Università degli Studi di Cagliari
Corsi di Laurea in Ingegneria Chimica e Ingegneria Meccanica

FONDAMENTI DI INFORMATICA

http://people.unica.it/gianlucamarcialis

A.A. 2018/2019

Docente: **Gian Luca Marcialis**

PYTHON: ESERCIZI DI PROGRAMMAZIONE PARTE 4

Esercizio sui dizionari

- Siano dati due dizionari che rappresentano altrettante tabelle di una base di dati relazionale
- Ogni attributo della tabella è rappresentato come chiave del dizionario, alla quale è associata una lista di valori, ciascuno associato ad una delle *tuple* presenti
- Esempio:

```
Aule={'Nome': ['B1', 'Z', 'V'],
      'Edificio': ['DIEE-B', 'Parcheggi', 'Parcheggi'],
      'Piano': [1, 0, 0]}
```

- Si scriva una funzione `prodotto_cartesiano` che a partire da due dizionari `d1` e `d2` generi un dizionario d che sia il prodotto cartesiano fra le tabelle rappresentate dai due dizionari.

Aule

Nome	Edificio	Piano
B1	DIEE-B	1
Z	Parcheggi	0
V	Parcheggi	0

Esempio di prodotto cartesiano

A1	A2	A3
a11	a21	a31
a12	a22	a32

A={A1: [a11, a12], A2: [a21, a22], A3: [a31, a32]}

B1	B2
b11	b21
b12	b22
b13	b23

B={B1: [b11, b12, b13], B2: [b21, b22, b23]}

Tabella prodotto cartesiano:

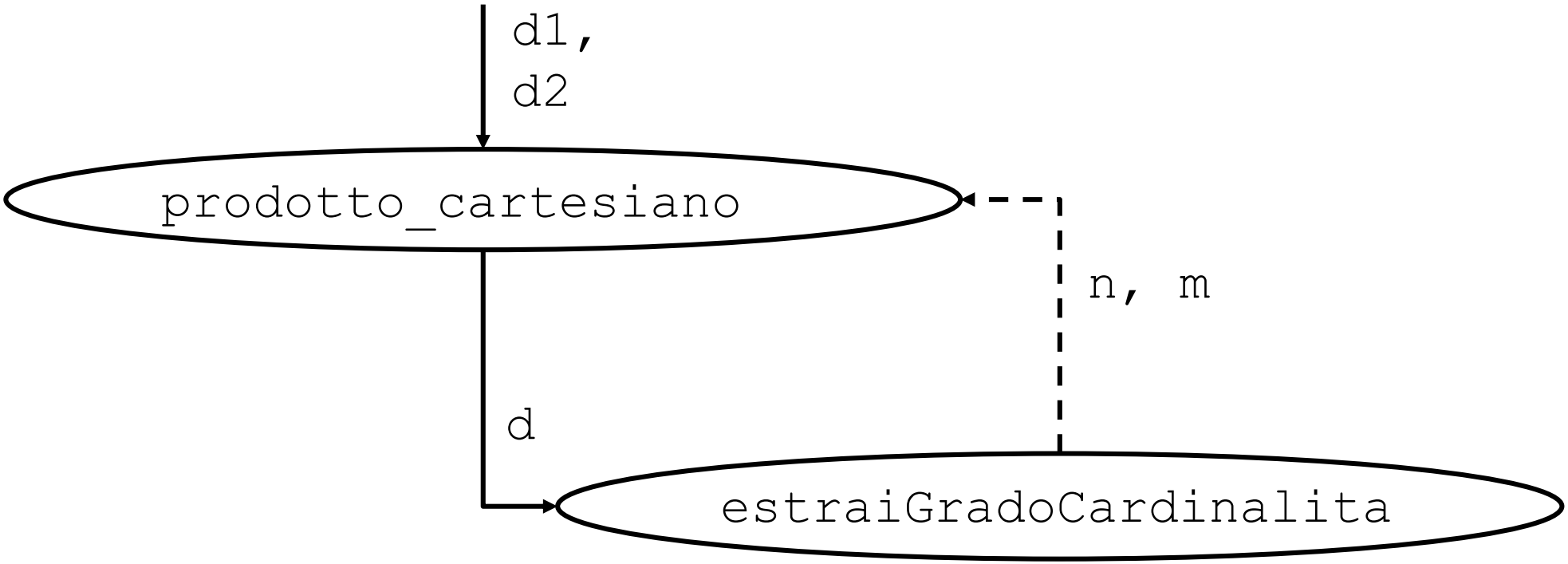
A1	A2	A3	B1	B2
a11	a21	a31	b11	b21
a11	a21	a31	b12	b22
a11	a21	a31	b13	b23
a12	a22	a32	b11	b21
a12	a22	a32	b12	b22
a12	a22	a32	b13	b23

