Esercizio.

- Scrivere una funzione **sum_digits(n)** che, dato un numero naturale **n**, restituisce la somma delle cifre in posizione pari e la somma delle cifre in posizione dispari di **n** (contando da destra).

Esempio n=2184571 sommapari=1+5+8+2, sommadispari=7+4+1

- Dato un naturale n, siano si(n) e sp(n) rispettivamente la somma delle cifre in posizione pari e dispari in n. (dove la 0-esima cifra è quella di minor peso). n è divisibile per 11 se il valore assoluto della differenza tra sp(n) e si(n) cioè abs(sp(n)-si(n))) è divisibile per 11 (*nota: n=0 e' divisibile per 11*). Scrivere una funzione ricorsiva **undici(n)** per decidere se un numero naturale dato è divisibile per 11. E' consentito l'uso dell'operatore modulo % solo per dividere per 10.

In [1]:
```
def sum_digits(n):
    """Restituisce la somma delle cifre pari e
la somma delle cifre dispari di n"""
    si,sp = 0,0
    aux = n
    while aux>0:
        si += aux%10
        aux /= 10
        sp += aux%10
        aux /= 10
    return si,sp
```

In [2]:
```
def undici(n):
    if n<10: # caso base: n ha una sola cifra
        return n == 0
    else:
        s_p,s_d = sum_digits(n)
        return undici(abs(s_d-s_p))
```

In [10]:
```
print undici(11*233+3)
print undici(11*23123)
```

```
False
True
```

Esercizio. Date le definizioni

```
a = []
b = []
```

Stabilire quali delle seguenti istruzioni producono la stampa

```
[0,1,2,3,4]
```

In [56]:
```
def f(x):
    if x == []:
        for x in range(3):
            a.append(x) # a e' una variabile globale: la modifica persiste
            b = [3,4] # b e' una variabile locale!
```

```
f(a) # print a -> [0,1,2]
a.extend(b) # b e' ancora una lista vuota, quindi a rimane invariata
print a
```

```
[0, 1, 2]
```

In [54]:
```
def f(x):
    if x == []:
        for x in range(3):
            a.append(x) # a e' una variabile globale
            # attenzione: poiche' il return e' nel ciclo for,
            # l'istruzione a.append(x) viene eseguita (al piu')
            # una sola volta!
            return [3,4]

a.extend(f(a)) #  l'interprete valuta prima f(a), quindi a diventa [0],
# poi estende a con il valore ritornato da f(a), ovvero con [3,4]
print a
```

```
[0, 3, 4]
```

In [12]:
```
def f(b):
    for x in range(3):
        a.append(x) # a e' una variabile globale
    b.extend([3,4]) # b e' un riferimento all'oggetto
    # passato come argomento ad f
    return a

f(a).extend(b) #  nella chiamata f(a), a diventa prima [0,1,2],
# e poi [0,1,2,3,4]. b rimane una lista vuota, quindi .extend(b)
# non altera a
print a
```

```
[0, 1, 2, 3, 4]
```

In [2]:
```
def f(x,lst):
    if x == []:
        for x in range(5): # da qui in poi x si riferisce a un intero
            lst.append(x)

f(a,b) # b fa la parte di lst in f(x,lst)
print b
```

```
[0, 1, 2, 3, 4]
```

Esercizio

Definire una funzione ricorsiva **to_base2(n)** per la conversione di un numero $n \in \mathbb{N}$ da base 10 a base 2.

(La funzione deve ritornare un oggetto string)

Vi ricordo che per convertire un numero da base 10 a base 2 dovete:

- finchè x è maggiore di 0 dividere x per 2. Il resto della divisione indica la prossima cifra del numero binario (tali

cifre devono essere scritte da destra a sinistra)

Esempio x=6

6/2 = 3, 6%2 = 0 la cifra piu' a destra del numero binario e' 0

3/2 = 1, 3%2 = 1 la seconda cifra (a partire da destra) del numero binario e' 1

1/2 = 0, 1%2 = 1 la terza cifra (a partire da destra) del numero binario e' 1

6 in base due e' 110

In [14]:
```
def to_base2(n):
    if n == 0:
        return ''
    else:
        return to_base2(n/2) + str(n%2)
```

