Cicli

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

Università di Milano Bicocca

Calcolo della media

- Immaginiamo di dover calcolare la media . . .

. . . di 2 numeri interi

```
1 a1=int(input('Numero? '))
2 a2=int(input('Numero? '))
3 me=float(a1+a2)/2 # il float
4 #serve in Py2 per avere
5 #risultati con la virgola
6 print('Media: %f' % (me))
```

```
Numero? 1
Numero? 2
Media: 1.500000
```

. . . di 3 numeri interi

```
1 a1=int(input('Numero? '))
2 a2=int(input('Numero? '))
3 a3=int(input('Numero? '))
4 me=(a1+a2+a3)/3.0
5 print('Media: %f' % (me))
```

. . . di n numeri interi

```
1 a1=int(input('Numero? '))
2 a2=int(input('Numero? '))
3 ...
4 an=int(input('Numero? '))
5 me=(a1+a2+...+an)/float(...)
6 print('Media: %f' % (me))
```

- Svantaggio dell'approccio usato: al variare del numero di elementi, occorre modificare il codice in diversi punti

- Altro svantaggio: Nel modificare il codice, richio di commettere qualche errore

Media di 2 numeri interi

```
1 a1=int(input('Numero? '))
2 a2=int(input('Numero? '))
3 me=(a1+a2)/2.0
4 print('Media: %f' % (me))
```

```
Numero? 1
Numero? 2
Media: 1.500000
```

Media di 4 numeri interi

```
1 a1=int(input('Numero? '))
2 a2=int(input('Numero? '))
3 a3=int(input('Numero? '))
4 a4=int(input('Numero? '))
5 me=(a1+a2+a3+a4)/3.0  # errore
6 print('Media: %f' % (me))
```

- C'è qualche errore?
- A riga 5, dovrebbe esserci 4.0 al denominatore

```
1  num=0 #numeratore
2  den=0 #denominatore
3
4  v=int(input('Numero? '))
5  num=num+v
6  den=den+1
7
8  ...
9
10 v=int(input('Numero? '))
11 num=num+v
12 den=den+1
13
14 me = num / float(den)
15 print('Media: %f' % (me))
```

- Le variabili *num* e *den* sono due contatori
- Uso dei contatori
  - den=den+1 # dal punto di vista matematico non avrebbe senso
  - Dal punto di vista informatico:
    - prendi il (vecchio) valore di *den*
    - aumentalo di 1
    - memorizza il risultato come (nuovo) valore di *den*
    - di fatto *den* viene aumentato di 1
  - num=num+v #aumento num del valore v

- Miglioramento: il blocco di codice (da riga 4 a riga 6) può essere ripetuto senza modifiche (copia incolla)
- Tuttavia situazione ancora non ottimale

Focus sui contatori

Le istruzioni

- num=num+v
- den=den+1

possono essere riscritte anche nel modo seguente:

- num+=v
- den+=1

ottenendo gli stessi risultati

```
1 var=0
2 var=var+1
3 print('1ma volta',var)
4 var+=1
5 print('2da volta',var)
```

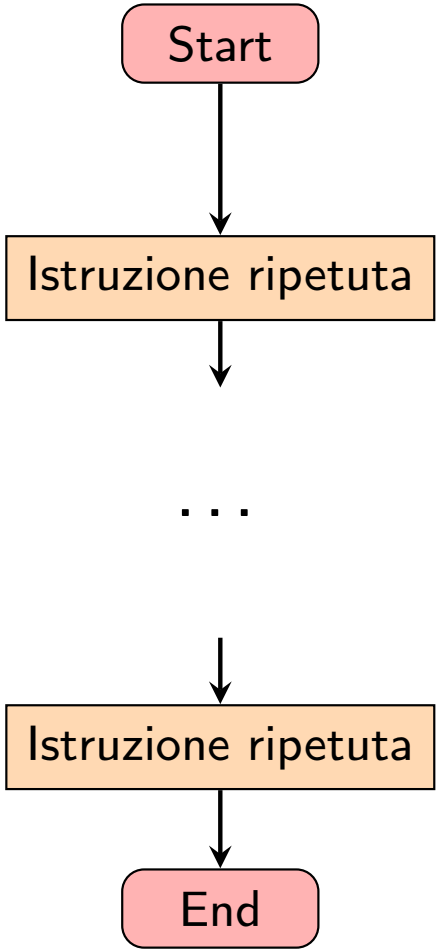
```
('1ma volta', 1)
('2da volta', 2)
```

