Cicli e Iterazioni

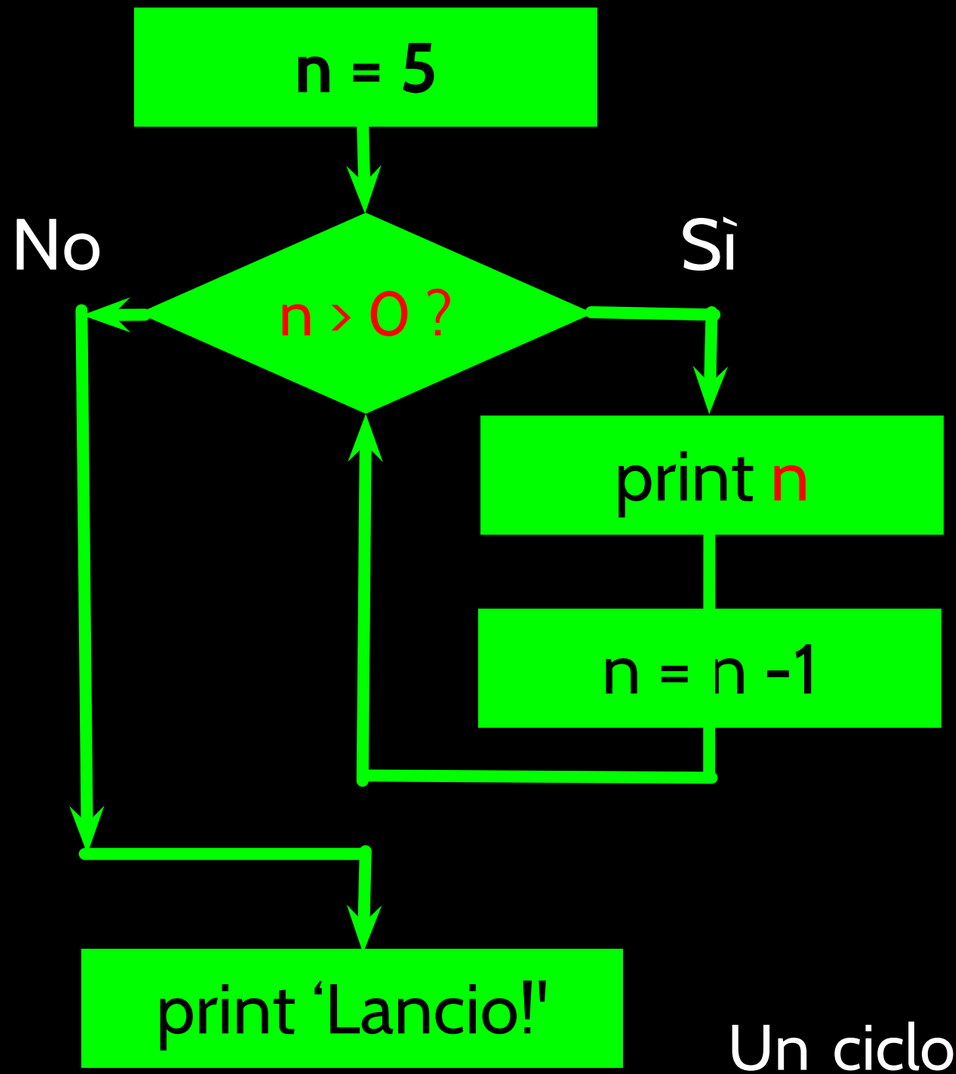
(Loops and Iteration)

Capitolo 5

Python for Informatics: Exploring Information
www.pythonlearn.com

Ripetizione di istruzioni

(iterazione o ciclo)



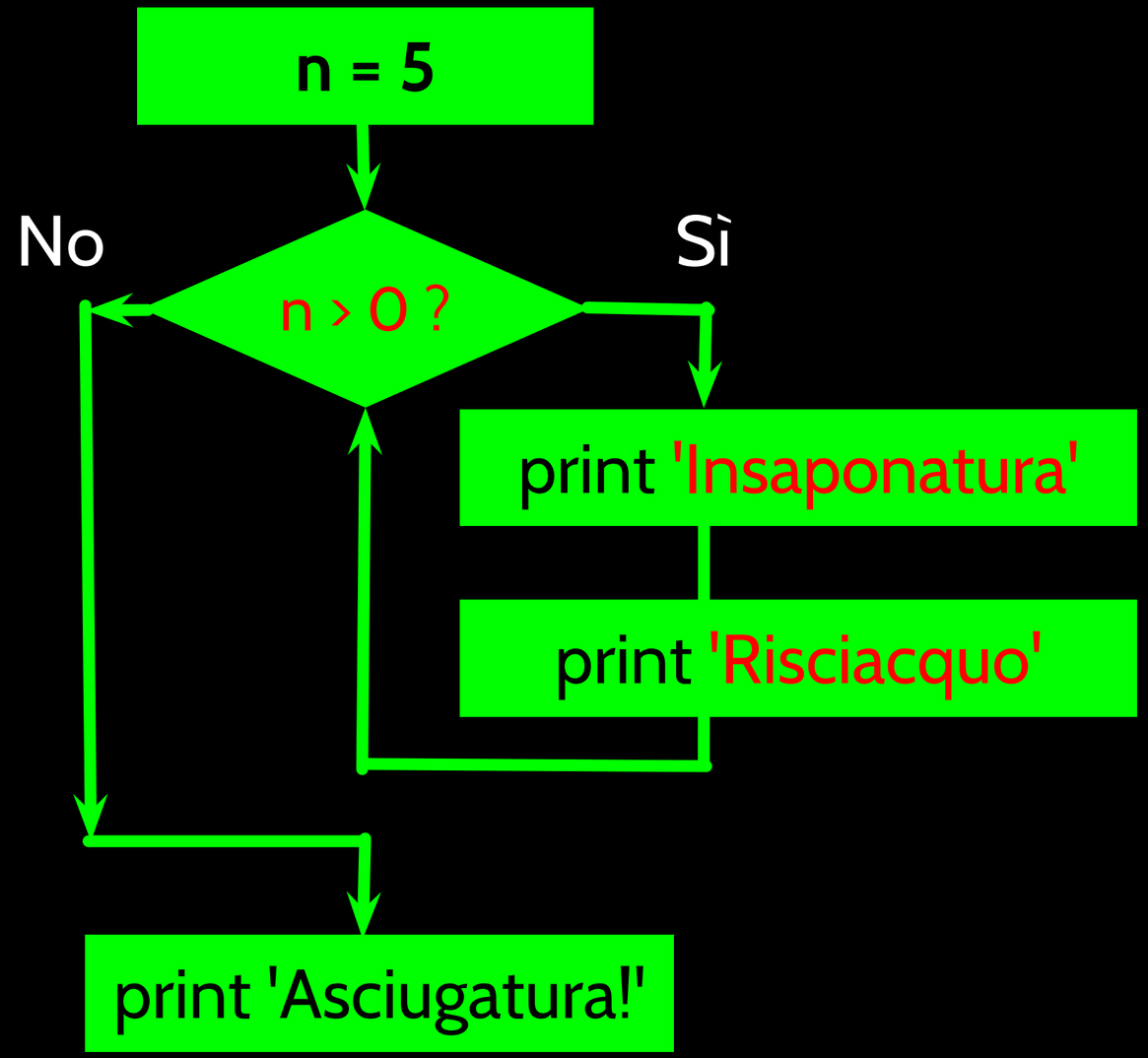
Programma:

```
n = 5
while n > 0 :
    print n
    n = n - 1
print 'Lancio!'
print n
```

