UNIVERSITÀ DI PARMA

SVILUPPO RISORSE PER L'AZIENDA
CISITA
PARMA

python

le basi di python

- python: il linguaggio
- variabili e tipi di dato
- input output
- controllo del flusso di esecuzione (if – while)

python install

- ***Python*** – (https://www.python.org/)
  - Python 3.x …
  - download … install for all users … add Python to environment variables
- ***prompt dei comandi***
  - (Windows) **`cmd`** – (Linux) Applicazioni > Accessori > Terminale – (Mac) Terminale
- ***NumPy*** (*Numeric Python*)
  - **`pip3 install numpy`**
- ***Pandas*** (*Python Data Analysis*)
  - **`pip3 install pandas`**
- ***MatplotLib***
  - **`pip3 install matplotlib`**
- ***Jupiter Notebook***
  - **`pip3 install notebook`**

primi ambienti di sviluppo per iniziare

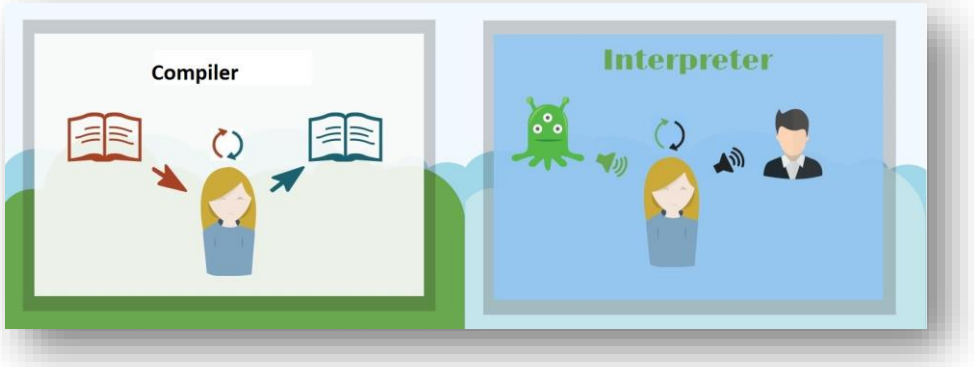
- *in attesa di completare l'installazione …*
  - ***playground***
    - non necessaria nessuna installazione di software
    - http://www.ce.unipr.it/brython/
- dopo aver installato solo Python
  - ***shell interattiva***
    - IDLE

Jupyter Notebook

- web application open source
- permette di creare e condividere documenti che contengono
  - codice eseguibile
  - testo formattato notazione scientifica (markdown, LaTeX)
  - grafici
- documentazione (https://jupyter-notebook.readthedocs.io/en/stable/)
- esecuzione dal prompt dei comandi
  - **`jupyter notebook`**

python

- ideato da Guido van Rossum nel ***1989***
- è molto utilizzato come linguaggio di ***scripting***
  - *insieme a R è il più utilizzato nel* ***machine learning***
- è un linguaggio ***interpretato*** a tipizzazione dinamica
- è dotato di un ***ambiente interattivo***
- l'***indentazione*** è ***obbligatoria*** (leggibilità del codice)



caratteristiche

- ***free***
  - è possibile utilizzarlo e distribuirlo senza restrizioni di copyright (***open-source***)
- ***facile*** da usare
  - linguaggio di alto livello (semplice e potente)
  - sintassi facile da imparare
- ***portabile***
  - linguaggio interpretato, il codice può essere eseguito su qualsiasi piattaforma purché abbia l'interprete Python installato (Unix, Linux, Windows, macOS, Android, iOS …)
- ***multi-paradigma***
  - supporta sia la programmazione procedurale, la programmazione ad oggetti e diversi elementi della programmazione funzionale

diffusione del linguaggio

tipi di dato

- un ***tipo di dato*** specifica
  - un insieme di valori
  - un insieme di operazioni ammesse su questi valori
- **`int, float, bool, str`**
- operazioni aritmetiche e logiche, confronti

```
2  * 5 =  10
2 ** 5 =  32
2 ** 0.5 =  1.4142135623730951
29 // 7 =  4
29  % 7 =  1
29  / 7 =  4.142857142857143
3 >  3 =  False
3 >= 3 =  True
True and False =  False
True  or False =  True
not True =  False
```

