La Ricorsione

E' piuttosto frequente avere funzioni che richiamino altre funzioni è meno comune ma perfettamente lecito che una funzione possa richiamare sé stessa.

Una funzione ricorsiva è una funzione che richiama se stessa.

def f(x):

```
print(x)

f(x)
```

Se una ricorsione non ha una condizione di stop la chiamata alla funzione viene eseguita all'infinito ed in teoria il programma non giunge mai alla fine (questa situazione è conosciuta come ricorsione infinita). Nella realtà un programma con una ricorsione infinita non viene eseguito senza fine, dato che ogni chiamata ad una funzione impegna un po' di memoria del computer e questa memoria prima o poi finisce.

Python stampa un messaggio d'errore quando è stato raggiunto il massimo livello di ricorsione possibile.

In [1]:

```python
def f(x):
    print(x)
    f(x)
```

In [2]:

```python
f(2)
```

```
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
```

Per evitare la ricorsione infinita è necessario avere una condizione di stop.

In [ ]:

```
def ContoAllaRovescia(n):
  if n == 0:
    print("Via!")
  else:
    print(n)
    ContoAllaRovescia(n-1)
```

In [44]:

```
ContoAllaRovescia(5)
```

```
5
4
3
2
1
Via!
```

In [3]:

```
def stampa(lista):
    '''stampa i contenuti di lista'''
    for x in lista:
        print(x)
```

In [7]:

```
lista=['a','b','c','d']
stampa(lista)
```

```
a
b
c
d
```

In [8]:

```
def stampa(lista):
    '''stampa i contenuti di lista'''
    if lista==[] :return
    print(lista[0])
    stampa(lista[1:])
```

In [9]:

```
lista=['a','b','c','d']
stampa(lista)
```

```
a
b
c
d
```

In [13]:

```
def stampa_rov(lista):
    '''stampa i contenuti di lista'''
    if lista==[] :return
    stampa_rov(lista[1:])
    print(lista[0])
```

In [14]:

```
lista=['a','b','c','d']
stampa_rov(lista)
```

```
d
c
b
a
```

In [15]:

```
def pali(stringa):
    '''Restituisce True se la stringa e' palindroma, False altrimenti'''
    i,f=0,len(stringa)-1
    while i<f:
        if stringa[i]!=stringa[f]: return False
        i+=1
        f-=1
    return True
```

In [16]:

```
print(pali('anna'), pali('antenna'))
```

```
True False
```

In [17]:

```
def pali(stringa):
    '''Restituisce True se la stringa e' palindroma, False altrimenti'''
    if len(stringa) < 2: return True
    if stringa[0]==stringa[-1]: return pali(stringa[1:-1])
    return False
```

In [18]:

```
print(pali('anna'), pali('antenna'))
```

```
True False
```

ESERCIZIO: progettare la funzione canc(stringa) che, data una stringa, ne restituisce copia privata di spazi

In [19]:

```
def canc(stringa):
    '''Restituisce una stringa uguale alla prima esclusi i caratteri
    spazio che sono cancellati'''
    if len(stringa) == 0: return stringa
    a = canc(stringa[1:])
    if stringa[0] ==' ': return a
    return stringa[0] + a
```

In [20]:

```
canc('domani è un altro giorno')
```

Out[20]:

```
'domanièunaltrogiorno'
```

ESERCIZIO: progettare la funzione binarie(n) che, dato un intero, restituisce l'insieme di tutte le stringhe binarie lunghe n

In [23]:

```
def binarie(n):
    if n==0: return set()
    if n==1: return {'0','1'}
    a=binarie(n-1)
    b=set()
    for x in a:
        x0=x + '0'
        x1=x + '1'
        b.add(x0)
        b.add(x1)
    return b
```

In [24]:

```
binarie(3)
```

Out[24]:

```
{'000', '001', '010', '011', '100', '101', '110', '111'}
```

ESERCIZIO: progettare la funzione anagrammi(parola) che, data una stringa, restituisce la lista di tutti gli anagrammi che si possono ottenere dalla stringa.

Al ptimo posto dell'anagramma può ritrovarsi qualunque elemento della parola. Perottenere tutti gli anagrammi che hanno un dato elemento in testa basta estrarre l'elemento dalla parola e trovare gli anagrammi della parola più corta. A questi anagrammi verrà poi aggiunto in testa l'elemento estratto.