Output:

5
4
3
2
1
Lancio!
0

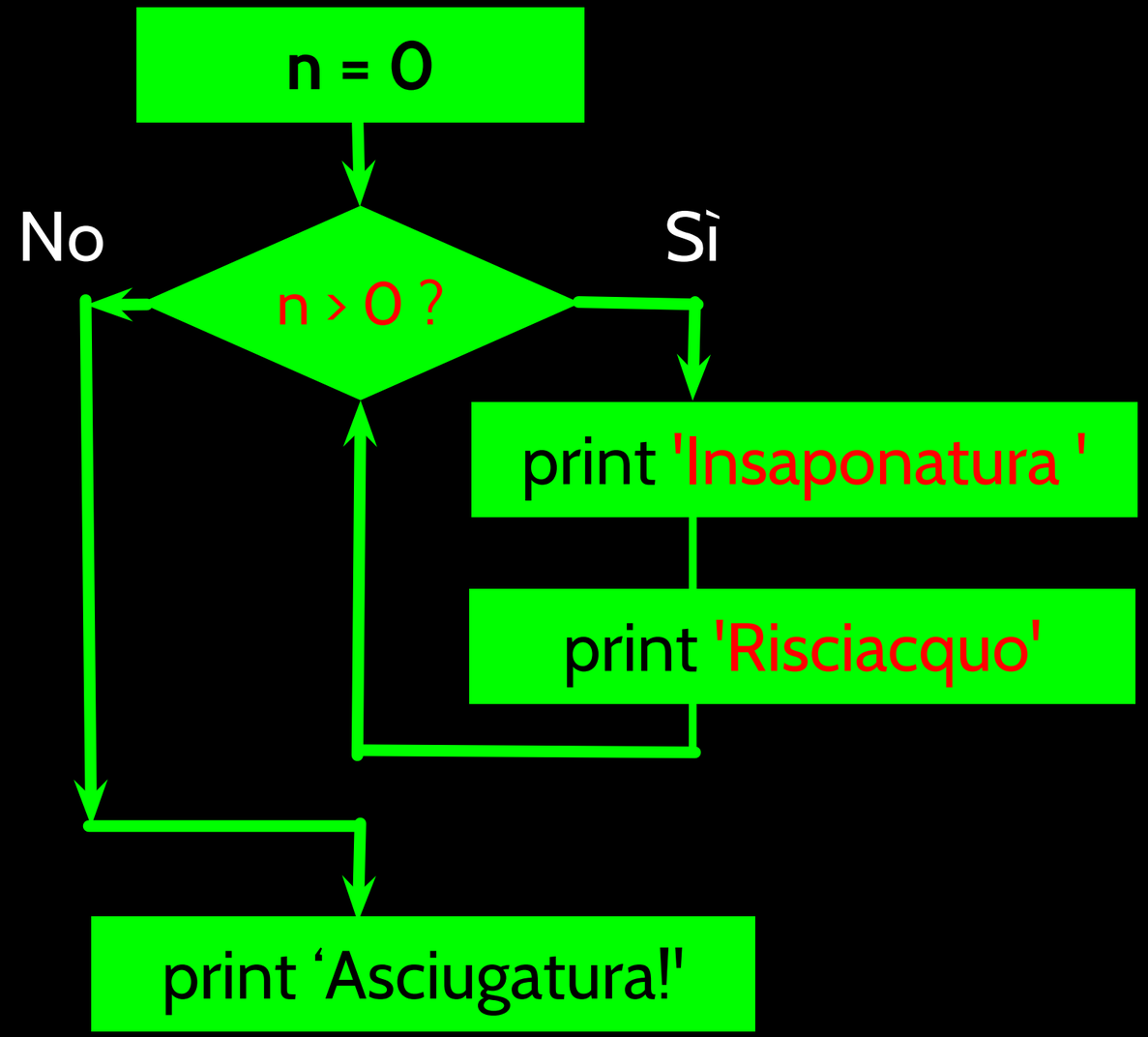
Un ciclo o loop (istruzioni ripetute) è caratterizzato da una variabile di iterazione che cambia valore ad ogni esecuzione del corpo del ciclo. Spesso i valori assunti formano una successione di numeri.



