

UNIVERSITÀ
DI PARMA








algoritmi in python 3



python

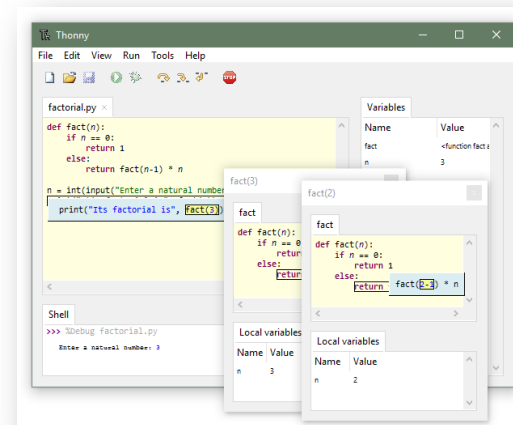
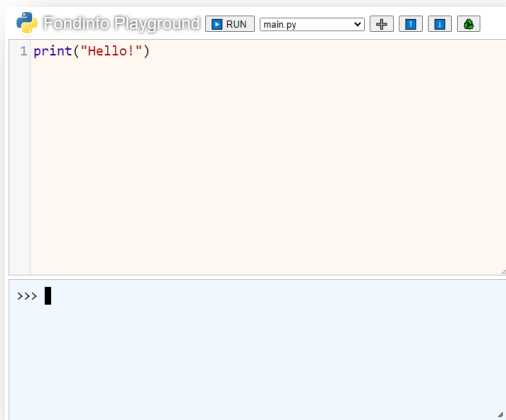
- python è un linguaggio ideato da Guido van Rossum nel 1989
- è molto utilizzato come linguaggio di scripting
- è un linguaggio interpretato
- è dotato di un ambiente interattivo in cui dare comandi e ottenere immediatamente risultati
- caratteristica (quasi) unica: l'indentazione è obbligatoria

perché python

- python è uno dei linguaggi di programmazione ***più popolari al mondo***
- usato da aziende come ***Google, Netflix, NASA ...*** ma ***facile da usare***
 -  Data Science e AI
 -  Sviluppo Web
 -  Game Development
 -  App e Automazione
 -  Ricerca scientifica
-  è ***multiplatforma*** funziona su Windows, macOS e Linux
-  è completamente gratuito e open source

cosa serve

- **playground** (<https://fondinfo.github.io/play/>)
 - sufficiente un qualunque browser (chrome, safari, firefox ...)
- alternativa **thonny** (<https://thonny.org/>)
 - installabile su Windows, Mac, Linux
- interfaccia interattiva **REPL** (Read-Eval-Print Loop)



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

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

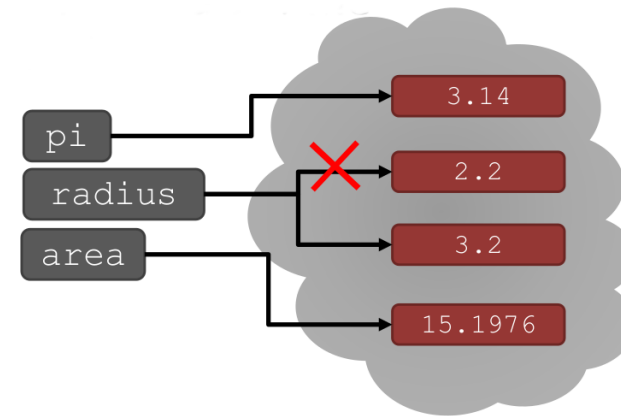
valori numerici e booleani

- **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 restituiscono 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
```

assegnamento (assegnazione)

- una variabile serve per far riferimento a un valore
- **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!
```

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**

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


stringhe di testo

- **str**

- sequenze di caratteri
- racchiusi fra apici o doppi apici
- concatenazione
- appartenenza (in)
- confronto
(segue l'ordine lessicografico)
- operatori di confronto
<, <=, >, >=, ==, !=

```
>>> str1 = "Monty Python's "  
>>> str2 = 'Flying Circus'  
>>> result = str1 + str2  
>>> result  
"Monty Python's Flying Circus"  
>>> "lyin" in result  
True  
>>> len(result)  
28  
>>> 'first' < 'second'  
True  
>>> 'Second' < 'first'  
True
```

funzioni predefinite (bult-in)