Struttura dell'algoritmo

- Dati due dizionari, d1 e d2, con grado e cardinalita n1, m1 e n2 e m2, rispettivamente:
  - Si può considerare d2 come set di tuple «meno significative» (come nelle tabelle di verità si ordina dai bit meno significativi ai più significativi) ➔ moltipicheremo i contenuti di d2 tante volte quante sono le tuple di d1 (m1).
  - Dopo di che aggiungeremo le tuple di d1 prendendole *singolarmente* e replicandole tante volte quante sono le tuple di d2 (m2).

	A1	A2	A3	B1	B2	
x m2	a11	a21	a31	b11	b21	x m1
	a11	a21	a31	b12	b22	
	a11	a21	a31	b13	b23	
x m2	a12	a22	a32	b11	b21	
	a12	a22	a32	b12	b22	
	a12	a22	a32	b13	b23	

Struttura della funzione



Funzione `estraiGradoCardinalita`

```
def estraiGradoCardinalita(d):

    k=d.keys()
    grado=len(k)
    cardinalita=len(d[k[0]])

    return grado, cardinalita
```

Funzione prodotto_cartesiano

```
def prodotto_cartesiano(d1,d2):
  d={}

  n1,m1=estraiGradoCardinalita(d1)
  n2,m2=estraiGradoCardinalita(d2)

  for k in d2:      #tuple meno "significative"
    l=d2[k] * m1
    d[k]=l

  for k in d1:     #tuple più "significative"
    l=[]
    for el in d1[k]:
      l=l+[el] * m2
    d[k]=l

  return d
```

Esercizio sulle matrici (cronometrarsi)

