Hoja de ejercicios 4

Repaso

4 de diciembre de 2013

Informática
Año 2013/2014
Facultad de CC.
Matemáticas

▷ 1. Todas la funciones booleanas de dos argumentos

Una función booleana de dos argumentos (por ejemplo la conjunción lógica, `and`) transforma un par de valores booleanos en un valor booleano. Comprueba que hay 16 funciones distintas de este tipo. Escribe en Python la expresión que define a cada una de ellas.

Pista: Una manera muy conveniente de representar las funciones booleanas es mediante *tablas de verdad*. En la primera columna aparecen los posibles valores del primero de los argumentos. En la primera fila aparecen los posibles valores del segundo de los argumentos. El punto donde se unen una fila y una columna indica el valor de la función que se está definiendo si se aplica a argumentos que tienen como valor el primero de la fila y de la columna, respectivamente.

La siguiente figura muestra la tabla de verdad para el operador lógico de conjunción:

conjunción	verdadero	falso
verdadero	verdadero	falso
falso	falso	falso

Esta tabla aglutina todos los resultados posibles para la función booleana de conjunción cuando se le aplica a dos valores booleanos. Por ejemplo, la conjunción de verdadero y falso da como resultado falso. Para describir fácilmente todos los operadores booleanos utiliza la propiedad de que cada operador tiene una tabla de verdad diferente.

Solución

Toda función booleana, f, de dos argumentos puede definirse mediante una tabla de verdad del estilo:

f	verdadero	falso
verdadero		
falso		

Donde el valor de $f(p, q)$ viene determinado por el valor de la tabla en la fila p y la columna q. Puesto que en cada una de los cuatro huecos que define la tabla puede ir el valor verdadero o falso, el número total de posibles funciones es 2^4.

Mostramos, abreviando verdadero por v y falso por f, las 16 posibles tablas de verdad:

f_1	v	f
v	v	v
f	v	v

f_2	v	f
v	v	v
f	v	f

f_3	v	f
v	v	v
f	f	v

f_4	v	f
v	v	v
f	f	f

f_5	v	f
v	v	f
f	v	v

f_6	v	f
v	v	f
f	v	f

f_7	v	f
v	v	f
f	f	v

f_8	v	f
v	v	f
f	f	f

f_9	v	f
v	f	v
f	v	v

f_{10}	v	f
v	f	v
f	v	f

f_{11}	v	f
v	f	v
f	f	v

f_{12}	v	f
v	f	v
f	f	f

f_{13}	v	f
v	f	f
f	v	v

f_{14}	v	f
v	f	f
f	v	f

f_{15}	v	f
v	f	f
f	f	v

f_{16}	v	f
v	f	f
f	f	f

Por ejemplo la tabla para el operador de conjunción es la f_8 y la tabla para la implicación es la f_5. El valor de cada función se define a partir de los valores de dos variables booleanas, `p,q` Para cada una de las tablas damos una expresión que la calcula, muchas otras expresiones son también posibles:

```
def f_0(p,q):
    return True

def f_1(p,q):
    return p or q

def f_2(p,q):
    return p or (not q)

def f_3(p,q):
    return p

def f_4(p,q):
    return (not p) or q

def f_5(p,q):
    return q

def f_6(p,q):
    return not ((p or q) and not(p and q))

def f_7(p,q):
    return p and q

def f_8(p,q):
    return (not p) or (not q)

def f_9(p,q):
    return (p or q) and not(p and q)

def f_10(p,q):
    return not q

def f_11(p,q):
    return p and (not q)
```

```
def f_12(p,q):
    return not p

def f_13(p,q):
    return (not p) and q

def f_14(p,q):
    return (not p) and (not q)

def f_15(p,q):
    return False
```

▷ **2. Expresiones relacionadas con una formación rectangular de números**
Considera la siguiente formación de números

$$\begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{array}$$

Para formaciones de este estilo con N columnas, escribe expresiones para los siguientes cálculos y situaciones:

Cálculo del número El número que aparece en la fila i y columna j ($i \geq 1$ y $1 \leq j \leq N$).

Fila de un número La fila que ocupa un número $p \geq 1$.

Columna de un número La columna que ocupa un número $p \geq 1$.

¿Están en la misma fila? Cuándo dos números p y q ($p, q \geq 1$) están en la misma fila.

¿Están en la misma columna? Cuándo dos números p y q ($p, q \geq 1$) están en la misma columna.

¿Están en la misma diagonal? Cuándo dos números p y q ($p, q \geq 1$) están en la misma diagonal.

Solución

Cálculo del número El primer número de la fila i es $N(i-1)+1$, el segundo será $N(i-1)+2$, y el j-ésimo será $N(i-1)+j$. El número p buscado se calculará de la siguiente forma:

```
p = (i-1)*n + j
```

Fila de un número Como ya hemos visto, los elementos en la fila i son de la forma $N(i-1)+j$, por lo que para calcular la fila será necesario calcular el cociente de la división entre $N(i-1)+j$ y N. Pero como $1 \leq j \leq N$, esta división daría un resultado incorrecto cuando $j = N$, restaremos 1 antes de hacer la división; para calcular la fila será necesario por tanto sumar 1. Si el número del que se desea calcular la fila es `p`, la expresión es:

```
fila = (p-1)/N + 1
```

Columna de un número Puesto que los elementos en la columna j son de la forma $N * (i-1)+j$, para calcular la columna será necesarior recurrir a la operación de resto. Si calculásemos directamente el módulo N a ese número, nos daría como resultado una columna entre 0 y $N-1$, donde el valor 0 se correspondería con la columna N. Para evitar ese problema, podemos restar uno antes de hacer la operación de módulo y, después, volver a sumar 1, obteniendo así directamente una columna entre 1 y N.

```
columna = (p-1)%N + 1
```

¿Están en la misma fila? Teniendo en cuenta el cálculo de la fila, la solución es muy sencilla, basta con comprobar que el resultado de ambos calculos coincide.

```
mismaFila = (p-1)/N + 1 == (q-1)/N + 1
```

Donde `p` y `q` son los valores de los números que tenemos que comprobar si pertenecen a la misma fila. El resultado obviamente es un valor booleano.