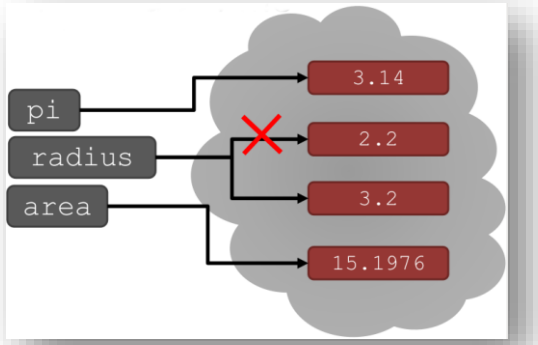
valori numerici e booleani

- ***int*** per numeri interi
- ***float*** per i numeri reali
  - operazioni di base `+`, `-`, `*`, `/`
  - divisione intera `//`, resto della divisione intera `%`, potenza `**`
  - confronti: `<`, `<=`, `>`, `>=`, `==`, `!=`
- ***bool*** per valori booleani: `True`, `False`
  - operazioni: `and`, `or`, `not`
  - i confronti danno un risultato booleano

```
-2 // 3 =  -1
-2  % 3 =  1
2 ** 1000 =
107150860718626732094842504906000181056140481170553360744375038837035105112493612249319837881569585812759467291755314682518714528569231404359845775746985748039345677748242309854210746050623711418779541821530464749835819412673987675591655439460770629145711964776865421676604298316526243868372056680693 76
```

variabili e operazione di assegnamento

- una ***variabile*** può essere definita come un contenitore per memorizzare un risultato
- **assegnamento**, operatore **=**
  - a sinistra una variabile
  - a destra un valore (o una espressione)
  - l'assegnamento definisce il valore e il tipo di una variabile
- non confondere il confronto di uguaglianza **==** con l'assegnamento **=**




```
pi = 3.14  # assignment
radius = 2.2
area = pi * (radius ** 2)
print(area)  # 15.1976
radius = radius + 1
# guess radius... and area!
```

variabili

- una ***variabile*** è definita da
  - un ***nome*** (etichetta - identificatore)
  - associato ad un ***valore*** (oggetto)
- un oggetto assegnato a più variabili: ***non viene copiato, ma riceve più etichette***
- il ***tipo*** della variabile ***dipende*** dal ***valore*** attualmente assegnato *(tipizzazione dinamica)*
- una variabile ***non*** deve essere ***dichiarata***, ma ***deve essere inizializzata***

nei linguaggi a tipizzazione dinamica le variabili hanno tipi che possono cambiare durante l'esecuzione di un programma, di solito a causa di assegnamenti
i linguaggi a tipizzazione dinamica sono spesso interpretati
linguaggi a tipizzazione dinamica: JavaScript, PHP, Prolog, Python, Ruby …

stringhe di testo

- **str**
  - sequenze di ***uno o più caratteri*** Unicode
  - primi 128 codici Unicode == ASCII
  - a capo: '\n' (10, o 13-10...)
  - tabulazione: '\t' (9)

Unicode era stato originariamente pensato come una codifica a 16 bit per codificare 65.535 (2^16 -1) caratteri.
Ora lo standard Unicode prevede una codifica fino a 21 bit e supporta un repertorio di codici numerici che possono rappresentare circa un milione di caratteri

```
s1 = "Monty Python's"     # apici o doppi apici per delimitare stringhe
s2 = 'Flying Circus'
s3 = s1 + ' ' + s2        # concatenazione
print(s3)                 # Monty Python's Flying Circus
s3 = s3.replace('o','_')  # sostituzione di tutte le occorrenze
print(s3)                 # M_nty Pyth_n's Flying Circus
```

confronto tra stringhe

- il confronto tra stringhe segue l'ordine lessicografico
  - case sensitive
- operatori di confronto: **<, <=, >, >=, ==, !=**