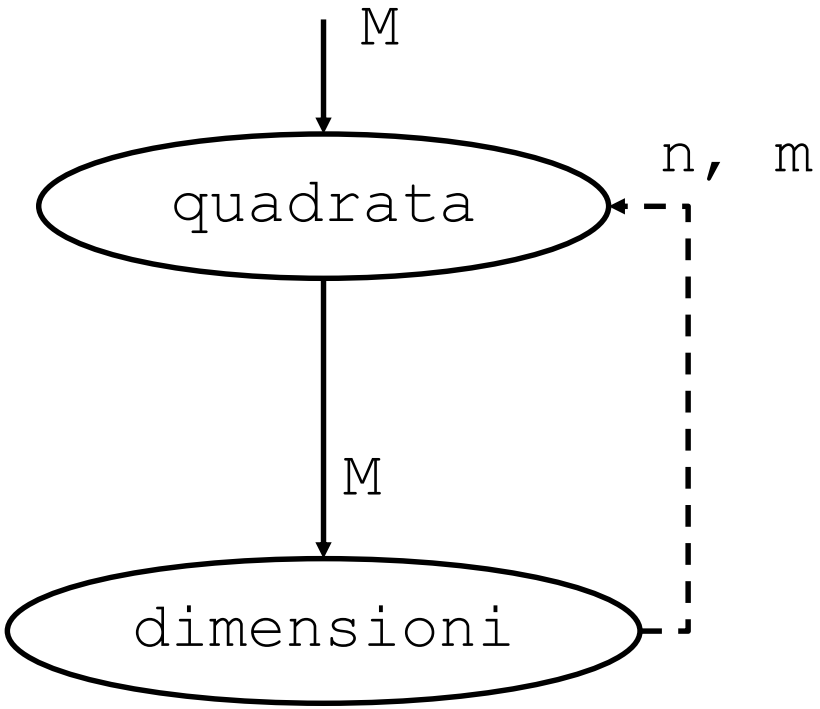
- E' possibile rappresentare una matrice in Python attraverso una lista di liste, ciascuna sottolista rappresenta una riga della matrice data
- Scrivere le seguenti funzioni:
  - `trasposta(M)`: genera la matrice `T`, trasposta di `M`, ricevuta in ingresso
  - `quadrata(M)`: *`True`* se `M` è quadrata, *`False`* altrimenti
  - `dimensioni(M)`: restituisce il numero di righe e il numero di colonne di `M`
  - `somma(M1,M2)`: somma due matrici
  - `prodotto(M1,M2)`: calcola il prodotto di due matrici
  - `leggi(nomefile)`: legge da file una matrice memorizzata per riga da file di nome `nomefile` e la restituisce
  - `scrivi(nomefile,M)`: scrive su file indicato la matrice M

Cosa fare di fronte ad un compito del genere

- Si cerca di mettere gli esercizi in ordine di difficoltà progressivo. Per esempio, usando il rosso per difficoltà alte, giallo per medie e verde per basse:
  - `trasposta(M)`: genera la matrice `T`, trasposta di `M`, ricevuta in ingresso
  - `quadrata(M)`: *`True`* se `M` è quadrata, *`False`* altrimenti
  - `dimensioni(M)`: restituisce il numero di righe e il numero di colonne di `M`
  - `somma(M1,M2)`: somma due matrici
  - `prodotto(M1,M2)`: calcola il prodotto di due matrici
  - `leggi(nomefile)`: legge da file una matrice memorizzata per riga da file di nome `nomefile` e la restituisce
  - `scrivi(nomefile,M)`: scrive su file indicato la matrice M

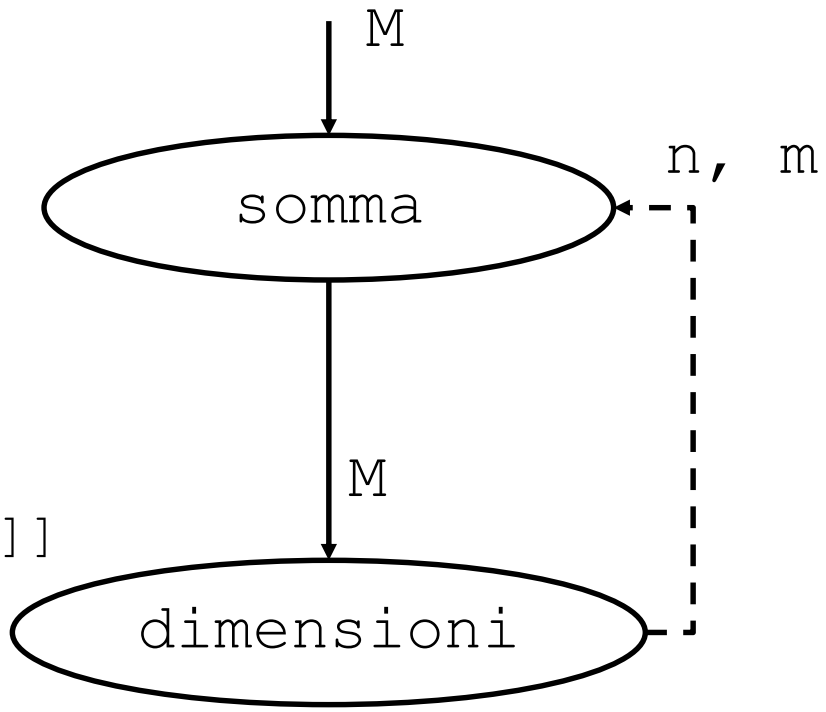
Le funzioni «verdi»

```
def dimensioni(M):
    nr=len(M)
    nc=len(M[0])
    return nr, nc

def quadrata(M):
    n,m=dimensioni(M)
    return n==m
```



