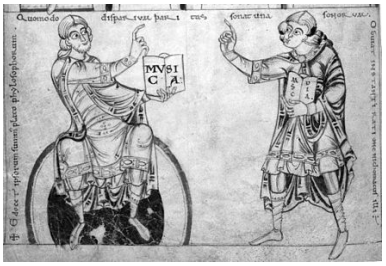
Matematica con Python

Mauro Saita

e-mail: maurosaita@tiscalinet.it

Versione provvisoria. Gennaio 2017.[1]

Alcuni programmi



Nicomaco di Gerasa e Platone.

Indice

1 Trasformazione di un intervallo di tempo da 'secondi' a 'ore, minuti, secondi'. **2**

2 Scrivere tre numeri in ordine decrescente **3**

3 Massimo comun divisore di tre numeri **5**

4 Alla ricerca di numeri perfetti **7**

4.1 Alcune note storiche sui numeri perfetti 8

5 Test di primalità **12**

[1] Nome File: python_esempi.tex

1 Trasformazione di un intervallo di tempo da 'secondi' a 'ore, minuti, secondi'.

Questo programma acquisisce in input un tempo espresso in secondi e lo restituisce (output) nel formato 'hh mm ss'(ore, minuti, secondi).

```
# Nome programma tempo01.py

# Main del programma
nsec = input('Inserire un tempo espresso in secondi: ')
nsec = int(nsec)
# Il quoziente hh tra n e 3600 fornisce il numero di ore.
hh = nsec//3600
# Il resto r di nsec e 3600 dà i secondi restanti
r  =  nsec % 3600
# Il quoziente mm tra r e 60 fornisce il numero di minuti.
mm = r // 60
# Il resto ss di r e 60 fornisce i secondi.
ss = r % 60

#Tempo in hh mm ss
print('Tempo espresso in ore, minuti, secondi: ')
print(str(hh) + ' hh ' + str(mm)+ ' mm ' + str(ss)+ ' ss ' )
```

Qui di seguito lo stesso programma realizzato utilizzando una funzione (denominata 'hhmmss'):

```
# Nome programma tempo02.py

def hhmmss(nsec):
    # Il quoziente hh tra n e 3600 fornisce il numero di ore.
    hh = nsec//3600
    # Il resto r di nsec e 3600 dà i secondi restanti
    r = nsec % 3600
    # Il quoziente mm tra r_1 e 60 fornisce il numero di minuti.
    mm = r // 60
    # Il resto ss di r e 60 fornisce i secondi.
    ss = r % 60
    return str(hh)+ ' hh ' + str(mm)+ ' mm '  + str(ss) + ' ss '

# Main del programma
n = input('Inserire un tempo espresso in secondi: ')
n = int(n)
tempo=hhmmss(n)
print('Tempo espresso in ore, minuti, secondi: ' + tempo)
```

2 Scrivere tre numeri in ordine decrescente

Il seguente programma acqisisce tre interi e li scrive in ordine decrescente (dal più grande al più piccolo). La funzione 'scambia' inverte l'ordine della coppia ordinata x, y. Per 'chiamare' la funzione dal programma si scrive il nome della funzione seguito dai due numeri, cioè: scambia(x,y). Attenzione!, x, y possono essere due costanti, per esempio scambia(12, 37); oppure due variabili contenenti i numeri che si vogliono scambiare. I nomi delle variabili possono essere diversi da x e y, in questo programma la funzione viene invocata così: scambia(a,b), scambia(b,c), scambia(a,c) .

L'output è un un vettore con due componenti (più precisamente una 'tupla'). Il programma acquisisce l'output e denomina il vettore con la lettera 't' (si scrive $(t) = (a, b)$). La prima componente di 't' è $(t)[0]$, mentre la seconda è $(t)[1]$.