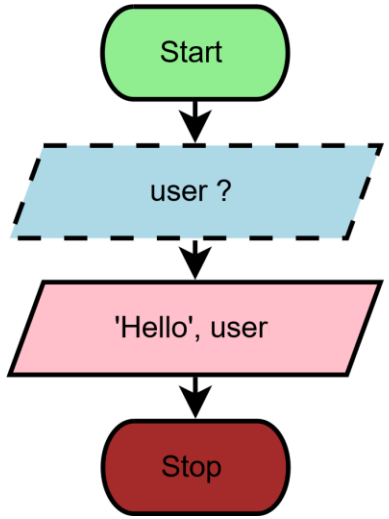
```
print("'first' < 'second' = ",'first' < 'second')
print("'first' < 'Second' = ",'first' < 'Second')
print("chr(90) = ",chr(90))
print("ord('Z') = ",ord('Z'))
```

```
'first' < 'second' =  True
'first' < 'Second' =  False
chr(90) =  Z
ord('Z') =  90
```

input - output

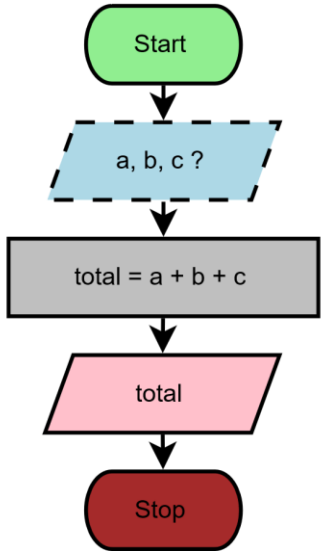
- **input**
  - ***legge*** una riga di ***testo*** (dallo standard input – ***tastiera***)
  - e la inserisce in una variabile
  - ***è di tipo str***
  - prima mostra un messaggio
- **print**
  - ***scrive*** una serie di valori sullo standard output (***schermo***)
  - tra i valori (parametri) viene inserito uno spazio

```
user = input("What's your name? ")

print("Hello,", user)
```

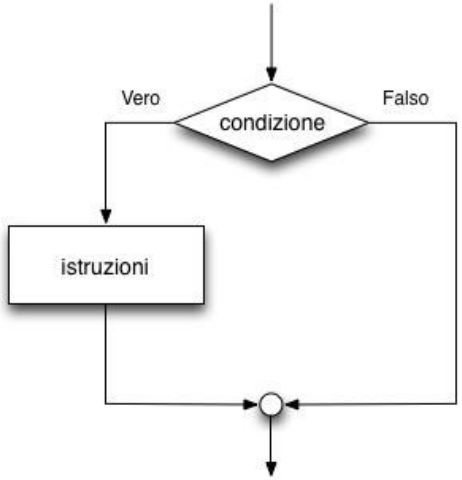


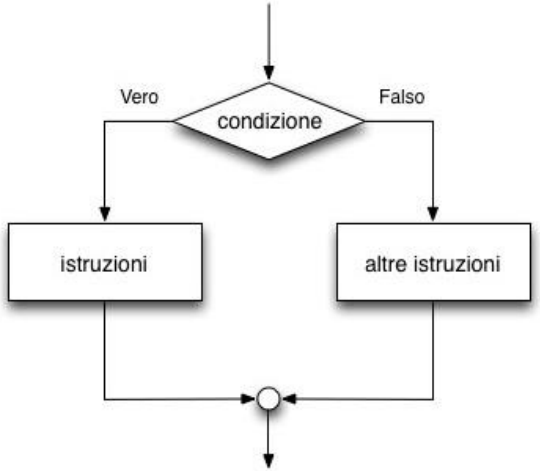
esempio: somma di 3 valori numerici

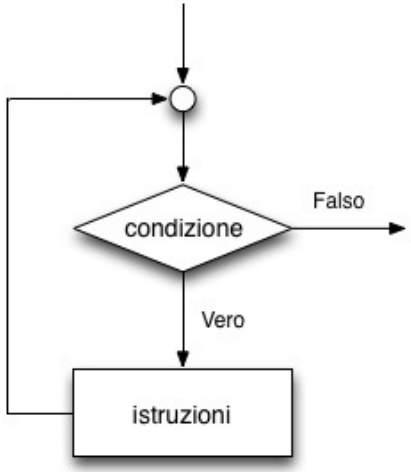
```
a = float(input("Insert 1st val: "))
b = float(input("Insert 2nd val: "))
c = float(input("Insert 3rd val: "))

total = a + b + c

print("The sum is", total)
```