- **max**, **min**, **abs**, **len**, **round** ...
- funzioni per conversione di tipo (cast): **int**, **float**, **str** ...
- parametri delle funzioni racchiusi tra parentesi e separati da virgole
- risultato può essere assegnato a variabile

```
>>> max(3, 5)
5
>>> m = min(6, 4)
>>> m
4
>>> "5" + 3
TypeError: can only concatenate str (not "int") to str
>>> int("5") + 3
8
>>> "5" + str(3)
"53"
```

- in Python tutti i valori sono **oggetti**
 - tipi diversi → operazioni diverse, come metodi
- **attivazione** di un metodo di un oggetto
 - oggetto e metodo separati da “.”
 - parentesi obbligatorie
 - eventuali parametri tra parentesi
- metodi per gli oggetti di tipo str
 - upper, lower, count, ...

```
>>> txt = "Monty Python"
>>> shout = txt.upper() # new string returned, `txt` unchanged
>>> shout
"MONTY PYTHON"
>>> txt.count("y")
2
```

input - output

- **input**

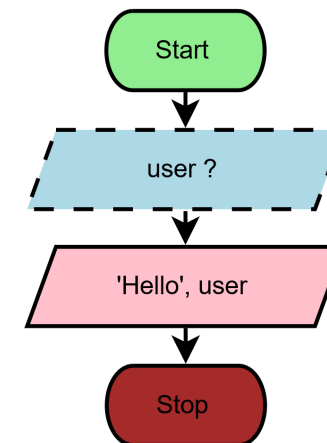
- *legge* una riga di *testo* (dallo standard input – *tastiera*)
- e la inserisce in una variabile
- prima mostra un messaggio

- **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)
```



liste

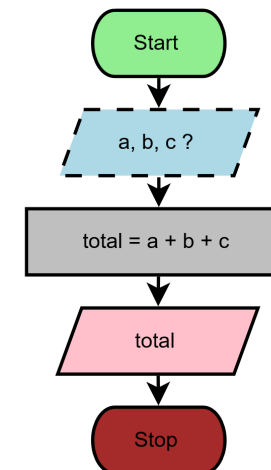
- **sequenza** mutabile di valori omogenei
 - **elementi** tra quadre, separati da virgole
- aggiunta, rimozione: `append`, `remove`
- lunghezza: **`len`**
- test di appartenenza: **`in`**

```
>>> groceries = ["spam", "egg", "beans"]
>>> groceries.append("sausage") # add "sausage" at the end
>>> len(groceries) # size has grown
4
>>> "egg" in groceries # membership test
True
>>> groceries.remove("egg") # remove "egg"
>>> len(groceries) # size has shrunk
3
>>> groceries
["spam", "beans", "sausage"]
```



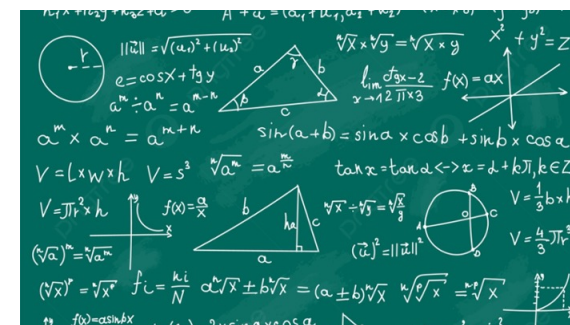
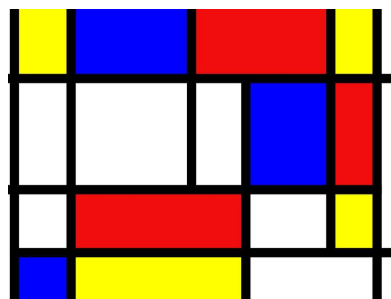
somma di 3 valori numerici