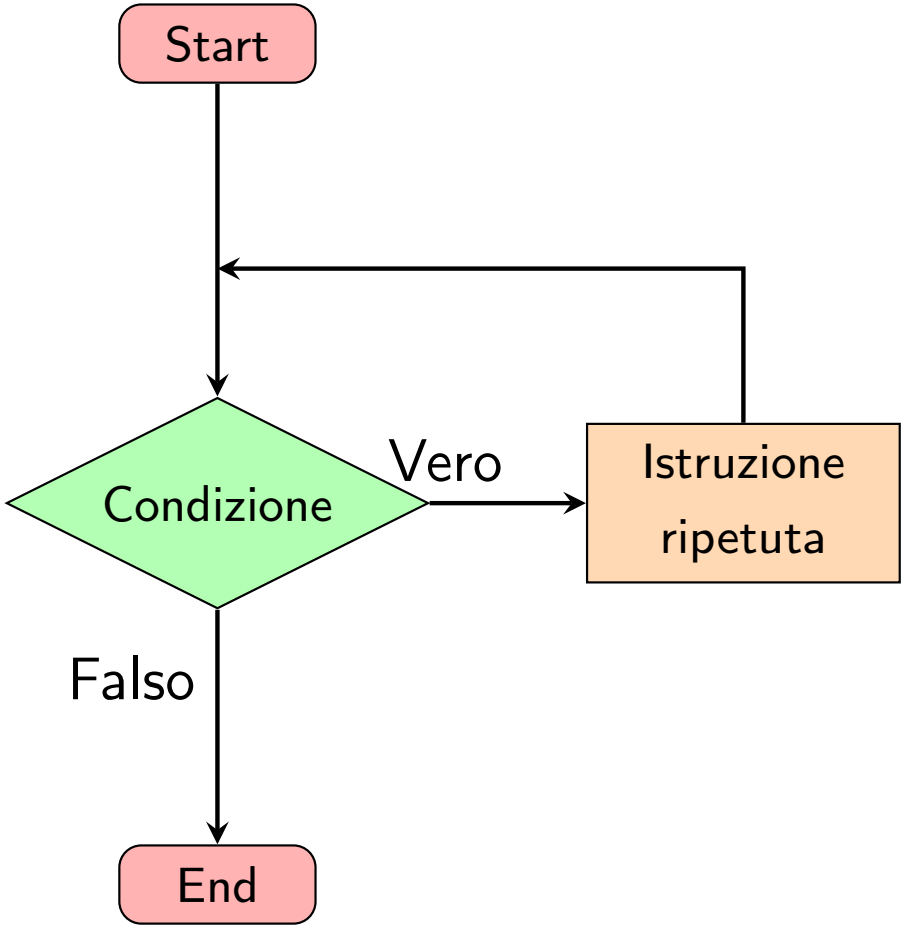
- Riprendiamo il codice qua a lato
  - Blocco di istruzioni ripetuto tante volte quanti sono i numeri della media
  - Se cambia la numerosità dei valori, devo modificare il programma
  - Come gestire 1000 numeri?
- Le ripetizioni di codice vanno evitate
  - Se trovo un errore devo modificare N blocchi
  - Se mi dimentico di aggiornare/modificare qualche blocco?

```
1  num=0 #numeratore
2  den=0 #denominatore
3
4  v=int(input('Numero? '))
5  num=num+v
6  den=den+1
7
8  ...
9
10 v=int(input('Numero? '))
11 num=num+v
12 den=den+1
13
14 me = num / float(den)
15 print('Media: %f' % (me))
```