controllo del flusso di esecuzione



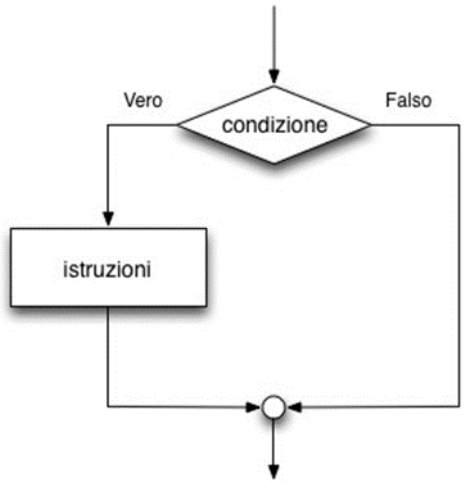




selezione - if

- ***indentazione*** del corpo di if o else ***necessaria*** (per sintassi), non opzionale!
- clausola else: opzionale
- eseguita solo se la condizione non è verificata

```
age = int(input("Age? "))

if age < 14:
    print("You're too young for driving a scooter...")
    print("But not for learning Python!")
```

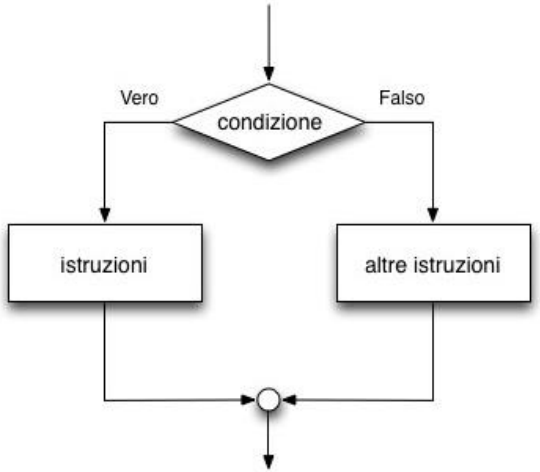


the zen of python

Beautiful is better than ugly.	Bello è meglio che brutto.
Explicit is better than implicit.	Esplicito è meglio di implicito.
Simple is better than complex.	Semplice è meglio che complesso.
Complex is better than complicated.	Complesso è meglio di complicato.
Flat is better than nested.	Lineare è meglio che nidificato.
Sparse is better than dense.	Sparso è meglio di denso.
Readability counts.	La leggibilità conta.
Special cases aren't special enough to break the rules.	I casi speciali non sono abbastanza speciali per infrangere le regole.
Although practicality beats purity.	Anche se la praticità batte la purezza.
Errors should never pass silently.	Gli errori non dovrebbero mai essere ignorati.
Unless explicitly silenced.	A meno che non vengano esplicitamente messi a tacere.
In the face of ambiguity, refuse the temptation to guess.	In caso di ambiguità, rifiuta la tentazione di indovinare.
There should be one-- and preferably only one --obvious way to do it.	Ci dovrebbe essere un modo ovvio, e preferibilmente uno solo, di fare le cose.
Although that way may not be obvious at first unless you're Dutch.	Anche se questo potrebbe non essere ovvio da subito, a meno che non siate olandesi.
Now is better than never.	Ora è meglio che mai.
Although never is often better than *right* now.	Sebbene mai sia spesso meglio che proprio adesso.
If the implementation is hard to explain, it's a bad idea.	Se l'implementazione è difficile da spiegare, l'idea è pessima.
If the implementation is easy to explain, it may be a good idea.	Se l'implementazione facile da spiegare, idea può essere buona.
Namespaces are one honking great idea -- let's do more of those!	I namespace sono una grandissima idea, usiamoli il più possibile!

