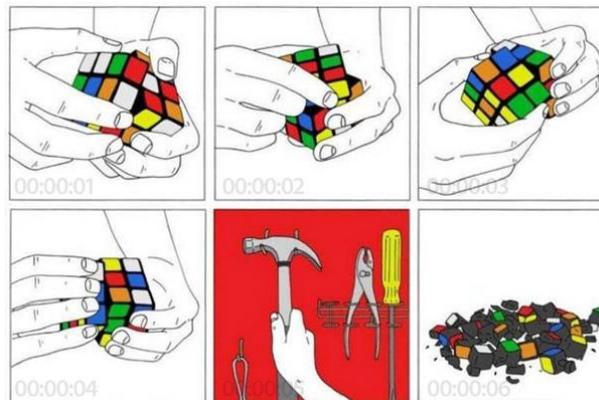




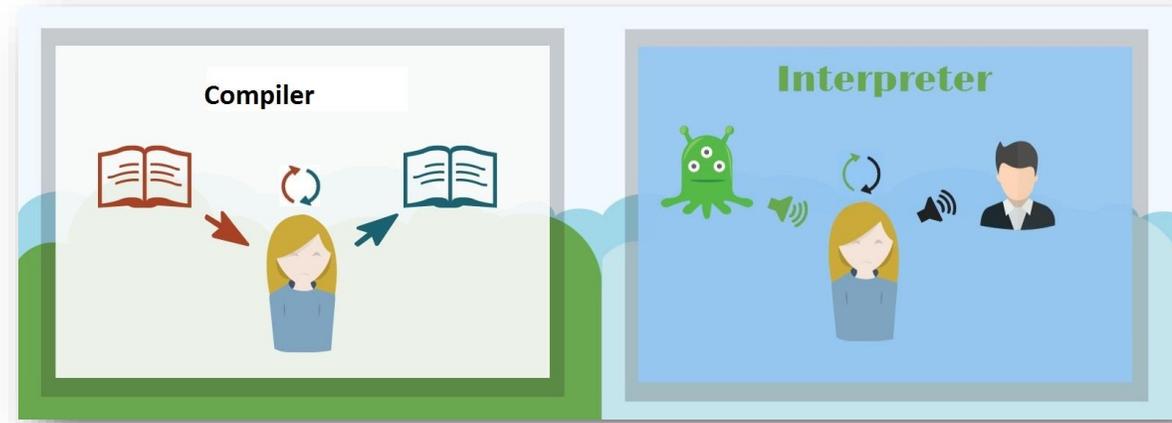
**UNIVERSITÀ  
DI PARMA**

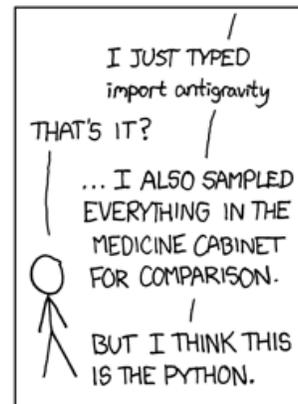
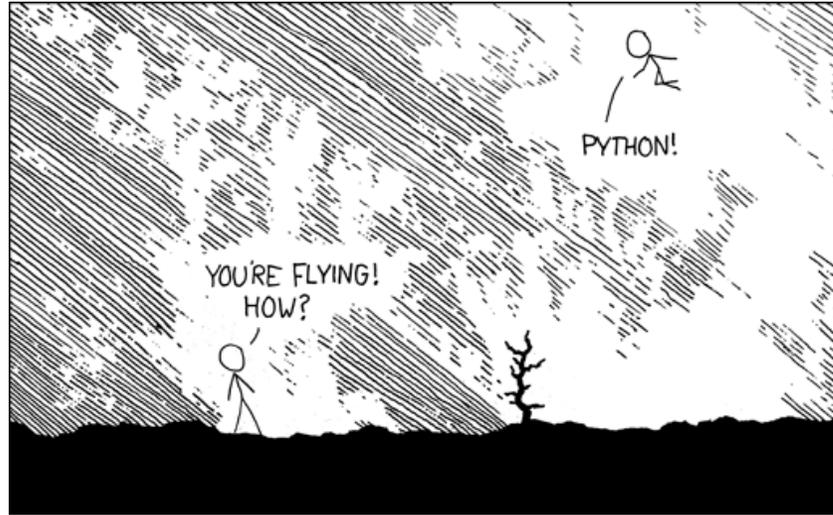
# **algoritmi in python 3**