```
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)
```



Cosa succede se eliminiamo la conversione di tipo (cast)?

https://fondinfo.github.io/play/?c02_sum3.py

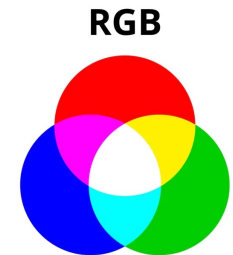
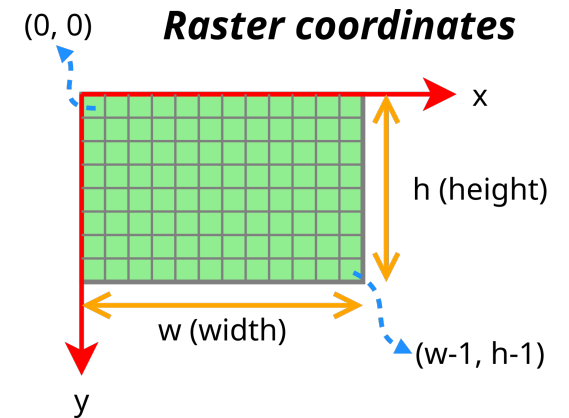


algoritmi in python 3 moduli



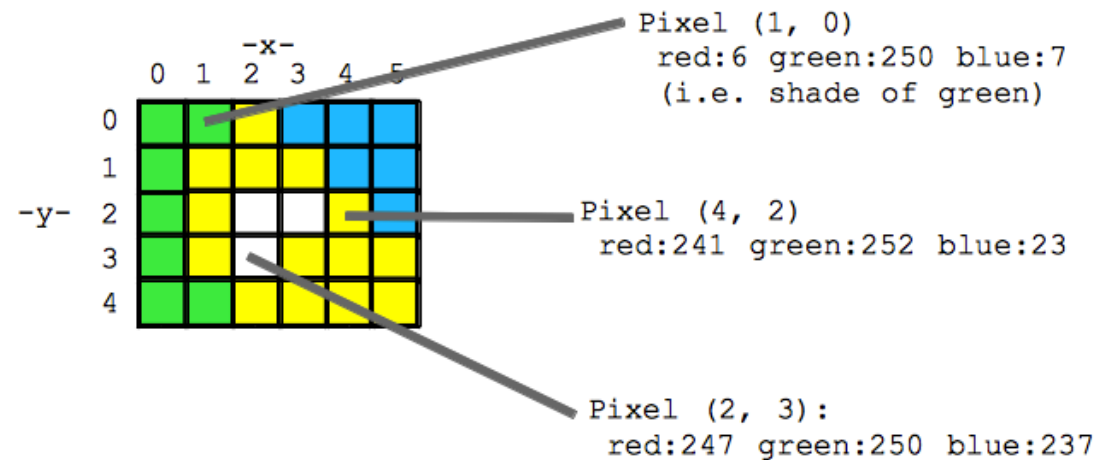
grafica 2d

- **disegno** su canvas
 - coordinate raster
 - origine in alto a sinistra
- sintesi additiva dei **colori**
 - primari: red, green, blue
- semplice modulo con funzioni per grafica bidimensionale **g2d**
 - integrato nel playground (<https://fondinfo.github.io/play/>)
 - per esecuzione in locale necessario copiare il file g2d.py nella cartella di lavoro
- dove trovo g2d?
 - <https://fondinfo.github.io/g2d/> documentazione e file
 - <http://github.com/fondinfo/fondinfo/archive/master.zip> g2d e tutti gli esempi di codice



tupla

- sequenza immutabile di valori
 - anche di tipo diverso
- spesso tra parentesi
 - per separarla da altro codice
- tupla utile per la grafica 2d
 - posizione: (x, y)
 - dimensione: (width, height)
 - colore: (red, green, blue)
 - ogni componente nel range 0..255



```
center_pt = (320, 240) # packing
window_size = (640, 480)
bluette_color = (47, 102, 207)
x, y = center_pt # sequence unpacking
```

esempio: rettangoli e cerchi

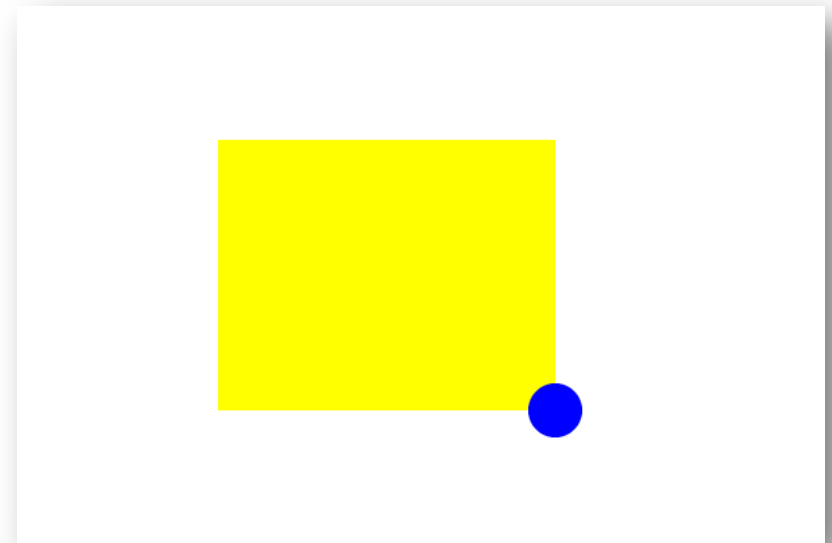
```
import g2d