Un ciclo infinito

```
n = 5
while n > 0 :
    print 'Insaponatura'
    print 'Risciacquo'
print 'Asciugatura!'
```

Cosa c'è di sbagliato in questo ciclo?



Un altro ciclo

```
n = 0
while n > 0 :
    print 'Insaponatura'
    print 'Risciacquo'
print 'Asciugatura!'
```

Cosa fa questo ciclo?

Uscire da un ciclo con `break`

- L'istruzione **`break`** termina il ciclo corrente e salta all'istruzione immediatamente successiva al ciclo.

- È come un test del ciclo verificabile ovunque nel corpo del loop.

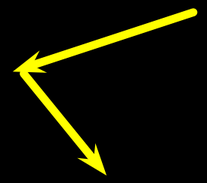
```
while True:
    line = raw_input('> ')
    if line == 'fatto' :
        break
    print line
print 'Già Fatto??!!'
```

```
> Ciao !
Ciao !
> finito
finito
> fatto
Già Fatto??!!
```

Uscire da un ciclo con **`break`**

- L'istruzione **`break`** termina il ciclo corrente e salta all'istruzione immediatamente successiva al ciclo.

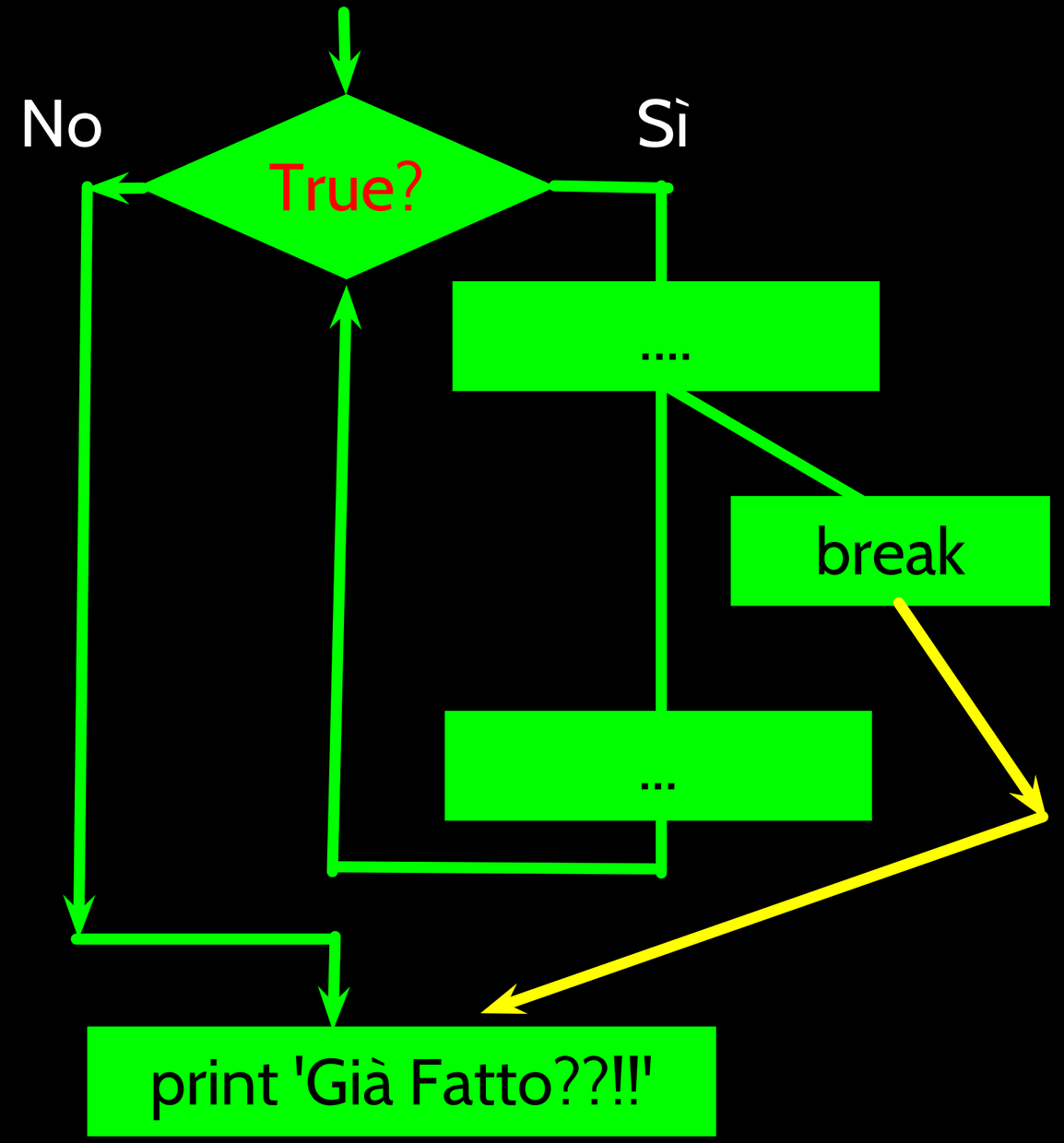
- È come un test del ciclo verificabile ovunque nel corpo del loop.



```
while True:
    line = raw_input('> ')
    if line == 'fatto' :
        break
    print line
print 'Già Fatto??!!'
```

```
> Ciao !
Ciao !
> finito
finito
> fatto
Già Fatto??!!
```

```
while True:
    line = raw_input('> ')
    if line == 'fatto' :
        break
    print line
print 'Già Fatto??!!'
```



http://it.wikipedia.org/wiki/Teletrasporto_(Star_Trek)

Terminare una iterazione con ‘`continue`’

L’istruzione **`continue`** termina l’iterazione corrente e salta all’inizio del ciclo per iniziare l’iterazione successiva.

