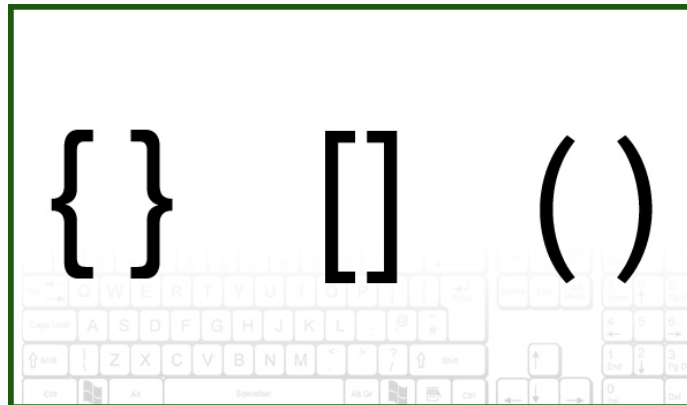




UNIVERSITÀ
DI PARMA

liste e tuple

informatica e laboratorio di programmazione



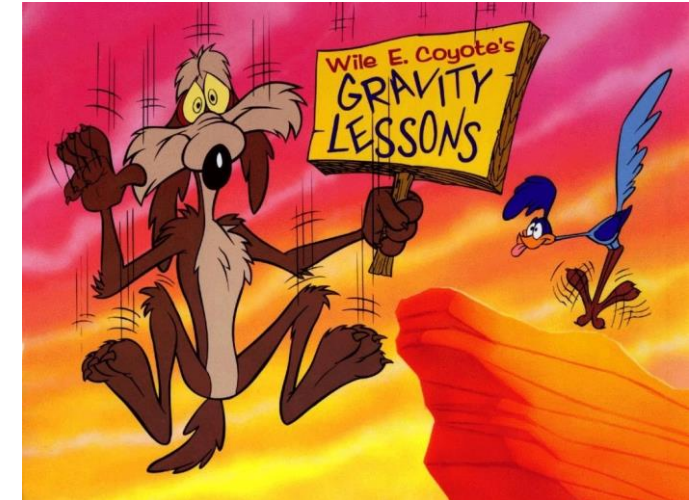
- ***sequenza*** di elementi (*di solito dello stesso tipo*)
- l'intera lista può essere assegnata ad una variabile, così diamo un nome alla lista
- i singoli ***elementi*** sono ***numerati*** per ***posizione***
- gli ***indici*** partono da ***0***

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

rainfall_data = [13, 24, 18, 15]

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

- 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
```

```
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
```

```
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
```



```
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...)
```



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

```
txt = "Monty Python's Flying Circus"
txt[0]      # 'M'
txt[1]      # 'o'
txt[-1]     # 's'
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"]
```

- il ciclo *for* permette di iterare su qualsiasi tipo di sequenza
 - *lista, stringa, tupla, range...*
- *nell'esempio* ad **ogni iterazione**, alla variabile *el* è assegnato un elemento della lista *corsi*

```
corsi = ['LISI', 'LIET', 'Fisica']  
  
for el in corsi:  
    print(el)
```



```
def limita_valori(lis, limite):  
    '''  
    fissa un limite massimo ai valori della lista lis  
    '''  
    for i in range(len(lis)):  
        if lis[i] > limite:  
            lis[i] = limite  
  
def stampa_valori(lis):  
    for i, val in enumerate(lis):  
        print('indice', i, 'valore', val)  
  
dati = [5, 4, 2]  
limita_valori(dati, 4)  
print(dati)  
stampa_valori(dati)
```

The enumerate() method adds counter to an iterable and returns it (the enumerate object)

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

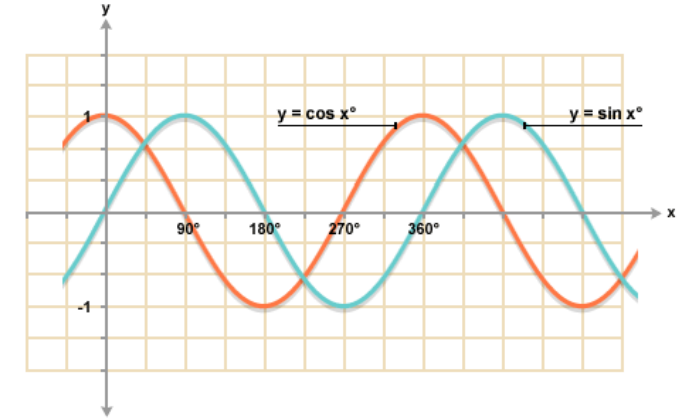
```
# Tuple packing
pt = 5, 6, "red"
pt[0]  # 5
pt[1]  # 6
pt[2]  # "red"

