UNIVERSITÀ
DI PARMA

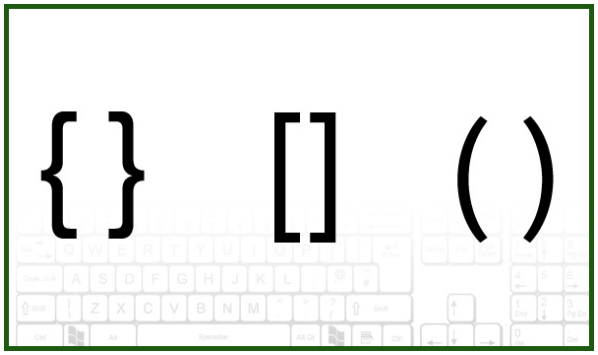
sequenze

Sequences and Collections

UNIVERSITÀ DI PARMA

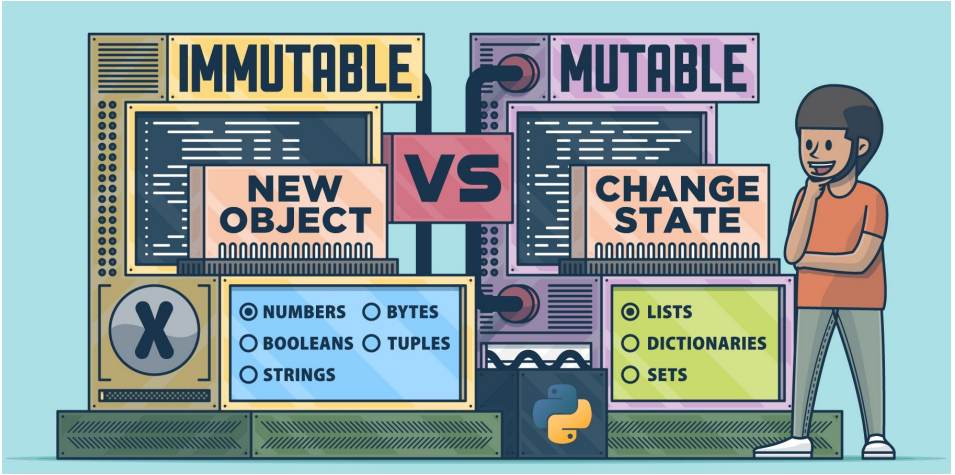
sequenze in Python

- ***sequenza*** = collezione di elementi accessibili mediante un indice
  - gli oggetti presenti mantengono l'ordine
  - tipicamente elementi tutti dello stesso tipo
- ***stringhe, liste, tuple***
- le sequenze sono iterabili (possibilità di scorrere gli elementi)



oggetti mutabili e immutabili

- in Python tutto è un ***oggetto***
- oggetti ***mutabili*** = possibile modificarne lo stato (valore)
- oggetti ***immutabili*** = non possibile modificarne lo stato
  - modifica produce un nuovo oggetto



UNIVERSITÀ
DI PARMA

lista

UNIVERSITÀ DI PARMA

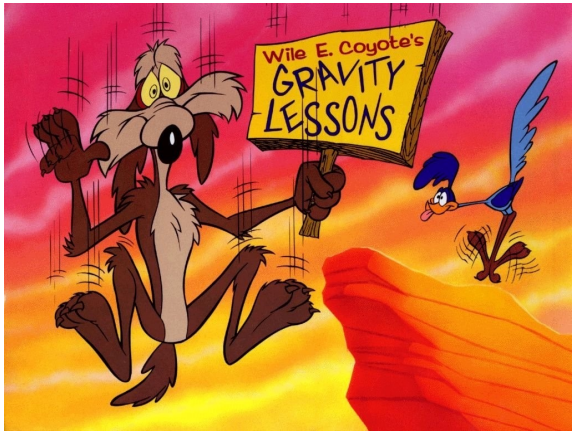
lista

- ***sequenza*** mutabile di elementi (*di solito omogenei - dello stesso tipo*)
- l'intera lista può essere assegnata ad una variabile (nome della lista)
- i singoli ***elementi*** sono ***numerati*** e accessibili per ***posizione***
  - gli ***indici*** partono da ***0***

```
to_buy = ["spam", "eggs", "beans"]

rainfall_data = [13, 24, 18, 15]

months = ["Jan", "Feb", "Mar",
          "Apr", "May", "Jun",
          "Jul", "Aug", "Sep",
          "Oct", "Nov", "Dec"]

results_by_month = [0] * 12  # 12 times 0 (list repetition)
```

accesso agli elementi

- attenzione ad usare ***indici validi!***
  - lunghezza attuale di una lista x: **`len(x)`**
  - elementi numerati da **`0`** a **`len(x)-1`**
  - indici negativi contano dalla fine




```
n = len(months)                 # 12
months[3]                       # "Apr"
months[-2]                      # "Nov", same as n - 2
to_buy = ["spam", "eggs", "beans"]

to_buy[1] = "ketchup"           # replace an element
```