In [7]:

```
def anagramma(parola):
    '''Ritorna la lista di tutti gli anagrammi  della sequenza seq'''
    if len(parola) <= 1: return {parola}
    insieme=set()
    for i in range(len(parola)):
        parola1=parola[:i]+parola[i+1:]
        insieme1=anagramma(parola1)
        for x in insieme1:
            insieme.add(parola[i]+x)
    return insieme
```

In [8]:

```
anagramma('abc')
```

Out[8]:

```
{'abc', 'acb', 'bac', 'bca', 'cab', 'cba'}
```

In [9]:

```
1 anagramma('1234')
```

Out[9]:

```
{'1234',
 '1243',
 '1324',
 '1342',
 '1423',
 '1432',
 '2134',
 '2143',
 '2314',
 '2341',
 '2413',
 '2431',
 '3124',
 '3142',
 '3214',
 '3241',
 '3412',
 '3421',
 '4123',
 '4132',
 '4213',
 '4231',
 '4312',
 '4321'}
```

In [10]:

```
1 anagramma('ananas')
```

Out[10]:

```
{'aaanns',
 'aaansn',
 'aaasnn',
 'aanans',
 'aanasn',
 'aannas',
 'aannsa',
 'aansan',
 'aansna',
 'aasann',
 'aasnan',
 'aasnna',
 'anaans',
 'anaasn',
 'ananas',
 'anansa',
 'anasan',
 'anasna',
 'annaas',
 'annasa',
 'annsaa',
```

```
 'ansaan',
 'ansana',
 'ansnaa',
 'asaann',
 'asanan',
 'asanna',
 'asnaan',
 'asnana',
 'asnnaa',
 'naaans',
 'naaasn',
 'naanas',
 'naansa',
 'naasan',
 'naasna',
 'nanaas',
 'nanasa',
 'nansaa',
 'nasaan',
 'nasana',
 'nasnaa',
 'nnaaas',
 'nnaasa',
 'nnasaa',
 'nnsaaa',
 'nsaaan',
 'nsaana',
 'nsanaa',
 'nsnaaa',
 'saaann',
 'saanan',
 'saanna',
 'sanaan',
 'sanana',
 'sannaa',
 'snaaan',
 'snaana',
 'snanaa',
 'snnaaa'}
```

ESERCIZIO: progettare la funzione cambio(tot,lista) che, data una cifra e una lista di tagli di monete, restituisce il numero di modi differenti che si hanno per cambiare la cifra.

Ad esempio: cambio(8, [1, 5]) restituisce 2 i possibili cambi sono (1,1,1,1,1,1,1,1) (1,1,1,5)

In [27]:

```
def cambio(tot,lista):
    if tot == 0: return 1
    if len(lista)==1:
        if tot%lista[0]==0:return 1
        else: return 0
    count=0
    for i in range(len(lista)):
        if lista[i]<= tot: count += cambio(tot-lista[i],lista[i:])
    return count
```

In [28]:

```
cambio(22,[7,13])
```

Out[28]:

0

In [29]:

```
cambio(15, [1, 5, 10])
```

Out[29]:

6

In [30]:

```
cambio(100, [1, 5, 10, 20, 50])
```

Out[30]:

343

ESERCIZIO: progettare la funzione digital_sum(n) che, dato un numero, restituisce la somma delle cifre che lo compongono.

Ad esempio:

digital(324) retituisce 9=3+2+4.

In [31]:

```
def digital_sum(n):
    if n<10: return n
    return n%10 + digital_sum(n//10)
```

```
ESERCIZIO: progettare la funzione partizione(liste) che, data una lista di interi, produce una tupla di tre interi.

Il primo elemento della tripla è il numero di interi strettamente positivi presenti nella lista.

Il secondo elemento della tripla e' il numero di zeri presenti nella lista

Il secondo elemento della tripla è il numero di elementi strattamente negativi presenti nella tupla.
```

In [39]:

```
def partizione(lista):
    if lista == []: return (0,0,0)
    a, b, c = partizione(lista[1:])
    if lista[0] > 0: return (a + 1, b,c)
    elif lista[0] == 0: return (a, b + 1,c)
    else: return  (a,b,c+1)
```

In [40]:

```
partizione([3,0,-1,2,4,-1])
```

Out[40]:

```
(3, 1, 2)
```

In [ ]:

```

```