Esercizi Python

Corso di Bioinformatica 2012

Esercizio 1

Tipi contenitore

- Creare due tuple che rappresentino i due elenchi di nomi e cognomi descritti sotto:

  nomi: Numa, Tullo, Anco
  cognomi: Pompilio, Ostilio, Marzio

- Ottenere una lista in cui ogni elemento è un dizionario {'nome': nome, 'cognome': cognome}, che accoppia nomi e cognomi in base all'ordine.

Soluzione 1

Tipi contenitore

```
>>> nomi = ('Numa','Tullo','Anco')
>>> cognomi = ('Pompilio','Ostilio','Marzio')
>>> l = []
>>> for nome, cognome in zip(nomi,cognomi):
...     l.append({'nome': nome, 'cognome': cognome})
...
>>> l
[{'cognome': 'Pompilio', 'nome': 'Numa'},
 {'cognome': 'Ostilio', 'nome': 'Tullo'},
 {'cognome': 'Marzio', 'nome': 'Anco'}]
```

Una possibile alternativa:

```
>>> l = [{'nome': nome, 'cognome': cognome} for nome,
cognome in zip(nomi, cognomi)]
```

Esercizio 2

Tipi contenitore

- Creare un dizionario che contenga come chiavi 'nome' e 'cognome', inserendo i propri dati come valori
- Aggiungere 'matricola'
- Aggiungere 'esami', provando ad immaginare che tipi di dato usare per rappresentare sia nome che voto degli esami

Soluzione 2

Tipi contenitore

```
>>> d = {'nome':'Pinco','cognome':'Pallino'}
>>> d['matricola'] = 258115
>>> d['esami'] = [{'nome':'Bioinformatica','voto': 30},
{'nome':'Analisi','voto': 18}]
>>> d
{'nome':'Pinco','cognome':'Pallino','matricola':
258115,'esami':[{'nome':'Bioinformatica','voto': 30},
{'nome':'Analisi','voto': 18}]}
```

Una possibile alternativa:

```
>>> d['esami'] = {'Bioinformatica': 30, 'Analisi': 18}
>>> d
{'nome':'Pinco','cognome':'Pallino','matricola':
258115,'esami':{'Bioinformatica': 30, 'Analisi': 18}}
```

Esercizio 3

Istruzione if

Scrivere un programma che:

- prenda una stringa in input da tastiera, rappresentante un nucleotide (A,C,G,T)
- stampi a video il nucleotide complementare

Assicurarsi che il programma funzioni correttamente sia con input maiuscolo che minuscolo.

Soluzione 3

Istruzione if

```
nucleotide = raw_input('Inserisci un nucleotide (A,C,G,T):
')

if nucleotide == 'A' or nucleotide == 'a':
    print 'T'
elif nucleotide == 'C' or nucleotide == 'c':
   print 'G'
elif nucleotide == 'G' or nucleotide == 'g':
   print 'C'
elif nucleotide == 'T' or nucleotide == 't':
   print 'A'
```

Una possibile alternativa:

```
nucleotide = nucleotide.capitalize()
```

Esercizio 4

Iterazione

- Calcolare la somma dei primi 500 numeri naturali (da 0 a 500 escluso)

Soluzione 4

Iterazione

```
>>> n = 0
>>> for i in range(0,500):
...     n += i
...
>>> n
124750
```

Possibili alternative:

```
>>> sum(range(0,500))
124750
>>> (499*500)/2 # Gauss!
124750
```

Esercizio 5

Iterazione

- Data la stringa `'abcdefghi'`, scrivere un programma che analizzi la stringa e stampi a video:

```
Lettera 1: a
Lettera 2: b
...
```

  E così via.

- Modificare poi il programma in modo da leggere la stringa da tastiera.