Le funzioni «gialle»

```
def somma(M1, M2):
    n, m=dimensioni(M1)

    M=[]
    i=0
    while i<n:
        rigaM1=M1[i]
        rigaM2=M2[i]
        j=0
        r=[]
        while j<m:
            r = r + [rigaM1[j]+rigaM2[j]]
            j=j+1
        M=M+[r]
        i=i+1

    return M
```



Le funzioni «gialle»

```
def leggi(nomefile):
  f=open(nomefile,"r")

  M=[]
  riga=f.readline()
  while riga!="":
    riga=riga.split()
    l=[]
    for el in riga:
      l=l+[int(el)]
    M=M+[l]
    riga=f.readline()
  f.close()
  return M
```

```
def scrivi(nomefile,M):
  f=open(nomefile,"w")

  for riga in M:
    linea=""
    for el in riga:
      linea=linea+str(el)+" "
    f.write(linea)

f.close()
```

Le funzioni «rosse»

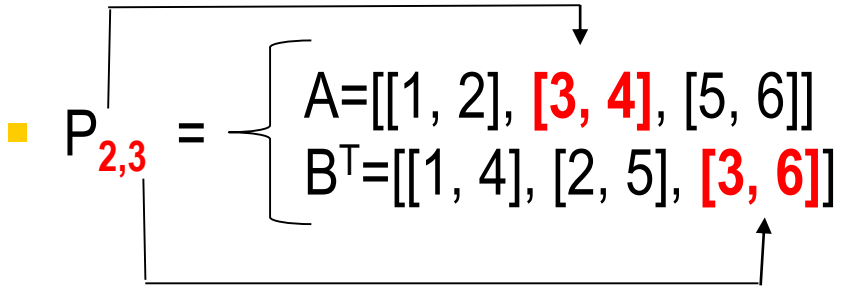
```
def trasposta(M):
    n,m=dimensioni(M)

    T=[]        #inizializzazione di T
    i=0
    while i<m:
        T=T+[[0]*n]
        i=i+1

    i=0         #assegnazione dei valori di T
    while i<n:
        j=0
        while j<m:
            T[j][i]=M[i][j]
            j=j+1
        i=i+1
    return T
```