## **informatica e laboratorio di programmazione**



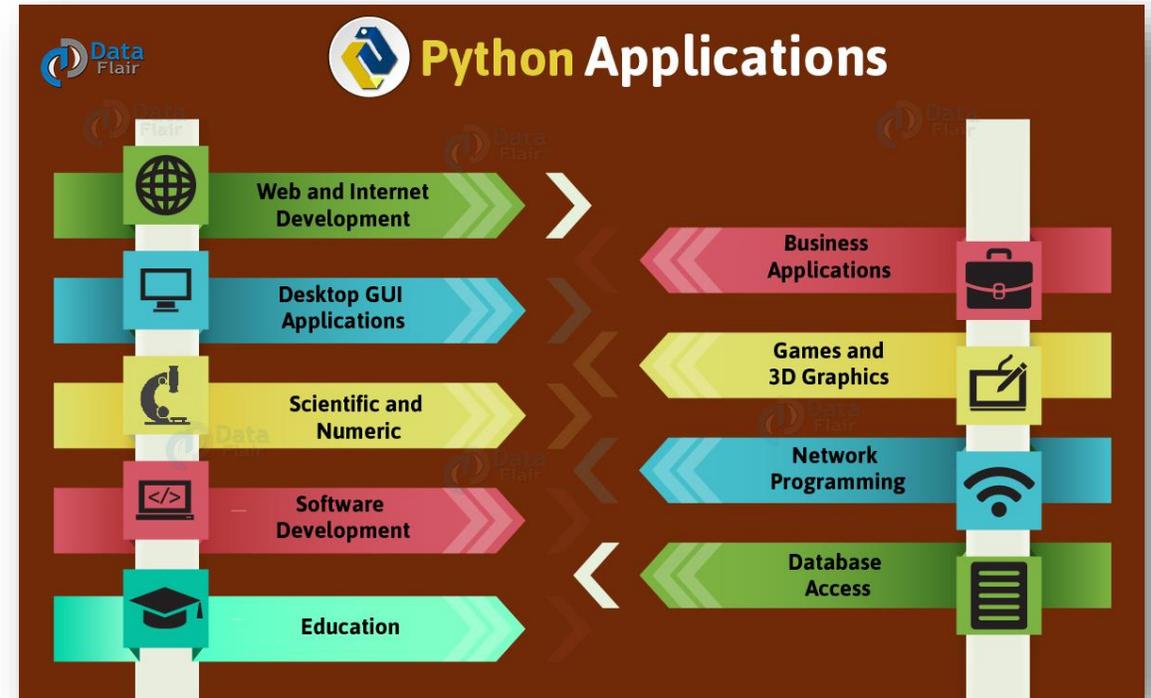
- python è un linguaggio ideato da Guido van Rossum nel **1989**
- è molto utilizzato come linguaggio di **scripting**
- è un linguaggio **interpretato**
- è dotato di un **ambiente interattivo** in cui dare comandi e ottenere immediatamente risultati
- caratteristica (quasi) unica: l'**indentazione** è **obbligatoria**





- **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

- NASA
- Yahoo!
- Google
- Youtube
- Dropbox
  
- web, data analysis, scripting, teaching, games



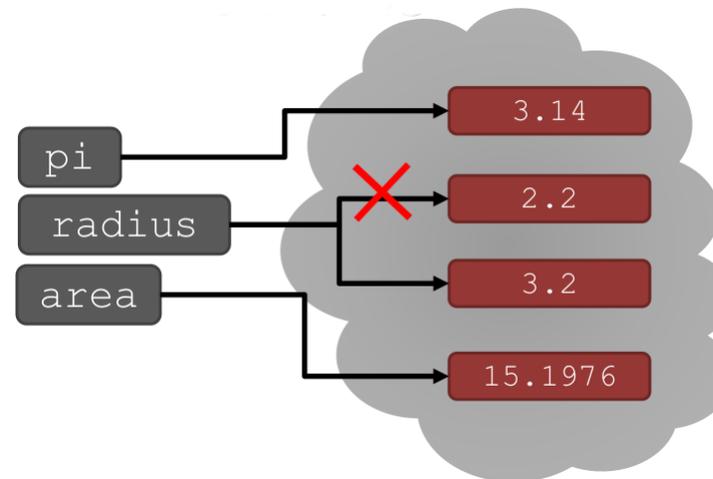
- un *tipo di dato* specifica
  - un insieme di valori
  - un insieme di operazioni ammesse su questi valori
- **int, float, bool, str**
- operazioni aritmetiche e logiche, confronti

```
>>> 10-3
7
>>> 10/3
3.3333333333333335
>>> 10//3
3
>>> 10>3
True
>>> 10<=3
False
>>> 10 > 3 or not True
True
>>>
```

- *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 # floored integer division
-1
>>> -2 % 3 # reminder is always positive
1
>>> 2 ** 1000 # no limits (but memory)
107150860718626732094842504906000181056140
481170553360744375038837035105112493612249
319837881569585812759467291755314682518714
528569231404359845775746985748039345677748
242309854210746050623711418779541821530464
749835819412673987675591655439460770629145
711964776865421676604298316526243868372056
68069376
```

- 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)
- non confondere il confronto di uguaglianza **==** con l'assegnamento **=**



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

- 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**

```
>>> x = 100
>>> DELTA = 5    # Constants: UPPER_CASE
>>> x = x + DELTA # Variables: lower_case
>>> next_position = x
# Use explicative names!
```

*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 ...*

- **str**

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

```
>>> str1 = "Monty Python's "  
>>> str2 = 'Flying Circus'  
>>> result = str1 + str2  
>>> result  
"Monty Python's Flying Circus"
```

*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*



Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	\$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

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