Soluzione 5

Iterazione

```
for i, letter in enumerate('abcdefghi'):
    print 'Lettera %d: %s' % (i+1, letter)
```

```
s = raw_input('Inserisci una stringa: ')
for i, letter in enumerate(s):
    print 'Lettera %d: %s' % (i+1, letter)
```

Esercizio 6

Iterazione

- Scrivere un programma che stampi la lunghezza delle stringhe fornite dall'utente, finchè l'utente non inserisce la stringa `'exit'`

Soluzione 6

Iterazione

```
while True:
    line = raw_input('Inserisci una stringa: ')
    if line == 'exit':
        break
    print len(line)
```

Esercizio 7

Funzioni

- Riprendere l'esercizio 3, e risolverlo definendo una funzione complementare (...), che ritorni il nucleotide complementare a quello passato come argomento
- Provare a invocare la funzione così definita dalla console interattiva

Soluzione 7

Funzioni

```
def complementare(nucleotide):
    nucleotide = nucleotide.capitalize()
    if nucleotide == 'A':
        return 'T'
    elif nucleotide == 'C':
        return 'G'
    elif nucleotide == 'G':
        return 'C'
    elif nucleotide == 'T':
        return 'A'

nucleotide = raw_input('Inserisci un nucleotide (A,C,G,T): ')
print complementare(nucleotide)
```

```
>>> complementare('A')
'T'
```

Esercizio 8

Funzioni

- Aggiungere al programma precedente una funzione filamento_opposto(...) che utilizzi la funzione complementare(...) per ritornare il filamento opposto a quello passato come argomento

```
>>> filamento_opposto('CTAATGT')
'GATTACA'
```

Soluzione 8

Funzioni

```
def filamento_opposto(filamento):
    opposto = ''

    for nucleotide in filamento:
        opposto += complementare(nucleotide)

    return opposto
```

Esercizio 9

Iterazione

- Scrivere un programma che legga un intero `n` da tastiera e stampi a schermo:

```
1
1 2
1 2 3
1 2 3 4
...
```

  E così via fino a `n`.
- Per stampare senza andare a capo, aggiungere una virgola in fondo alla riga con l'istruzione `print`:
  ```
  print 'foo',
  ```
- Per andare a capo, dare l'istruzione `print` senza argomenti:
  ```
  print
  ```

Soluzione 9

Iterazione

```
n = int(raw_input('Inserisci un numero n: '))

for i in range(1,n+1): # l'ultimo indice e' escluso!
    for j in range(1,i+1): # l'ultimo indice e' escluso!
        print j,
    print
```

Esercizio 10

Iterazione

- Scrivere un programma che, dati i due elenchi di numeri sottostanti, crei la matrice dei loro prodotti:

  v1: 1,2,3,4,5
  v2: 6,7,8,9,10
  mat:
  1*6 1*7 1*8 ...
  2*6 2*7 2*8 ...
  ...

- Completare il programma con una stampa della matrice riga per riga:

  [6, 7, 8 ...]
  [12, 14, 16 ...]
  ...

Soluzione 10

Iterazione

```
v1 = (1,2,3,4,5)
v2 = (6,7,8,9,10)
mat = []
for i, x1 in enumerate(v1):
    mat.append([])
    for x2 in v2:
        mat[i].append(x1*x2)

for row in mat:
    print row
```

Una possibile alternativa:

```
mat = [[x1*x2 for x2 in v2] for x1 in v1]
```

Esercizio 11

Funzioni

- Scrivere un programma che definisca una funzione stampa_matrice(mat), che migliori la stampa dell'esercizio precedente:

```
>>> stampa_matrice(mat)
  6    7    8    9   10
 12   14   16   18   20
 18   21   24   27   30
 ...
```

- Per fare in modo che i numeri siano stampati allineati, usare per ogni numero:

```
print '%3i' % num
```

Soluzione 11

Funzioni

```
def stampa_matrice(mat):
    for row in mat:
        for num in row:
            print '%3i' % num,
        print
```

Esercizio 12

Funzioni

- Aggiungere al programma precedente una funzione square(val,n), che ritorni una matrice quadrata di dimensione n con il valore val in ogni cella

- Utilizzare la funzione stampa_matrice(mat) per stampare nella console interattiva una matrice 6x6 con il valore 0 in ogni cella

Soluzione 12

Funzioni

```
>>> stampa_matrice(square(0,6))
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
  0   0   0   0   0   0
```

Soluzione 12

Funzioni

```
def square(val, n):
    mat = []
    for i in range(0,n):
        row = []
        for j in range(0,n):
            row.append(val)

        mat.append(row)

    return mat
```

Possibili alternative:

```
def square(val, n):
    return [[val for j in range(0,n)] for i in range(0,n)]
```

```
def square(val, n):
    return [[val] * n] * n
```

Esercizio 13

- Utilizzare la list comprehension per creare la lista dei primi dieci cubi:
  `[0,1,8,27...]`
- Applicare filter() alla lista per ottenere una nuova lista con solo numeri pari.
- Applicare map() sulla lista per ottenere una nuova lista che contenga gli elementi della precedente moltiplicati per 3.