g2d.init_canvas((600, 400)) # width, height

g2d.set_color((255, 255, 0)) # red + green = yellow
g2d.draw_rect((150, 100), (250, 200)) # left-top, size

g2d.set_color((0, 0, 255))
g2d.draw_circle((400, 300), 20) # center, radius

g2d.main_loop() # manage the window/canvas
```



esempio: linee e testi

```
import g2d

g2d.init_canvas((600, 400)) # width, height

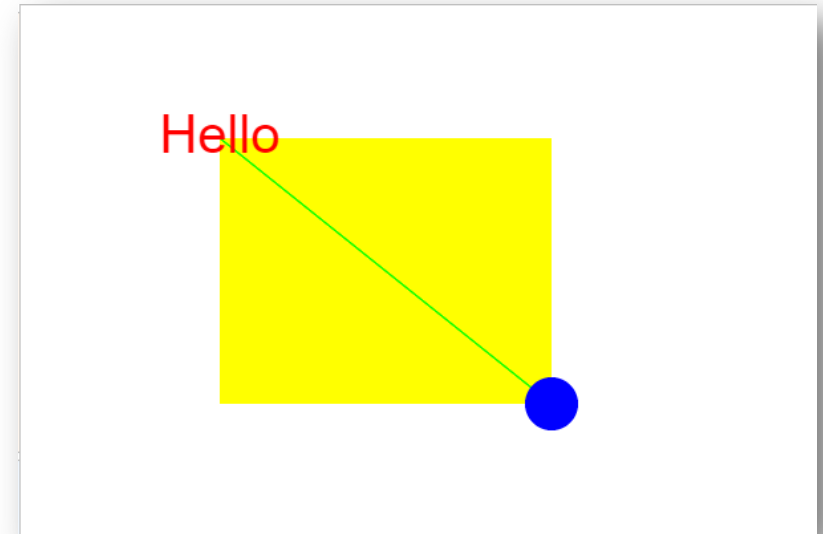
g2d.set_color((255, 255, 0)) # red + green = yellow
g2d.draw_rect((150, 100), (250, 200)) # left-top, size

g2d.set_color((0, 0, 255))
g2d.draw_circle((400, 300), 20) # center, radius

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, center, font-size

g2d.main_loop() # manage the window/canvas
```



https://fondinfo.github.io/play/?c02_draw.py

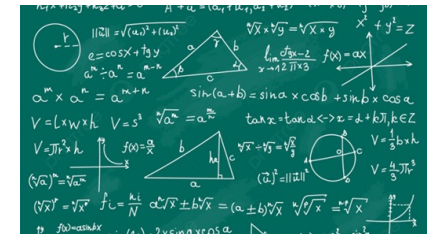
finestre di dialogo

- **g2d.prompt**: richiesta di input, in finestra, risultato str
- **g2d.alert**: visualizzazione messaggio, singolo parametro str
- **g2d.confirm**: richiesta di conferma, risultato bool

```
import g2d
g2d.init_canvas((600, 400))
name = g2d.prompt("What's your name?")
g2d.alert("Hello, " + name + "!")
g2d.main_loop()
```

moduli battery included

- Python Standard Library (<https://docs.python.org/3/library/>)
 - normalmente inclusa nelle installazioni di Python
 - contiene moduli con librerie e costanti
- modulo **math**
 - **sqrt**, **log**, **sin**, **pi**, e ...



