```
import pickle # leggere/scrivere i file, non sempre necessario

import statistics as st # libreria per la statistica
    data = [] #lista di dati
    st.mean(data) # media
    st.stdev(data) # standard deviation
    st.median(data) # mediana
    st.variance(data) # varianza

import numpy as np # libreria tuttofare
    data = [] #lista di dati
    np.mean(data) # media
    np.std(data) # standard deviation
    np.median(data) #mediana

    np.linspace(*args) # return list of evenly spaced numbers, see doc
    np.polyfit(x,y,order) # generate the coefficients for a order-degree poly_fit of x,y
                                                    arrays
    np.polyval(coeff, x) # p(x) dove p polinomio con coeff.

    A = np.array([[], [], []]) # crea una matrice, tutte le funzioni non le modificano
    A.dot(B) # prodotto per una matrice B
    A.transpose() # matrice trasposta
    np.linalg.inv(A) # matrice inversa
    np.linalg.det(A) # determinante della matrice
    np.linalg.solve(A,b) # risolve il sistema lineare, con b matrice colonna b = np.array([[1
                                                    ],[2]])


import matplotlib.pyplot as plt # libreria per i grafici, lavora con COPPIE DI PUNTI
    # le liste "funzionano bene" per le operazioni puntuali
    x = [] # ascisse dei punti da stampare (lista)
    y = [] # ordinate dei punti da stampare (lista)

    plt.plot(x, y) # crea il plot, see doc
    plt.title('Plotting Differential Equation Solution')
    plt.xlabel('t')
    plt.ylabel('x(t)')
    plt.grid()
    plt.axis([0, 25, 0, 1])
    plt.show() # stampa il plot

import scipy # ottimizzazione, lavora con le FUNZIONI
    def f(x):
        return ## funzione
    scipy.optimize.fminbound(f, xmin, xmax) # trova il punto minimo della funzione
    scipy.integrate.odeint(f, valore_iniziale, vettore_tempi) # genera il vettore della
                                                    soluzione
    scipy.interpolate.interp1d(x,y) # input coppie di punti, output funzione
    scipy.integrate.quad(f,a,b) # integra la funzione f tra a e b
    scipy.optimize.curve_fit() #see doc

# Numeri Immaginari
(x,y) = (2,3)
z = x + yj = complex(x,y)
x == z.real
y == z.imag

import cmath
    cmath.polar(z) # converte in polari
    cmath.rect(r,rho) # converte in cartesiano
```