```
>>> 'first' < 'second'
```

```
True
```

```
>>> 'Second' < 'first'
```

```
True
```

```
>>> chr(90)
```

```
'Z'
```

```
>>> ord('Z')
```

```
90
```

- **input**

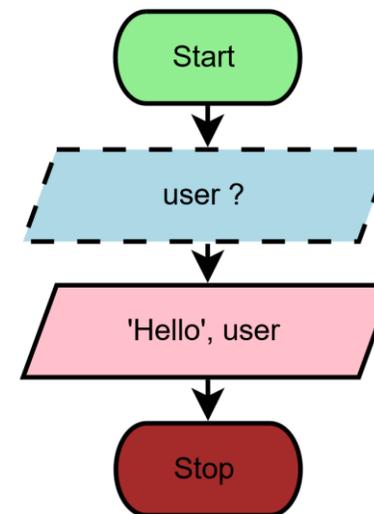
- *legge* una riga di *testo* (dallo standard input – *tastiera*)
- e la inserisce in una variabile
- prima mostra un messaggio

- **print**

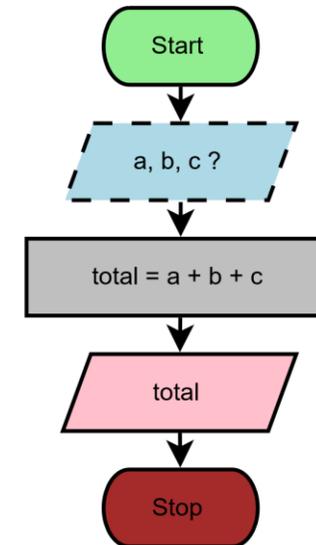
- *scrive* una serie di valori sullo standard output (*schermo*)
- tra i valori (parametri) viene inserito uno spazio

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

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



```
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)
```



algoritmi in python 3

# strutture di controllo

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

```
# File tooyoung.py

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

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

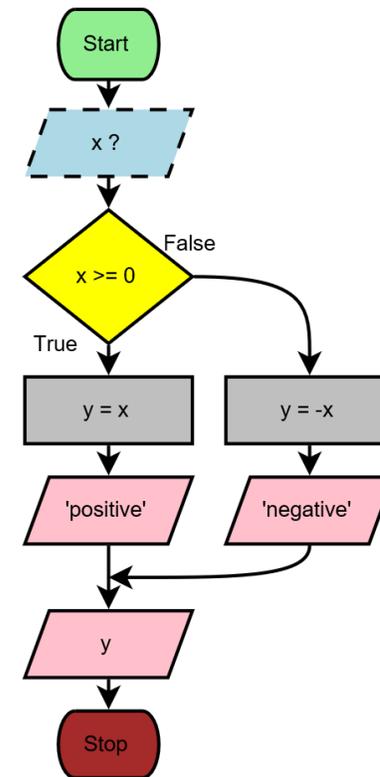
*readability counts (The Zen of Python) ... [import this]*

Beautiful is better than ugly.  
Explicit is better than implicit.  
Simple is better than complex.  
Complex is better than complicated.  
Flat is better than nested.  
Sparse is better than dense.  
Readability counts.  
Special cases aren't special enough to break the rules.  
Although practicality beats purity.  
Errors should never pass silently.  
Unless explicitly silenced.  
In the face of ambiguity, refuse the temptation to guess.  
There should be one-- and preferably only one --obvious way to do it.  
Although that way may not be obvious at first unless you're Dutch.  
Now is better than never.  
Although never is often better than *\*right\** now.  
If the implementation is hard to explain, it's a bad idea.  
If the implementation is easy to explain, it may be a good idea.  
Namespaces are one honking great idea -- let's do more of those!

Bello è meglio che brutto.  
Esplicito è meglio di implicito.  
Semplice è meglio che complesso.  
Complesso è meglio di complicato.  
Lineare è meglio che nidificato.  
Sparso è meglio di denso.  
La leggibilità conta.  
I casi speciali non sono abbastanza speciali per infrangere le regole.  
Anche se la praticità batte la purezza.  
Gli errori non dovrebbero mai essere ignorati.  
A meno che non vengano esplicitamente messi a tacere.  
In caso di ambiguità, rifiuta la tentazione di indovinare.  
Ci dovrebbe essere un modo ovvio, e preferibilmente uno solo, di fare le cose.  
Anche se questo potrebbe non essere ovvio da subito, a meno che non siate olandesi.  
Ora è meglio che mai.  
Sebbene mai sia spesso meglio che proprio adesso.  
Se l'implementazione è difficile da spiegare, l'idea è pessima.  
Se l'implementazione facile da spiegare, idea può essere buona.  
I namespace sono una grandissima idea, usiamoli il più possibile!