UNIVERSITÀ DI PARMA

ciclo for su una lista

- nell'esempio ad ogni iterazione viene assegnato alla variabile ***val*** un elemento della lista ***values***
- ciclo ***for*** permette di iterare su qualsiasi tipo di sequenza
  - list, str, tuple, range…

```
values = [2, 3, 5, 7, 11]

print("Cubes:")

for val in values:
    cube = val ** 3
    print(cube, end="\t")
```

```
8 27 125 343 1331
```

UNIVERSITÀ DI PARMA

appartenenza, inserimento, rimozione

```
to_buy = ["spam", "eggs", "beans"]

"eggs" in to_buy              # True, to_buy contains "eggs"
to_buy.append("bacon")        # add an element to the end
to_buy.pop()                  # remove (and return) last element

to_buy.insert(1, "bacon")     # other elements shift
removed = to_buy.pop(1)       # remove (and return) element at index

to_buy.remove("eggs")         # remove an element by value
```

UNIVERSITÀ DI PARMA

slice: porzioni di lista

```
spring = months[2:5]        # ["Mar", "Apr", "May"]
quart1 = months[:3]         # ["Jan", "Feb", "Mar"]
quart4 = months[-3:]        # ["Oct", "Nov", "Dec"]
whole_year = months[:]      # Copy the whole list

list1 = ["spam", "eggs", "beans"]
list2 = ["sausage", "mushrooms"]
to_buy = list1 + list2      # List concatenation
so_boring = [1, 2] * 3      # List repetition:
                            # [1, 2, 1, 2, 1, 2]
results_by_month = [0] * 12
```

uguaglianza e identità

```
a = [3, 4, 5]
b = a[:]          # b = [3, 4, 5] -- a new list!
b == a            # True, they contain the same values
b is a            # False, they are two objects in memory
                  # (try and modify one of them...)
c = a
c is a            # True, same object in memory
                  # (try and modify one of them...)
```


```
# Lexicographical comparison of lists (or strings, tuples...)
# Compare the first two *different* elements
[2, 0, 0] > [1, 2, 0]  # True: 2 > 1
[2, 1, 0] > [2, 0, 1]  # True: 2 == 2, 1 > 0
```

funzioni su liste

```
def append_fib(data: list[int]):
    val = len(data)
    if val >= 2:
        val = data[-2] + data[-1]
    data.append(val)

def main():
    values = []
    for _ in range(12):
        append_fib(values)
    print(values)  # let's see what's going on

main()
```

unpacking (spacchettamento

- qualsiasi sequenza può essere spacchettata su un numero corrispondente di variabili

```
a, b, c = [1, 2, 3]
```

- assegnamento con stella per catturare una sequenza in una lista
  - provare a omettere first, second, o last

```
>>> first, second, *middle, last = [0, 1, 2, 3, 4, 5]  # range(6)
>>> first
0
>>> second
1
>>> middle
[2, 3, 4]
>>> last
5
```

stringhe e liste

UNIVERSITÀ DI PARMA

stringhe e liste

- ***stringa***: sequenza ***immutabile*** di caratteri
- ***join*** e ***split***: da lista a stringa e viceversa

```
txt = "Monty Python's Flying Circus"
txt[3]    # "t"
txt[-2]   # "u"
txt[6:12] # "Python"
txt[-6:]  # "Circus"

days = ["tue", "thu", "sat"]
txt = "|".join(days)  # "tue|thu|sat"

days = "mon|wed|fri".split("|")  # ["mon", "wed", "fri"]
```

UNIVERSITÀ DI PARMA

ciclo for su stringhe

- il ciclo ***for*** scorre i valori di qualsiasi sequenza
  - una stringa è una sequenza di caratteri

```
line = input("Text? ").lower()
digits, vowels = 0, 0

for c in line:
    if "0" <= c <= "9":  # char comparison
        digits += 1
    elif c in "aeiou":  # membership test
        vowels += 1
```