Soluzione 13

```
>>> l1 = [x**3 for x in xrange(10)]
>>> l1
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]

>>> l2 = filter(lambda x: not x%2, l1)
>>> l2
[0, 8, 64, 216, 512]

>>> map(lambda x: x*3, l2)
[0, 24, 192, 648, 1536]
```

Esercizio 14

- Scrivere una funzione che, ricevendo in ingresso il nome di un file fasta, restituisca una stringa con la sequenza in esso contenuta.

```
>gi|251831106|ref|NC_012920.1| Homo sapiens mitochondrion, complete
genome
GATCACAGGTCTATCACCCTATTAACCACTCACGGGAGCTCTCCATGCATTTGGTATTTTCGTCTGGGGG
GTATGCACGCGATAGCATTGCGAGACGCTGGAGCCGGAGCACCCTATGTCGCAGTATCTGTCTTTGATTC
CTGCCTCATCCTATTATTTATCGCACCTACGTTCAATATTACAGGCGAACATACTTACTAAAGTGTGTTA
...
```

Soluzione 14

```
def get_seq(fastafile):
    f = open(fastafile)
    lines = f.readlines()
    f.close()
    lines = [row.strip()  for row in lines]
    lines = [row.replace('\n','') for row in lines if not row.startswith('>')]
    return ''.join(lines)

print get_seq('mitochondrion.fasta')
```

Single-line solution!

```
def get_seq(fastafile):
    return ''.join(row.strip().replace('\n','') for row in
open(fastafile)  if not row.startswith('>'))
```

Una soluzione così compatta non è necessariamente migliore!
Potrebbe anzi risultare meno leggibile.

Esercizio 15

- Definire una funzione eratostene(n) che ritorni tutti i numeri primi da 2 a n inclusi, utilizzando il procedimento del Crivello di Eratostene:
  - Si crea un elenco di tutti i numeri naturali da 2 a n, detto setaccio.
  - Si aggiunge il primo numero del setaccio all'elenco dei numeri primi trovati, e si eliminano dal setaccio tutti i suoi multipli. Si prosegue in questo modo fino ad esaurire i numeri nel setaccio.

Soluzione 15

```
def eratostene(n):
    setaccio = range(2, n+1)
    primi = []
    while setaccio: # finche' ci sono numeri nel setaccio
        primi.append(setaccio[0])
        setaccio = filter(lambda x: x%primi[-1], setaccio)
    return primi
```

Esercizio 16

- Creare una stringa ripetendo 10,000 volte la stringa 'CGAT'.
- Creare una stringa di 40,000 caratteri scelti a caso dall'alfabeto 'CGAT'.
- Utilizzare la funzione compress() del modulo zlib per comprimere le due stringhe.
- Calcolare le lunghezze delle nuove stringhe così ottenute ed i relativi rapporti di compressione.

Soluzione 16

```
>>> s1 = 'CGAT'*10000
>>> import random
>>> s2 = ''.join([random.choice('CGAT') for i in xrange
(10000)])
>>> import zlib
>>> zs1 = zlib.compress(s1)
>>> zs2 = zlib.compress(s2)
>>> len(zs1)
68
>>> len(zs2)
12055
>>> len(zs1)/float(len(s1))
0.0017
>>> len(zs2)/float(len(s2))
0.301375
```

Esercizio 17

- Caricare in una stringa una sequenza contenuta in un file fasta, come fatto nell'esercizio 14.
- Calcolare il rapporto di compressione della sequenza.

Soluzione 17

```
>>> seq = get_seq('mitochondrion.fasta')
>>> import zlib
>>> zseq = zlib.compress(seq)
>>> len(seq)
16569
>>> len(zseq)
4992
>>> len(zseq)/float(len(seq))
0.3012855332246967
```

Esercizio 18

- Scrivere un programma che definisca la funzione dh(s, t), che implementi il calcolo della distanza di Hamming tra due stringhe s e t.

- Aggiungere la funzione dhplus(s,t), che generalizzi dh (s,t) al caso di stringhe di diversa lunghezza.