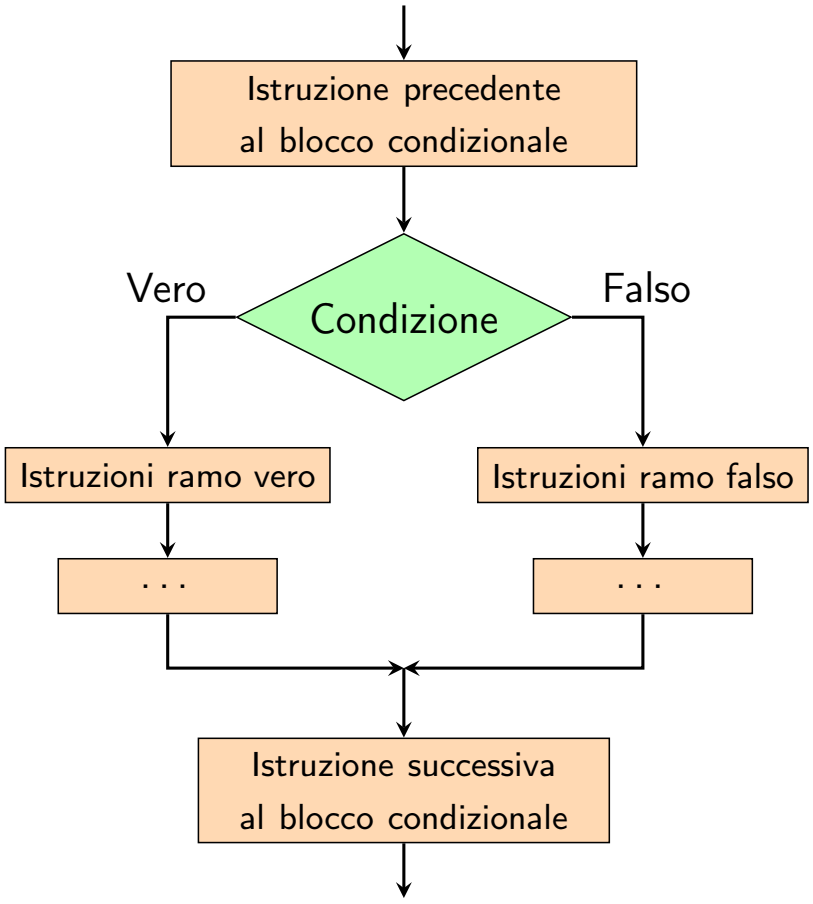
Come risolvere il problema

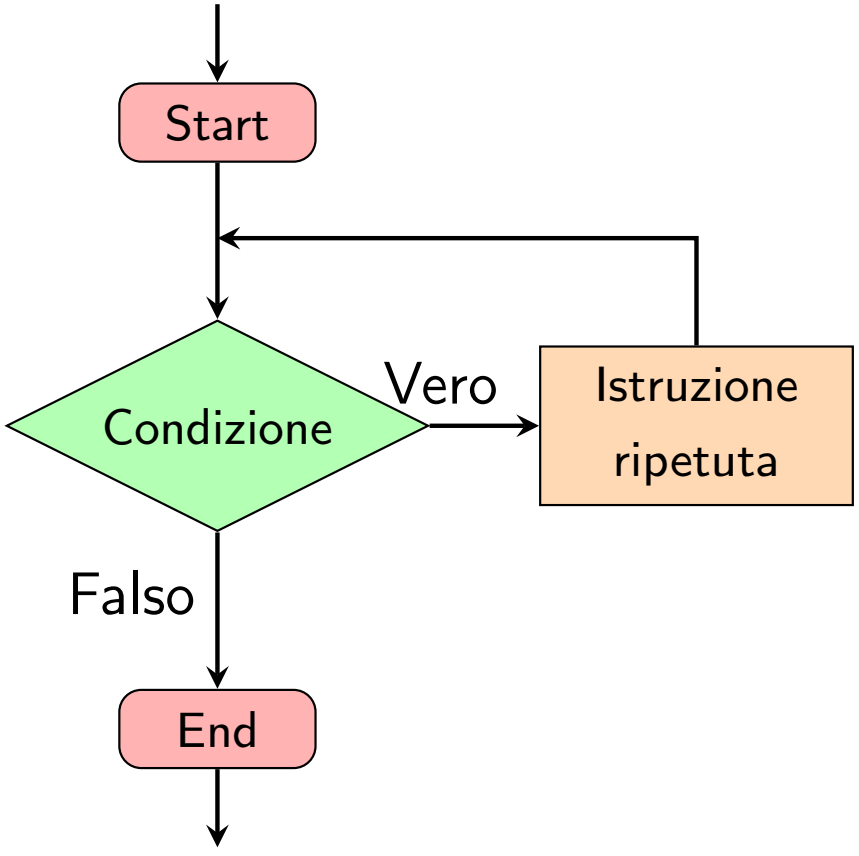
- La soluzione a destra è chiamata ciclo
- La *condizione* deve assumere valore VERO fintanto che c'è da ripetere l'istruzione . . . per poi diventare falsa





Posso usare le istruzioni condizionali per realizzare un ciclo?





- Risposta: NO. Anche se entrambe le tipologie di istruzioni usano le espressioni booleane . . .