Esistono modi molto più semplici e raffinati per ordinare 3 numeri (o, più in generale, n numeri). La soluzione qui proposta ha come unico scopo quello di acquisire familiarità con le funzioni.

```
# Nome programma ordina01.py
# Il programma acquisisce tre interi e li ordina in ordine decrescente
# (non crescente).

def scambia(x,y):
    x_1=x
    x=y
    y=x_1
    return (x,y)

# MAIN DEL PROGRAMMA
a = int(input("Digitare un intero positivo. n = "))
b = int(input("Digitare un intero positivo. m = "))
c = int(input("Digitare un intero positivo. k = "))

# Se a<b scambia a e b
if (a<b):
    (t)=scambia(a,b)
    a=(t)[0]
    b=(t)[1]

# Se b<c<a scambia b e c
if (b<c and c<a):
    (t)=scambia(b,c)
    b=(t)[0]
    c=(t)[1]

# Se c>=a scambia a e c e poi scambia c e b
if (c>=a):
    (t)=scambia(a,c)
```

```
    a=(t)[0]
    c=(t)[1]

    (t)=()
    (t)=scambia(b,c)
    b=(t)[0]
    c=(t)[1]

print(a,b,c)
```

3 Massimo comun divisore di tre numeri

Il seguente programma funziona così

- Acquisisce tre interi a, b, c.
- Calcola il massimo comun divisore di (a, b) e scrive il risultato nella variabile denominata 'd' : $d = MCD(a, b)$.
- Calcola il massimo comun divisore di (d, c). $MCD(d, c)$ è il risultato cercato.

L'algoritmo si basa sul seguente fatto: $MCD(a, b, c) = MCD(MCD(a, b), c)$.

```
# Nome programma mcd03.py
# Calcolo del Massimo Comun Divisore di tre numeri

def scambia(x,y):
    x_1=x
    x=y
    y=x_1
    return (x,y)

def mcd(x,y):
    i=1
    while(i!=0):
        resto= x%y
        if (resto==0):
            mcd = y
            return mcd
        else:
            x = y
            y = resto


# MAIN DEL PROGRAMMA
n = int(input("Digitare un intero positivo. n = "))
m = int(input("Digitare un intero positivo. m = "))
k = int(input("Digitare un intero positivo. k = "))

# Si utilizzano le variabili ausiliarie a,b,c perché si vuole conservare l'ordine
# con il quale sono stati inseriti i numeri da tastiera
a = n
b = m
c = k

# Se a<b scambia a e b
if (a<b):
```

```
    (t)=scambia(a,b)
    a=(t)[0]
    b=(t)[1]
# Calcola il MCD di a,b
d = mcd(a,b)

# Se d<c scambia d e c
if (d<c):
    (t)=scambia(d,c)
    d=(t)[0]
    c=(t)[1]
# Calcola il MCD di d,c.
e = mcd(d,c)

# Scrive il massimo comun divisore di tre numeri
print('MCD', (n,m,k), ' = ',str(e))
```

4 Alla ricerca di numeri perfetti

Il seguente programma stampa i numeri perfetti compresi in un range deciso dall'utente. Importando il modulo time è stato possibile misurare il tempo di esecuzione del programma. Per non avere tempi di attesa troppo lunghi conviene inserire un range di valori piuttosto piccolo ...

```
# Nome programma perfetti02.py

import time

def num_perfetto(x):
    somma_divisori=0
    for i in range(1, x):
          if x % i == 0:
              somma_divisori= somma_divisori +i
    if (x==somma_divisori):

         return True
    else:
         return False

#Main del programma
print('IL PROGRAMMA CERCA I NUMERI PERFETTI COMPRESI TRA GLI INTERI n E m, n<m.' )

n = input('Digitare un intero positivo n: ')
m = input('Digitare un intero positivo m: ')

inizio=time.time()
i=1
n = int(n)
m = int(m)
conta_numeri_perfetti = 0
if (n > 0 and m>0 and n<m):
     print('I numeri perfetti compresi tra ' + str(n) + ' e ' + str(m) )
     print(' sono i seguenti: ')
     i = n
     while (i<m+1):
         if  num_perfetto(i)==True:
             conta_numeri_perfetti = conta_numeri_perfetti + 1
             print(str(i))
         i=i+1
     if (conta_numeri_perfetti == 0):
         print("Non ci sono numeri perfetti compresi nell'intervallo specificato")
     fine=time.time()
     durata=fine-inizio
```

```
    print('(Tempo di esecuzione: ' + str(durata) + ' secondi.)')
else:
    print('Numeri non corretti. Digitare due interi positivi n, m con n<m.')
```

