Hoja de ejercicios 3

Bucles

4 de diciembre de 2013

Informática
Año 2013/2014
Facultad de CC.
Matemáticas

▷ 1. Número de cifras

Escribe un programa que determine cuántas cifras tiene un entero dado.

Solución

Una solución sencilla consiste en ir eliminando la cifra menos significativa de un número sucesivamente hasta llegar a una única cifra. Es decir, si nos dan un número, por ejemplo el 3754, podemos comprobar si tiene una cifra o no, si tiene más de una cifra entonces podemos eliminar la última (el 4) y añadir uno al contador de cifras, de esta forma nos queda el 375 y seguimos contando sus cifras de igual manera. Casi cuesta más explicarlo que expresarlo en un programa:

```
def numCifras(m):
    cont=1
    while m >= 10:
        m = m / 10
        cont = cont +1
    return cont
```

El programa anterior es *bastante* correcto, pero es fácilmente mejorable. ¿Se te ocurre cómo? (Piensa un poco antes de seguir...) En primer lugar, el código que acabamos de escribir no funciona correctamente con números negativos. Eso se soluciona fácilmente. Además, siempre que aparezca una constante numérica en el código, como en este caso es el 10, deberíamos preguntarnos que papel juega. En este caso, el 10 es fruto de una suposición no exigida en el enunciado: estamos suponiendo que los números están expresados en base 10. De hecho, con el mismo esfuerzo podríamos solucionar el problema de contar las cifras de una determinada cantidad expresada en cualquier base. El código siguiente tiene en cuenta estas mejoras:

```
def numCifras(m,base):
    cont=1
    while m >= base:
        m = m / base;
        cont = cont +1;
    return cont
```

Observa la instrucción `int m = abs(n)` que es la que se encarga de hacer que nuestro programa pueda funcionar con números negativos.

▷ 2. Número de cifras en una determinada base

Escribe un programa que determine cuántas cifras tiene una cantidad C si la expresamos en una determinada base b.

Solución

Una solución sencilla consiste en ir eliminando la cifra menos significativa de un número

sucesivamente hasta llegar a una única cifra. Es decir, si nos dan un número, por ejemplo el 3754, podemos comprobar si tiene una cifra o no, si tiene más de una cifra entonces podemos eliminar la última (el 4) y añadir uno al contador de cifras, de esta forma nos queda el 375 y seguimos contando sus cifras de igual manera. Esto se expresa mejor en un programa:

```
def numCifras(m,base):
    cont=1
    while m >= base:
        m = m / base
        cont = cont +1
    return cont
```

El programa anterior es *bastante* correcto, pero es fácilmente mejorable. ¿Se te ocurre cómo? (Piensa un poco antes de seguir...) En primer lugar, el código que acabamos de escribir no funciona correctamente con números negativos. Eso se soluciona fácilmente. Además, siempre que aparezca una constante numérica en el código, como en este caso es el 10, deberíamos preguntarnos que papel juega. En este caso, el 10 es fruto de una suposición no exigida en el enunciado: estamos suponiendo que los números están expresados en base 10. De hecho, con el mismo esfuerzo podríamos solucionar el problema de contar las cifras de una determinada cantidad expresada en cualquier base. El código siguiente tiene en cuenta estas mejoras:

```
def numCifras(m,base):
    cont=1
    m = abs(m)
    while m >= base:
        m = m / base;
        cont = cont +1;
    return cont
```

Observa la instrucción `int m = abs(n)` que es la que se encarga de hacer que nuestro programa pueda funcionar con números negativos.

▷ 3. Varianza

La varianza de n números $x_1, \ldots, x_n$ es

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

donde $\bar{x}$ es la media.

Se necesita un programa que lea una secuencia de números y calcule su varianza. Los números se introducirán usando el teclado y se acabarán con un 0, que obviamente no será tenido en cuenta para el cálculo de la varianza. En cualquier caso los números no podrán ser almacedados en una lista.

Pista: La fórmula de la varianza parece indicar que necesitamos hacer dos pasadas sobre los datos: una para calcular primero la media y otra para aplicar esa fórmula. Porque no es razonable pedir a quien use nuestro programa que escriba dos veces los mismo datos, parece que necesitamos almacenarlos internamente. Afortunadamente, un poco de esfuerzo mental y un poco de aritmética básica te permitirán reescribir la fórmula de la varianza, de forma que se pueda calcular de una sola pasada y puedas evitar todas las complejidades que produce el almacenamiento de unos datos arbitrariamente largos.

