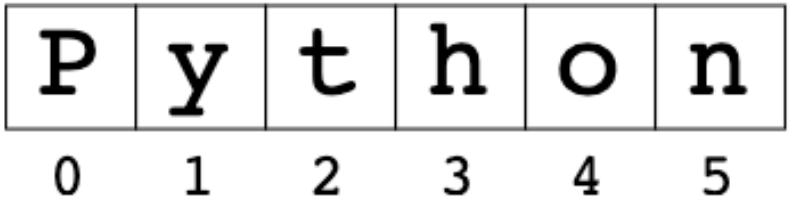
UNIVERSITÀ
DI PARMA

stringhe
e
iterazioni

stringhe

'Python' (len 6)



UNIVERSITÀ DI PARMA

confronti fra stringhe

- operatori **<, <=, >, >=, ==, !=**
- confronto in ordine lessicografico deciso dal primo carattere diverso
- ordine definito dal codice di rappresentazione
  - primi 128 codici Unicode == ASCII prima le cifre, poi le maiuscole, poi le minuscole
- operatore **in**: test di appartenenza (sottostringa)

```
>>> "art" < "arc"
False
>>> "art" < "arts"
True
>>> "arT" < "arc"
True
>>> "Py" in "Monthy Python"
True
>>> chr(83)
"S"
>>> ord("S")
83
```

UNIVERSITÀ DI PARMA

tabella ASCII

Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char	Code	Char
0	NUL	16	DLE	32	SPC	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

stringhe - cicli

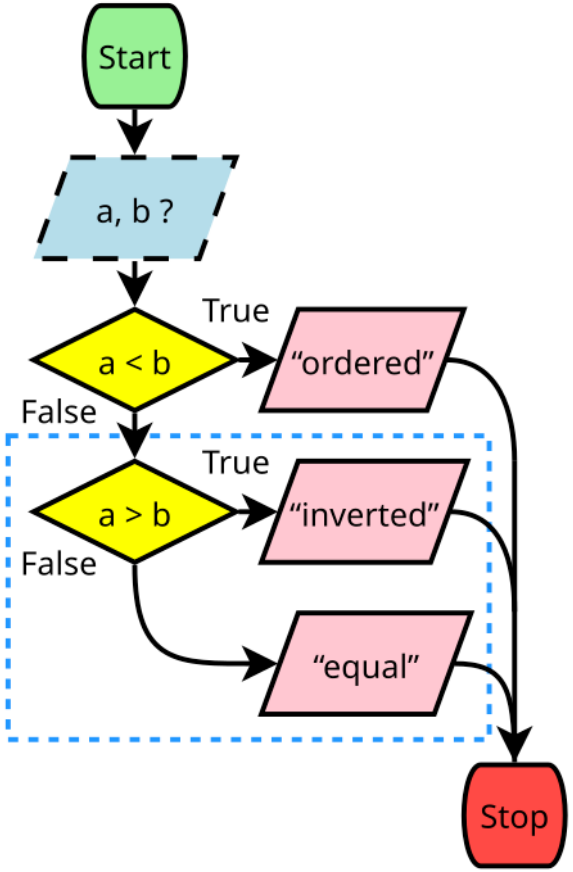
- il ciclo ***for*** scorre i valori di qualsiasi sequenza
  - una stringa è una ***sequenza di caratteri***
  - ogni ***carattere*** è ancora di tipo ***str*** con lunghezza 1

```
riga = input("inserisci lettere e cifre ").lower()
cifre, vocali = 0, 0

for c in riga:
    if "0" <= c <= "9":
        cifre += 1
    elif c in "aeiou":  # test di appartenenza
        vocali += 1

print('cifre:',cifre,'vocali',vocali)
```

stringhe - confronto

- in input due stringhe → 3 alternative
  - sono in ordine
  - sono invertite
  - sono uguali

```
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")
```