4.1 Alcune note storiche sui numeri perfetti

1. `Pitagora (Samo, ∼ 570 a.C. - Metaponto, ∼ 495 a.C.)` Le prime testimonianze sullo studio dei numeri perfetti risalgono a Pitagora e alla scuola pitagorica. Secondo Pitagora la perfezione numerica andava posta in relazione con i divisori del numero. Chiamò *eccedenti* quei numeri la cui somma dei divisori risultava maggiore del numero stesso e *difettivi* quelli per cui risultava minore. I numeri restanti, molto rari, erano detti *perfetti*. Pitagora intuì che la proprietà di perfezione (la somma dei divisori di un numero è uguale al numero stesso) doveva essere legata al numero 2. Si accorse infatti che le potenze 2^n di 2 sono lievemente difettive, per esempio

Numeri del tipo 2^n	Divisori	Somma dei divisori
$2^1 = 2$	1	1
$2^2 = 4$	$1, 2$	3
$2^3 = 8$	$1, 2, 4$	7
$2^4 = 16$	$1, 2, 4, 8$	15
$2^5 = 32$	$1, 2, 4, 8, 16$	31
$\dots$	$\dots$	$\dots$

   Più in generale, il numero $N = 2^n$ ha i seguenti divisori (minori di N):

$$1,\ 2^1,\ 2^2,\ \dots,\ 2^{n-1} \tag{4.1}$$

   e la loro somma vale[2]

$$1 + 2^1 + 2^2 + \cdots + 2^{n-1} = 2^n - 1 \quad n \in \mathbb{N},\ n > 1 \tag{4.2}$$

   Questo dimostra che *tutte* le potenze 2^n di 2 sono difettive esattamente per una unità .

2. `Euclide (∼ 300 a.C.)` Euclide ha dimostrato questo importante risultato di teoria dei numeri

   Teorema 4.1 (Elementi, Libro IX, Proposizione 36)**.** *Se, per qualche $n > 1$, $2^n - 1$ è primo allora $(2^n - 1)2^{n-1}$ è un numero perfetto.*

[2] L'uguaglianza 4.2 era nota a Pitagora. Essa è un caso particolare dell'uguaglianza

$$(1 - q)(1 + q^1 + q^2 + \cdots + q^{n-1}) = q^n - 1, \qquad q, n \in \mathbb{N},\ q > 1,\ n > 1$$

la cui dimostrazione consiste nel fare la moltiplicazione che compare a primo membro dell'uguaglianza, eseguire tutte le semplificazioni possibili e verificare che ciò che resta è $q^n - 1$, ossia il secondo membro dell'uguaglianza. Posto $q = 2$, si ottiene

$$1 + 2^1 + 2^2 + \cdots + 2^{n-1} = 2^n - 1$$

ossia l'uguaglianza 4.2

Per esempio $2^3 - 1 = 7$ è primo, quindi il numero $2^{3-1}(2^3 - 1) = 4 \cdot 7 = 28$ è perfetto. Anche $2^5 - 1 = 31$ è primo, quindi $2^{5-1}(2^5 - 1) = 16 \cdot 31 = 496$ è perfetto, eccetera.

Dimostrazione. Si consideri il numero $N = p2^{n-1}$, con p primo.

I divisori di N (minori di N) sono

$$1,\, 2^1,\, 2^2,\, \dots,\, 2^{n-1} \qquad \text{i divisori di } 2^n \tag{4.3}$$

$$+$$

$$p,\, p\,2^1,\, p\,2^2,\, \dots,\, p\,2^{n-2} \tag{4.4}$$

$$\tag{4.5}$$

La somma S di tutti i divisori di N è

$$\begin{aligned} S &= \underbrace{1 + 2^1 + 2^2 + \cdots + 2^{n-1}}_{2^n-1} + \underbrace{p(1 + 2^1 + 2^2 + \cdots + 2^{n-2})}_{p(2^{n-1}-1)} \\ &= 2^n - 1 + p\,(2^{n-1} - 1) \end{aligned} \tag{4.6}$$

Se $2^n - 1$ è primo si può scegliere $p = 2^n - 1$. Allora la somma S dei divisori di N diventa

$$\begin{aligned} S &= (2^n - 1) + (2^n - 1)\,(2^{n-1} - 1) \\ &= (2^n - 1)\big(1 + (2^{n-1} - 1)\big) \\ &= (2^n - 1)2^{n-1} \\ &= N \end{aligned} \tag{4.7}$$

Segue che $N = (2^n - 1)2^{n-1}$ è un numero perfetto. ∎

3. `Nicomaco di Gerasa (~ 60 d.C. - ~ 120 d.C.)`. Matematico greco dell'età ellenistica, vissuto in Palestina. La sua opera più importante giunta fino a noi è l'*Introduzione all'aritmetica* nella quale si occupò di numeri primi e di numeri perfetti e ne indagò il loro significato. Di formazione pitagorica, Nicomaco raccolse, senza riportare le dimostrazioni, le principali proprietà scoperte dalla scuola pitagorica sui numeri perfetti ossia

   (a) L'n-esimo numero perfetto ha n cifre.