Solución

La varianza es la media aritmética de los cuadrados de la distancia de cada punto a la media; es decir

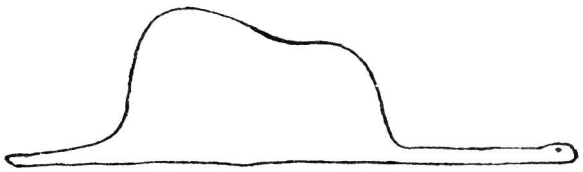
$$\frac{1}{n}\sum_{i=1}^{n}(x_i-\bar{x})^2=\frac{1}{n}((x_1-\bar{x})^2+\cdots+(x_n-\bar{x})^2)$$

Con esta ecuación difícilmente se calcula la varianza en una sóla pasada, pero si se expande queda:

$$\frac{1}{n}\sum_{i=1}^{n}x_i^2-\frac{1}{n^2}\Big(\sum_{i=1}^{n}x_i\Big)^2$$

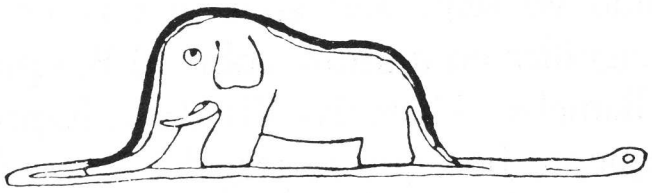
▷ 4. El principito

Mi dibujo número uno. Era así:



Mostré mi obra maestra a las personas mayores y les pregunté si mi dibujo les daba miedo. Me contestaron: "¿Por qué nos habría de asustar un sombrero?"

Pero mi dibujo no representaba un sombrero, sino una serpiente boa que digería un elefante. Dibujé entonces el interior de la serpiente boa, a fin de que las personas adultas pudiesen comprender, pues los adultos siempre necesitan explicaciones. Mi dibujo número dos era así:



Las personas mayores me aconsejaron abandonar los dibujos de serpientes boas abiertas o cerradas y que pusiera más interés en la geografía, la historia, el cálculo y la gramática. Y así fue como a la temprana edad de seis años, abandoné una magnífica carrera de pintor, desalentado por el fracaso de mis dibujos números uno y dos. Las personas mayores nunca comprenden por sí solas las cosas, y resulta muy fastidioso para los niños tener que darles continuamente explicaciones.

El principito, Antoine de Saint-Exupéry

Al principito le gustan mucho los números y prefiere las fracciones así:

$$\frac{5687171}{18686419}$$

ya que puede ver y disfrutar de muchas cifras. Sin embargo los mayores prefieren ver cosas más simples y explicadas:

$$\frac{5687171}{18686419} = \frac{7 \cdot 812453}{23 \cdot 812453} = \frac{7}{23}$$

Pista: Se trata de simplificar al máximo una fracción $\frac{n}{d}$. Para ello, tenemos que calcular el máximo común divisor del numerador y del denominador, $\mathrm{mcd}(n, d)$, que es el mayor número por el que podemos dividir tanto n como d.

Solución

Para no tener que dar explicaciones a los mayores el principito ha decidido hacer un programa que simplifique las fracciones y así, si alguien viene y se asusta de ver largas ristras de números, el programa se encargará de mostrar la fracción más reducida posible.

Para ello, como aprendió en sus primeras clases de matemáticas, basta con calcular el máximo común divisor entre el numerador y el denominador y dividir tanto el numerador como el denominador por dicho número. Es decir, si $m = \mathrm{mcd}(n, d)$ y $n' = \frac{n}{m}$ y $d' = \frac{d}{m}$, entonces $\frac{n}{d} = \frac{n'}{d'}$ y la fracción $\frac{n'}{d'}$ es irreducible.

Notas bibliográficas En el siglo III a. C., Euclides escribió los *Elementos* [**?**, **?**, **?**], dividida en trece volúmenes, que ha sido la obra matemática por excelencia durante más de dos mil años. En su libro VII [**?**], aparece el conocido algoritmo de Euclides para encontrar el máximo común divisor de dos números. En [**?**] también se relata el texto original de Euclides explicando el algoritmo. Puede encontrarse la implementación de este algoritmo, y de otros similares, en infinidad de libros básicos de programación, como [**?**].