- . . . le istruzioni condizionali (in Python) non permettono di "tornare indietro"

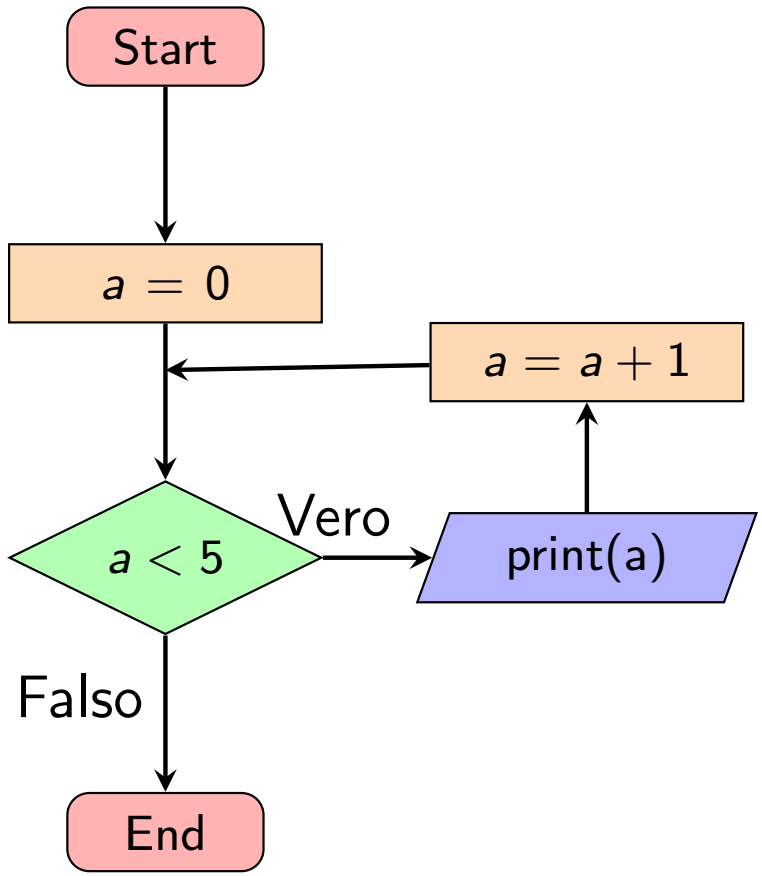
Cicli

- Python mette a disposizione 2 istruzioni per gestire l'esecuzione ripetuta di blocchi di codice
  - while
  - for
- Vengono anche chiamate istruzioni per la *gestione dei cicli* . . .
- . . . o più semplicemente *cicli* (*loop* in inglese)