¿Están en la misma columna? Teniendo en cuenta el cálculo de la columna de un número, la expresión buscada es

```
mismaColumna = (p-1)%n + 1 == (q-1)%n + 1
```

Donde `p` y `q` son los valores de los números que tenemos que comprobar si pertenecen a la misma columna. El resultado obviamente es un valor booleano.

¿Están en la misma diagonal? Procederemos por partes. En primer lugar, dos números estarán en la misma diagonal de izquierda a derecha si la diferencia entre la fila y la columna coincide tanto en p como en q; es decir,

```
mismaDiagonalIzDer = (filaP-columnaP) == (filaQ-columnaQ)
```

siendo `columnaP`, `columnaQ`, `filaP` y `filaQ` las calculadas en los apartados anteriores. Análogamente, dos números estarán en la misma diagonal de derecha a izquierda si coincide la suma de la fila y la columna tanto en p como en q:

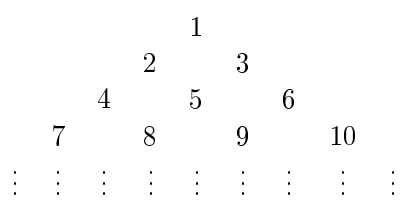
```
mismaDiagonalDerIz = (filaP+columnaP) == (filaQ+columnaQ)
```

Por último, los números están en la misma diagonal si están en una de las dos diagonales anteriores:

```
mismaDiagonal = mismaDiagonalIzDer or mismaDiagonalDerIz
```

▷ 3. Expresiones relacionadas con una formación triangular de números

Considera la siguiente formación de números:



Número en posición Calcula el número que aparece en la posición j de la fila i ($i \geq 1$ y ¿qué condición debe cumplir j?).

Fila de un número Calcula la fila que ocupa un número p.

¿Están en la misma fila? Indica si dos números p y q están en la misma fila.

¿Están en la misma diagonal? Indica si dos números p y q están en la misma diagonal.

Solución

Número en posición En primer lugar, observemos que en la fila n existen exactamente n números por lo que j deberá cumplir esta acotación:

$$1 \leq j \leq i$$

El último número de la fila n es el n-ésimo número triangular, a saber $\frac{n(n+1)}{2}$ por lo que el número buscado será $j + \frac{(i-1)i}{2}$. Además, todas las divisiones que aparecen tienen resultado entero. Por tanto el número p se calcula así:

```
p = j + ((i-1)*i)/2;
```

Fila de un número Puesto que el último número de la fila n es el n-ésimo número triangular y todo número natural está entre dos números triangulares, será necesario calcular el único número n de forma que

$$\frac{(n-1)n}{2} < p \leq \frac{n(n+1)}{2}$$

de lo que obtenemos lo siguiente:

$$\frac{(n-1)^2}{2} < p < \frac{(n+1)^2}{2}, \quad (n-1)^2 < 2p < (n+1)^2, \quad (n-1) \leq \lfloor \sqrt{2p} \rfloor < n+1.$$

Llamando $k = \lfloor \sqrt{2p} \rfloor$ tenemos que $n = k+1$ o $n = k$. ¿Cómo resolvemos esa disyunción? Basta con fijarnos en lo siguiente:

$$\begin{array}{ll} \frac{k(k-1)}{2} < p \leq \frac{k(k+1)}{2} & \text{si } n = k \\ \frac{k(k+1)}{2} < p \leq \frac{(k+1)(k+1)}{2} & \text{si } n = k+1 \end{array}$$

Entonces podemos asegurar que $n = k+1$ si $k(k+1) < 2p$ y $n = k$ en caso contrario. Por tanto la solución será ésta:

```
k = int(math.sqrt(2*p));
fila = k+1 if  k*(k+1) < 2*p else k
```

¿Están en la misma fila? Para ver si los elementos p y q están en la misma fila, habrá que tomar la expresión del apartado anterior para cada número y comprobar su igualdad. Lo normal será contar con una función que calcule la fila de un número utilizando el código del apartado anterior

```
def fila(numero):
    k = int(math.sqrt(2*numero));
    fila = k+1 if  k*(k+1) < 2*numero else k
```

```
    return fila
```

Una vez que tenemos esta función, definir la igualdad de fila de dos valores p y q es trivial:

```
mismaFila = fila(p) == fila(q);
```

¿Están en la misma diagonal? Procederemos por partes: primero escribiremos expresiones para ver si dos números están en la misma diagonal de izquierda a derecha, despúes haremos lo mismo para la otra diagonal y, por último, daremos solución al apartado combinando las dos soluciones anteriores.

Para ver si los dos números están en la misma diagonal de izquierda a derecha conviene distribuir los elementos de otra forma:

```
1
2  3
4  5  6
7  8  9  10
```

Las diagonales de izquierda a derecha según la distribución original siguen siendo diagonales según la distribución actual; salvo que ahora se observa que dos elementos están la misma diagonal si la diferencia entre filas coincide con la diferencia entre columnas (en la nueva disposición). Si `filaP` es la fila del número p, el número triangular anterior a p será `filaP`(`filaP`-1) partido por 2 y por tanto la columna de p será p menos `filaP`(`filaP`-1) dividido entre 2. Si definimos una función que nos devuelve la columna que ocupa un valor en la nueva distribución,

```
def columna(numero):
    fp = fila(numero);
    return numero - (fp*(fp-1))/2
```

decidir si p y q están en la misma diagonal de izquierda a derecha lo decide la expresión:

```
mismaDiagonalIzDer = columna(p)-fila(p) == columna(q)-fila(q)
```

Para determinar si dos elementos están en la otra diagonal observamos que, si distribuimos los elementos como antes, las diagonales de derecha a izquierda se transforman en una columna, y basta entonces con comprobar las columnas:

```
mismaDiagonalDerIz = columna(p) == columna(q)
```

Por último, para averiguar si dos números están en la misma diagonal, es suficiente con calcular la disyunción de las dos expresiones anteriores:

```
mismaDiagonal = mismaDiagonalDerIz or mismaDiagonalIzDer
```

▷ 4. Congruencia de Zeller

Calcular el día de la semana al que pertenece una determinada fecha no es tarea fácil. Afortunadamente, Christian Zeller (1824–1899) propuso en 1882 una expresión que permite calcularlo fácilmente.