▷ 5. Suma marciana

Se ha encontrado en Marte la siguiente operación de sumar, resuelta en una roca:

$$\begin{array}{ccc} \clubsuit & \diamondsuit & \spadesuit \\ & \clubsuit & \heartsuit \\ \hline \diamondsuit & \spadesuit & \clubsuit \end{array}$$

Se desea descifrar el significado (o sea, el valor) de esos símbolos, suponiendo que se ha empleado el sistema de numeración decimal.

Solución

Una posibilidad consiste en producir cada una de las combinaciones posibles,

♣	♢	♠	♡
0	0	0	0
0	0	0	1
...	...	...	...
0	0	0	9
0	0	1	0
...	...	...	...
0	9	9	9
1	0	0	0
...	...	...	...

y examinar cuáles de ellas verifican esa cuenta.

Para concretar un poco más el problema, tendremos en cuenta lo siguiente:

- Por estar en el sistema de numeración decimal, los posibles valores de cada uno de los símbolos que intervienen en la cuenta son los valores del cero al nueve.
- Como se ha empleado el sistema de numeración decimal, la grafía ♣♢♠ representa la cantidad $100♣ + 10♢ + ♠$.

Entonces, el algoritmo descrito se traduce a Python fácilmente así:

```
def martian_sum():
    club = 0
    while club <10:
        diamond = 0
        while diamond<10:
            spade = 0
            while spade<10:
                heart = 0
                while heart<10:
                    sum1 = 100*club + 10*diamond + spade
                    sum2 = 10*club + heart
                    sum = 100 * diamond + 10 * spade + club
                    if sum1+sum2 == sum:
                        print "  ", heart,diamond,spade
                        print "+   ",club,heart
                        print "---------"
                        print "  ",diamond,spade,club
                    else:
                        print sum1,sum2,sum
                    heart +=1
                spade +=1
            diamond += 1
        club += 1
```

Ahora, pueden hacerse dos observaciones que nos permiten limitar un poco los tanteos:

- Los símbolos ♣ y ♢ no pueden ser nulos, ya que están al principio de los números.

- Los cuatro símbolos empleados por los habitantes de Marte pueden suponerse distintos.

▷ 6. Aproximaciones al número π

Desde que el ser humano se ha preocupado por conocer el entorno y explicar el porqué de las cosas que lo rodean, ha habido personas que han intentado calcular la relación existente entre la longitud de la circunferencia y el radio (o diámetro) que la define.

Largo ha sido el periplo de los matemáticos en torno a este número. En este ejercicio te proponemos utilizar las siguientes fórmulas matemáticas para construir programas que permitan calcular aproximaciones al número π.

- François Viète (1540–1603) en 1593:

$$\frac{2}{\pi} = \sqrt{\frac{1}{2}} \cdot \sqrt{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{1}{2}}} \cdot \sqrt{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{1}{2} + \frac{1}{2}\sqrt{\frac{1}{2}}}} \cdots$$

- John Wallis (1616–1703) en 1656:

$$\frac{4}{\pi} = \frac{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots}{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots}$$

- Gottfried Wilhelm Leibniz (1646–1716) en 1673:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \cdots = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}$$

- Borwein en 1987:

$$x_0 = \sqrt{2} \qquad y_1 = 2^{\frac{1}{4}} \qquad \pi_0 = 2 + \sqrt{2}$$

$$x_{n+1} = \frac{1}{2}\left(\sqrt{x_n} + \frac{1}{\sqrt{x_n}}\right) \qquad y_{n+1} = \frac{y_n\sqrt{x_n} + \frac{1}{\sqrt{x_n}}}{y_n + 1} \qquad \pi_n = \pi_{n-1}\frac{x_n + 1}{y_n + 1}$$

Tiene una convergencia muy rápida: $\pi_n - \pi < 10^{-2^{n+1}}$.

Notas bibliográficas π es sin duda el más famoso de los números, y por eso la bibliografía sobre él es extensísima. Dos libros llenos de curiosidades sobre π son [**?**] y [**?**]. En [**?**] se dedica un capítulo a los diversos métodos usados a lo largo de la historia para calcular π. Tan distinguido número no podía faltar tampoco en Internet:

`http://www.joyofpi.com/pilinks.html`
`http://www-groups.dcs.st-and.ac.uk/~history/HistTopics/Pi_through_the_ages.html`

En las dos primeras direcciones hay cientos de referencias diversas dedicadas a π; en la tercera puedes encontrar muchas, pero que muchas, cifras decimales de π.