While: esempio

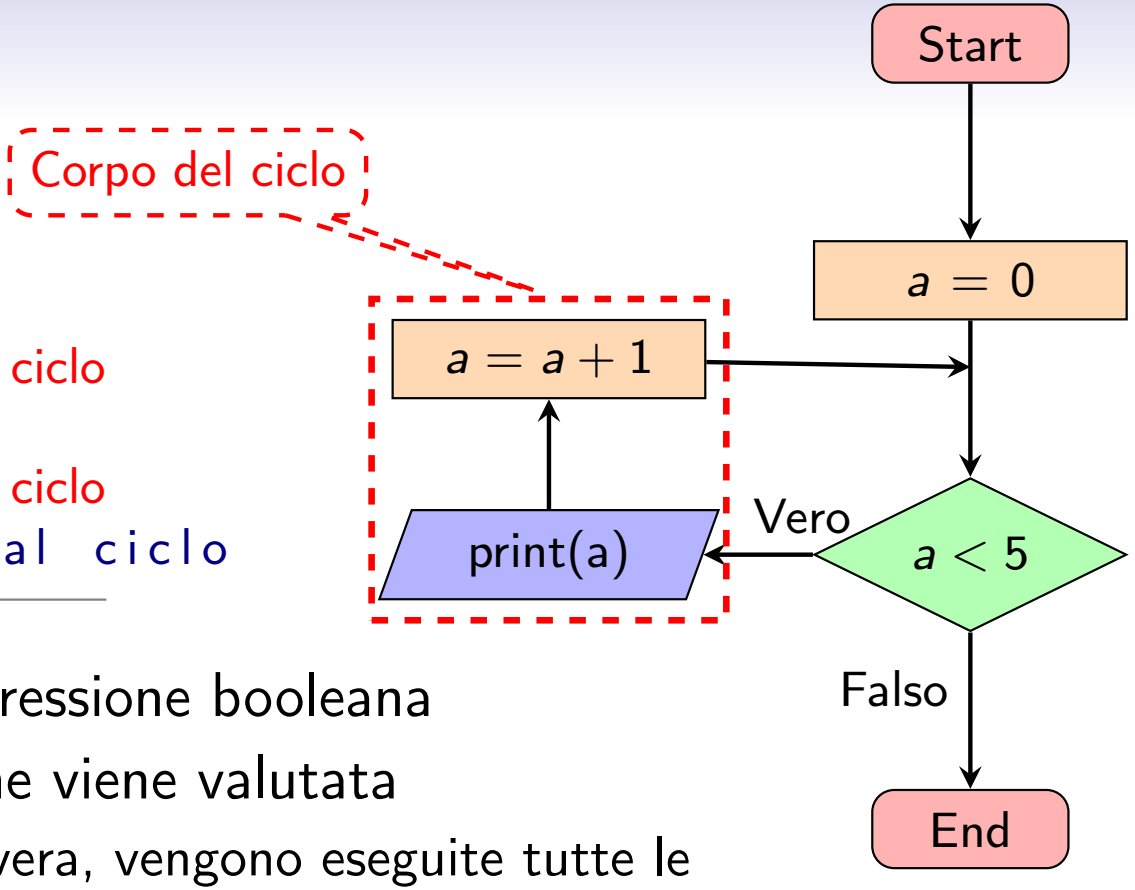
```
1 a = 0
2 while a < 5:
3     print(a)
4     a = a + 1
```

```
0
1
2
3
4
```