UNIVERSITÀ DI PARMA

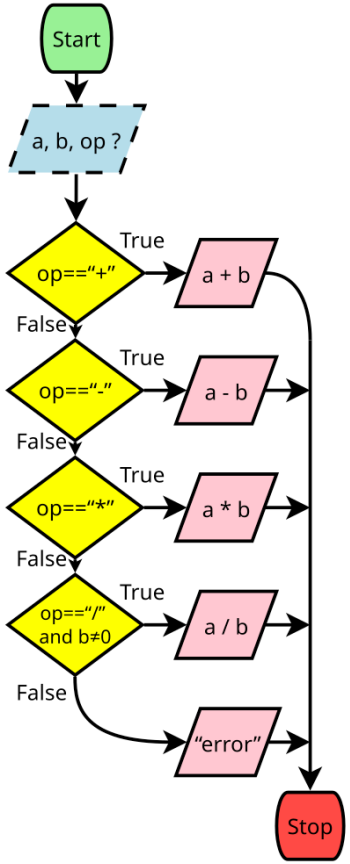
algoritmo alternativo – senza elif

- se la prima stringa non precede la seconda…
  - c'è un'altra condizione, if annidato
  - codice più complicato, meno leggibile
  - molte condizioni → annidamento crescente

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

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

operatori aritmetici

```
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")
```



UNIVERSITÀ DI PARMA

caratteri speciali per la stampa

- **\n** : ritorno a capo (codice 10 dec)
- **\t** : tabulazione orizzontale (codice 9 dec)

```
>>> print("one\ttwo\tthree\nfour\tfive\tsix")
one  two  three
four five six
```

- parametri opzionali per print
  - **end** : sequenza finale, default \n
  - **sep** : separatore tra parametri, default spazio

```
>>> for i in range(3):
print(1, 2, 3, sep=".", end=";")
1.2.3;1.2.3;1.2.3;
```

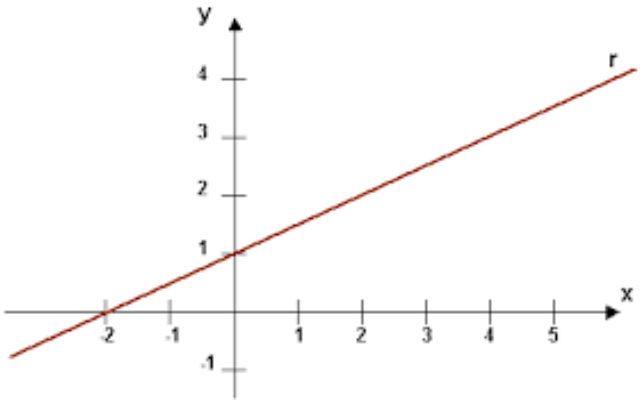
stringhe formattate

- concatenazione di stringhe : op. **+**
  - comporre stringhe con molti dati è complesso
- alternativa : **stringhe formattate** (**f-string**)
  - **f** prima delle virgolette
  - espressioni in stringa tra graffe
  - sostituite da loro rappresentazione testuale

```
radius = 2.5
area = math.pi * radius**2
# msg = ("The circle with radius " + str(radius) +
# " has area " + str(area) + ".")
msg = f"The circle with radius {radius} has area {area}."
print(msg)
```

UNIVERSITÀ
DI PARMA

cicli e linearità



linearità

- in ciclo **`for i in range(n)`**…
- ***legame lineare*** tra una grandezza v e i

$$v = m \cdot i + q$$

- se sono noti il primo e l'ultimo valore v_{first} e v_{last}:

$$\begin{cases} v_{first} = m \cdot 0 + q & \text{per } i = 0 \\ v_{last} = m \cdot (n-1) + q & \text{per } i = n-1 \end{cases} \Longrightarrow \begin{cases} q = v_{first} \\ m = \frac{v_{last} - v_{first}}{n-1} \end{cases}$$

- ***i*** *è un numero intero ed* ***m*** *è la differenza tra due istanze*
- *es. se v è la posizione di quadrati disegnati in sequenza*
- *⇒ m è la distanza tra due quadrati successivi*

UNIVERSITÀ DI PARMA

esempio: disegno di n quadrati in sequenza

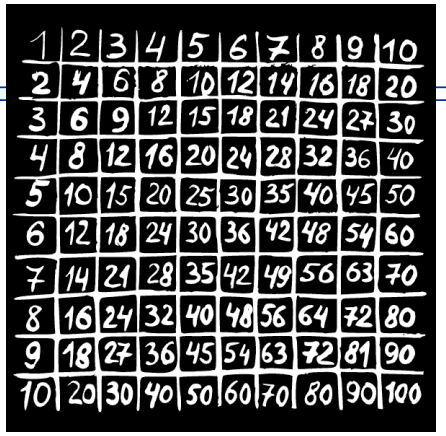
- L : misura del lato del canvas (noto)
- l : misura del lato dei quadrati (noto)
- ⇒ posizioni primo e ultimo quadrato:
  - *vfirst=0 vlast=L−l*

```
pos_fst, pos_lst = 0, L - l
pos_m = (pos_lst - pos_fst) / max(n - 1, 1) # ⚠ div by 0
for i in range(n):
    pos = pos_m * i + pos_fst
    g2d.draw_rect((pos, pos), (l, l))
```