```
# import module
import math
y = math.sin(math.pi / 4) # use namespace 'math' as prefix
print(y)

# import functions and constants from a module
from math import sin, pi
print(sin(pi / 4)) # no prefix for 'sin' 'pi'
```

<https://docs.python.org/3.13/library/math.html>

modulo random

- presente in Python Standard Library
- **randint**, **randrange**, **random**, **choice**, **shuffle**...

```
from random import randint, randrange, choice

die1 = randint(1, 6) # like rolling a die
die2 = randint(1, 6) # like rolling a die

one_of_three = randrange(3) # 0, 1, or 2

prime = choice(('alfa', 'beta', 'gamma')) # one from a sequence
```

algoritmi in python 3

strutture di controllo

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!")
```

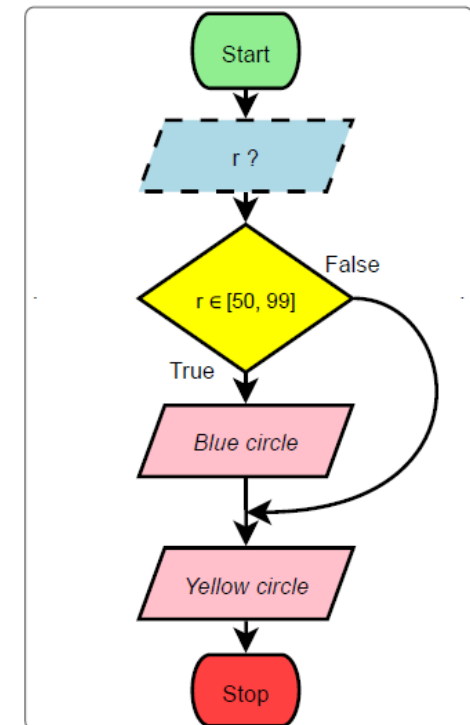
readability counts (The Zen of Python) ... [import this]

if e connettivi logici

```
...  
r = int(g2d.prompt("Radius? [50-99]"))  
if 50 <= r and r <= 99:  
    g2d.set_color((0, 0, 255))  
    g2d.draw_circle((200, 200), r)  
g2d.set_color((255, 255, 0))  
g2d.draw_circle((200, 200), 25)  
...
```

quale dei due?

```
...  
r = int(g2d.prompt("Radius? [50-99]"))  
if 50 <= r and r <= 99:  
    g2d.set_color((0, 0, 255))  
    g2d.draw_circle((200, 200), r)  
g2d.set_color((255, 255, 0))  
g2d.draw_circle((200, 200), 25)
```

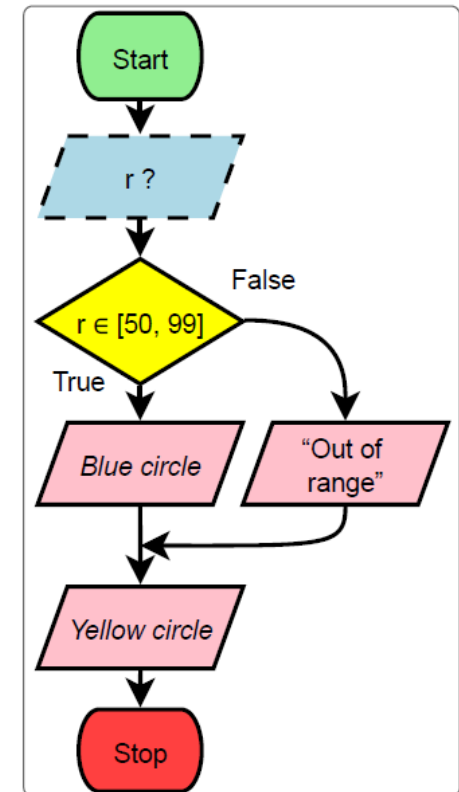


clausola else

```
...  
r = int(g2d.prompt("Radius? [50-99]"))  
if 50 <= r <= 99: # i.e.: 50 <= r and r <= 99  
    g2d.set_color((0, 0, 255))  
    g2d.draw_circle((200, 200), r)  
else:  
    g2d.alert("Out of range!")  
    g2d.set_color((255, 255, 0))  
    g2d.draw_circle((200, 200), 25)  
...
```

quale dei due?