UNIVERSITÀ DI PARMA

esempio: testo fra marcatori

- visualizza solo la porzione del testo interna ai marcatori <..>

```
text = input("Text? ")
inside = False

for c in text:
    if c == "<" and not inside:
        inside = True
    elif c == ">" and inside:
        inside = False
        print()
    elif inside:
        print(c, end="")
```

https://fondinfo.github.io/play/?c08_brackets.py

UNIVERSITÀ DI PARMA

esempio: lista di contatori

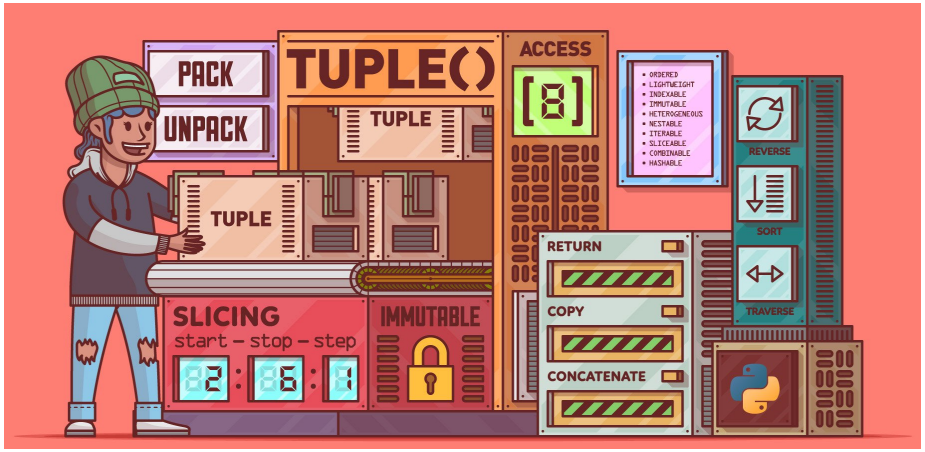
- contare separatamente le cifre in un testo
  - quanti 0? quanti 1? …
    - 10 condizioni, 10 variabili contatore
    - oppure lista di 10 elementi

```
text = input("Text? ")
counters = [0] * 10

for c in text:
    if "0" <= c <= "9":
        counters[int(c)] += 1

print(counters)
```

https://fondinfo.github.io/play/?c08_counters.py

UNIVERSITÀ
DI PARMA

tupla



tupla

- sequenza ***immutabile*** di valori (*anche di tipo diverso*)

```
# Tuple packing
pt = (5, 6, "red")
pt[0]  # 5
pt[1]  # 6
pt[2]  # "red"

# Sequence unpacking (from a list, string, tuple...)
x, y, colour = pt
a, b = 3, 4
a, b = b, a
```