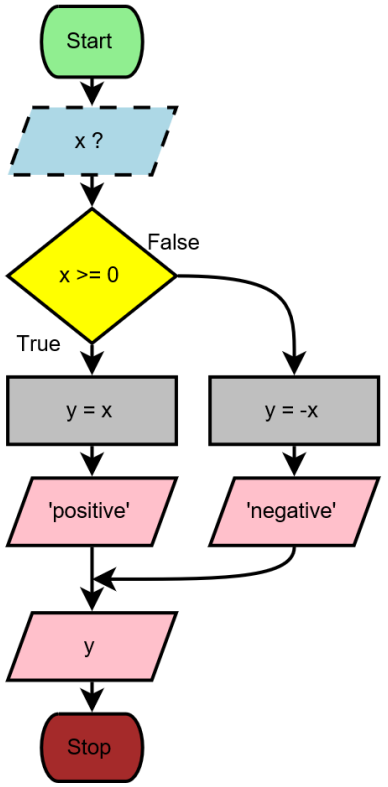
selezione – if --- else

- clausola ***else*** eseguita nel caso **`False`**
- nel corpo di if o else: è possibile inserire qualsiasi istruzione
  - anche altri blocchi if o altre strutture di controllo annidate!



esempio: valore assoluto

```
x = float(input("insert a value: "))

if x >= 0:
    y = x
    print(x, "is positive")
else:
    y = -x
    print(x, "is negative")

print("abs =", y)
```



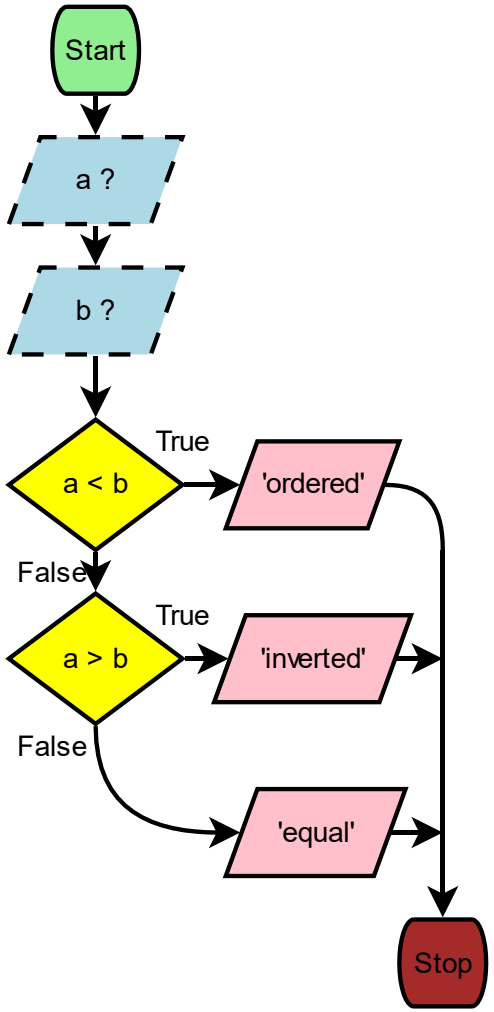
condizione con connettivi logici (espressioni booleane)

```
birth_year = int(input("Birth year? "))
birth_month = int(input("Birth month? "))
birth_day = int(input("Birth day? "))
current_year = int(input("Current year? "))
current_month = int(input("Current month? "))
current_day = int(input("Current day? "))

if (current_month > birth_month
    or (current_month == birth_month and current_day >= birth_day)):
    age = current_year - birth_year
else:
    age = current_year - birth_year - 1

print("Your age is", age)
```

selezione - elif

- **elif**: clausola ***else*** che contiene un altro ***if***
- in Python non è presente il costrutto switch (e nemmeno do-while)