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'fatto' :
        break
    print line
print 'Già Fatto??!!'
```

```
> Ciao a tutti
Ciao a tutti
> # non stampare questo
> stampa questo!
stampa questo!
> fatto
Già Fatto??!!
```

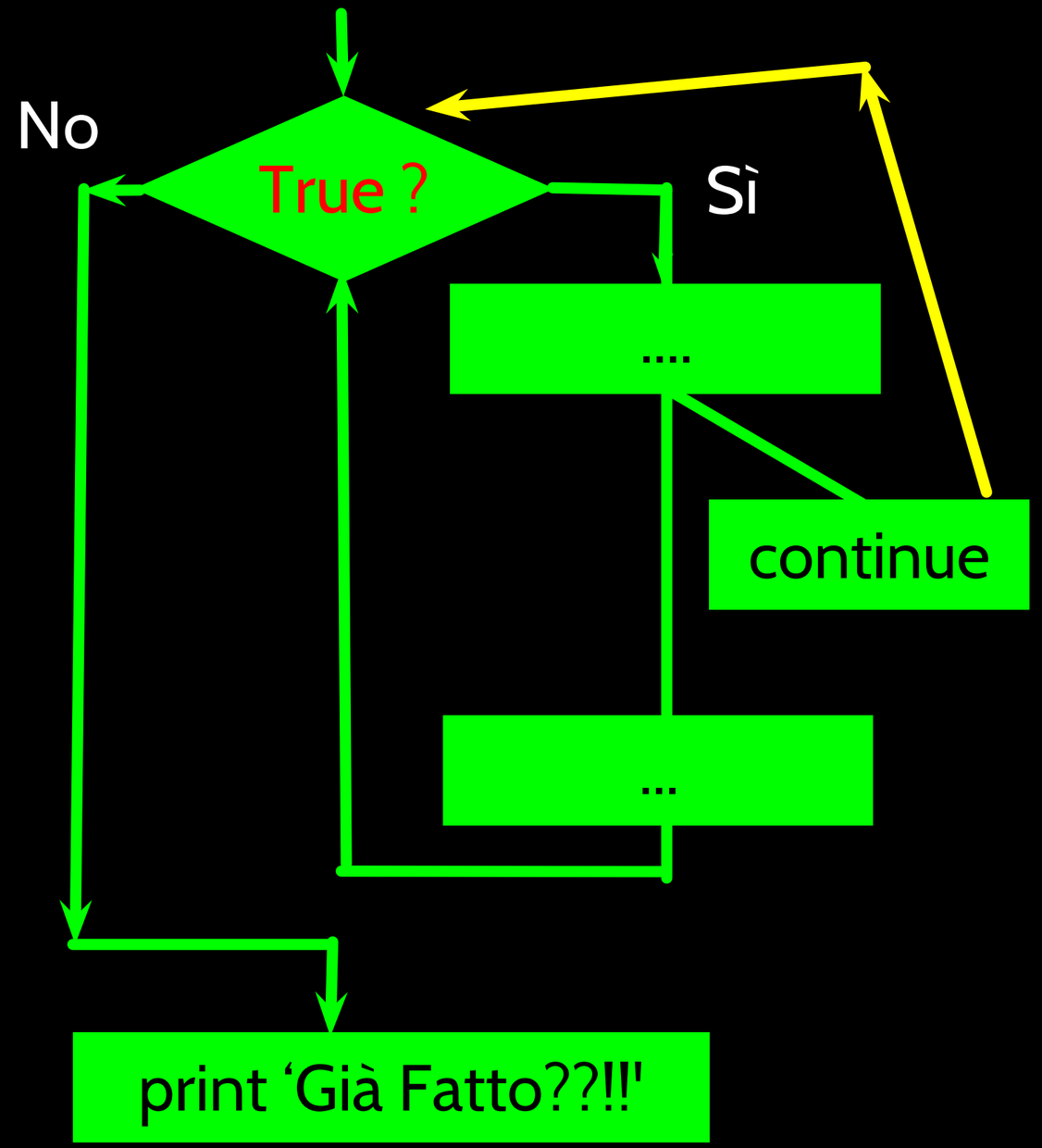
Terminare una iterazione con ‘`continue`’

L’istruzione **`continue`** termina l’iterazione corrente e salta all’inizio del ciclo per iniziare l’iterazione successiva.

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'fatto' :
        break
    print line
print 'Già Fatto??!!'
```

```
> Ciao a tutti
Ciao a tutti
> # non stampare questo
> stampa questo!
stampa questo!
> fatto
Già Fatto??!!
```

```
while True:
    line = raw_input('> ')
    if line[0] == '#' :
        continue
    if line == 'fatto' :
        break
    print line
print 'Già Fatto??!!'
```



Cicli indefiniti

si esce dal ciclo al verificarsi di una data condizione

- I cicli While sono chiamati “cicli indefiniti" a causa del fatto che vengono ripetuti (iterati) sino a che il Test non restituisce False
- Gli esempi di cicli (e relativi Test) sin a qui visti sono molto semplici ed è immediato vedere se essi terminano o no diventando cicli infiniti.
- A volte può essere complesso asserire con certezza che il ciclo avrà fine.

Cicli definiti

si conosce a priori il numero di iterazioni

- Abbastanza spesso si ha a che fare con elenchi o liste, list appunto, pensiamo alle linee in un file, in effetti un insieme finito di vari elementi.
- In Python, usando l'istruzione for è possibile scrivere un loop che viene eseguito per ogni singolo elemento di un insieme.
- Poiché gli elementi di un insieme (finito) sono in un numero noto, questi cicli sono detti ''cicli definiti'': essi verranno eseguiti un numero esatto di volte.
- In conclusione: '' le variabili di iterazione in cicli definiti prendono i loro valore dagli elementi di insiemi finiti.''