Un poco de historia El calendario que actualmente utilizamos se conoce como Calendario Gregoriano y fue introducido en el año 1582 por el Papa Gregorio XIII (1572–1585). De esta forma se sustituía el calendario Juliano y se perfeccionaba así el ajuste entre el calendario y el año solar.

El Calendario Gregoriano establece 97 años bisiestos cada 400 años y, para ajustar el desfase desde el año 45 a.C. y el año de su aplicación, se adelantaron 10 días: el 5 de octubre de 1582 se contó como el 15 de octubre.

El calendario se adoptó inmediatamente en España, Italia, Polonia y Portugal, otros países fueron adoptándolo más tarde. Por ejemplo, este calendario no se utilizó en Gran Bretaña hasta 1752, en Rusia hasta 1918 y en Turquía hasta 1927.

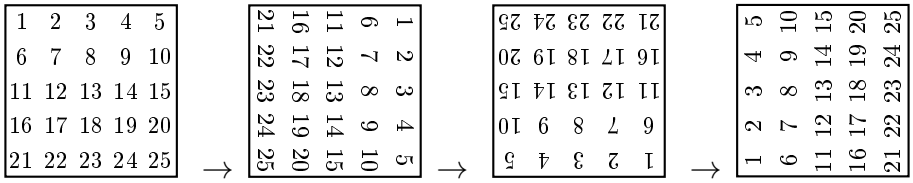
Solución

```
def day_week(day, month, year):
    """
    This function returns the day of the week of the corresponding day/month/year.
    The day must be posterior to 15/oct/1582. It returns an integer indicating:
    the day:
    1 -> Monday
    2 -> Tuesday
    3 -> Wednesday
    4 -> Thursday
    5 -> Friday
    6 -> Saturday
    7 -> Sunday

    @type day: int
    @type month: int
    @type year: int
    @rtype: int
    """
    if month < 3:
        month += 12
        year -= 1
    zeller = (day + (13 * year - 27)/5 + year + year/4
              - year/100 + year/400) % 7
    return ( (zeller+5)%7 ) + 1
```

▷ **5. ¿Pueden las matrices dar volteretas?**

(*) Pues parece que sí:



Voltereta Suponiendo que la matriz es cuadrada, de tamaño N, y numerada como en la figura, se pide una función que transforme un número en el que lo desplazará cuando la matriz dé una voltereta.

Ejemplo Consideremos una matriz de tamaño 5, y el número 12, que ocupa la casilla $(3, 2)$. El número que desplaza al 12 cuando se da un giro es el 18; el número que desplaza al 18 es el 14; y el que desplaza al 14 es el 8.

$$12 \to 18 \to 14 \to 8$$

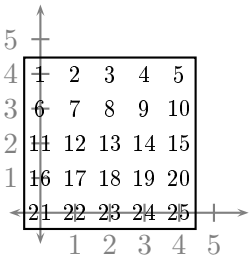
Por tanto, la función que buscamos deberá convertir el 12 en 18, el 18 en 14, etc.

Vuelta completa Comprueba que, aplicando la función cuatro veces sobre cualquier elemento de la matriz, dicho elemento no varía.

Solución

Voltereta Si las matrices dan volteretas, como sugiere el enunciado, para encontrar la solución a este problema tenemos que poder razonar con giros. Para ello, recurrimos a nuestros conocimientos de geometría básica.

Imaginemos que la matriz está *apoyada* sobre los ejes de coordenadas del plano.

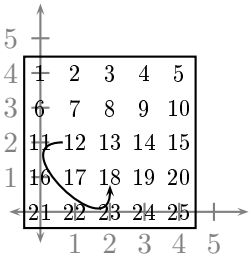


Entonces, cada número que pertenece a la matriz puede identificarse por las coordenadas del plano que ocupa. El 21 tendría coordenadas $(0, 0)$, el 10 coordenas $(4, 3)$ y el 12 estaría en las coordenadas $(1, 2)$.

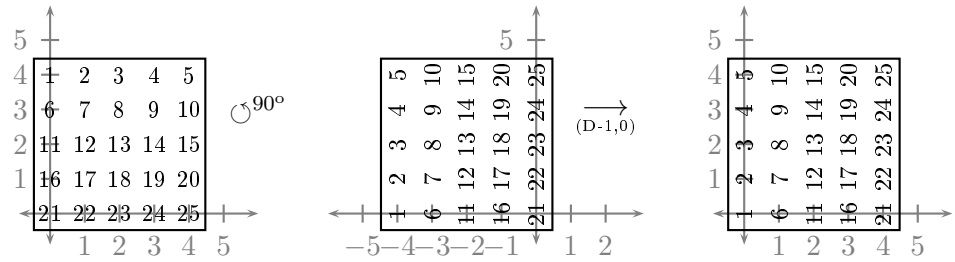
Encontrar las coordenadas que corresponden a un número no es misión imposible, sólo requiere un poco de atención. La coordenada x tiene que ver con el resto de la división del número que consideremos por la dimensión de la matriz. En efecto, si n es el número y D es la dimensión, entonces $(n-1)$ mód D nos da la coordenada x. En nuestro ejemplo concreto, con $D = 5$, la coordena x para $n = 12$ es $(12 - 1)$ mód 5 que es 1, para $n = 9$ es $(9 - 1)$ mód 5 que es 3.

De forma similar, la coordenada y está relacionada con el cociente de la división del número por la dimensión. Como antes, si n es el número y D es la dimensión, con la expresión $(D - 1) - ((n - 1)$ div $D)$ se obtiene la coordenada y que identifica la posición de n en la matriz. Si $D = 5$, la coordenada y para $n = 12$ es $(5 - 1) - ((12 - 1)$ div $5)$ que es 2, para $n = 9$ es $(5 - 1) - ((9 - 1)$ div $5)$ que da como resultado 3.

El poder pensar en los números de la matriz como coordenadas del plano es muy aducado para hacerlos girar. En efecto, si como indica el enunciado, queremos saber que número sustituirá al 12 en la matriz al girarla en el sentido de las agujas del reloj, una forma de encontrarlo es girar la matriz al revés, es decir, girarla en el sentido contrario a las agujas del reloj y ver cual es la posición que ocupa el 12 después de dicho giro.