- nel corpo di if o else: è possibile inserire qualsiasi istruzione
  - anche altri blocchi if o altre strutture di controllo annidate!

```
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)
```



- **and, or, not** *connettivi logici* per 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)
```

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

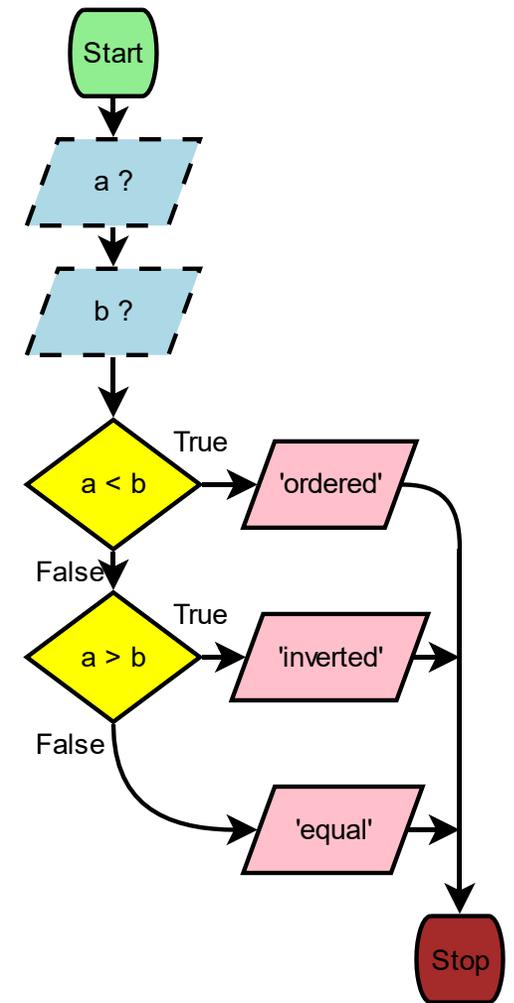
```

a = input("First word? ")
b = input("Second word? ")

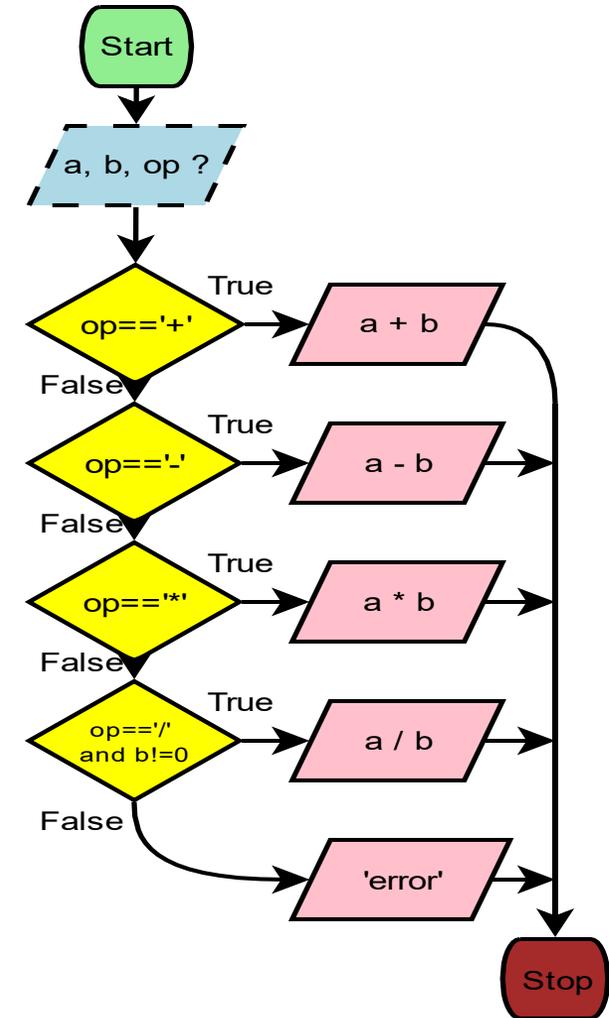
if a < b:
    print("The words are ordered")
elif a > b:
    print("The words are inverted")
else:
    print("The words are equal")

```

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



```
a = float(input())  
b = float(input())  
op = input()  
  
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")
```

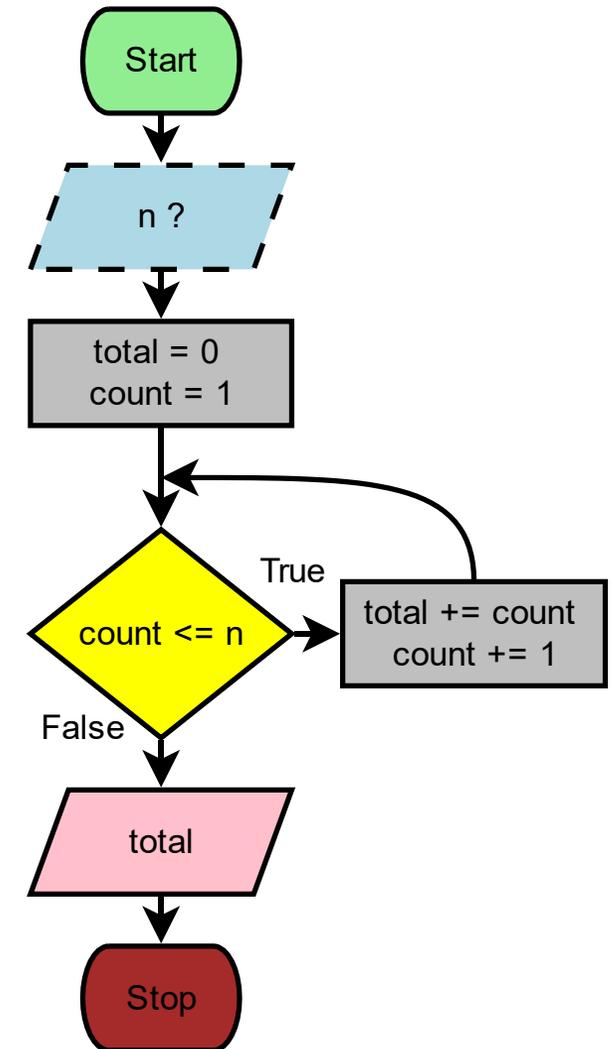


- **condizione** booleana di **permanenza** nel ciclo
- controllo preliminare (precondizione)
  - *possibile che il corpo non sia mai eseguito*

