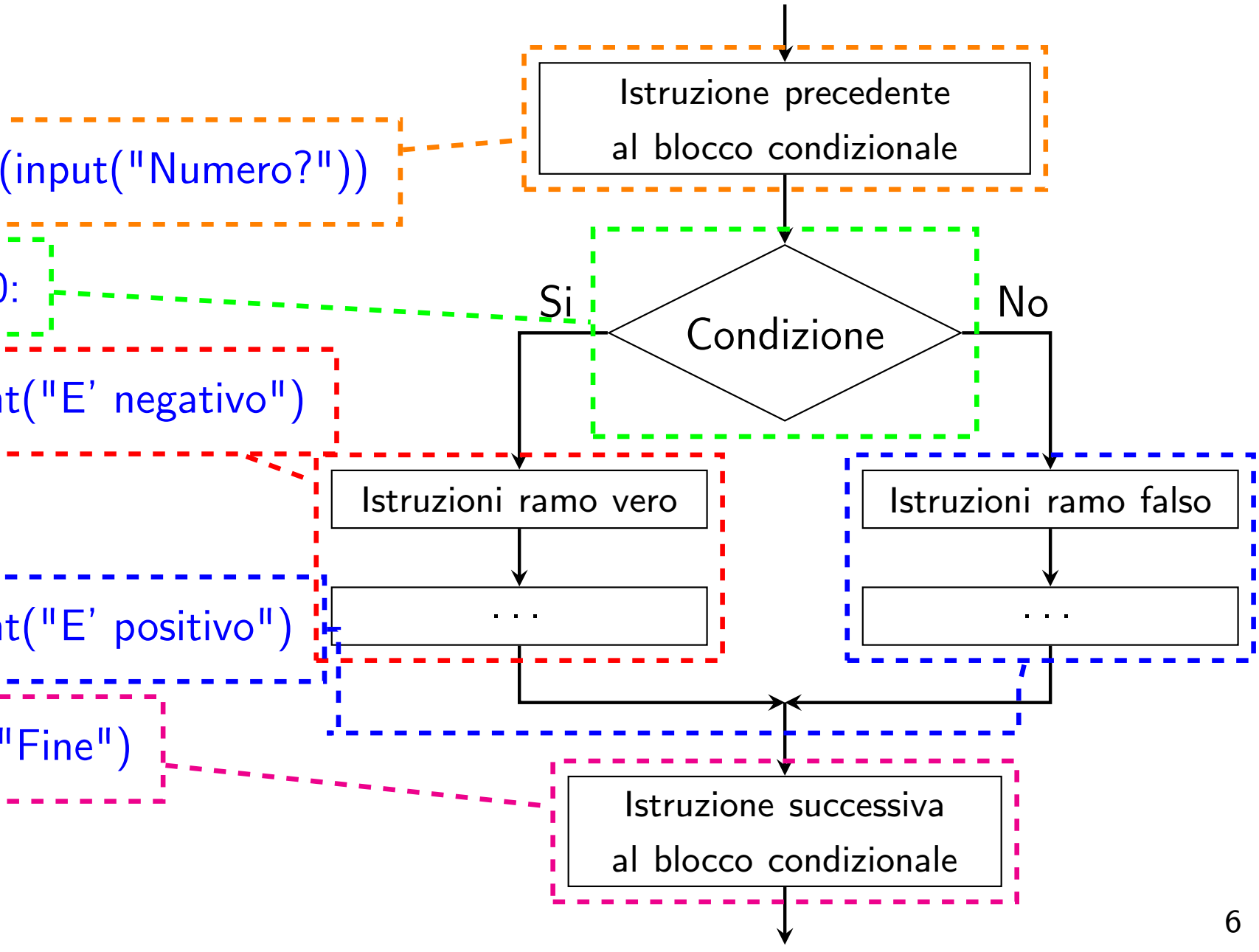
Ramo True

Ramo False

Istruzione condizionale

Istruzioni condizionali in Python

```
n=int(input("Numero?"))
if n<0:
    print("E' negativo")
else:
    print("E' positivo")
print("Fine")
```



Ramificazioni

```
1  if condizione:
2      prina riga di istruzioni  # Blocco
3      ...                       # di istruzioni
4      ultima riga di istruzioni # indentato
5  else:
6      prima riga di istruzioni  # Blocco
7      ...                       # di istruzioni
8      ultima riga di istruzioni # indentato
9  prima istruzione fuori dal blocco condizionale
```

- Il valore booleano della condizione determina quale dei due blocchi sarà eseguito