Girar la matriz un cuarto de vuelta en sentido contrario a las agujas del reloj equivale a rotar en torno al origen de coordenadas 90º y luego trasladar utilizando el vector de traslación $(D - 1, 0)$.



Tras este giro, las coordenadas que pasará a ocupar el número que estamos considerando, en este caso el 12, serán aquellas que pertenecen al número que ocupará la posición del 12 tras un giro de la voltereta, en el sentido de las agujas del reloj.

Encontrar la trasformación de coordenadas que suponen las operaciones que muestra la figura anterior es sencillo si se hace paso a paso. La rotación de 90^o trasforma las coordenadas (x, y) en las coordenadas $(-y, x)$ y sobre estas coordenas realizamos la traslación de vector $(D-1, 0)$ y obtenemos las coordenadas $(D-1-y, x)$.

En el caso concreto que estamos considerando de $D = 5$ y $n = 12$, las coordenadas de 12 son $(1, 2)$, la rotación las trasforma en $(-2, 1)$ y la traslación en $(2, 1)$. El número 9 tiene las coordenas $(3, 3)$ que se trasforman en las coordenadas $(5-1-3, 3)$, es decir $(1, 3)$.

Para completar el enunciado queda únicamente por determinar que número corresponde a dichas coordenadas trasformadas. Éste número será el que al girar para dar una voltereta (en el sentido de las agujas del reloj) ocupará la posición inicial que habíamos considerado. El de recuperar el número a partir de las coordendas es el *inverso* a encontrar las coordenadas que corresponden a un número en una matriz. Recordamos la forma de obtener las coordenadas de un número n en una matriz de dimensión D:

$$n \longrightarrow ((n-1) \text{ mód } D, (D-1) - (n-1) \text{ div } D)$$

Si lo que conocemos son las coordenas (x, y) y sabemos que se verifican las siguientes ecuaciones:

$$x = (n-1) \text{ mód } D$$
$$y = (D-1) - (n-1) \text{ div } D$$

Se trata de despejar n en función de x e y, por ejemplo

$$x - D \cdot y = -D \cdot (D-1) + D \cdot ((n-1) \text{ div } D) + (n-1) \text{ mód } D)$$

como el valor de $D \cdot ((n-1) \text{ div } D) + (n-1) \text{ mód } D$ es precisamente $n-1$, simplificamos

$$x - D \cdot y = -D \cdot (D-1) + n - 1$$

con lo que obtenemos

$$D \cdot (D-1-y) + x + 1 = n$$

Resumiendo, si $D = 5$, el número 12 ocupa las coordenadas $(1, 2)$, al girar al revés de lo que proponen las volteretas dichas coordenadas pasarían a ser $(2, 1)$ y esta posición corresponde al número $5 \cdot ((5-1-1) + 2 + 1$ que es 18. Por tanto el 18 es el número que pasará a ocupar la posición del 12.

Otro ejemplo, si $D = 5$, el número 9 ocupa las coordenadas $(3, 3)$ que al girar al revés se trasforman en las coordenadas $(1, 3)$ que corresponden al número $5 \cdot ((5-1-3) + 1 + 1$

que es 7. Por tanto, el número que al dar un giro de voltereta pasará a ocupar la posición del 9 es el 7.

El código del siguiente programa, por claridad, muestra los pasos intermedios, aunque pueden unificarse en una única expresión.

```
def flip(n, dim):
    """
    This function returns the number that appears after a square
    dim-dimensional matrix performs a flip in the position where
    originaly the number n was located
    @type n: integer
    @param n: M{dim>=n>=1}
    @type dim: integer
    @param dim: M{dim>=0}
    @rtype: integer
    """

    # The coordinates of number n
    x = (n-1)%dim
    y = (dim-1) - (n-1)/dim

    #The coordinates of the number that will be in the postition of n
    x_flipped = dim - 1 - y
    y_flipped = x

    return  (dim-1-y_flipped)*dim + x_flipped+1
```

▷ 6. Resolución de un sistema de ecuaciones

Desarrolla un programa que estudie la naturaleza de las soluciones del sistema de ecuaciones siguiente,

$$\begin{array}{rcrcl} a_1x & + & b_1y & = & c_1 \\ a_2x & + & b_2y & = & c_2 \end{array}$$

determinando si es un sistema incompatible, compatible indeterminado o compatible determinado, presentando las soluciones correspondiente en este último caso.

Pista: Observa que el problema planteado requiere averiguar el rango de las matrices de coeficientes (M) y ampliada (A)

$$M = \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \end{pmatrix} \qquad A = \begin{pmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \end{pmatrix}$$

Además, para el cálculo de esos rangos puede ser útil tener presentes algunas de las siguientes observaciones:

- El rango de una matriz es cero si y sólo si tiene todos sus coeficientes nulos.
- El rango de una matriz cuadrada $M_{n\times n}(\mathbb{R})$ es n si y sólo si su determinante es no nulo.
- Para las matrices M y A descritas, $rg(M) \leq rg(A) \leq rg(M) + 1$.
- Si en la matriz A descrita, los coeficientes de la primera fila son proporcionales a los de la segunda, se tiene que $rg(A) \leq 1$.

En resumen, las posibilidades son las siguientes:

$$rg(M) = \begin{cases} 0 \Rightarrow \text{si } rg(A) = \begin{cases} 0 \Rightarrow \textit{Sist. Compatible Indeterminado} & (\infty \text{ soluciones}) \\ 1 \Rightarrow \textit{Sist. Incompatible} & (\nexists \text{ solución}) \end{cases} \\ 1 \Rightarrow \text{si } rg(A) = \begin{cases} 1 \Rightarrow \textit{Sist. Compatible Indeterminado} & (\infty \text{ soluciones}) \\ 2 \Rightarrow \textit{Sist. Incompatible} & (\nexists \text{ solución}) \end{cases} \\ 2 \Rightarrow \textit{Sistema Compatible Determinado} \quad (\exists! \text{ solución}) \end{cases}$$

Un poco de historia La regla de Cramer se llama así en honor al matemático suizo Gabriel Cramer (1704–1752). A los 18 años ya era doctor y a los 20 profesor de universidad. Cramer introdujo una gran innovación: utilizó el francés, en lugar del latín, para dar sus clases, en contra de lo que era habitual en aquella época.