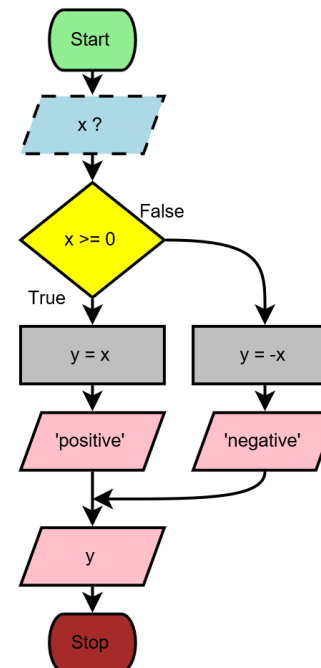
```
...  
r = int(g2d.prompt("Radius? [50-99]"))  
if 50 <= r <= 99: # i.e.: 50 <= r and r <= 99  
    g2d.set_color((0, 0, 255))  
    g2d.draw_circle((200, 200), r)  
else:  
    g2d.alert("Out of range!")  
    g2d.set_color((255, 255, 0))  
    g2d.draw_circle((200, 200), 25)  
...
```



esempio: valore assoluto

- 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)
```

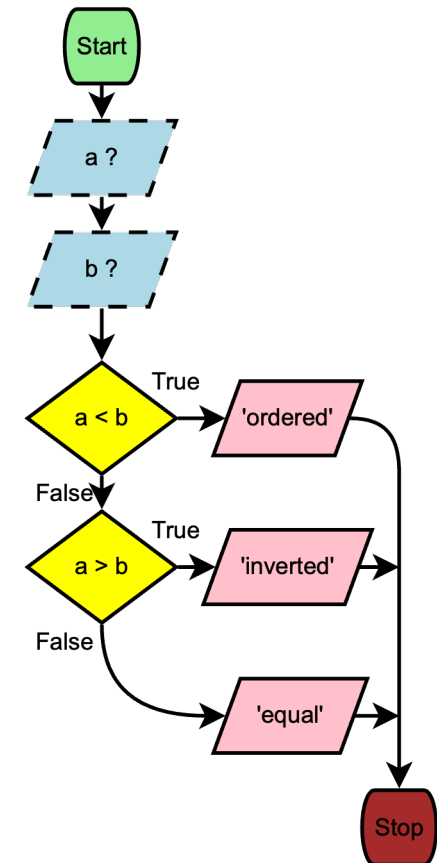


elif

- **elif**: (contrazione di *else if*) clausola *else* che contiene un altro *if*
- es. confronto fra stringhe

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

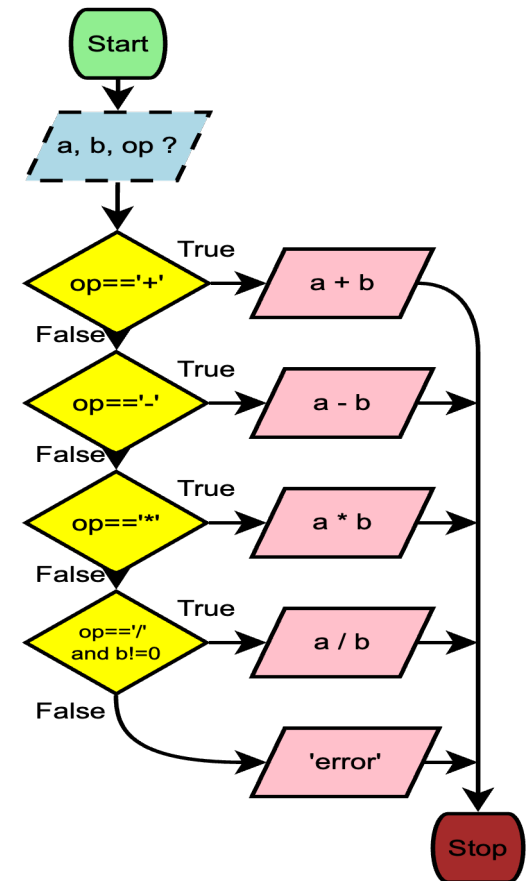
if a < b:
    print("The words are ordered")
elif a > b:
    print("The words are inverted")
else:
    print("The words are equal")
```