La funzione prodotto: idea

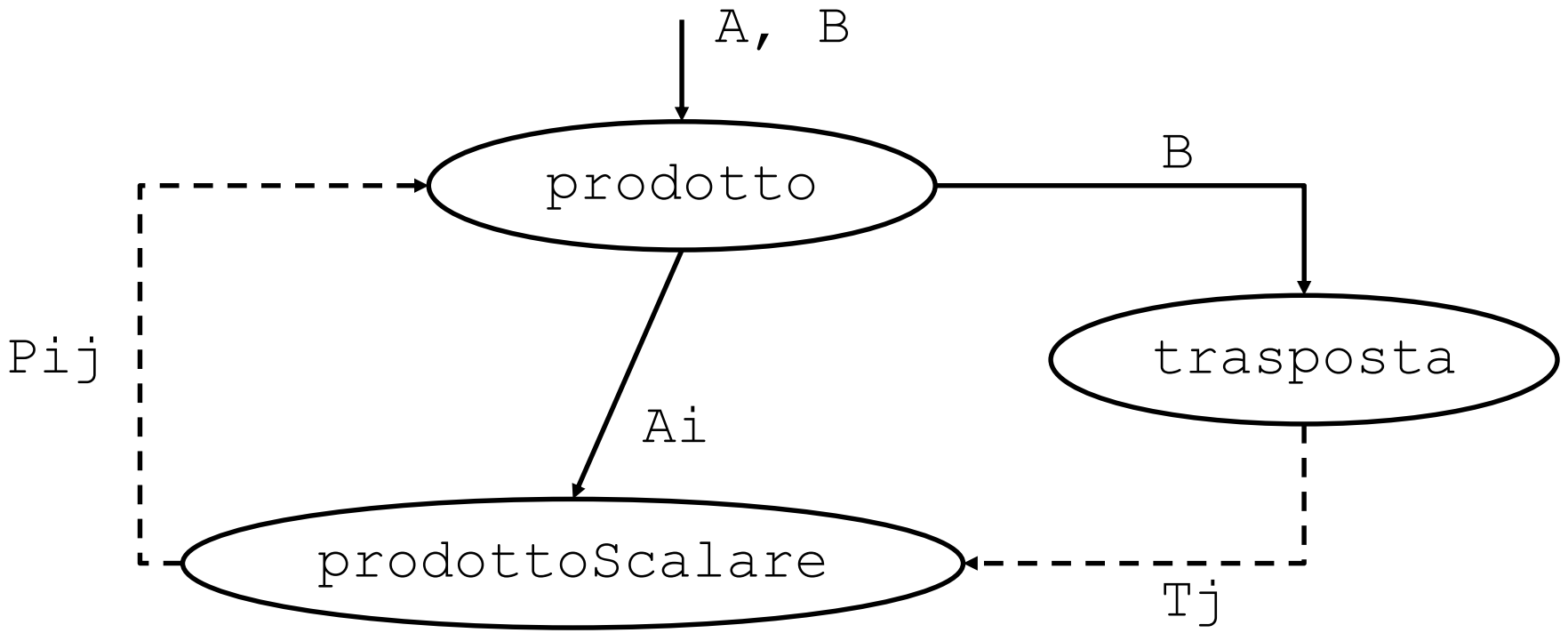
- A=[[1, 2], [3, 4], [5, 6]] ➔ 3x2
- B=[[1,2,3],[4,5,6]] ➔ 2x3

- P=AxB ➔ 3x3

- Se traspongo B, posso fare il prodotto incrociato delle righe in modo separato. Per esempio:



Struttura



La funzione `prodottoScalare`

```
def prodottoScalare(u,v):
    n=len(u)

    p=0
    i=0
    while i<n:
        p=p+u[i]*v[i]
        i=i+1

    return p
```

La funzione prodotto

```
def prodotto(A,B):
  n,m=dimensioni(A)
  c,p=dimensioni(B)

  if m!=c:
    return False

  T=trasposta(B)

  P=[]
  while i<n:
    riga1=A[i]
    rigap=[]
    j=0
    while j<p:
      riga2=T[j]
      pr=prodottoScalare(riga1,riga2)
      rigap=rigap+[pr]
      j=j+1
    P=P+[rigap]
    i=i+1

  return P
```

Esercizio sulle funzioni (da facebook)

- Scrivere un programma Python che legge da file "funzione.txt", una sequenza di coppie t, f(t), dove t rappresenta l'istante di tempo nel quale è stato campionata una certa misura f(t) (si pensi per esempio al segnale vocale visto a lezione, un segnale di tensione o di corrente, di potenza dissipata e così via). Le coppie lette, che nel file sono separate dal carattere "a capo" (quindi due valori per riga), devono essere memorizzate in una lista di liste.

Esercizio sulle funzioni

- Il programma deve poi salvare in un file "elaborazione.txt" la tripla di valori rappresentanti la derivata della funzione che all'istante `t[i]` è calcolata come:

$$derivata[i] = \frac{f[i] - f[i-1]}{t[i] - t[i-1]}$$

- …e l'integrale della funzione che all'istante `t[i]` è calcolato come:

$$integrale[i] = \sum_{k=1}^{i} f[k] \cdot (t[k] - t[k-1])$$

- In entrambi i casi `i` è compreso tra 1 e il numero complessivo di campioni di `f` memorizzati.