Solución

La solución a este problema va a ser una función que devuelva una tupla de tres elementos. El primer elemento va a ser un entero según se indica en la siguiente tabla

0	si el sistema es compatible indeterminado
1	si el sistema es incompatible
2	si el sitema es compatible determinado.

Para ello definiremos 3 constantes

```
CONSISTENT_DEPENDENT = 0
NON_CONSISTENT = 1
CONSISTENT_INDEPENDENT = 2
```

Los dos elementos siguientes de la tupla tendrán sentido si el sistema es compatible determinado y serán las soluciones del sistema. Si el sistema no es compatible determinado devolverá `None`.

En primer lugar para poder hacer el análisis de rangos de las matrices usaremos una función auxiliar que calcula el determinante de una matriz cuadrada de 2x2

```
def det(a,b,c,d):
    """
    This function computes the determinant
    of the matrix

    |a  b|
    |c  d|
    """
    return a*d - c*b
```

Con todo esto la función que calcula las soluciones del sistema queda como sigue:

```
def solve(a,b,c,d,e,f):
    """
    Solves the linear equation represented by de matrix

    [a b| e]
    [c d| f]

    M = [a b]
        [a d]

    A = [a b e]
        [c d f]
```

```
    This function returns a 3 element tuple, the first element in the
    tuple is an integer:
    CONSISTENT_DEPENDENT -> dependent consistent system
    NON_CONSISTENT -> non consistent sistem
    CONSISTENT_INDEPENDENT -> consistent independent system

    If the system is consistent and independent,
    then the next two elements of the tuple are the unique solution of
    the system. Otherwise the next two elements are None.
    """
    if a==b and b==c and c==d and d==0: #rank of M = 0
        if e==0 and f==0: # rank of A = 0
            solution = CONSISTENT_DEPENDENT, None, None
        else: # rank A =1
            solution = NON_CONSISTENT, None, None
    else:
        detM = det(a,b,c,d)
        if detM==0: # rank of M = 1
            if det(a, e, c, f) == 0: # rank of A = 1
                solution =  CONSISTENT_DEPENDENT, None, None
            else:
                solution =  NON_CONSISTENT, None, None
        else: #rank of M = 2
            x = float(det(e, b, f, d)) / detM
            y = float(det(a, e, c, f)) / detM
            solution =  CONSISTENT_INDEPENDENT, x, y
    return solution
```

▷ 7. Fechas correctas

Se desean escribir programas que permitan determinar la correción de números que describen fechas.

Día y mes Si consideramos únicamente dos números, correspondientes a un día y un mes, deseamos hacer un programa que indique si dichos números se refieren a una fecha válida. Podemos suponer que febrero tiene siempre 28 días.

Ejemplo Los números 31 4 no corresponden a una fecha válida pues el mes de abril sólo tiene 30 días.

Día, mes y año Si consideramos tres números enteros, para día, mes y año, escribe un programa que indique si dichos números se refieren a una fecha correcta.

Ejemplo Los datos `31 12 1970` corresponden a una fecha correcta. Sin embargo, `29 2 1999` no forman una fecha correcta pues el año 1999 no es bisiesto.

Pista: Define una expresión para decidir cuando un año es o no bisiesto. Recuerda que se considera bisiesto un año cuyo número es divisible por cuatro excepto los años que son múltiplos de cien a no ser que lo sean de 400.

Solución

Día, mes y año

```
def max_days_month(month, year):
    """
    This function returns the munber of days that a given
    month has. We need the year because of the leap years.
```

```
        @type month: int
        @param month: 1<=month<=12
        @type year: int
        @param year: year>=0
        @rtype: int

        """
        if month == 2:
            max_days = 28
            if year%4 == 0 and (year%100 != 0 or year%400 == 0): # year is a leap ye
                max_days = 29
        elif month == 4 or month == 6 or month == 9 or month == 11:
            max_days = 30
        else:
            max_days = 31
        return max_days

    def is_correct_date(day, month, year):
        """
        This function indicates if the date indicated with day month year is correct
        @type day: int
        @type month: int
        @type year: int
        @rtype: boolean
        """

        if year < 0:
            answer = False
        elif month < 1 or month > 12:
            answer = False
        else:
            answer = 1 <= day and day <= max_days_month(month, year)
        return answer
```

▷ 8. Operaciones aritméticas

Escribe un programa que nos permita introducir dos números naturales y una operación (suma, resta, multiplicación o división), y calcule el resultado de dicha operación, presentándolo en pantalla en el siguiente formato:

```
   1234
*    25
  -------
  30850
```

Solución

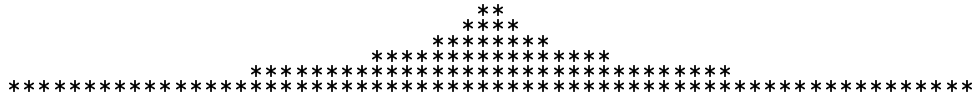
Para hacer más sencillo la presentación usaremos el descriptor de formato `% d` adecuadamente

```
n,m=int(raw_input('valor del primer operando:')),int(raw_input('valor del segundo
operando:'))
operacion=raw_input('operador:')
if operacion=='+':
    resultado=n+m
elif operacion=='-':
    resultado=n-m
```

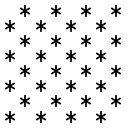
```
elif operacion=='*':
    resultado=n*m
elif operacion=='/':
    cociente=n/m
    resto=n%m
if operacion=='/':
    print '%16d'%(n),'|',m
    print 18*' '+16*'-'
    print '%16d'%(resto),' ',cociente
else:
    print '%32d'%(n)
    print  15*' '+operacion+'%16d'%(m)
    print  16*' '+16*'-'
    print  16*' '+'%16d'%(resto)
```

▷ 9. Dibujos con asteriscos

Volcán Escribe un programa que dibuje en la pantalla la siguiente figura, compuesta por líneas de 2, 4, 8, 16, 32 y 64 asteriscos respectivamente:

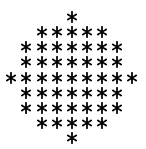


Mosaico Escribe un programa que dibuje en la pantalla la figura siguiente,



compuesta por una matriz de 8×8 caracteres, blancos o negros, como en un tablero de ajedrez.

