Programmazione 1 — Soluzioni Esercitazione 2

Le soluzioni sono anche fornite in un unico script python sul sito:

`https://github.com/mathcoding/programming/tree/master/scripts`

1. Scrivere una procedura che calcoli l'ennesimo numero di Fibonacci usando un **processo iterativo** (si veda il `Lab 4` per la definizione di processo iterativo).

```
def FibRec(n):
    """ Calcolo di Fibonacci con PROCESSO RICORSIVO """
    if n == 0 or n == 1:
        return n
    return FibRec(n-1) + FibRec(n-2)

def FibIter(n):
    """ Calcolo di Fibonacci con PROCESSO ITERATIVO """
    def FibI(a, b, counter):
        if counter < 2:
            return a
        return FibI(a+b, a, counter-1)

    return FibI(1, 0, n)
```

2. Una funzione f è definita dalla regola seguente:

$$f(n) = \begin{cases} n & \text{if } n < 3 \\ f(n-1) + 2\,f(n-2) + 3\,f(n-3) & \text{if } n \geq 3 \end{cases} \tag{1}$$

   (a) Si scriva una procedura che calcoli f usando un **processo ricorsivo**.

   (b) Si scriva una procedura che calcoli f usando un **processo iterativo**.

```
def FnRec(n):
    """ Calcolo di f(n) = f(n-1)+2*f(n-2)+3*f(n-3), con f(0)=0, f(1)=1, f(2)=2
        (uso di processo ricorsivo) """
    if n < 3:
        return n
    return FnRec(n-1) + 2*FnRec(n-2) + 3*FnRec(n-3)

def FnIter(n):
    """ Calcolo di f(n) = f(n-1)+2*f(n-2)+3*f(n-3), con f(0)=0, f(1)=1, f(2)=2
        (uso di processo iterativo) """
    def FnI(a, b, c, counter):
        if counter < 3:
            return a
        return FnI(a+2*b+3*c, a, b, counter-1)

    return FnI(2, 1, 0, n)
```

3. Si scrive un predicato che ci dica se un dato numero n sia un numero primo. Si può usare la definizione che n è un numero primo, se e solo se n è il suo più piccolo divisore maggiore di uno (suggerimento: scrivere prima una funzione che trova il più piccolo divisore di n).

   È possibile scrivere questa procedura in modo tale che l'ordine di crescita per il numero di operazioni richieste sia $\Theta(\sqrt{n})$?

```
def IsPrime(n):
    """ Predicato che restituisce "True" quando "n" e' un numero primo """
    def MinorDivisore(a, n):
        if a*a > n:
            return n
        if n % a == 0:
            return a
        return MinorDivisore(a+1, n)

    if n == 0:
        return False
    if n == 1:
        return True
    return MinorDivisore(2,n) == n
```

4. La procedura `Sommatoria` vista nel `Lab 7` genera un processo ricorsivo lineare. La stessa procedura può essere riscritta in modo tale che il processo generato sia iterativo: scrivere la procedura che genera un processo iterativo lineare.

```
def SommatoriaIter(F, a, Next, b):
    def SommatoriaI(a, b, accumulator):
        if a > b:
            return accumulator
        return SommatoriaI(Next(a), b, accumulator+F(a))

    return SommatoriaI(a, b, 0)
def IntegraleIter(F, a, b, dx):
    def AddDx(x):
        return x + dx
    return dx*SommatoriaIter(F, a+dx/2, AddDx, b)
```

5. In maniera analoga alla funzione `Sommatoria`, si può definire una funzione `Produttoria` che restituisce i valori dei prodotti di una funzione valutata in un insieme di punti definiti da un dato intervallo $[a, b]$:

$$\prod_{n=a}^{b} f(n) = f(a) \cdot ... \cdot f(b)$$

   Scrivere tale funzione e mostrare come sia possibile usarla per calcolare $n!$

```
def ProduttoriaRec(F, a, Next, b):
    if a > b:
        return 1
    else:
        return F(a) * ProduttoriaRec(F, Next(a), Next, b)
print("Es. 2.5 => ProduttoriaRec(1, 5): ", ProduttoriaRec(lambda x: x, 1, Inc, 5))
```

6. Eseguire il grafico sovrapposto delle funzioni seguenti:

$$R(n) = n, R(n) = 10^5\, n, R(n) = n^2, R(n) = \log n, R(n) = n \log n, R(n) = 1.6^n, R(n) = 2^n.$$

ATTENZIONE: Questa procedura potrebbe non funzionare correttamente all'interno di Spyder. Tuttavia uno script python può essere eseguito anche da linea di comando, come visto in laboratorio. Per i grafici, si consiglia di vedere pochi grafici alla volta con velocità di crescita simili.

```
from numpy import linspace
from matplotlib.pyplot import plot, xlabel, ylabel, show, legend, clf
from math import log

def PlotFunzioni():
    D = linspace(1, 10, 100)
    clf()
    #plot(D, [x for x in D], label="y=x")
    plot(D, [x*x for x in D], label="y=x^2")
    plot(D, [log(x) for x in D], label="y=log(x)")
    plot(D, [x*log(x) for x in D], label="y=x*log(x)")

    #D = linspace(1, 10, 100)
    #plot(D, [1.6**x for x in D], label="y=1.6^x")
    #plot(D, [2**x for x in D], label="y=2^x")
    xlabel("x")
    ylabel("y=f(x)")
    legend(loc="upper left", shadow=True)
    show()
```