https://fondinfo.github.io/play/?c03_squares.py

UNIVERSITÀ DI PARMA

cicli annidati



- un ciclo all'interno di un altro ciclo
- es. stampa della tavola pitagorica
  - problema più semplice: stampa di una sola riga

```
y = int(input("Insert a value: "))
for x in range(1, 11):
    print(x * y, end="\t") # keep printing on the same line
print() # print just a `newline`
```

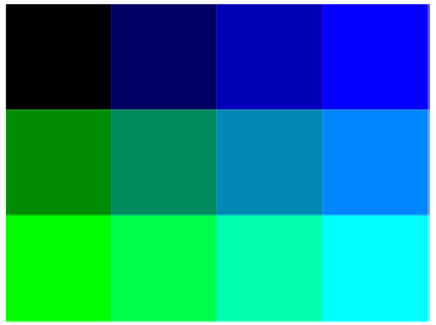
  - generalizzazione per tutte le righe

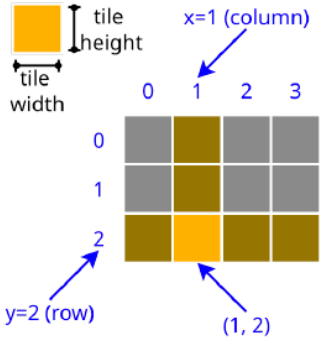
```
for y in range(1, 11):
    for x in range(1, 11):
        print(x * y, end="\t")
    print()
```

https://fondinfo.github.io/play/?c03_tables.py

griglia di colori

- visualizzare una griglia rows × cols di rettangoli
  - in orizzontale, blu cresce da 0 a 255
  - in verticale, verde cresce da 0 a 255
- suggerimenti:
  - due cicli annidati (vedi tavola pitagorica)
  - valori lineari rispetto alle variabili x o y






```
w, h = CANVAS_W / max(cols, 1), CANVAS_H / max(rows, 1)
g, b = 255 / max(rows - 1, 1), 255 / max(cols - 1, 1)
for y in range(rows):
    for x in range(cols):
        g2d.set_color((0, g * y, b * x))
        g2d.draw_rect((w * x, h * y), (w, h))
```

https://fondinfo.github.io/play/?c03_grid.py

UNIVERSITÀ
DI PARMA

cicli con sentinella

UNIVERSITÀ DI PARMA

sentinella

- ***sentinella* = *valore particolare*** che determina l'uscita dal ciclo
  - acquisizione primo valore in input
  - finché valore acquisito ≠ sentinella
  - nel ciclo elaborazione del valore
  - nel ciclo acquisizione del valore successivo
- il valore sentinella non deve essere elaborato

```
v = int(input("Value (-1 to end)? "))
while v != -1:
    print(v ** 3) # <-- in general, process v here
    v = int(input("Value (-1 to end)? "))
```

UNIVERSITÀ DI PARMA

quale ciclo usare?

- for per tutte le sequenze e gli iterabili
  - range, tuple, str, list… è più semplice e chiaro!
- while con sentinella o in casi più particolari
- esempio: conteggio da 1 a n-1

```
n = int(input("n? "))
for count in range(n):
    print(count)
```

```
n = int(input("n? "))
count = 0
while count < n:
    print(count)
    count += 1
```

UNIVERSITÀ DI PARMA

sentinella con conteggio

```
count = 0
v = int(input("Value (-1 to end)? "))
while v != -1:
    count += 1
    print(v ** 3) # <-- process here v as needed
    v = int(input("Value (-1 to end)? "))
print("Number of processed values:", count)
```

UNIVERSITÀ DI PARMA

somma di valori

- es. sommare tutti i valori in input fino a sentinella
- utilizzo di variabile per totale parziale
- acquisizione nuovo valore → aggiunto al totale

```
total = 0
v = int(input("Value (-1 to end)? "))
while v != -1: # -1 is the sentinel
   total += v
   v = int(input("Value (-1 to end)? "))
print("Sum of all values:", total)
```

- esercizio: calcolare anche la media

https://fondinfo.github.io/play/?c03_sentinel.py

formula di Gauss

- sommare tutti i numeri naturali da 1 a n

```
n = int(input("n? "))
total = 0
for i in range(1, n + 1):
    total += i
print(total)
```