Círculo Este dibujo es un simple círculo, del tamaño que elija el usuario, y con su borde adaptado a la rejilla discreta que nos proporciona el monitor:



Solución

Volcán Para trazar el dibujo descrito, usaremos un bucle que escribe una línea en cada vuelta,

```
i = 0
while i <= numFilas:
  [[Dibujar la fila |i|]]
```

donde el trazado de la fila i-ésima consiste en

```
[[Poner los blancos de la izquierda]]
[[Poner los asteriscos]]
[[Salto de linea]]
```

El fragmento de *poner los asteriscos* es fácil, porque en cada fila hay el doble que en la anterior, de forma que, partiendo de `numAst = 1` antes de la primera fila, basta con duplicar el número de ellos cada vez y ponerlos, uno a uno,

Mosaico La cosa consiste en escribir ocho filas:

```
y = 0
s = ""
while y<rows:
  [[anyadir a s la siguiente fila]]
```

En cada línea `i`, los caracteres `j` (de 1 a `tamanyo`) son blancos o negros dependiendo de la paridad de `i+j`. Cada línea acaba con un salto a la siguiente:

```
        x = 0
        while x<rows:
            if (x+y)%2==0:
                s += "*"
            else:
                s += " "
            x += 1
```

El programa completo queda como sigue:

```
def chess(rows):
    """
    @type rows: int
    @rtpe: string

    Example:
>>> print(chess(10))
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
* * * * *
 * * * * *
     """
    y = 0
    s = ""
    while y<rows:
        x = 0
        while x<rows:
            if (x+y)%2==0:
                s += "*"
            else:
                s += " "
            x += 1
```

```
            s += "\n"
            y += 1
        return s
```

Círculo En este caso la estrategia es un poco diferente, no nos preocuparemos de definir explícitamente los asteriscos que hay que escribir. Mejor, utilizaremos las fórmulas de la geometría analítica para decidir si un determinado punto pertenece o no a un círculo, de esa manera podremos *interpolar* el círculo.

```
def circle(radius):
    """
    @type radius: int
    @rtpe: string

    Example
>>> print circle(10)
..........*..........
......*********......
....**********....
...**********...
..**********..
..**********..
.**********.
.**********.
.**********.
.**********.
**********
.**********.
.**********.
.**********.
.**********.
..**********..
..**********..
...**********...
....**********....
......*********......
..........*..........
    """
    s = ""
    y = -radius
    while y<=radius:
        x = -radius
        while x<=radius:
            if x*x+y*y <= radius*radius:
                s + ="*"
            else:
                s + ="."
            x += 1
        s += "\n"
        y += 1
    return s
```

▷ 10. Juego de adivinación

Consideremos el siguiente juego entre los jugadores A (adivino) y P (pensador): P piensa un número comprendido entre 1 y N (digamos 1000, por ejemplo), y A trata de adivinarlo,

mediante tanteos sucesivos, hasta dar con él. Por cada tanteo de A, P da una respuesta orientativa de entre las siguientes:

```
Fallaste. El número pensado es menor que el tuyo.
Fallaste. Mi número es mayor.
Acertaste al fin.
```

Naturalmente, caben diferentes estrategias para el adivino:

- Una posibilidad, si el adivino no tiene prisa en acabar, consiste en tantear números sucesivamente: primero el 1, después el 2, etc. hasta acertar.
- Otra estrategia, sin duda la más astuta, consiste en tantear el número central de los posibles de modo que, al fallar, se limiten las posibilidades a la mitad (por eso se llama *bipartición* a este modo de tantear).
- También es posible jugar caprichosamente, tanteando un número al azar entre los posibles. Al igual que la anterior, esta estrategia también reduce el intervalo de posibilidades sensiblemente.

Se plantea desarrollar tres programas independientes: uno deberá desempeñar el papel del pensador; otro el del adivino (usando la estrategia de bipartición), y un tercero deberá efectuar la simulación del juego, asumiendo ambos papeles (y donde el adivino efectúa los tanteos a capricho).

Solución

Sin duda lo más interesante que propone el enunciado es resolver el problema de encontrar un número en un intervalo contando con la información de si el número buscado es mayor o menor que el propuesto.

Utilizaremos la variable `li` para indicar el límite inferior del intervalo de búsqueda y `ls` para indicar el límite superior. Estas variables serán inicializadas convenientemente.

En principio, el programa sigue un esquema de búsqueda, se tantean números hasta que se acierta, cada vez que no se acierta se modifican los límites de búsqueda:

```
while (!encontrado) {
  [[proponer un candidato a numero pensado]]
  [[si no se acierta actualizar los limites de busqueda]]
}
```

Proponer un candidato es sencillo, basta encontrar el punto medio `pm` del intervalo de búsqueda (`pm = (li + ls) / 2;`). Por otro lado, determinar el acierto o actualizar los límites no tiene grandes misterios:

```
switch (c) {
case '+' :
  li = pm + 1;
  break;
case '-' :
  ls = pm - 1;
  break;
case 's' :
  encontrado = true;
  break;
}
```

Un posible problema de nuestra solución es que nos *fiamos* del usuario y esperamos que éste diga la verdad sobre el número que le proponemos. No cuesta mucho mejorar el programa para que podamos controlar en cierta medida a los mentirosos compulsivos o a aquellos que tratan de ser más *listos* que nuestro programa. Podemos saber si alguien está jugando sucio si en algún momento el intervalo de búsqueda deja de tener sentido, es decir, si el límite inferior supera al límite superior. Esta nueva condición puede ser añadida al bucle:

```
while ( (li<=ls) && (!encontrado) ) {
  [[proponer un candidato a numero pensado]]
  [[si no se acierta actualizar los limites de busqueda]]
}
```

El siguiente código aglutina todas las consideraciones anteriores, incluida la detección de mentirosos.

```
#include <iostream>
using namespace std;

const int MIN=1;
const int MAX=100;

main() {
  cout << "Piensa un numero entre "<< MIN << " y " << MAX << endl;
  int cont = 0;
  bool encontrado = false;
  int li = MIN, ls = MAX;

  while ( (li<=ls) && (! encontrado) ) {
    cout << "buscando entre " << li << " -- " << ls << endl;
    int pm = (li + ls) / 2;
    cout << "Es tu numero el " << pm << " (- es menor, + es mayor, s si)";
    char c;
    cin >> c;
    switch (c) {
    case '+' :
      li = pm + 1;
      break;
    case '-' :
      ls = pm - 1;
      break;
    case 's' :
      encontrado = true;
      break;
    }
    cont ++;
  }
  if (encontrado) cout << "Acerte en "<<cont<<" intentos!!" << endl;
  else cout << "Me has mentido" << endl;
}
```

▷ 11. Ecuación diofántica

Se llama ecuación *diofántica* a cualquier ecuación algebraica, generalmente de varias variables, planteada sobre el conjunto de los números enteros $\mathbb{Z}$ o los números naturales $\mathbb{N}$.