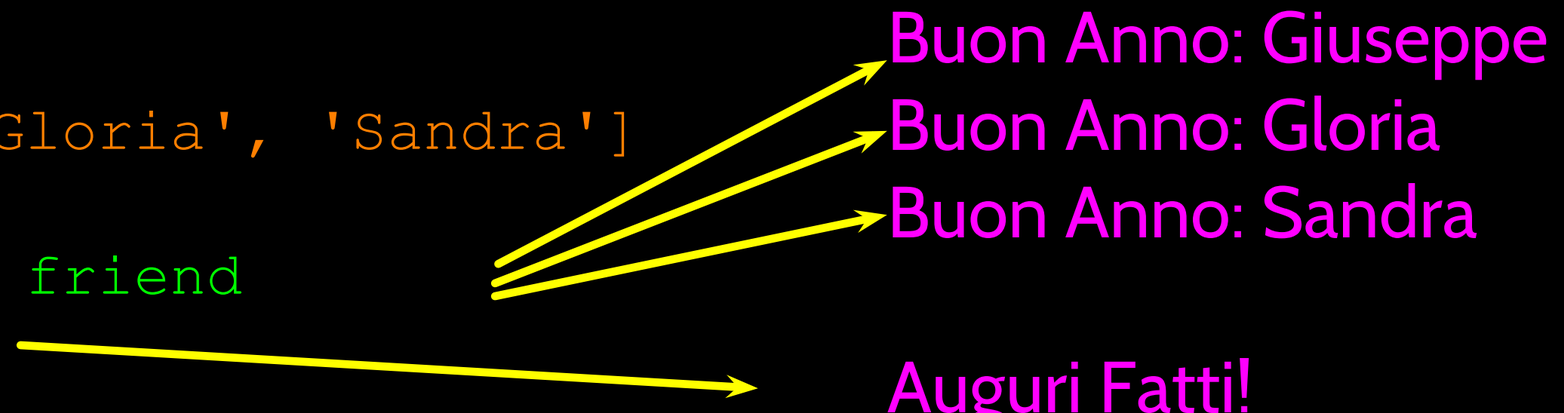
Un semplice ciclo definito

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'Lancio Missile!'
```

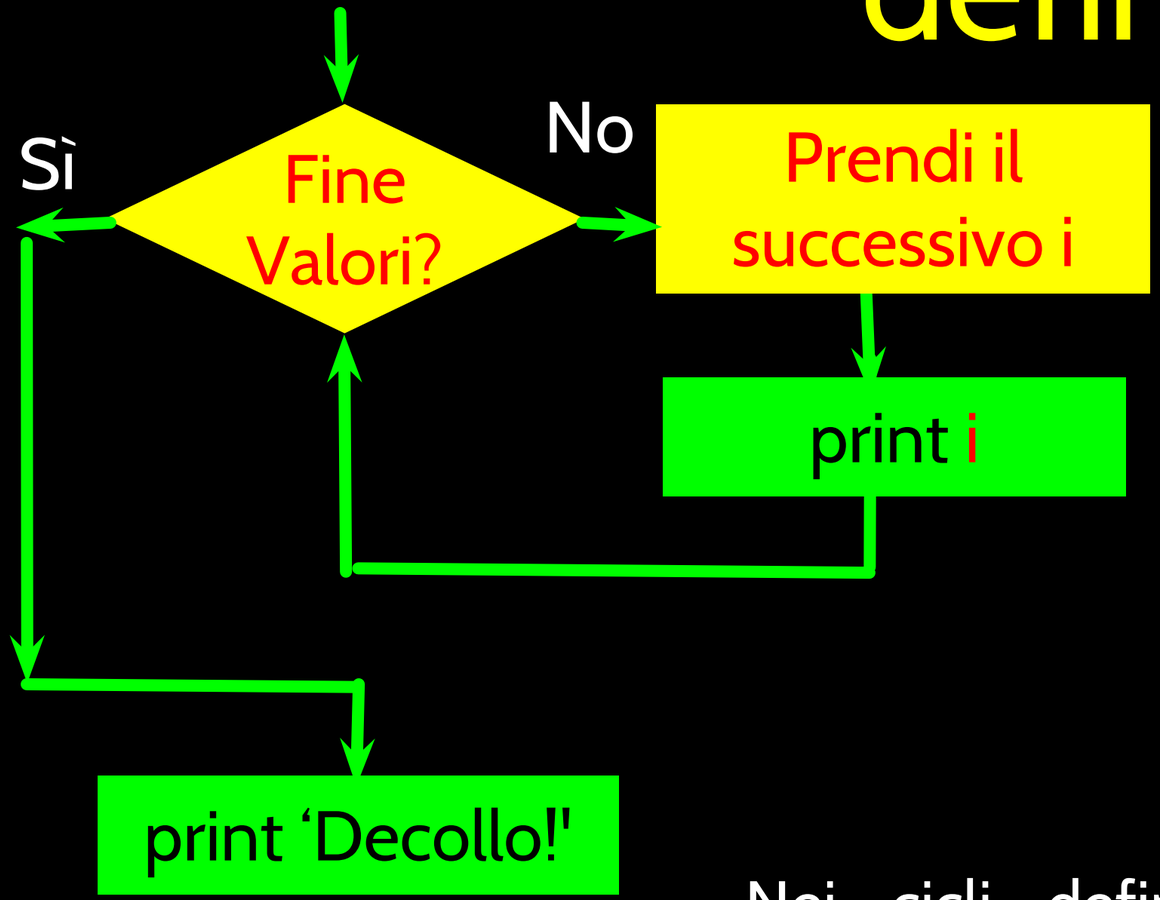
```
5
4
3
2
1
Lancio Missile!
```

Un ciclo definite con Stringhe