```
# 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)
```

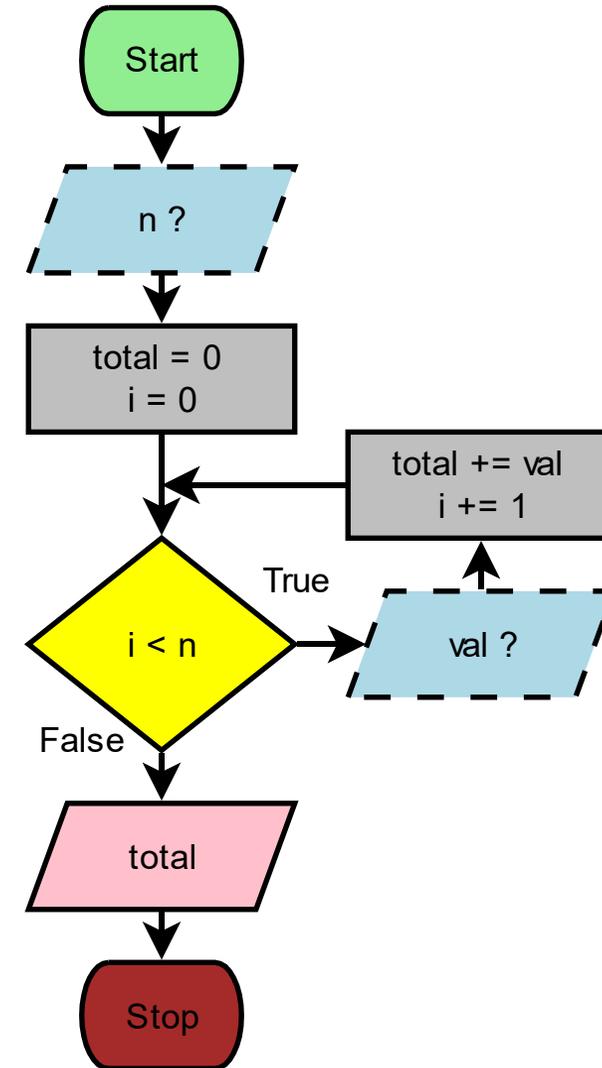


```
n = int(input("How many values? "))
total = 0
i = 0

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

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

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

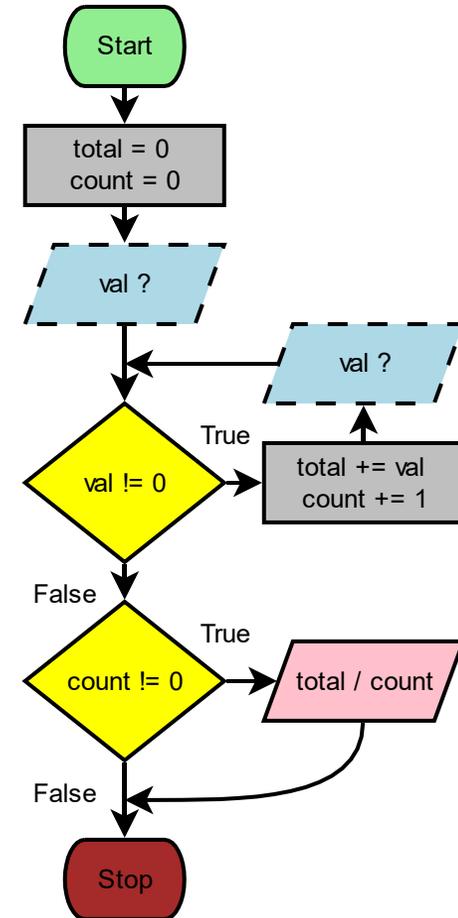


```
total = 0
count = 0

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

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

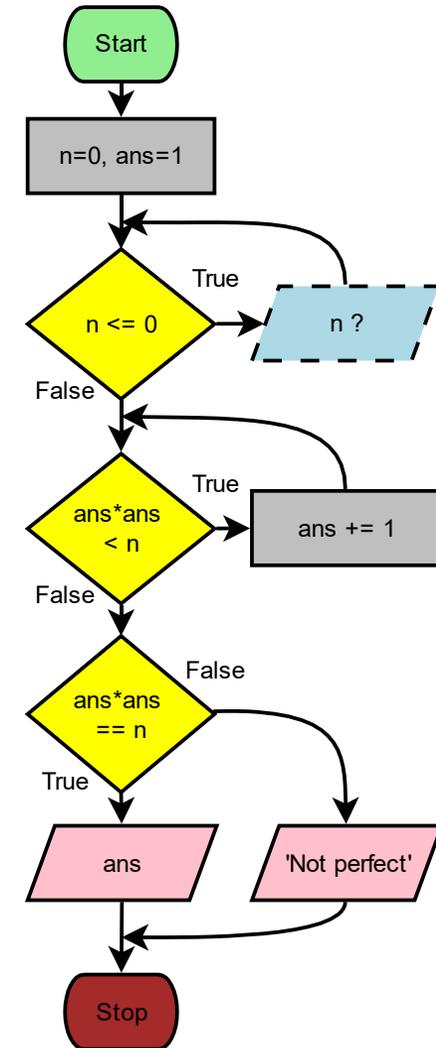
if count != 0:
    print("The average is", total / count)
```



```
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")
```