- Escribe un programa que calcule las maneras diferentes en que un número natural n se puede escribir como suma de otros dos números naturales. Es decir, que calcule las

soluciones de la ecuación diofántica $x + y = n$.

- Escribe un programa que calcule las maneras diferentes en que un número natural n se puede escribir como producto de dos números naturales. Es decir, que calcule las soluciones de la ecuación diofántica $xy = n$.
- Escribe un programa que, dado un número natural n, calcule la cantidad de soluciones de la ecuación diofántica: $x^2 - y^2 = n$

Pista: Ten en cuenta que toda solución de la anterior ecuación nos produce una factorización del entero n:

$$n = (x + y)(x - y)$$

Ya sólo nos queda dilucidar cuáles de esas factorizaciones producen una solución de la ecuación.

Un poco de historia La palabra *diofántica* hace referencia al matemático griego del siglo III de nuestra era Diofanto de Alejandría.
(http://es.wikipedia.org/wiki/Diofanto_de_Alejandr%C3%ADa)

▷ 12. Un bonito triángulo

Anidando bucles y con los dígitos $\{0, \ldots, 9\}$ se pueden escribir triángulos tan interesantes como el siguiente:

```
1
232
34543
4567654
567898765
67890109876
7890123210987
890123454321098
90123456765432109
0123456789876543210
```

¿Te atreves?

Solución

Vamos a hacer una función que devuelva el triángulo. Esencialmente la función será un bucle que escriba cada línea

```
def triangle(n):
  s += ""
  i = 1
  while i<n:
    current_line = [[ escribe la linea actual ]]
    s += current_line +"\n"
    i += 1
```

Para poder escribir la línea actual vamos a a realizar otra función. Esa función debe de conocer cuál es el número total de líneas y la línea actual en la que nos encontramos. La vamos a hacer en tres partes, primero que añada los blancos necesarios al comienzo de la línea, luego que escriba la secuencia de números creciente y luego la decreciente

```
def line(total_lines, current_line):
  s = [[ blancos iniciales ]]
  [[ anyadir a s la secuencia creciente ]]
  [[ anyadir a s la secuencia decreciente ]]
```

El número de blancos iniciales coincide con la expresión `total_lines-current_line`. Para escribir las secuencias creciente y decreciente haremos un bucle para cada una. haremos una bucle. Por ejemplo para la 11ª fila primero escribiremos `1234567890` y luego `10987654321`. Llevaremos una variable `i` para saber cuántos dígitos tenemos que escribir y otra variable `d` que será el dígito actual que tenemos que escribir. La primera siempre se incrementará en 1 unidad, mientras que con la segunda hay que incrementarla en el primer bucle y disminuirla en el segundo. Además hay que tener cuidado de que la variable `d` siempre esté entre 0 y 9. Con todo el cuerpo de la función `line` queda como sigue:

```
    s = " "*(total_lines-current_line)
    i = 1
    d = current_line % 10 # the digit to be printed
    while i<=current_line-1:
        s += str(d)
        d = (d + 1) % 10
        i += 1

    i = 1
    while i<=current_line:
        s += str(d)
        d = (d - 1) % 10
        i += 1
    return s
```

Juntándo todo, nos queda el programa

```
#!/bin/python
# $Id$

def line(total_lines, current_line):
    """
    This function returns a string containing the current_line of a
    total_line isosceles triangle. It also returns the current digit
    to be printed.

    @type total_lines: int
    @param total_lines: total_lines>0
    @type current_line: int
    @param current_line: 0<current_line<=total_lines
    @rtype: string
    @return: the string with the line
    """
    s = " "*(total_lines-current_line)
    i = 1
    d = current_line % 10 # the digit to be printed
    while i<=current_line-1:
        s += str(d)
        d = (d + 1) % 10
        i += 1

    i = 1
```

```
    while i<=current_line:
        s += str(d)
        d = (d - 1) % 10
        i += 1
    return s

def triangle(n):
    """
    This function returns a string containig a n-line isosceles triangle
    with digits from 0 to 9
    @type n: int
    @param n: n>0
    @rtype: string
    """

    s = ""
    i = 1
    while i<=n:
        current_line = line(n,i)
        s += current_line+"\n"
        i += 1
    return s

def main(n):
    s = triangle(n)
    print s

if __name__=="__main__":
    main(10)
```

▷ 13. Espectro natural

(*) El *espectro natural* de una circunferencia de radio r es el conjunto de puntos con coordenadas naturales (n, m) tales que $n^2 + m^2 = r^2$. El método obvio para calcular el espectro necesitaría dos bucles anidados en donde se irían explorando las abcisas y ordenadas de los puntos desde 0 a r. Afortunadamente existe un método más eficiente, que además aprovecha la simetría del problema: si (n, m) está en el espectro también está (m, n). Definimos

$$B(x, y) = \{(n, m) \mid n^2 + m^2 = r^2 \wedge x \leq n \leq m \leq y\}$$

Obviamente el espectro es $B(0, r) \cup \{(m, n) \mid (n, m) \in B(0, r)\}$ que a su vez se puede calcular usando las siguientes reglas:

$$B(x, y) = \begin{cases} B(x+1, y), & \text{si } x^2 + y^2 < r^2 \\ \{(x, y)\} \cup B(x+1, y-1), & \text{si } x^2 + y^2 = r^2 \\ B(x, y-1), & \text{si } x^2 + y^2 > r^2 \end{cases}$$

Escribe un programa iterativo que calcule el espectro natural de una circunferencia de radio r utilizando exclusivamente las reglas dadas en el párrafo anterior. El programa pedirá el número entero r y a continuación pasará a escribir los puntos del espectro en la pantalla.