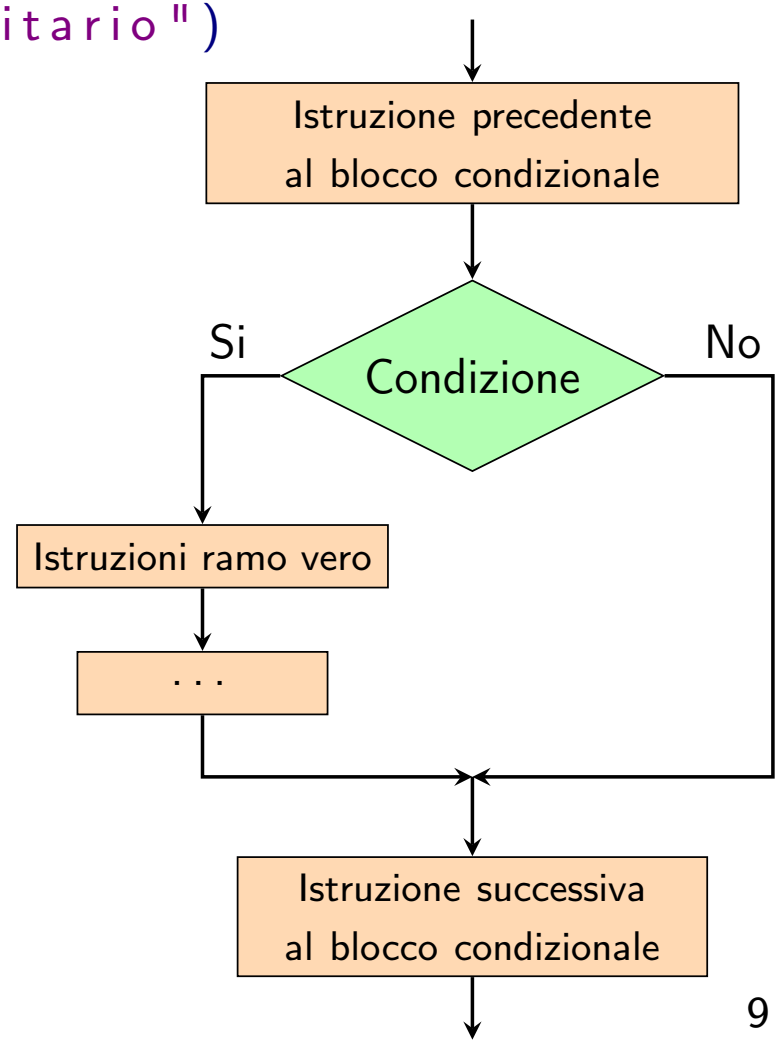
Inizio e fine del "ramo vero"

```
1  if a>=0:
2      print("Numero positivo")                # Ramo
3      print("Valore assoluto: "+str(a))       # "vero"
4  else:
5          print("Numero negativo")              # Ramo
6          print("Valore assoluto: "+str(-a))    # "falso"
7  print("Fine programma")
```

- Le due alternative sono chiamate ramificazioni, rappresentano flussi alternativi di esecuzione
- Python usa l'indentazione per distinguere dove iniziano e terminano le due ramificazioni
- Non è necessario che i due rami siano indentati allo stesso modo, basta che i blocchi siano indentati in maniera coerente al loro interno
- Nella prima riga i : indicano dove termina la condizione

Il ramo else è opzionale

```
1 n = int(input("Inserisci un numero "))
2 if n < 0:
3     n = -n #cambio segno al valore
4 print("adesso riprende il flusso unitario")
5 print(n)
```



L'istruzione pass

- Il *ramo vero* deve sempre contenere qualcosa

```
1 if x>0:
2 else:
3     print('Ciao')
```

else: . . . IndentationError: expected an indented block (. . . prima dell'else)

- In questo caso può essere usata l'istruzione pass, che funge da segnaposto.

```
1 if x>0:
2     pass
3 else:
4     print('Ciao')
```

- pass è utile anche durante lo sviluppo, in attesa che il codice mancante venga scritto

- Non è buono stile scrivere una condizione come la seguente (il pass è tollerato solo se usato temporaneamente)