operazioni aritmetiche

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

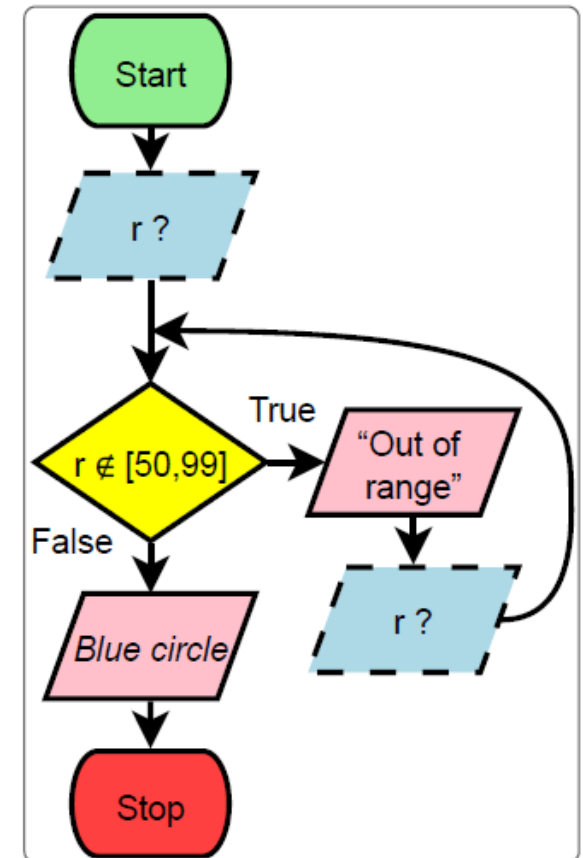
if op == '+':
    print(a + b)
elif op == '-':
    print(a - b)
elif op == '*':
    print(a * b)
elif op == '/' and b != 0:
    print(a / b)
else:
    print("Operation not allowed")
```



iterazione: while

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

```
...  
r = int(g2d.prompt("Radius? [50-99]"))  
  
while r < 50 or r > 99:  
    g2d.alert("Out of range!")  
    r = int(g2d.prompt("Radius? [50-99]"))  
  
g2d.set_color((0, 0, 255))  
g2d.draw_circle((200, 200), r)
```

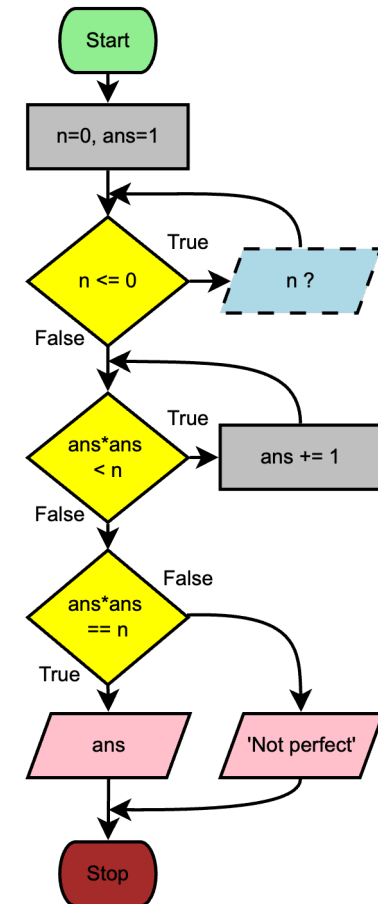


quadrato perfetto

```
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")
```



iterazione for

- opera solo su sequenze e iterabili
 - list, tuple, str, range...
- numero di iterazioni = lunghezza sequenza
- variabile di iterazione
 - a ogni iterazione = nuovo valore da sequenza

```
values = [2, 3, 5, 7, 11]
for val in values: # list
    print(val ** 3) # 8 27 125 343 1331
```



```
for r in (200, 175, 150): # tuple
    color = (randrange(256),
             randrange(256), randrange(256))
    g2d.set_color(color)
    g2d.draw_circle((200, 200), r)
```


intervallo di valori

- range : intervallo di valori aperto a destra
 - estremi: inferiore incluso (0), superiore escluso
 - se estremo inferiore $\neq 0$, servono due parametri
- reversed : sequenza rovesciata