```
friends = ['Giuseppe', 'Gloria', 'Sandra']
for friend in friends :
    print 'Buon Anno: ',  friend
print 'Auguri Fatti!'
```



Un semplice ciclo definito



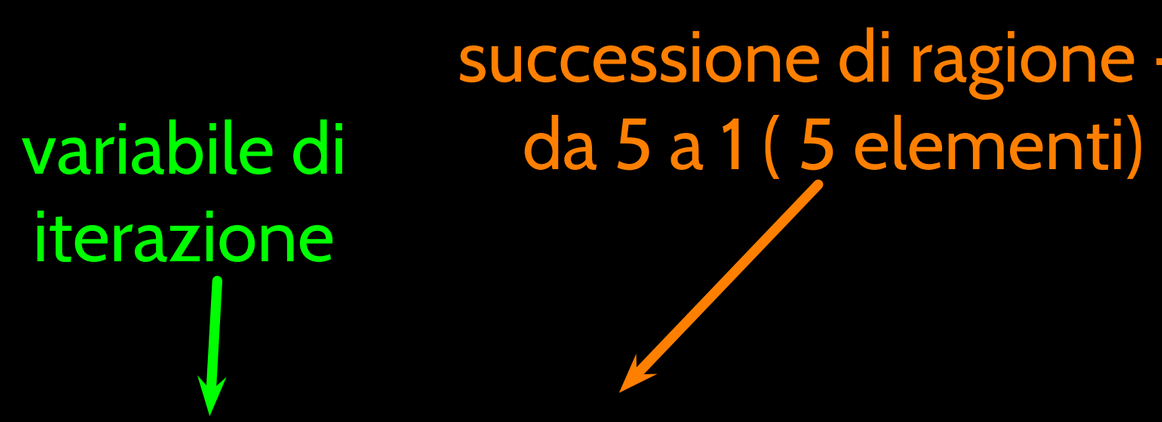