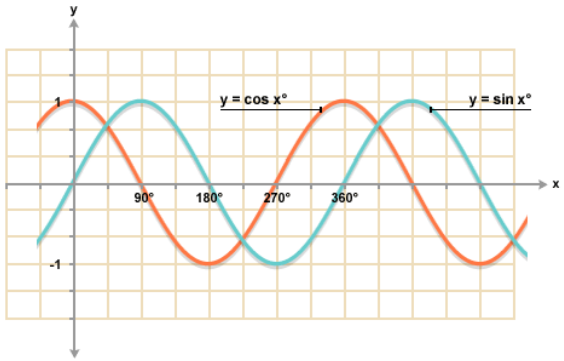
UNIVERSITÀ
DI PARMA

liste e tuple

esercizi

valori precalcolati

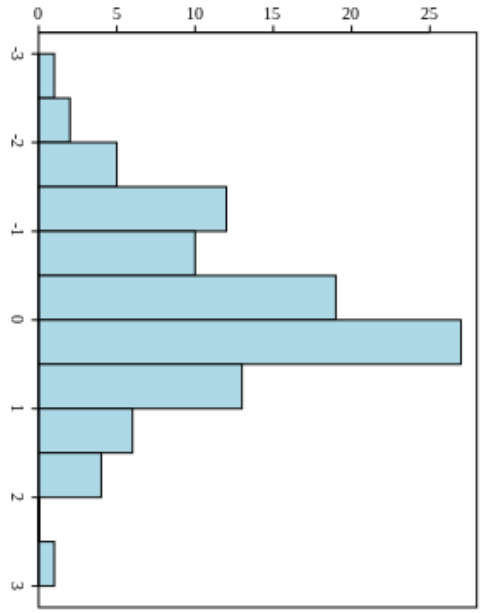
- riempire una lista con i valori di sin(x)
  - 360 elementi, indice x tra 0 e 359
- poi, ciclicamente...
  - chiedere un angolo all'utente
  - visualizzare il corrispondente valore precalcolato del sen
- *nota*
  - math.sin opera su radianti
  - calcolare math.sin(x * math.pi / 180), anzichè math.sin(x)



https://github.com/albertoferrari/info_lab/blob/master/codice_lezioni/sl05_03_es_01_sin.py

istogramma con barre orizzontali

- chiedere all'utente una lista di valori positivi
- la lista termina quando l'utente inserisce il valore 0
- mostrare un istogramma
- larghezza di ciascuna barra proporzionale al valore corrispondente
- la barra più lunga occupa tutto lo spazio disponibile



https://github.com/albertoferrari/info_lab/blob/master/codice_lezioni/sl05_03_es_02_istogramma.py

risultati casuali

- simulare n lanci di una coppia di dadi
  - n scelto dall'utente
- contare quante volte si presenta ciascun risultato
  - risultati possibili: da 2 a 12 (somma dei due dadi)
- per conteggiare i vari risultati, usare una lista di (almeno) 11 valori

https://github.com/albertoferrari/info_lab/blob/master/codice_lezioni/sl05_03_es_03_dadi.py

UNIVERSITÀ DI PARMA

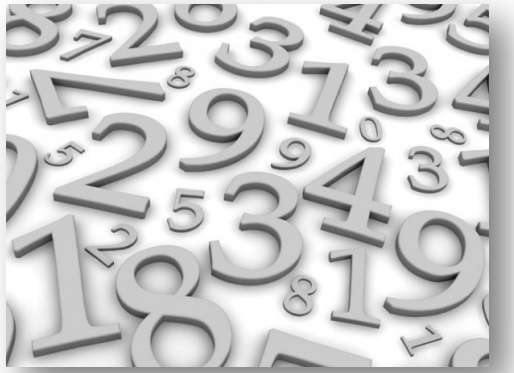
conteggio caratteri

- chiedere una riga di testo all'utente
- contare separatamente le occorrenze di ciascuna lettera maiuscola (da 'A' a 'Z')
- creare una lista (array) di 26 elementi
  - inizialmente tutti posti a 0
- ciascun elemento è il contatore per una certa lettera
- l'indice del contatore corrispondente ad una lettera val può essere ottenuto come ***ord(val) - ord('A')***

https://github.com/albertoferrari/info_lab/blob/master/codice_lezioni/sl05_03_es_04_caratteri.py

UNIVERSITÀ DI PARMA

fattori primi



- funzione che trova tutti i fattori primi di un numero n
  - parametro: n
  - risultato: lista, contenente i fattori primi di n
- algoritmo: scorrere tutti i valori d'interesse, e cercare i divisori
  - x è divisore di n sse n % x == 0
  - non considerare i fattori non primi
- provare la funzione con valori inseriti dall'utente

quando si trova un divisore x, dividere ripetutamente n per x, finché resta divisibile

valutare l'uso di un ciclo while, anzichè for

https://github.com/albertoferrari/info_lab/blob/master/codice_lezioni/sl05_03_es_05_fattori_primi.py

memory

- l'utente sceglie righe e colonne
- allocare una lista di dimensione n=righe×colonne (pari)
- inserire in ordine le prime lettere dell'alfabeto
  - ciascuna ripetuta due volte
- mescolare le celle
  - per ciascuna cella, scegliere una posizione a caso e scambiare il contenuto delle celle
- mostrare la lista, andando a capo per ogni riga
- usare una lista semplice, ma nella visualizzazione introdurre dei ritorni a capo

cella a inizio riga: il suo indice i è multiplo di colonne, ossia i % colonne == 0
cella a fine riga: i % colonne == colonne – 1
per cominciare, inserire nella lista valori numerici crescenti, anziché lettere

https://github.com/albertoferrari/info_lab/blob/master/codice_lezioni/sl05_03_es_06_memory.py