In [24]:
```
to_base2(2**6+15)
```

Out[24]:
```
'1001111'
```

Cronometrare il codice

Un modo per cronometrare il codice è quello di utilizzare il modulo time come nel seguente esempio:

In [63]:
```
import time

start = time.time() #salva la data corrente nella variabile start
#istruzione 1
runtime = time.time()-start
print "l'istruzione 1 è stata eseguita in %f secondi" % runtime
```

```
l'istruzione 1 è stata eseguita in 0.000107 secondi
```

Esercizio (fattoriale)

1) Definire una funzione ricorsiva per il calcolo del fattoriale

2) Misurare il tempo medio impiegato per calcolare il fattoriale per x. 0 <= x < 500

3) Estendere la funzione fattoriale affinchè utilizzi un dizionario per evitare di ricalcolare piu' volte il fattoriale di uno stesso valore. Ad esempio una volta calcolato il fattoriale di 4, il dizionario, chiamiamolo C, conterrà i seguenti valori:

C={1:1, 2:2, 3:6, 4:24}

4) Confrontare il tempo medio impiegato per calcolare il fattoriale con i due metodi per x. 0 <= x < 500

In [1]:
```
def factorial(n):
    if n < 2:
        return 1
    else:
        return n*factorial(n-1)

C = {}
```

```
def factorial_cached(n):
    if n < 2:
        return 1
    else:
        if not (n in C.keys()):
            C[n] = n*factorial_cached(n-1)
        return C[n]
```

In [22]:
```
import time
n = 500
start = time.time()
for x in range(n):
    factorial(x)
runtime = time.time() - start
print "factorial average runtime: %f"%(runtime/n)
```

```
factorial average runtime: 0.000174
```

In [23]:
```
start = time.time()
for x in range(n):
    factorial_cached(x)
runtime_cached = time.time() - start
print "cached factorial runtime: %f"%(runtime_cached/n)
```

```
cached factorial runtime: 0.000019
```

In [25]:
```
print "speed_up: %f"%(runtime/runtime_cached)
```

```
speed_up: 8.929714
```

Esercizio (esponenziale)

1) Definire una funzione ricorsiva per il calcolo dell'esponenziale di x^y, con x ed y interi e y>0

2) Estendere la funzione precedente al caso $y \in \mathbb{Z}$

In [2]:
```
def powN(base,esp):
    if esp == 1:
        return base
    else:
        return base*powN(base,esp-1)

def powZ(base,esp):
    if esp == 0:
        return 1
    elif esp > 1:
        return powN(base,esp)
    else:
        return 1/float(base)*powZ(base,esp+1)
```

In [7]:
```
powN(3,4)
```

```
powZ(2,-5)
```

Out[7]: 0.03125

Esercizio. Cosa calcola la seguente funzione?

In [9]:
```
def funzionemisteriosa(x,y):
    global i
    if y<0:
        print "ERRORE"
    if y==0:
        return 1
    g=funzionemisteriosa(x,y/2)
    if y%2==1:
        return x*g*g
    else:
        return g*g
```

Confrontate il tempo di esecuzione di funzionemisteriosa per x=2, y=100 con una funzione (che avete già visto in precedenza) che calcola la stessa cosa. Quale risulta essere la più veloce? Perchè?

In [59]:
```
# la funzione misteriosa calcola la potenza y di x.
# risulta più veloce di powZ perche' calcola al piu'
# 2*ceil(log2(y)) prodotti (dove ceil e' la funzione
# che approssima all'intero piu' grande) invece di y-1
# prodotti
import time
n = 1000
x,y = 2,100
start = time.time()
for x in range(n):
    powN(x,y)
runtime = time.time() - start
print "powN runtime: %f"%(runtime)

start = time.time()
for x in range(n):
    funzionemisteriosa(x,y)
runtime_f = time.time() - start
print "funzionemisteriosa runtime: %f"%(runtime_f)

print "speed_up: %f"%(runtime/runtime_f)
```

```
powN runtime: 0.071012
funzionemisteriosa runtime: 0.012168
speed_up: 5.835998
```