```
for i in [5, 4, 3, 2, 1] :
    print i
print 'Decollo!'
```

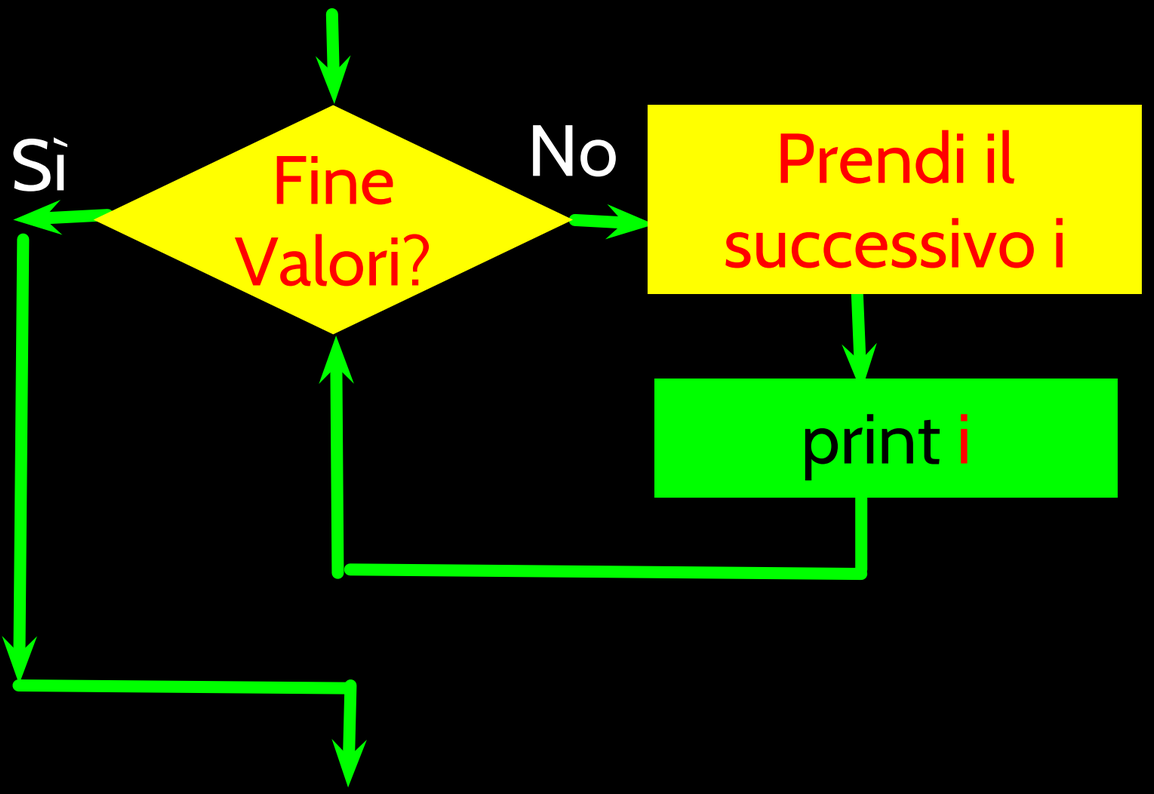
```
5
4
3
2
1
Decollo!
```

Nei cicli definiti (cicli for) le variabili di iterazione, espressamente dichiarate, assumono i loro valori dagli elementi (ordinati) di opportuni insiemi finiti.

Uno sguardo a `in` ...

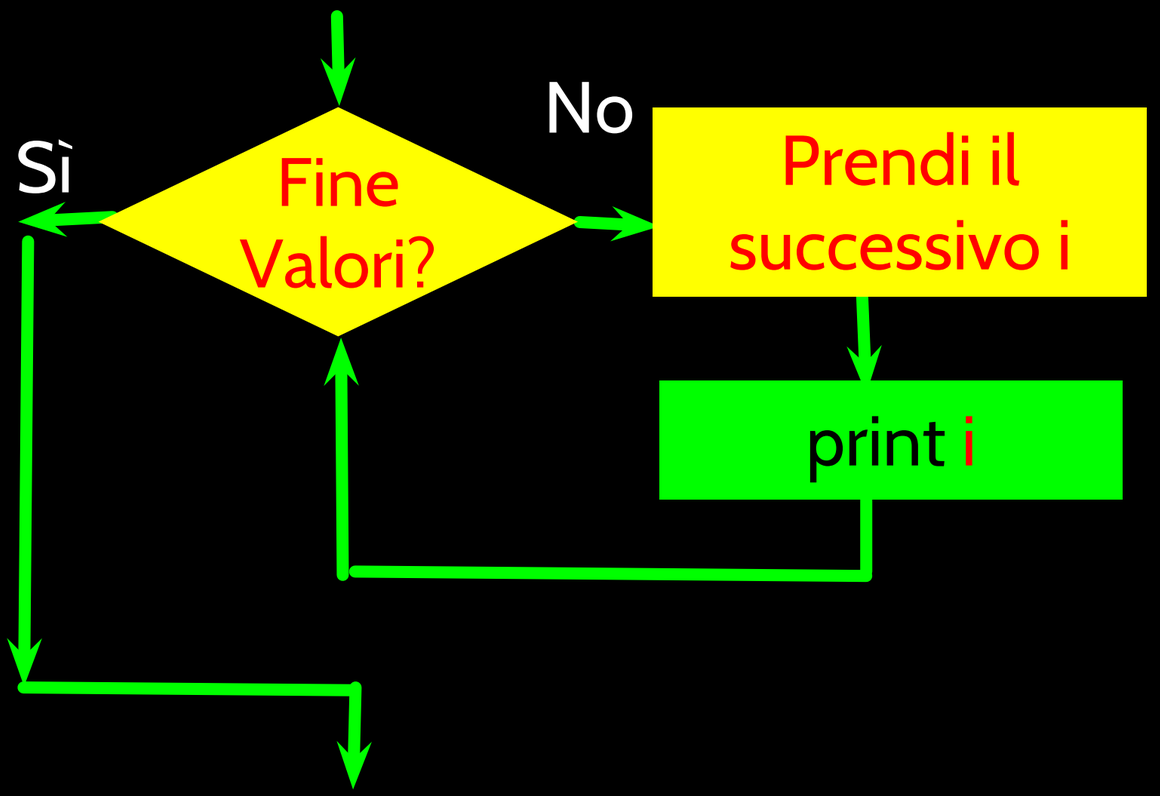
- La variabile di iterazione "itera" (prende i valori) tra quelli di una successione. (insieme ordinato)
- Il blocco delle istruzioni ripetute (corpo) è eseguito per ogni valore che è **in**cluso nella successione.
- La variabile di iterazione passa da un valore all'altro spostandosi ordinatamente dal primo all'ultimo elemento della successione.



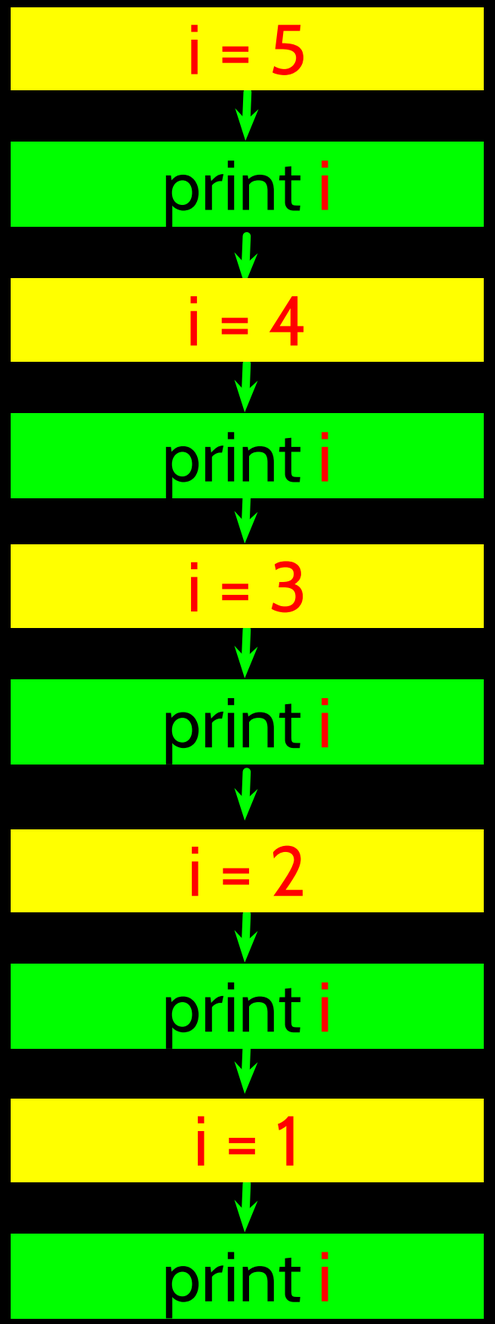



```
for i in [5, 4, 3, 2, 1] :
    print i
```

- La variabile di iterazione “itera” (prende i valori) tra quelli di una successione. (insieme ordinato)
- Il blocco delle istruzioni ripetute (corpo) è eseguito per ogni valore che è **in**cluso nella successione.
- La variabile di iterazione passa da un valore all’altro spostandosi ordinatamente dal primo all’ultimo elemento della successione.