   (b) Tutti i numeri perfetti sono pari.

   (c) Tutti i numeri perfetti terminano con 6 e con 8 in modo alternato.

   (d) Ogni numero perfetto si può scrivere nella forma $2^{n-1}(2^n - 1)$ per qualche $n > 1$, dove $2^n - 1$ è primo.

   (e) Esistono infiniti numeri perfetti.

Queste proposizioni, nonostante fossero prive di dimostrazione, vennero per un lungo periodo assunte per vere. Oggi si sà che le asserzioni (a) e (c) sono false, mentre le altre non sono ancora state nè provate nè confutate.

Nicomaco fornì inoltre l'elenco dei primi quattro numeri perfetti: $6, 28, 496, 8128$.

4. `Ibn al−Hasan (Bassora, 965 circa ˘ Il Cairo, 1039).` Fu uno dei più importanti e geniali scienziati del mondo islamico e dell'inizio del secondo millennio (si legga a questo proposito l'interessante biografia riportata da wikipedia: https://it.wikipedia.org/wiki/Alhazen). Sui numeri perfetti arrivò a conclusioni simili a quelle di Euuclide.

5. `Ibrahim ibn Fallus (1194-1239).` Matematico arabo che venne a conoscenza degli studi greci sui numeri perfetti. Nel suo trattato, che si ispira all'*Introduzione all'aritmetica* di Nicomaco studiò i numeri da un punto di vista puramente matematico rinunciando a ogni interpretazione morale o religiosa. Elencò inoltre quelli che secondo lui erano i primi dieci numeri perfetti. I primi sette sono in effetti corretti (e sono i primi sette numeri perfetti), mentre gli altri sono inesatti. Tale scoperta restò sconosciuta ai matematici occidentali, che riuscirono a trovare i successivi numeri perfetti soltanto qualche secolo più tardi.

6. `Hudalrichus Regius.` Nel 1536 provò che $2^{13} - 1 = 8191$ è primo. Questo fatto gli permise di rendere pubblico il quinto numero perfetto $2^{12}(2^{13} - 1) = 33550336$ e di dimostrare così che la prima proposizione di Nicomaco è falsa (il quinto numero perfetto ha 8 cifre e non 5).

7. `Cataldi (Bologna, 15 aprile 1552 - Bologna, 11 febbraio 1626).` Trovò il sesto numero perfetto $2^{16}(2^{17} - 1) = 8589869056$ e con ciò dimostrò che la terza proposizione di Nicomaco è falsa, poichè il quinto e il sesto numero perfetto terminano entrambi con 6. Trovò anche il settimo numero perfetto, ossia $2^{18}(2^{19} - 1) = 137438691328$.

8. `Eulero (Basilea, 15 aprile 1707 - San Pietroburgo, 18 settembre 1783).` Nel 1732 trovò l'ottavo numero perfetto $2^{30}(2^{31} - 1) = 2305843008139952128$, 125 anni dopo la scoperta del settimo e nel 1738 corresse Cataldi, provando che $2^{29} - 1$ non era primo. Il risultato più importante trovato da Eulero in questo campo è il seguente:

   Teorema 4.2. *Ogni numero perfetto* pari *è del tipo* $2^{n-1}(2^n - 1)$ *con* n *primo.*

   Ciò significa che la quarta asserzione di Nicomaco è vera nel caso di numeri pari. Utilizzando questo risultato, egli dimostrò anche che i numeri perfetti terminano con le cifre 6 oppure 8, anche se non alternativamente (quindi riformulò e dimostrò l'asserzione (b) di Nicomaco). Trovò infine che $2^{30}(2^{31} - 1)$ è un numero perfetto; esso rimase il più grande numero perfetto conosciuto per altri 150 anni. Eulero, durante la sua vita, scrisse quasi 100 articoli sulla teoria dei numeri.

Per maggiori informazioni e curiosità sui numeri perfetti si consiglia di consultare i siti

webmath2.unito.it/paginepersonali/romagnoli/perfetti.pdf

http://www-groups.dcs.st-and.ac.uk/history/HistTopics/Perfect_numbers.html

che sono serviti per scrivere queste note.