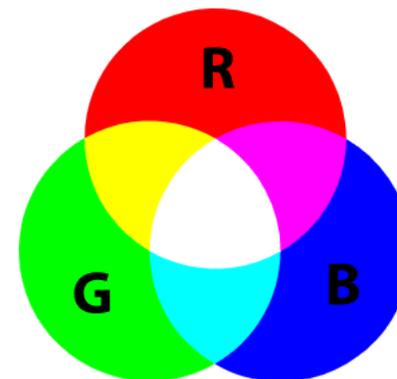
- Python Standard Library: <http://docs.python.org/3/library/>
- tutti gli import all'inizio dello script, per evidenziare le dipendenze

```
# import module
import math
y = math.sin(math.pi / 4)
print(y)

# import functions and constants from a module
from math import sin, pi
print(sin(pi / 4))

from random import randint
die1 = randint(1, 6) # like rolling a die
die2 = randint(1, 6) # like rolling a die
print(die1, die2)
```

- *sequenza immutabile* di valori, anche di tipo diverso
  - spesso tra parentesi, per separarla da altri valori
  - utili anche per grafica:
    - Color: (red, green, blue)  
Ogni componente nel range 0..255
    - Point: (x, y)
    - Size: (width, height)
    - Rect: (left, top, width, height)



```
center = (150, 100)
color = (10, 10, 200) # ~ blue
size = (640, 480)
rectangle = (150, 100, 200, 200) # square
```

- *transpiler* (*source-to-source compiler*): converte codice Python in JavaScript
  - sia il traduttore che il codice generato girano nel browser
- useremo un modulo ad-hoc: g2d
  - genera html e lancia web-server locale
  - definisce funzioni di disegno
- per l'esecuzione locale, copiare nella carella di lavoro il file g2d.py, da:  
[https://github.com/albertoferrari/info\\_lab/blob/master/esercitazioni/g2d.zip](https://github.com/albertoferrari/info_lab/blob/master/esercitazioni/g2d.zip)
- *brython.info*

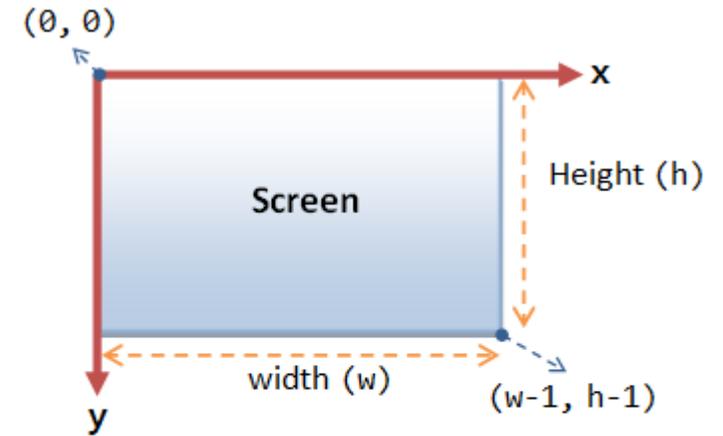
```
import g2d

# Create canvas, width=600, height=400
g2d.init_canvas((600, 400))

# Yellow rectangle, left=150, top=100, w=250, h=200
# red=255 (max), green=255 (max), blue=0 (min)
g2d.set_color((255, 255, 0))          # color
g2d.fill_rect((150, 100, 250, 200))  # rect

# Blue circle, center=(400, 300), radius=20
g2d.set_color((0, 0, 255))
g2d.fill_circle((400, 300), 20)      # circle

# Handle window events
g2d.main_loop()
```



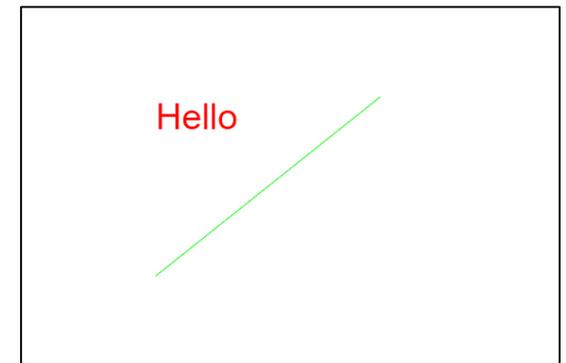
**The 2D Screen Coordinates:** The origin is located at the top-left corner, with x-axis pointing left and y-axis pointing down.

```
import g2d
g2d.init_canvas((600, 400))

g2d.set_color((0, 255, 0))
g2d.draw_line((150, 100), (400, 300)) # pt1, pt2

g2d.set_color((255, 0, 0))
g2d.draw_text("Hello", (150, 100), 40) # text, left-top, font-size

g2d.main_loop()
```