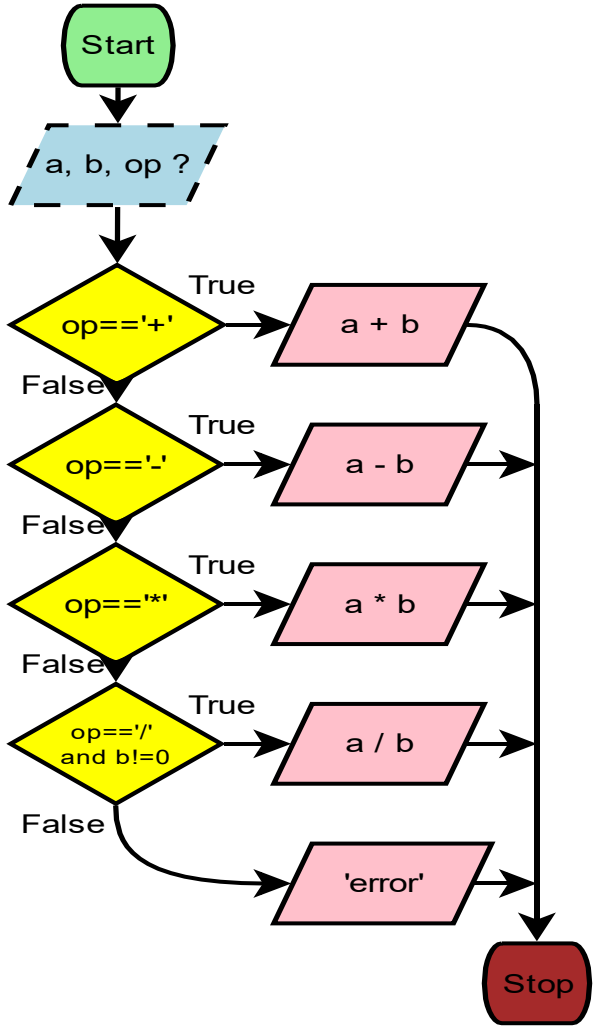
```
a = input("First word? ")
b = input("Second word? ")
if a < b:
    print("The words are ordered")
elif a > b:
    print("The words are inverted")
else:
    print("The words are equal")
```



There should be one -- and preferably only one -- obvious way to do it (ZoP)

esempio: operazioni aritmetiche

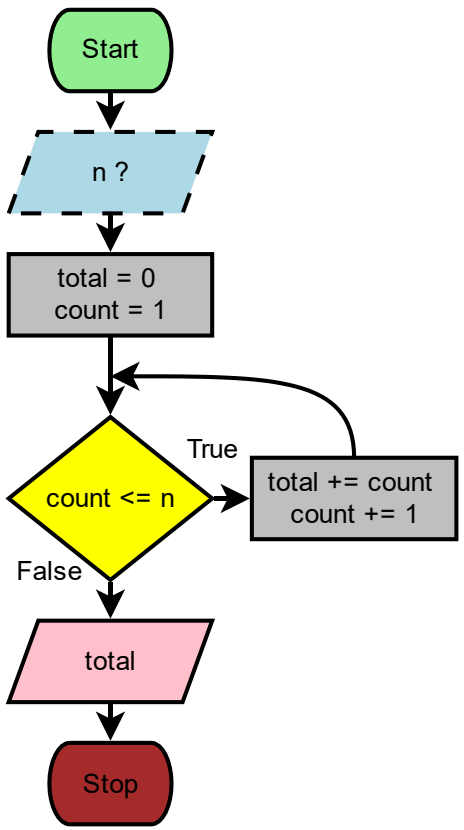
```
a = float(input('first  value: '))
b = float(input('second value: '))
op = input('operation (+-*/): ')

if op == '+':
    print(a + b)
elif op == '-':
    print(a - b)
elif op == '*':
    print(a * b)
elif op == '/' and b != 0:
    print(a / b)
else:
    print("Operation not allowed")
```



iterazione: while

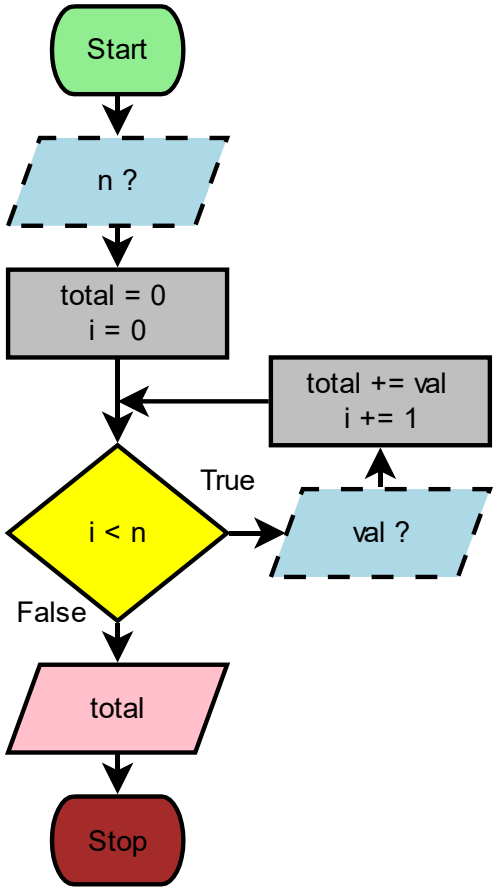
- ***condizione*** booleana di ***permanenza*** nel ciclo
- controllo preliminare (***precondizione***)
  - *possibile che il corpo non sia mai eseguito*

```
# Sum of the numbers from 1 to n
total = 0
count = 1
n = int(input("n? "))

while count <= n:
    total = total + count
    count = count + 1

print("The sum is", total)
```