# multiple assignments, from a tuple
x, y, colour = pt  # sequence unpacking
a, b = 3, 4
a, b = b, a
```

liste e tuple
esercizi



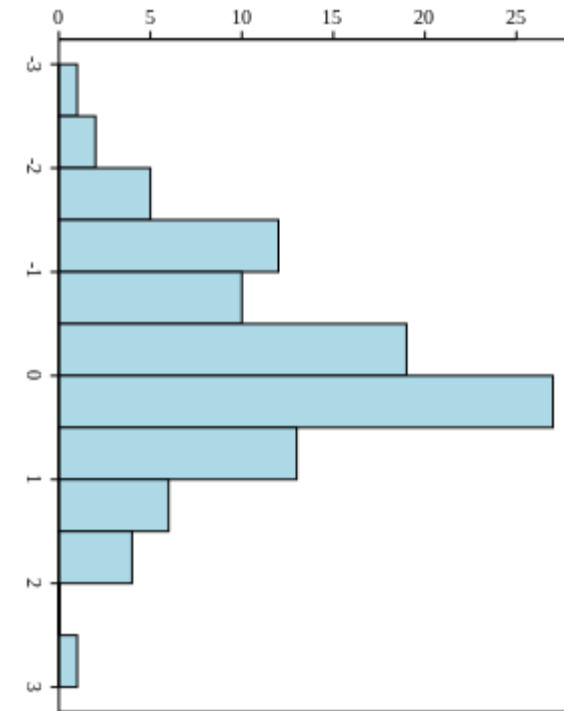
- 5.1 valori precalcolati
 - riempire una lista con i valori di $\sin(x)$
 - 360 elementi, indice x tra 0 e 359
 - poi, ciclicamente...
 - chiedere un angolo all'utente
 - visualizzare il corrispondente valore precalcolato del seno
 - *nota*
 - `math.sin` opera su radianti
 - calcolare `math.sin(x *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



http://www.ce.unipr.it/brython/?p3_list_sin.py

- 5.2 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

http://www.ce.unipr.it/brython/?p3_list_histogram.py



- 5.3 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



http://www.ce.unipr.it/brython/?p3_list_dice.py

- 5.4 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')***

<http://www.ce.unipr.it/brython/?p3 list letters.py>

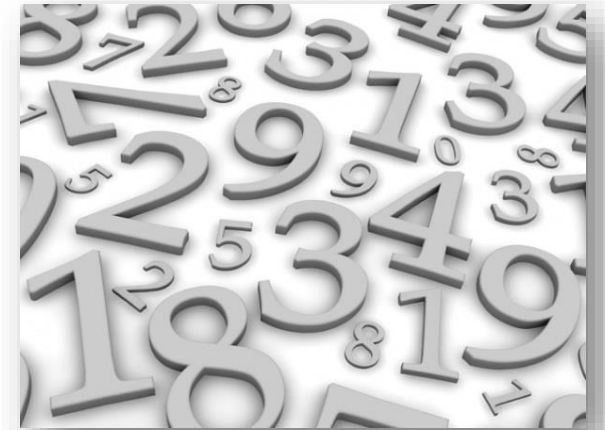


○ 5.4 fattori primi

- 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`, anziché `for`*

http://www.ce.unipr.it/brython/?p3_list_factors.py



○ 5.4 memory

- l'utente sceglie righe e colonne
- allocare una lista di dimensione $n = \text{righe} \times \text{colonne}$ (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 $i \% \text{colonne} == 0$
cella a fine riga: $i \% \text{colonne} == \text{colonne} - 1$
per cominciare, inserire nella lista valori numerici crescenti, anziché lettere*

http://www.ce.unipr.it/brython/?p3_list_couples.py