```
for i in [5, 4, 3, 2, 1] :
    print i
```



Cicli definiti

si conosce a priori il numero di iterazioni

- Abbastanza spesso si ha a che fare con elenchi o liste, list appunto, pensiamo alle linee in un file, in effetti un insieme finito di vari elementi.
- In Python, usando l'istruzione for è possibile scrivere un loop che viene eseguito per ogni singolo elemento di un insieme.
- Poiché gli elementi di un insieme (finito) sono in un numero noto, questi cicli sono detti ''cicli definiti'': essi verranno eseguiti un numero esatto di volte.
- In conclusione: '' le variabili di iterazione in cicli definiti prendono i loro valore dagli elementi di insiemi finiti.''

Struttura dei cicli:
occhio a quello che succede all'interno.

Nota: per quanto banali possano sembrare gli esempi di seguito riportati, essi ricalcano la struttura di qualsiasi tipo di ciclo.

Costruire cicli “intelligenti”

Quando ci si blocca a scrivere codice che esegue un passo alla volta il trucco è avere chiara la “conoscenza” di quello che deve avvenire nel ciclo.

Impostare variabili al valore iniziale

operando sui dati:

Eseguire sempre un compito alla volta: cercare, modificare o aggiornare una variabile

Controllare le variabili

Ciclo tra i valori di un insieme

```
print 'Prima'
for thing in [9, 41, 12, 3, 74, 15] :
     print thing
print 'Dopo'
```

```
$ python basicloop.py
Prima
9
41
12
3
74
15
Dopo
```

Qual è il maggiore?

Qual è il maggiore?

3 41 12 9 74 15

Il maggiore (sino ad ora): -1

Contatori e cicli

```
contatore = 0
print 'Prima', contatore
for numero in [9, 41, 12, 3, 74, 15] :
    contatore = contatore + 1
    print contatore, numero
print 'Dopo', contatore
```

```
$ python countloop.py
Prima 0
1 9
2 41
3 12
4 3
5 74
6 15
Dopo 6
```

Per contare quante volte un ciclo viene eseguito, introduciamo una variabile ‘contatore’ che è inizializzata a 0 ed alla quale aggiungiamo 1 (uno) ad ogni esecuzione del corpo del ciclo.

Sommatorie e cicli

```
accumulatore = 0
print 'Prima', accumulatore
for valore in [9, 41, 12, 3, 74, 15] :
    accumulatore = accumulatore + valore
    print accumulatore, valore
print 'Dopo', accumulatore
```

```
$ python sumloop.py
Prima 0
9 9
50 41
62 12
65 3
139 74
154 15
Dopo 154
```

Volendo sommare un valore che compare in un ciclo, introduciamo una variabile 'accumulatore' inizializzata a 0 e sommiamo ad essa il valore desiderato ad ogni iterazione richiesta dal ciclo.

Un ciclo per calcolare la media

```
count = 0
sum = 0
print 'Prima', count, sum
for value in [9, 41, 12, 3, 74, 15] :
    count = count + 1
    sum = sum + value
    print count, sum, value
print 'Dopo', count, sum, sum / count
```

```
$ python averageloop.py
Prima 0 0
1 9 9
2 50 41
3 62 12
4 65 3
5 139 74
6 154 15
Dopo 6 154 25
```

La media combina lo schema del conteggio con quello della sommatoria una volta calcolati i due valori divideremo ‘sum’ per ‘count’ alla fine del ciclo.

Filtrare i valori in un ciclo

```
print ‘Prima’
for value in [9, 41, 12, 3, 74, 15] :
    if value > 20:
       print ‘Sopra la soglia',value
print 'Dopo'
```

```
$ python filter1.py
Prima
Sopra la soglia 41
Sopra la soglia 74
Dopo
```

Per individuare / filtrare un valore particolare useremo l’ istruzione if all’interno del corpo del ciclo.

Logica della ricerca: le variabili Boolean

```
found = False
print 'Before', found
for value in [9, 41, 12, 3, 74, 15] :
    if value == 3 :
            found = True
     print found, value
print 'After', found
```

```
$ python search1.py
Before False
False 9
False 41
False 12
True 3
True 74
True 15
After True
```

Se durante una ricerca vogliamo essere informati se un valore è stato trovato, possiamo usare una variabile booleana inizializzata a False e posta a True non appena trovato quello che stavamo cercando.

Qual è il minore?

Qual è il minore?

9 41 12 3 74 15

Il minore per ora: -1

Qual è il minore?

9 41 12 3 74 15

Il minore per ora: None

```
smallest = None
print 'Before'
for value in [9, 41, 12, 3, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value
print 'After', smallest
```

Trovare il minimo valore

```
$ python smallest.py
Before
9 9
9 41
9 12
3 3
3 74
3 15
After 3
```

Eccoci alle prese con la variabile smallest che indica il minor valore trovato sino ad ora. La prima volta che si esegue il ciclo (la variabile) smallest è posta a None, quindi viene aggiornata e il primo (valore) value diventa smallest (il minimo).

Gli operatori (è) "is" e (non è) "is not"

```
smallest = None
print 'Before'
for value in [3, 41, 12, 9, 74, 15] :
    if smallest is None :
        smallest = value
    elif value < smallest :
        smallest = value
    print smallest, value
print 'After', smallest
```

- Python possiede un operatore "is" che viene usato nelle espressioni logiche.
- "is" 'è lo stesso di'
- Simile al confronto ==, ma più severo.
- 'is not' è anch'esso un operatore logico

Sommario

- Ciclo While (ciclo indefinito)
- Ciclo Infinito
- Uso di break
- Uso di continue
- Ciclo For (ciclo definito)
- Variabile di iterazione
- Maggiore o minore

Acknowledgements / Contributions

Initial Development: Charles Severance, University of Michigan School of Information

… Insert new Contributors here

...