Un poco de historia El primero que utilizó el símbolo π fue William Jones (1675–1749) en 1706. A Euler le gustó este símbolo, lo adoptó y difundió su uso. La fórmula de Leibniz es una particularización de la serie que define el arcotangente de un ángulo; James Gregory (1638–1675) la había descrito con anterioridad, pero no hay ninguna información de que la usase para aproximar el número π.

He aquí una tabla con la cronología del número de cifras decimales de π calculadas:

Número de decimales	Año	Autores	Sistema informático
2 037	1949	G.W. Reitwiesner, ...	ENIAC
10 000	1958	F. Genuys	IBM 704
100 265	1961	D. Shanks y J. Wrench	IBM 7090
1 001 250	1974	J. Guilloud y M. Bouyer	CDC 7600
16 777 206	1983	Y. Kanada, S. Yoshino y Y. Tamura	HITAC M-280H
134 214 700	1987	Y. Kanada, Y. Tamura, Y. Kubo, ...	NEC SX-2
6 442 450 000	1995	D. Takahashi y Y. Kanada	HITAC S-3800/480 (2 procesadores)
51 539 600 000	1997	D. Takahashi y Y. Kanada	HITACHI SR2201 (1024 procesadores)
206 158 430 000	1999	D. Takahashi y Y. Kanada	HITACHI SR8000 (128 procesadores)

Solución

- El método de Borwein para aproximar el valor de π es extemandamente rápido, tan sólo en la segunda iteración $n = 2$, el valor de π_2 difiere de π en menos de 10^{-8}, es decir, menos de una cienmillonésima. Haremos un programa que calcule los valores de la sucesión π_n hasta el valor de n que deseemos.

  El programa no requiere grandes habilidades de programación, basta con ser cuidadosos a la hora de plasmar las dependencias entre los cálculos de las diversas sucesiones implicadas, x_n, y_n y π_n.

```
from math import sqrt

def next_x(x):
    return 0.5*(sqrt(x)+1/sqrt(x))

def next_y(x,y):
    return (y*sqrt(x) + 1/sqrt(x))/(y+1)

def next_pi(x, y, pi):
    return pi * ((x+1)/(y+1))

def borewein(n):
    x = next_x(sqrt(2)) #x=x_1
    y = sqrt(sqrt(2)) #y=y_1
    pi = 2 + sqrt(2) #pi=π_0
    i = 0
    while i<n:
        # INV: pi = π_i, x=x_{i+1}, y=y_{i+1}
        pi = next_pi(x, y, pi) #pi=π_{i+1}
        y = next_y(x,y) #y=y_{i+2}
        x = next_x(x) #x=x_{i+2}
        i += 1
    return pi
```

▷ 7. Calculando sobre tablas

Especifica y diseñar algoritmos para:

- Llenar una tabla con datos aleatorios.
- Mostrar los valores de una tabla por pantalla.
- Calcular la media y la desviación típica de los elementos de la tabla.
- Calcular la cantidad de números pares.
- Calcular la cantidad de cuadrados perfectos.
- Calcular la cantidad de los números cuyo logaritmo en base 2 sea menor que otro número dado `log` > 0.
- Determinar si la tabla está ordenada de menor a mayor.
- Contar el número de *picos* que contiene. Un número es un pico si es estrictamente mayor que los dos que tiene a su lado.
- Calcular la cantidad de números que sean potencia de dos.
- Decidir si la tabla es *palíndroma*, es decir, puede leerse igual de izquierda a derecha que de derecha a izquierda.
- Calcular la cantidad de números primos que contiene.
- Si el nombre de la tabla es `t`, un algoritmo que calcule en otra tabla `s` las sumas parciales de los elementos de la primera:

$$s[1] = t[1],\ s[2] = t[1] + t[2],\ s[3] = t[1] + t[2] + t[3],\ \cdots \quad s[i] = \sum_{k=1}^{i} t[k]$$

Solución

- Para contar los número que verifican la propiedad de ser cuadrados perfectos tenemos que recorrer todo el vector, para ello lo más sencillo es utilizar un bucle `for`.

```
import math
def countPerfectSquare(t):
    count=0
    for n in t:
        sr = round(math.sqrt(n))
        if n == sr*sr:
            count +=1
    return count
```

- Para decidir si la tabla se encuentra ordenada utilizamos un bucle `while` puesto que si nos damos cuenta de que un par de posiciones no están ordenadas entonces detenemos el recorrido.