Soluzione 18

```
def dh(s, t):
    if len(s) != len(t):
        return -1
    return sum(es != et for es, et in zip(s, t))

def dhplus(s, t):
    sprime = s+'.'*(len(t)-1)
    tprime = '.'*(len(s)-1)+t
    dist = max(len(s), len(t))
    for x in xrange(len(s)+len(t)-1):
        sprime = sprime[-1]+sprime[:-1]
        pardist = dh(sprime, tprime)
        dist = min(dist, pardist)
    return dist
```

Esercizio 19

Funzioni

- Aggiungere al programma dell'esercizio 12 una funzione spirale(n), che ritorni una matrice a spirale oraria:

```
>>> stampa_matrice(spirale(4))
  0   1   2   3
 11  12  13   4
 10  15  14   5
  9   8   7   6
```

- La matrice riporta i numeri da 0 a n**2 escluso seguendo un andamento a spirale in senso orario.
- Si consiglia di fare uso della funzione square(val,n) appena definita per preparare una matrice a valori non validi (ad esempio -1), nella quale inserire i numeri.

Soluzione 19

Funzioni

```
def spirale(n):
    mat = square(-1, n)

    x,y = 0,0 # cella iniziale
    dx,dy = 1,0 # incrementi iniziali

    for num in range(0,n**2):
        mat[y][x] = num  # scrive il numero nella cella

        # evita di uscire dalla matrice o di scrivere su una cella piena
        if not(0 <= x+dx < n)  or not(0 <= y+dy < n)  or mat[y+dy][x+dx]  is not -1:
            dx,dy = -dy,dx  # ruota il vettore incremento

        # cambia cella
        x += dx
        y += dy

    return mat
```

Esercizio

Classi e oggetti

- Scrivere un programma che definisca una classe Punto2D, che rappresenti un punto del piano.
- Definire un metodo distanza_origine(), che ritorni la distanza del punto dall'origine.

Soluzione

Classi e oggetti

```
import math

class Punto2D:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def distanza_origine(self):
        return math.sqrt(self.x**2+self.y**2)
```

Per testare la classe:

```
>>> p = Punto2D(1,1)
>>> p.distanza_origine()
1.4142135623730951
```

Esercizio

- Scrivere un programma che definisca una classe Matrice, che rappresenti una matrice come visto negli esercizi precedenti.

- Definire un metodo stampa(), che stampi la matrice come nell'esercizio 11.

Soluzione

Classi e oggetti

```
class Matrice:
    def __init__(self, mat):
        self.mat = mat

    def stampa(self):
        for row in self.mat:
            for num in row:
                print '%3i' % num,
            print
```

Per testare la classe:

```
>>> m = Matrice([[1,2,3],[4,5,6],[7,8,9]])
>>> m.stampa()
  1   2   3
  4   5   6
  7   8   9
```

Esercizio

- Scrivere una funzione conta_caratteri(s) che ritorni un dizionario contenente il numero di occorrenze per ciascun carattere presente nella stringa s:

```
>>> conta_caratteri('aiuola')
{'a':2, 'i':1, 'u':1, 'o':1, 'l':1}
```

- *Facoltativo*: risolvere l'esercizio utilizzando i costrutti per il controllo delle eccezioni.

Soluzione

```
def conta_caratteri(s):
    conteggio = {}
    for c in s:
        if c in conteggio:
            conteggio[c] += 1
        else:
            conteggio[c] = 1

    return conteggio
```

Utilizzando i costrutti per il controllo delle eccezioni:

```
def conta_caratteri(s):
    conteggio = {}
    for c in s:
        try:
            conteggio[c] += 1
        except KeyError:
            conteggio[c] = 1

    return conteggio
```

Esercizio

- Scrivere un programma che stampi tutti i numeri perfetti inferiori ad un numero n dato in ingresso.

- Un numero naturale è perfetto se è uguale alla somma dei suoi divisori propri e dell'unità.

  Per esempio 6: i divisori propri di 6 sono 2 e 3, e la somma di 2, 3 e dell'unita' ha come risultato 6.

- 0 non è un numero perfetto.

Soluzione

```
def perfetto(n):
    if n == 0:
        return False

    divisori = []
    for i in range(1,n):
        if n % i == 0:
            divisori.append(i)

    return sum(divisori) == n

n = int(raw_input('Inserisci un numero positivo: '))

for i in range(0, n):
    if perfetto(i):
        print i
```