```
for i in range(5): # 0, 1, 2, 3, 4
    print(i)
```

```
for i in reversed(range(5)): # 4, 3, 2, 1, 0
    print(i)
```

moduli

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

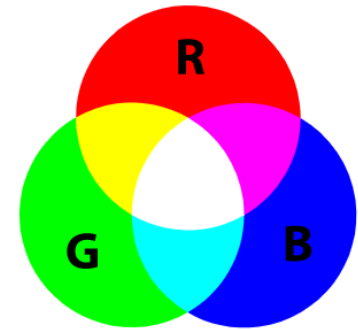
```
# import module
import math
y = math.sin(math.pi / 4)
print(y)

# 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)
```

tupla

- ***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
```

disegno su canvas

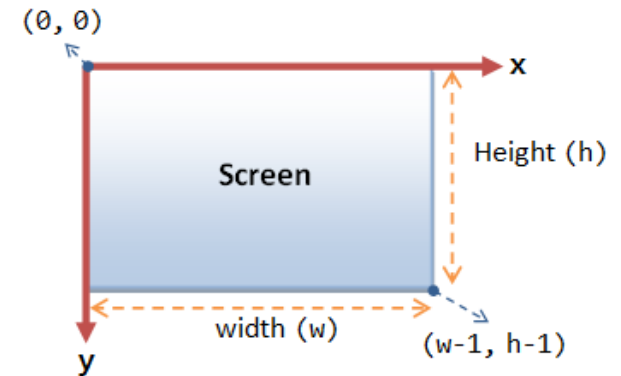
```
import g2d

# 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.

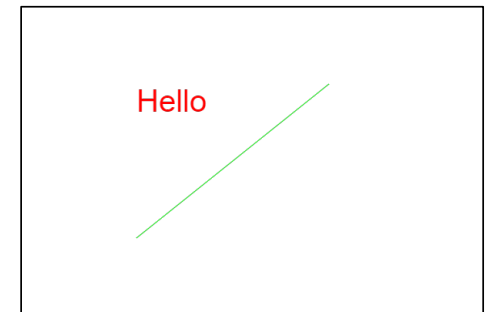
linee e testi

```
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()
```



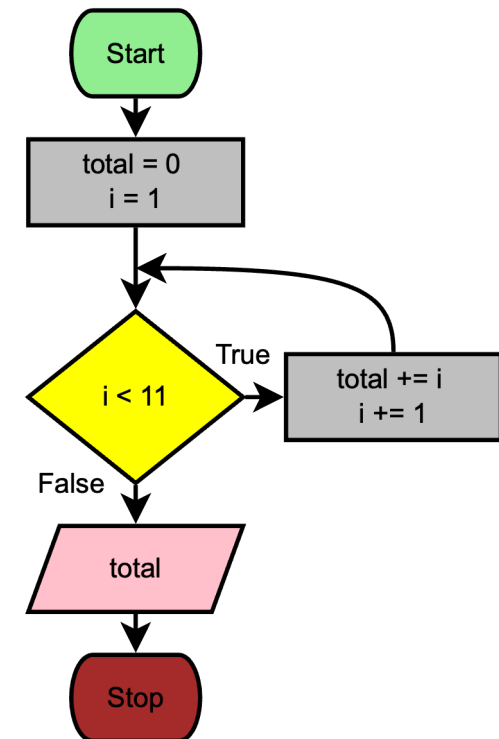
intervallo di valori: range

- 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
```

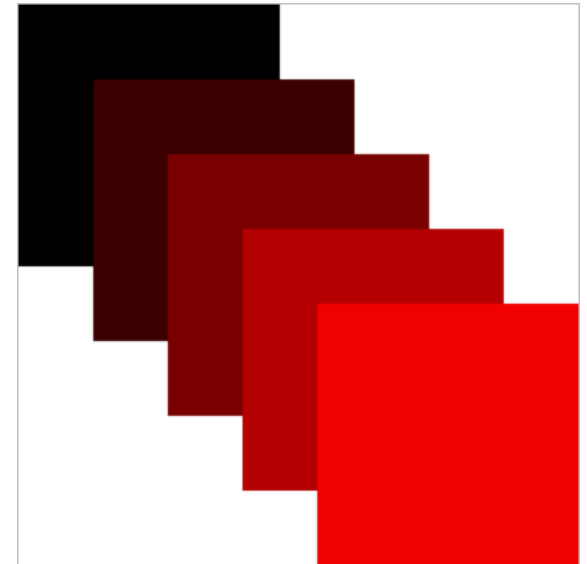


sequenza di quadrati

```
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

