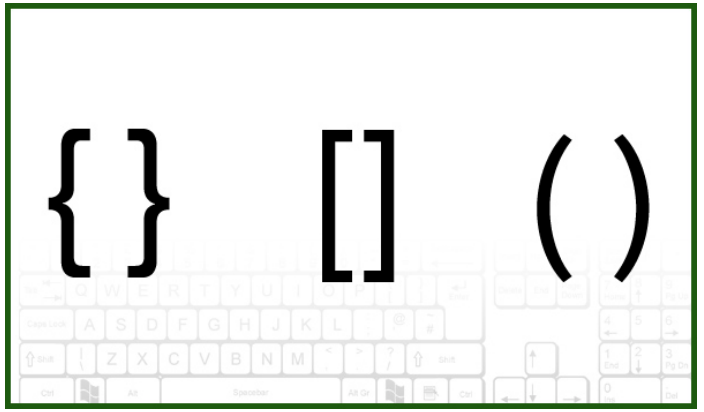
sequenze



sequenze

- liste
  - funzioni e metodi su liste
- uguaglianza e identità
- ciclo for
- range
- tuple

lista

- ***sequenza*** di elementi (*di solito dello stesso tipo*)
- l'intera lista può essere assegnata ad una variabile
- 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"]
```

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
print('length of months',n)
print('months[3] = ',months[3])          # "Apr"
print('months[-2] = ',months[-2])        # "Nov", same as n - 2

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

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
```

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...)
```

stringhe e liste

- ***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"]
```

for - ciclo su liste

- il ciclo ***for*** permette di iterare su qualsiasi tipo di sequenza
  - ***lista, stringa, tupla, range***…
- *nell'esempio* ad **ogni iterazione**, alla variabile ***t*** è assegnato un elemento della lista ***topics***

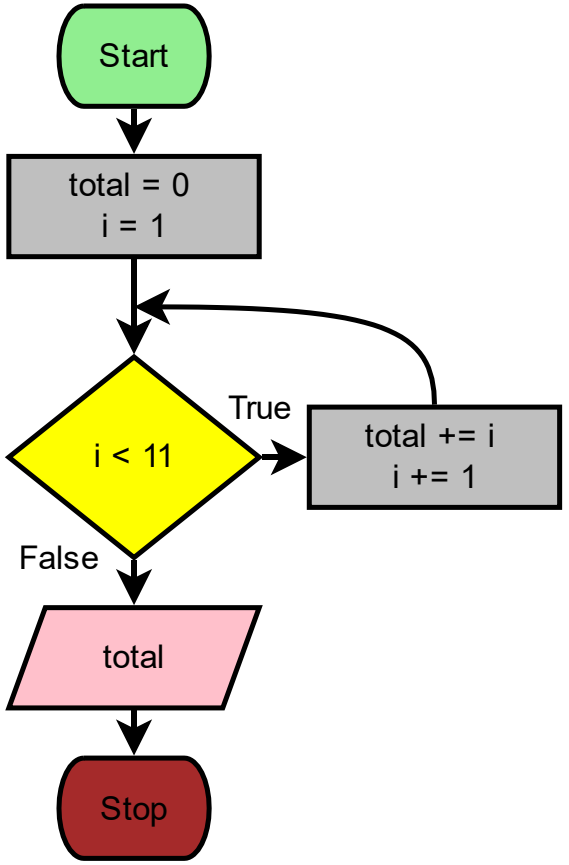
```
topics = ['Artificial Intelligence','Machine Learning','Deep Learning']
for t in topics:
    print(t)
```

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
```



funzioni su liste

```
def limitaValori(lis,limite):
    ''' fissa un limite massimo ai valori della lista lista '''
    for i in range(len(lis)):
        if lis[i] > limite:
            lis[i] = limite

def stampaValori(lis):
    for i, val in enumerate(lis):
        print('indice', i, 'valore', val)

dati = [5, 4, 2]
print(dati)
limitaValori(dati, 4)
print(dati)
stampaValori(dati)
```

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

tupla

- 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
```

passaggio dei parametri – call by object

- parametri passati «per oggetto»
  - se il parametro è una ***variabile*** le modifiche non si ripercuotono all'esterno
  - se il parametro è una ***lista*** o un ***oggetto*** le modifiche si ripercuotono

```
def inc(f):
    f = f + 1
    print(f) # 11


a = 10
inc(a)
print(a)     # 10
```

```
def inc(f):
    for i in range(0,len(f)):
        f[i] = f[i] + 1
    print(f) # [3,4,6]

a = [2,3,5]
inc(a)
print(a) # [3,4,6]
```

restituzione di più valori

- la funzione restituisce una tupla

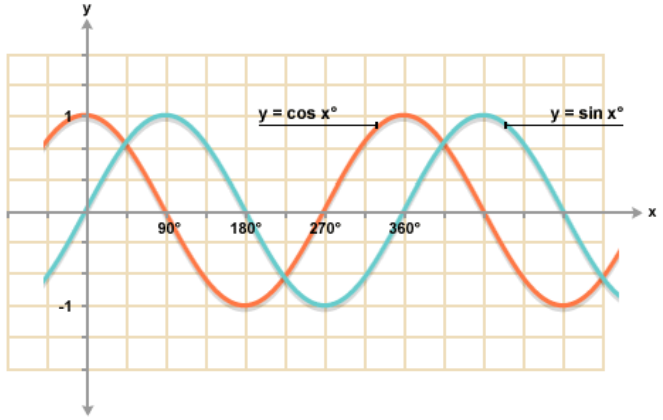
```
def min_max(f):
    '''
    restituisce valore minimo e massimo della lista f
    '''
    minimo = massimo = f[0]
    for i in range(1,len(f)):
        if f[i] < minimo:
            minimo = f[i]
        if f[i] > massimo:
            massimo = f[i]
    return minimo, massimo

a = [2,13,5,-3,8]
x , y = min_max(a)
print("minimo: ",x," massimo: ",y)
```

esercizi

sequenze

esercizi (1)



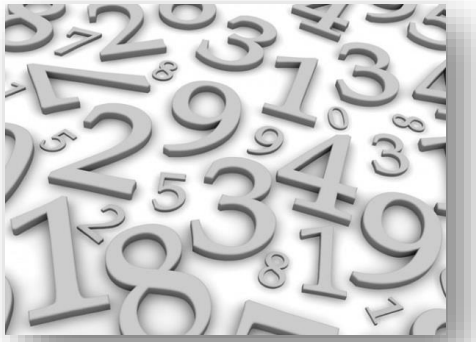
- valori precalcolati
  - riempire una lista con i valori di sin(x)
    - 360 elementi, indice x tra 0 e 359
  - poi, ciclicamente…
  - chiedere un angolo all'utente
  - visualizzare il corrispondente valore precalcolato del seno
  - *nota*
    - math.sin opera su radianti
    - calcolare math.sin(x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

esercizi (2)

- 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

esercizi (3)

- 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