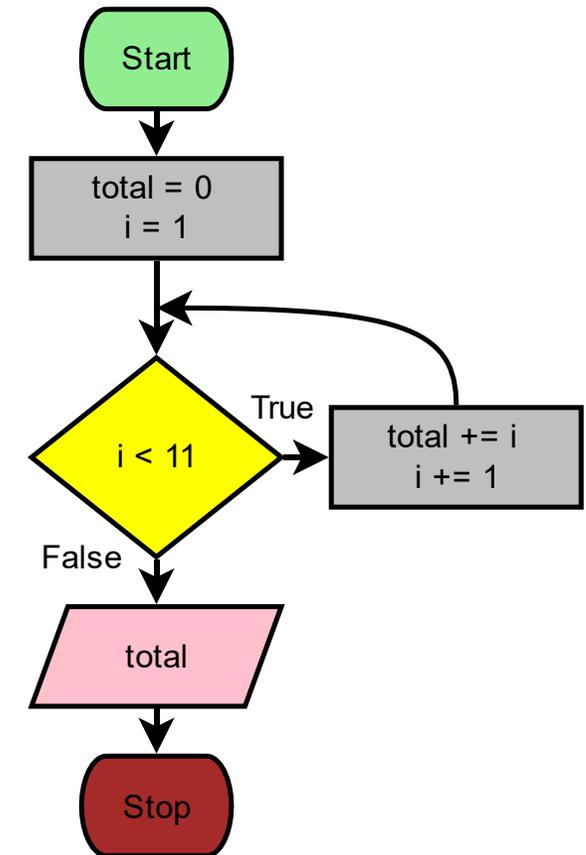


- range: intervallo di valori aperto a destra
  - estremo inferiore incluso
  - estremo superiore escluso
  - iterabile con un ciclo for

```
# Add up numbers from 1 to 10
```

```
total = 0  
for i in range(1, 11):  
    total += i  
print(total)
```

```
# total = 0; i = 1  
# while i < 11:  
#     total += i; i += 1
```

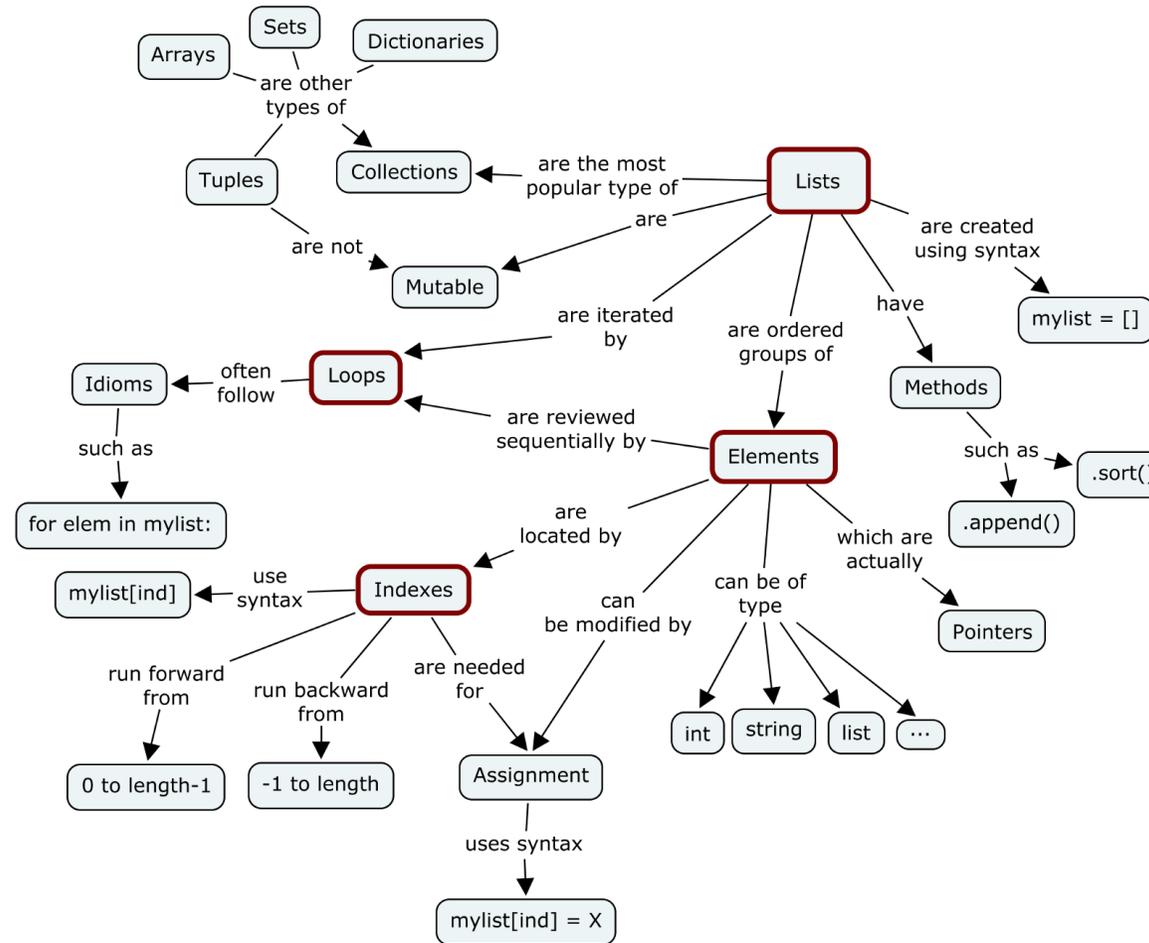


```
import g2d

g2d.init_canvas((300, 300))

for i in range(5): ## range(0, 5)
    x = i * 40
    y = x
    red = i * 60
    g2d.set_color((red, 0, 0))
    g2d.fill_rect((x, y, 100, 100))
g2d.main_loop()
```