- Sintassi

```
1 istruzione precedente
2 while condizione:
3     prima istruzione del corpo del ciclo
4     ...
5     ultima istruzione del corpo del ciclo
6 prima istruzione fuori dal ciclo
```



- La condizione è un'espressione booleana
- All'inizio[1], la condizione viene valutata
  - Se la condizione è vera, vengono eseguite tutte le istruzioni del corpo del ciclo
  - Se la condizione è falsa, nessuna istruzione del corpo del ciclo viene eseguita
- Dopo l'ultima istruzione del ciclo, si ripete la valutazione della condizione

[1] La prima volte che si incontra l'istruzione while

Esercizio

- Scrivete un programma che:
  - richiede un numero da tastiera
  - stampa i 5 numeri successivi
- Esempio di output

```
Immetti numero di partenza: 50

51
52
53
54
55
```

Implementazione . . .

Richiesta: programma che:

- richiede un numero da tastiera
- stampa i 5 numeri successivi

Esempio di implementazione

```
1 a = int(input("Immetti un n."))
2 cont = a
3 fine = a+5
4 while cont < fine:
5     print(cont)
6     cont = cont + 1
```

C'è un errore, dove?

Trace table

riga n.	a	cont	fine	print()
1	50	-	-	
2		50		
3			55	
5				50
6		51		
5				51
6		52		
5				52
6		53		
5				53
6		54		
5				54
6		55		

Implementazione corretta

Richiesta: programma che:

- richiede un numero da tastiera
- stampa i 5 numeri successivi

```
1 a = int(input("Immetti un n."))
2 cont = a + 1
3 fine = a+5
4 while cont <=  fine:
5     print(cont)
6     cont = cont + 1
```

riga n.	a	cont	fine	print()
1	50	-	-	
2		51		
3			55	
5				51
6		52		
5				52
6		53		
5				53
6		54		
5				54
6		55		
5				55
6		56		

Parte *difficoltosa*: progettare la condizione e le istruzioni in grado di influenzarne la valutazione, sia prima sia all'interno del ciclo

Focus su Trace table

- La trace table è uno strumento molto utile
- Utile per l'identificazione degli errori nei programmi (debugging)
- Vi aiuta a comprendere come l'interpret python lavora
  - Provate a creare le trace table degli script che scriverete
  - Vi richiederà un po' di tempo, ma ne vale la pena
- Suggerimenti pratici
  - Create tante colonne quante sono le variabili del vostro script
  - Simulate l'esecuzione delle istruzioni, un passo alla volta (non siate precipitosi)

Verifica delle condizioni

- Dovrete controllare attentamente
  - le istruzioni precedenti all'ingresso del ciclo
  - la condizione del ciclo while
- Le fasi più critiche sono:
  - L'ingresso nel ciclo
  - L'uscita dal ciclo
- Dopo aver scritto un ciclo . . . fate una verifica con un caso reale (es., trace table delle slide precedenti)

While ed if annidati

- Esempio

```
print("Stampa dei primi 5 numeri divisibili per 3")
cont=0
num=0
while cont < 5:
    if num % 3 == 0:
        print("%d e' divisibile per 3" % (num))
        cont = cont +1
    else:
        print("%d non e' divisibile per 3" % (num))
    num = num+1
print("fine")
```

- L'indentazione aiuta a capire di che blocco fa parte un'istruzione

Cicli while annidati

- E' possibile annidare cicli while tra di loro

```
1 while cond1:
2         ...
3         while cond2:
4                 ...
5                 ...
6         ...
7         ...
```

- Per ogni esecuzione del blocco istruzioni del ciclo1, il blocco istruzioni del ciclo 2 viene eseguito diverse volte
- Paragone: lancette delle ore e dei minuti dell'orologio
- Esempio: calcolo dei numeri primi compresi tra 2 ed N