- verificare risultato con la formula di Gauss

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

https://fondinfo.github.io/play/?c03_gauss.py

UNIVERSITÀ DI PARMA

valore massimo

- ricercare il massimo in una sequenza di valori
  - necessaria variabile per massimo temporaneo
  - valore iniziale: -math.inf (−∞)
  - aggiornata ogni volta che si trova un valore migliore (nuovo massimo temporaneo)

```
from math import inf
largest = -inf # -∞
for v in [3, 7, 5, 6]:
   if v > largest:
      largest = v
print(largest)
```

  - … le funzioni max e min accettano come parametro una sequenza

```
print(max([3, 7, 5, 6]))
```

inserimento valori in una lista

- non si conosce il numero di valori da inserire
  - → sentinella per terminare (nell'esempio linea vuota)

```
values = [] # list of floats
val = input("Val? (empty line to finish) ")
while val != "":
   values.append(float(val)) # float values
   val = input("Val? (empty line to finish) ")
```

non sempre è necessario memorizzare i valori
in alcuni casi è possibile elaborarli direttamente
esempio: calcolo della media dei valori in input

UNIVERSITÀ
DI PARMA

iterazioni

esercizi

somma delle potenze di 2

- chiedere all'utente un numero intero positivo ***n***
- sommare le prime ***n*** potenze di 2 $2^0+2^1+\ldots+2^n-1$
- *suggerimento: utilizzare totale parziale*
- *verificare risultato con la formula (… sempre di Gauss)*

$$\sum_{k=0}^{n-1} 2^k = 2^n - 1$$

cerchi in riga

- chiedere all'utente un valore intero n
- in un canvas $L \times L$, con $L = 500$ predefinito, disegnare una fila orizzontale di n cerchi
- i cerchi devono essere tutti di ugual dimensione e tangenti fra loro
- devono essere dimensionati e disposti in modo tale da occupare interamente la larghezza del canvas
- il loro colore deve variare linearmente da sinistra verso destra, dal nero fino al blu saturo

cerchi concentrici

- chiedere all'utente un valore intero n
- in un canvas $L \times L$, con $L = 500$ predefinito, disegnare n cerchi concentrici disposti al centro del canvas
- il diametro dei cerchi decresce linearmente, da L fino a L/n
- il colore varia linearmente dal rosso saturo del cerchio esterno fino al nero del cerchio interno

quadrati in diagonale

- chiedere all'utente un valore intero n
- in un canvas $L \times L$ con $L = 500$ predefinito, disegnare n quadrati
- quadrati tutti di ugual dimensione e posti sulla diagonale del canvas
- due quadrati successivi si sovrappongono sempre per metà del loro lato l
- i quadrati occupano l'intera diagonale del canvas
- il lato l dei quadrati non è predefinito ma dipende da n

distanza Manhattan

- in input w e h che rappresentano il numero di colonne e di righe di una tabella
- in ogni cella (x, y) della tabella inserire il valore intero calcolando la distanza di Manhattan rispetto all'angolo in alto a sinistra $(x_0, y_0) = (0, 0)$
- la distanza di Manhattan è la misura di quante celle bisogna spostarsi in orizzontale e verticale: $|\Delta x| + |\Delta y|$
- analogamente in una seconda matrice simile calcolare la distanza rispetto all'angolo in basso a sinistra: $(x0, y0) = (0, h - 1)$.

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

3	4	5	6
2	3	4	5
1	2	3	4
0	1	2	3

la stanza del mostro

- creare un gioco testuale in cui il giocatore si muove su una scacchiera virtuale di 5×5 celle
- le righe e le colonne della scacchiera sono numerate da 0 a 4
- il gioco inizia partendo da un angolo
- un ***tesoro*** e un ***mostro*** sono "nascosti" in due celle casuali
- il tesoro e il mostro non occupano la stessa cella né quella da cui inizia il gioco
- a ogni turno su chiede al giocatore una direzione (w/a/s/z) (alto, sinistra, destra, basso)
- e il giocatore si muove virtualmente di una cella
- si comunica al giocatore la sua posizione corrente: indice di colonna e indice di riga
- se il giocatore capita sulla cella del ***tesoro*** ha ***vinto***
- se invece capita sulla cella del ***mostro*** ha ***perso***

è sufficiente memorizzare tre coppie di coordinate (colonna, riga)
non è richiesto l'uso della grafica