C++
#include <iostream> <vector> <string> <fstream> <map> <algorithm> <cctype> <list>
using namespace std;

PYTHON
// divisione intera, ** è ^, %n riduzione modulo n, <= e >= e != NO =< e => e =!

DOPO if E for METTI : E INDENTATURA!!!

print(" cose \t %s cose" % string) \t spazio grande, \n va a capo, %s aggiunge una stringa da string
s stringa, s[i] dà il carattere i-esimo ma la stringa è immutabile. len(s) dà lunghezza, s.find(g,i,j) cerca la stringa g dalla posizone i alla j. Se non c'è dà -1
s.endswith(s1,..., sn) dà vero se la stringa finisce con una stringa tra s1,..., sn. s.replace
break permette di uscire da un ciclo while, contiune fa ricominciare il ciclo

open('percorso files', "r") apre file solo lettura. con "w" scrittura, con "w+" lettura e scrittura "a" aggiunge cose in fondo al file
s=f.read, f.write(s)

LISTE !INDICIZZATE DA 0!
	l[3:] #tutti gli elementi dopo il terzo compreso l[:3] #tutti gli elementi fino al terzo compreso. Con numeri negativi conta dalla fine
	matrix=[[1,2,3],[4,5,6],[7,8,9]]
	l[n]=valore #modifica la lista inserendo valore al posto n
	del(l[posizione o più posizioni indicate con :]) #cancella il/i termine/i dalla lista
	dir([]) #dà i metodi associati a list help([].metodo)
	l.sort() #ordina il vettore
	l.reverse() #ordina il vettore al contrario
	l.append(elemento) #aggiunge un elemento
	l.extend(lista di elementi) OPPURE l + lista di elementi #aggiunge lista di elementi
	l.insert(posizione, elemento) #aggiunge l'elemento nella posizione data

DIZIONARI
	D={} D={chiave1 : oggetto1 , ..., chiaven : oggetton}
	D.get(oggetto)
	len(D) #lunghezza dizionario
	D.keys() # dà UNA STRINGA con le chavi del dizionario
	D.values() # dà UNA STRINGA con i valori del dizionario
	D.items() # dà UNA STRINGA con coppie chaivi-valore del dizionario
	D[chiave] #dà il valore. E' modificabile con =valore e se la chiave non è presente, aggiunde chaive e valore al dizionario

SET
	S={chiave1, ..., chiaven} S=set() #crea set vuoto. {} crea dizionario
	S.add("oggetto") S.update([lista]) # aggiunge oggetto e lista
	S.discard(oggetto) S.remove(oggetto) #tolgono un oggetto. remove dà errore se l'oggetto non c'è
	A.union(B) A.intersection(B) A.difference(B)

TUPLE: set ordinato e immutabile. supporta somma e prodotto

import <libreria> as <sorannome libreria>
from <libreria> import <funzione>

numpy (np)
	np.array([[riga1],...,[rigan]]) #moltiplicazione matriciale A.dot(B), A+B, A.transpose()
	np.roots([a_n, ..., a_0]) #Trova radici del polinomio. a_n coefficiente di grado max
	np.set_printoptions(precision=16, suppress=True) #mostra il massimo numero di cifre decimali, suppress=True mostra il risultato non in notazione scientifica
	np.abs #valore assoluto
	np.exp(x)
	np.mean(dat) np.median(dat) np.std(dat) #media, mediana e deviazione std
	np.arange (tstart ,tstop, increment) #crea una lista da tstart a tstop-1 con passo increment. np.zeros(n) #crea vettore con n zeri
	np.interp(x_0 , x , y) #dà y_0 interpolando linearmente i dati (crea una spezzata)
	np.linspace(xmin, xmax, numero punti)
	np.trapz(funz , x , dx) con dx = (b - a)/N e x = np.linspace (a , b , N+1)
	np.diff(x) #calcola le differenze tra x_{i+1}-x_{i}. Differenze divise: np.diff(y)/np.diff(x)
	np.polyfit(x , y , grado) #trova il polinomio di approssimazione dei dati di grado n
	np.polyval(polinomio, punti) #valuta il polinomio nei punti

numpy.linalg (la)
	la.det(A) #determinante
	la.inv(A)
	la.solve(A, b) # risolve il sistema con coefficinti A e termine noto b

cmath
	z=complex(x,y) z.real z.imag z.conjugate
	cmath.polar(z) cmath.rect(r, theta) #converte z in forma polare e viceversa

scipy (sp)
	sp.optimize.fminbound (y , xmin , xmax) #dà il punto di minimo. Se richiesto fare f(xmin) per trovare ymin
	sp.fsolve(f, x0) #calcola lo zero x di una funzione f a partire da un'approssimazione iniziale x0;
	# Il calcolo degli zeri di un polinomio è in generale un problema mal condizionato in presenza di zeri multipli.
	sp.optimize.curve_fit(model(x, a_n,...,a_0), x, y) #dà i coefficienti a_i e la matrice di covarainza dei coefficienti. model è una funzione, di solito a_nx**n+...+a_0.
	sp.integrate.odeint(funzione, x_0, t) #t vettore dei tempi, si crea con np.arange. x_0=x(t_0)
	sp.interpolate.interp1d(x, y, kind='linear') #dà una funzione. kind può essere linear, quadratic, cubic)
	sp.integrate.quad(funz , a , b) #integrale numerico di funz da a a b. Il secondo valore è l'errore

matplotlib.pyplot (plt)
	PRIMA DI PLOTTARE DEFINISCI f(x) con
	def f(x):
		return funzione
	plt.rcParams["figure.figsize"] = (14,8)

	x=np.linspace(xmin, xmax, numero punti) #numero punti di solito è 100
	plt.plot(x, f(x), "r", label = r'parole label $codice LaTeX$', linewidth=2, punto a caso, f(punto a caso), 'or') #la lettera dopo f(x) dà il colore, r- è rosso tratteggiato, -. tratto e punto

	plt.plot (x , y , 'o' , xnew , f (xnew) , '-' , xnew , f2 (xnew) , '--') #esempio con tre plot e legenda
	plt.legend (['data' , 'linear' , 'cubic'] , loc='best')

	plt.grid() #crea una griglia nello sfondo
	plt.legend() #legenda
	plt.show() #mostra il plot
	plt.axis ([xmin , xmax , ymin , ymax])
	plt.xlabel ('x')
time
	time.sleep(n) #pausa n secondi

IPython.display.clear_output() #pulisce l'output

sympy (sym)
	sym.Rational Real Integer
	sym.pi #aggiungi .evalf(n) per un approssimazione di pi con n cifre sym.oo #infinito
	sym.Symbol('x')
	sym.expand((x+y)**2) complex=True
	sym.limit(funz, variabile, limite) # per limite a inftinito usare sym.oo
	sym.diff(funz, variabile, n) #n - derivata n-esima
	sym.cos , sin , tan
	sym.series(funz, variabile) #Serie Taylor
	sym.integrate(funz, var) #integrale indefinito
	sym.integrate(funz, (x,lower bound, upper bound))
	sym.solveset(funz, (variabile1,...,variabilen)) #dà l'insieme delle radici in var1,..,varn come t-upla
	sym.factor(polinomio) #fattorizza ma solo con coefiicienti interi
	sym.Matrix([col 1],...,[col n])
	f=sym.symbols('f', cls=sym.Function) #definisce f(x)
	f(x).diff(x,...,x) # derivata n-esima di f(x)
	sym.dsolve(espressione in f(x) e sue derivate, f(x)) #Risolve l'eq differenziale