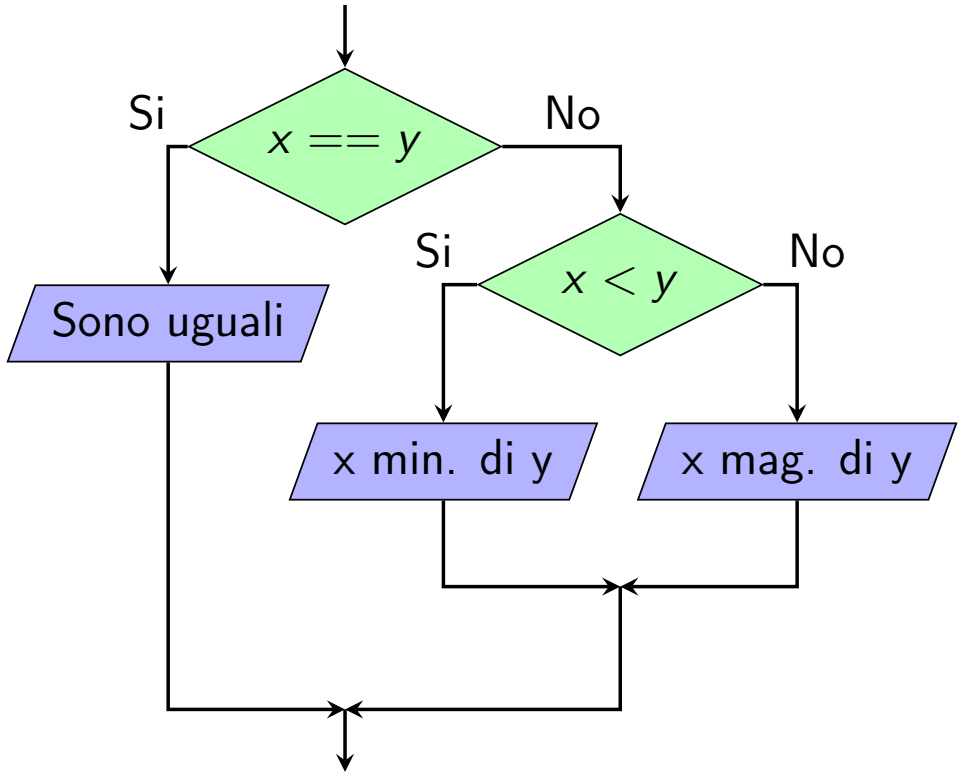
```
if x>=0:
    pass
else:
    print('Negativo')
```

- Sarebbe meglio riscrivere le istruzioni qua sopra in questo modo

```
if x<0: # equivalente a not(x>=0)
    print('Negativo')
```

Condizioni annidate

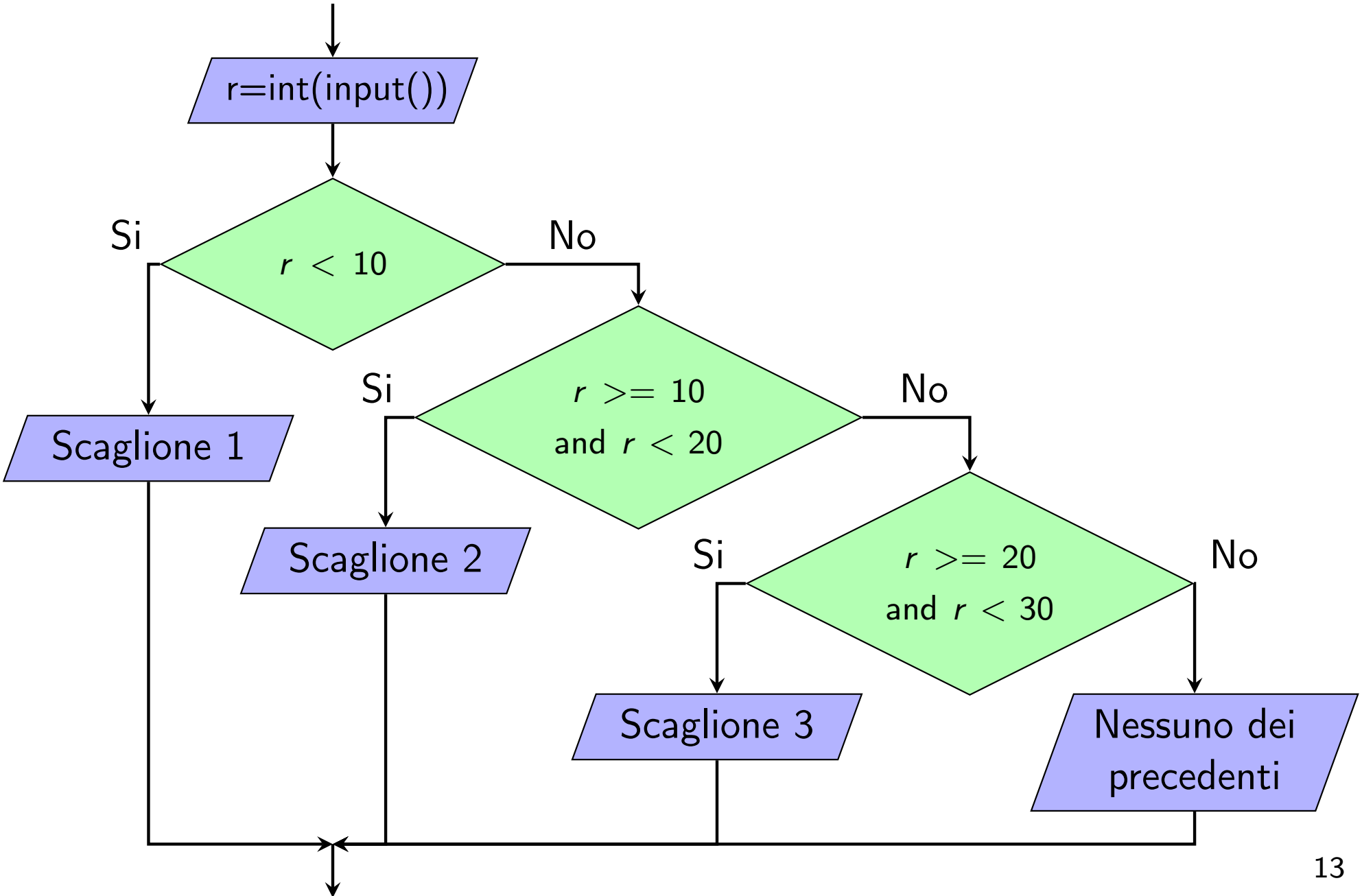
- Un'espressione condizionale può anche essere inserita nel corpo di un'altra espressione condizionale. Si parla in questo caso di condizione annidata




```
1 if x == y:
2     print("sono uguali")
3 else:
4     if x < y:
5         print(str(x)+"e' minore di"+str(y))
6     else:
7         print(str(x)+"e' maggiore di"+str(y))
```