algoritmi in python 3  
**esercizi**



- 1.1 cerchio
  - chiedere all'utente il valore del raggio  $r$  di un cerchio
  - mostrare il valore dell'area e della circonferenza
  - se  $r$  è negativo, però, mostrare un messaggio d'errore
- 1.2 minore e maggiore
  - generare e stampare tre numeri interi casuali:  $a$ ,  $b$ ,  $c$
  - ciascuno compreso tra 1 e 6
  - determinare e mostrare qual è il minore dei tre  
*controllare prima di tutto se  $a$  è minore degli altri due  
altrimenti controllare se  $b$  è minore di  $c$   
altrimenti ...*

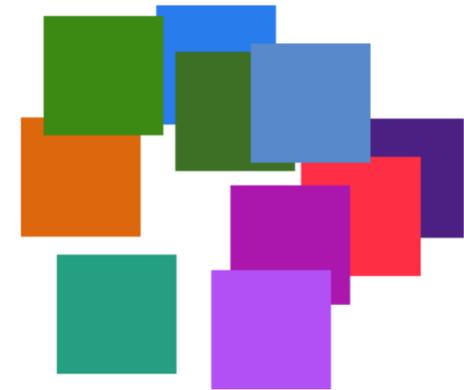
```
3.141592653589793238462643383279
5028841971693993751058209749445923
07816406286208998628034825342117067
9821 48086 5132
823 06647 09384
46 09550 58223
17 25359 4081
2848 1117
4502 8410
2701 9385
21105 55964
46229 48954
9303 81964
4288 10975
66593 34461
284756 48233
78678 31652 71
2019091 456485 66
9234603 48610454326648
2133936 0726024914127
3724587 00660631558
817488 152092096
```



- 1.3 quadrati casuali

- chiedere all'utente un numero  $n$
- disegnare  $n$  quadrati
  - tutti con lato di 100 pixel
  - ciascuno in posizione casuale
  - ciascuno con un colore casuale

*cominciare a disegnare un solo quadrato grigio, in posizione casuale*



- 1.4 numero segreto

- generare all'inizio del programma un numero “segreto” a caso tra 1 e 90
- chiedere ripetutamente all'utente di immettere un numero, finché non indovina quello generato
- ad ogni tentativo, dire se il numero immesso è maggiore o minore del numero segreto

## ○ 1.5 interesse composto

- dati di input: capitale iniziale, tasso d'interesse, durata investimento (anni)
- calcolare il capitale dopo ogni anno
- es. 100€ al 4.5%:

<b>Anno 0:</b>	<b>100.00€</b>
<b>Anno 1:</b>	<b>104.50€</b>
<b>Anno 2:</b>	<b>109.20€</b>
<b>Anno 3:</b>	<b>114.12€ ...</b>

## ○ 1.6 resistenze

- leggere, attraverso un ciclo, una sequenza di valori di resistenze elettriche
- la sequenza termina quando l'utente immette il valore 0
- alla fine, visualizzare la resistenza equivalente, sia nel caso di resistenze disposte in serie, che in parallelo

Formule utili:  
 $R_s = \sum R_i$   
 $1/R_p = \sum (1/R_i)$

- 1.7 orologio classico

- disegnare 12 tacche a raggiera, come in un orologio classico
- miglioramento: disegnare anche le tacche dei minuti, più piccole  
*usare  $\mathit{math.sin}$  e  $\mathit{math.cos}$  per determinare le posizioni in cui disegnare*



- 1.8 la stanza del mostro

- il giocatore si muove su una scacchiera di 5x5 celle, partendo da un angolo
- le righe e le colonne sono numerate da 0 a 4
- un tesoro ed un mostro sono nascosti in due posizioni casuali, all'inizio del gioco
  - ad ogni turno, il giocatore:
    - sceglie una direzione verso cui spostarsi (alto, basso, sinistra, destra)
    - se capita sulla cella del tesoro, ha vinto
    - se capita sulla cella del mostro, ha perso