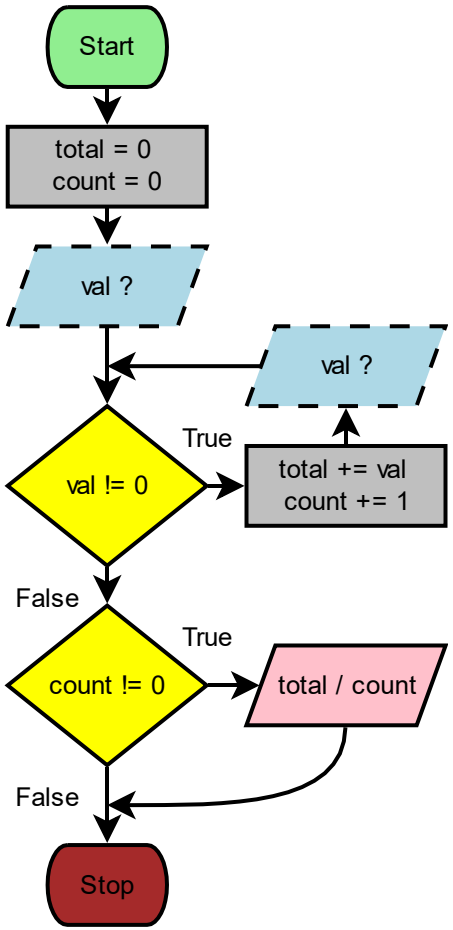
esempio: somma n valori in input

```
n = int(input("How many values? "))
total = 0
i = 0

while i < n:
    val = int(input("Val? "))

    total += val   # total = total + val
    i += 1         # i = i + 1

print("The sum is", total)
```



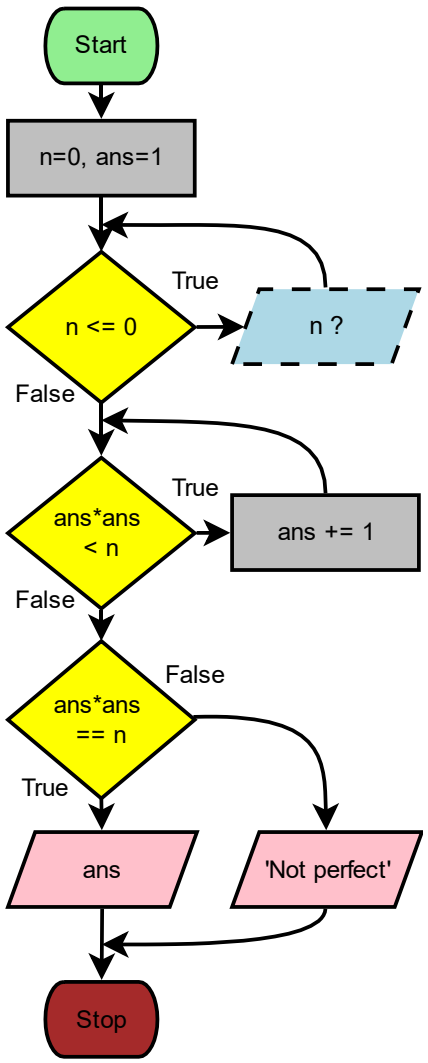
esempio: ciclo con sentinella

```
total = 0
count = 0

val = int(input("Val? (0 to finish) "))

while val != 0:
    total += val
    count += 1
    val = int(input("Val? (0 to finish) "))

if count != 0:
    print("The average is", total / count)
```



esempio: quadrato perfetto

```
n = 0
while n <= 0:
    n = int(input("Positive val? "))

ans = 1
while ans * ans < n:
    ans += 1

if ans * ans == n:
    print("Square root:", ans)
else:
    print("Not a perfect square")
```



mutable vs immutable