- Python usa l'indentazione per decidere a quale ramificazione appartiene un'istruzione

Istruzioni alternative a più vie



Istruzioni alternative a più vie

```
#Implementazione 1
r=int(input())
if r<10:
    print("scaglione 1")
else:
    if r>=10 and r<20:
        print("scaglione 2")
    else:
        if r>=20 and r<30:
            print("scaglione 3")
        else:
            print("nessuno ...")
print("fine")
```

```
#Implementazione 2
r=int(input())
if r < 10:
    print("scaglione 1")
elif r>=10 and r<20:
    print("scaglione 2")
elif r>=20 and r<30:
    print("scaglione 3")
else:
    print("nessuno dei prec.")
print("fine")
```

- Secondo voi, qualche condizione può essere semplificata?

Semplificazione delle condizioni

```python
#Implementazione 1
r=int(input())
if r<10:
    print("scaglione 1")
else:
    if r<20: # r>=10 and ...
      print("scaglione 2")
    else:
      if r<30: # r>=20 and ...
        print("scaglione 3")
      else:
        print("nessuno ...")
print("fine")
```

```python
#Implementazione 2
r=int(input())
if r < 10:
    print("scaglione 1")
elif r<20: # r>=10 and ...
    print("scaglione 2")
elif r<30: # r>=20 and ...
    print("scaglione 3")
else:
    print("nessuno dei prec.")
print("fine")
```

- Nei commenti le parti eliminate
- Le parti eliminate non servono (in questi 2 casi)

elif

- elif è l'abbreviazione di *else if*
- Non c'è alcun limite al numero di istruzioni elif
- In un blocco if . . . elif . . . l'istruzione *else*
  - è facoltativa,
  - se presente deve essere l'ultima dell'elenco
  - rappresenta l'azione da eseguire quando nessuna delle condizioni precedenti è stata soddisfatta
- NB: elif richiede una condizione booleana, else no

```
1 ...
2 if a < 10:
3     ...
4 elif a<20:
5     ...
6 else:
7     ...
```

Condizioni

- Scritte per mezzo di espressioni booleane
- Un'espressione booleana è un'espressione che può assumere solo valore vero o falso.
- Nota bene:
  - i valori booleani in python sono
    - True
    - False
  - e non
    - ~~"True"~~
    - ~~"False"~~

    In python le espressioni booleane si costruiscono usando: variabili, valori, operatori di confronto, ...

Operatori di confronto

- Operatore di confronto: == confronta due valori e produce un risultato di tipo booleano:
- Il >>> qua sotto rappresenta la riga di comando di python

```
>>> 5 == 5
True
```

```
>>> 5 == 6
False
```

Gli altri operatori sono:

```
1 x != y  # x diverso da y
2 x > y   # x maggiore di y
3 x < y   # x minore di y
4 x >= y  # x maggiore o uguale a y
5 x <= y  # x minore o uguale a y
```

Operatori logici

- AND
  per esempio, (x>0) and (x<10) è vera se e solo se le due condizioni sono valide contemporaneamente, cioè se x è più grande di 0 e contemporaneamente minore di 10.
- OR
  (x==0) or (x==10) è vera se almeno una delle condizioni è verificata, cioè se x è uguale a 0 oppure a 10
- NOT
  L'operatore not nega il valore di un'espressione booleana, trasformando in falsa un'espressione vera e viceversa.
  Così se 5>3 è vera, not(5>3) è falsa.