Per curiosità si riportano i primi 15 numeri perfetti:

```
6,
28,
496,
8128,
33550336,
8589869056,
137438691328,
2305843008139952128,
2658455991569831744654692615953842176,
191561942608236107294793378084303638130997321548169216,

1316403645856964833723975346045872291022347231838694311
17783728128,

1447401115466452442794637312608598848157367749147483581
8906635434913119915212 8,

2356272345726734706578954899670990498847754785839260071014302
7597506337283178622239730365539602600561360255566462503270175
0528925780432155433824984287771524270103944969186640286445341
2803383143979023683862403317143592235664321970310172071316352
7487298747400647801939587165936401087419375649057918549492160
555646976,

1410537837067120690632079580860631898814867435147156678388386
7599995486774265238011410419332903769025156195056870982932716
4087724366370087116731268159313652487450652439805877296207297
4467232951666582288469268077866528701889208678794514783645693
1392206037069506473607357237869517647305526682625328488638371
5072974324463835300053138429460296575143368065570759537328128,

5416252628436584741265446537439131614085649053903169578460392
0818387206994158534859198999921056719921919057390080263646159
2800138276054397462627889030573034455058270283951394752077690
4492443149486172943511312628083790493046274068171796046586734
8720992572190569465545299629919823431031092624244463547789635
4414813917198164416055867880921478866773213987566616247145517
2696430221755428178425481731961195165985555357393778892340514
6222324506715979193757372820860878214322052227584537552897476
2561793951766244263144803134469350852036575847982475360211728
8040378304860287362125931378999490033667394150374722496698402
8240806042108690077670395259231894666273615212775603535764707
9522501738583051710286030212348966478513639499289049732921451
0750597991145622151989934576498429132 8
```

5 Test di primalità

Il numero digitato è primo? Ecco un programma per scoprirlo.

```
# Nome file: primalita01.py

import math
import time

s=input('inserisci numero ')
n=eval(s)

#definizione e inizializzazione delle variabili
primo=True
#variabile per le divisioni successive
i=2

inizio=time.clock()

while(i<n):   # oppure(i<math.sqrt(n)):
    resto= n%i

    if (resto==0):
        primo=False
        break

    i=i+1

fine=time.clock()
if (primo==True):
    print ('il numero: ' + str(n) + ' è primo' )
else:
    print ('il numero: ' + str(n) + ' non è primo' )

tempo=fine-inizio
tempo2=format(tempo, '10.6f')
#print("il tempo di esecuzione è di " + '%.6f' %tempo + "secondi")

print("il tempo di esecuzione è di" + str(tempo2) + "secondi")
```

Questa seconda versione del programma chiama la funzione 'test_p' per effettuare il test di primalità, se la funzione ritorna il valore 'True' il numero è primo altrimenti, se ritorna il valore 'False', il numero è composto. Inoltre, è l'utente che decide quando uscire dal programma: digitando 's' alla domanda 'Si desidera uscire dal programma?' si interrompe l'esecuzione, in caso contrario si può interrogare il programma per un nuovo test di primalità.

```
# Nome programma primalita_02.py

import math
import time

def test_p(x):
    primo=True
    i=2
    while(i<n): #while(i<math.sqrt(n)):
        resto= n%i
        if (resto==0):
            primo=False
            break

        i=i+1

    return primo

# MAIN DEL PROGRAMMA
while True:
    n = int(input("Digitare un intero positivo. n = "))
    # La variabile inizio contiene il tempo subito prima la chiamata
    # della funzione test_p
    inizio=time.clock()
    p=test_p(n)
    # La variabile fine contiene il tempo subito dopo la chiamata della
    # funzione test_p
    fine=time.clock()
    # La variabile tempo contiene il tempo di esecuzione del programma
    tempo=fine-inizio
    tempo2=format(tempo, '10.6f')
    if (p==True):
        print ('Il numero: ' + str(n) + ' è primo' )
    else:
        print ('Il numero: ' + str(n) + ' non è primo' )
    #print("il tempo di esecuzione è" + '%.5f' %tempo + "secondi")
    print("Il tempo di esecuzione è" + str(tempo2) + " secondi")
    uscita = input("Si desidera uscire dal programma? (s) per sì, invio per continuare : ")
    if (uscita==('s' or 'S')):
        break
```