Esercizio sulle funzioni

- Nello sviluppare questo codice, si implementino le seguenti funzioni:
  - [4 punti] `leggiDati`, che, ricevendo in ingresso il nome di un file, restituisce una lista di di coppie di valori [t, f(t)].
  - [6 punti] `calcolaDerivata`, che, ricevendo in ingresso una lista formata dalle coppie [t, f(t)], restituisce una lista contenente le relative derivate, per ogni t, secondo la definizione data sopra.
  - [6 punti] `calcolaIntegrale`, che, ricevendo in ingresso una lista formata dalle coppie [t, f(t)], restituisce una lista contenente i relativi integrali, per ogni t, secondo la definizione data sopra.
  - [4 punti] `salvaDati` che, ricevendo in ingresso una lista con la tripla formata dai valori [t, d(f(t)), i(f(t))] (t, derivata di f(t), integrale di f(t)], li salva, riga per riga, nel file "elaborazione.txt".

Cominciamo dall'input/output

```
def leggiDati(nomeFile):
    fp=open(nomeFile,"r")

    dati=[]
    dato=fp.readline()
    while(dato!=""):
        valori=dato.split()
        valori[0]=float(valori[0])
        valori[1]=float(valori[1])
        dati=dati+[valori]
        dato=fp.readline()
    fp.close()

    return dati
```

```
def salvaDati(dati):
    fp=open("elaborazione.txt","w")

    for elemento in dati:
        stringa=""
        for dato in elemento:
            stringa=stringa+str(dato)+" "
        stringa=stringa+"\n"
        fp.write(stringa)

    fp.close()
```

La funzione derivata

```
def calcolaDerivata(dati):

    derivate=[]
    i=1
    while (i<len(dati)):
        d=(dati[i][1]-dati[i-1][1])
            /(dati[i][0]-dati[i-1][0])
        derivate=derivate+[d]
        i=i+1
    return derivate
```

La funzione integrale (versione efficiente)

```
def calcolaIntegrale(dati):
    integrali=[]
    i=1

    integrale=0.0
    while (i<len(dati)):
        vi=dati[i][1]*(dati[i][0]-dati[i-1][0])
        integrale=integrale+vi
        integrali=integrali+[integrale]
        i=i+1

    return integrali
```

Script principale

```
dati=leggiDati("funzione.txt")
derivate=calcolaDerivata(dati)
integrali=calcolaIntegrale(dati)
triple=[]
n=len(dati)
i=1
while i<n:
  triple=triple+
     [[dati[i][0],derivate[i-1],integrali[i-1]]]
  i=i+1
salvaDati(triple)
```

Generazione funzioni (fate voi)

- Scrivere altrettante funzioni che generino N campioni delle seguenti curve nell'intervallo desiderato [xmin, xmax] fornito in ingresso, assieme ai relativi parametri delle curve:
  - `generaRetta(a,b,xmin,xmax)`: $y = ax + b$
  - `generaParabola(a,b,c,xmin,xmax)`: $y = ax^2 + bx + c$
  - `salvaFunzione(x,y,nomefile)`: salva la curva stampando su file nomefile ogni valore di `x[i]` e relativo campione `y[i]`.

- Utilizzare le funzioni precedenti per calcolarne derivata e integrale, inserendo i parametri della curva desiderata da tastiera.

Per saperne di più...

- K.A. Lambert, *Programmazione in Python*, Apogeo (Maggioli), 2012.
- C. Horstmann, R.D. Necaise, *Concetti di informatica e fondamenti di Python*, Apogeo (Maggioli), 2014.
- F. Aiolli, Appunti di programmazione (scientifica) in Python, Esculapio, 2014.