Indica el número de pasos que lleva calcular el espectro en relación al radio r.

Solución

El espectro natural de una circunferencia de radio r, centrada en el origen, consiste en los puntos del plano con coordenadas naturales que pertenecen a dicha circunferencia. La figura 1

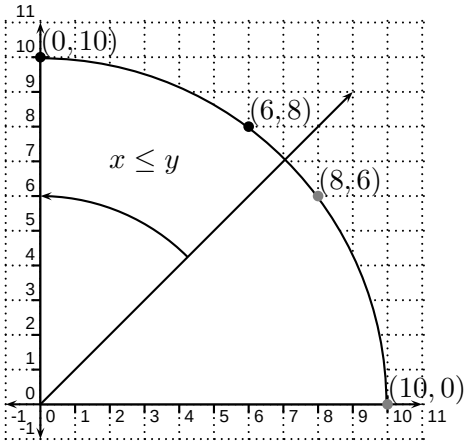


Figura 1: Puntos del espectro natural de una circunferencia de radio 10

muestra los puntos del espectro natural de la circunferencia de radio 10. Debido a la simetría del problema, basta con que busquemos los puntos que verifiquen que su coordenada x es menor o igual que su coordenada y. Si definimos el conjunto de puntos

$$B(x, y) = \{(n, m) \mid n^2 + m^2 = r^2 \wedge x \leq n \leq m \leq y\}$$

calcular el espectro de una circunferencia se reduce a encontrar los puntos que pertenecen $B(0, r)$ y para ello se dan las siguientes reglas:

$$B(x, y) = \begin{cases} B(x+1, y), & \text{si } x^2 + y^2 < r^2 \\ \{(x, y)\} \cup B(x+1, y-1), & \text{si } x^2 + y^2 = r^2 \\ B(x, y-1), & \text{si } x^2 + y^2 > r^2 \end{cases}$$

Un programa que solucione el problema debe, por tanto, tratar de seguir estas reglas para encontrar los puntos que pertenecen a $B(0, r)$. Para ello, como indican las reglas, si estamos en el punto (x, y), dependiendo de que dicho punto esté *dentro* de la circunferencia, justo *en* ella, o *fuera* de ella, tendremos que probar con $(x+1, y)$, $(x+1, y-1)$, o con $(x, y-1)$, respectivamente.

```
"""
Programa que calcula el espectro natural de un únmero natural
"""

def espectro(n):
    x = 0
    y = n
    l = []
    while x<=y:
        dist = x*x + y*y
        if dist == n*n:
            l.append((x, y))
            l.append((y, x))
            x = x+1
            y = y-1
        elif dist > n*n:
            y = y-1
        else:
            x = x+1
    return l
```

▷ 14. Los cubos de Nicómaco

(*) Considera la siguiente propiedad descubierta por Nicómaco de Gerasa:

> *Sumando el primer impar, se obtiene el primer cubo;*
> *sumando los dos siguientes impares, se obtiene el segundo cubo;*
> *sumando los tres siguientes, se obtiene el tercer cubo, etc.*

Comprobémoslo:

$$\begin{array}{lcll} 1^3 & = & 1 & = & 1 \\ 2^3 & = & 3+5 & = & 8 \\ 3^3 & = & 7+9+11 & = & 27 \\ 4^3 & = & 13+15+17+19 & = & 64 \end{array}$$

Desarrolla un programa que escriba los n primeros cubos utilizando esta propiedad. El valor de n puede ser un valor que se pide por el teclado o lo puedes declarar en tu programa como una constante.

Un poco de historia Nicómaco de Gerasa vivió en Palestina entre los siglos I y II de nuestra era. Escribió *Arithmetike eisagoge* (Introducción a la aritmética) que fue el primer tratado en el que la aritmética se consideraba de forma independiente de la geometría. Este libro se utilizó durante más de mil años como texto básico de aritmética, a pesar de que Nicómaco no demostraba sus teoremas, sino que únicamente los ilustraba con ejemplos numéricos.

Solución

Utilizaremos la variable `impar` para que vaya tomando los valores de los números impares. Su valor inicial será 1 ya que así, al incrementarla sucesivamente en 2 unidades, se irán generando los valores impares. Debemos calcular los n primeros cubos:

Sabemos que el primer cubo se calcula sumando el primer impar; el cálculo del cubo i-ésimo, cuando $i \geq 0$, se realiza sumando los i siguientes impares; necesitaremos, por lo tanto, además de ir generando números impares consecutivos (`impar += 2`), una variable que vaya acumulando su suma (`suma += impar`) y un bucle que controle que estas operaciones se realizan el número adecuado de veces.

```
def cubos_nicomaco(n):
    """
    Esta ófuncin devuelve una lista de listas cubos.
    cubos[i] -> lista de impares del (ié+1)-simo únmero cubo, el último elemento
                de la lista es la suma de los anteriores.

    Ejemplo n=4
    [[1,1],[3,5,8],[7,9,11,27],[13,15,17,19,64]]
    """
    cubos = []
    i = 0
    impar = 1
    while i < n:
        cubos.append([])
        j = 0
        suma=0
        while j<=i:
            cubos[i].append(impar)
            suma += impar
            impar += 2
            j += 1
        cubos[i].append(suma)
```

```
        i += 1
    return cubos

def main(n=4):
    i = 0
    for l in cubos_nicomaco(n):
        print "{0}^3 = {1} = {2}".format(i, "+".join(map(str,l[:-1])), l[-1])
        i += 1
```

▷ 15. Criba de Eratóstenes

Un método para encontrar todos los números primos entre 1 y n es la *criba de Eratóstenes*. Considera una lista de números entre 2 y n. El número 2 es el primer primo, pero todos los múltiplos de 2 (4, 6, 8, ...) no lo son, y por tanto pueden ser tachados de la lista. El primer número después de 2 que no está tachado es 3, el siguiente primo. Tachamos entonces de la lista los siguientes múltiplos de 3, por supuesto, a partir de 3×3 ya que los anteriores están ya tachados (9, 12, ...). El siguiente número no tachado es 5, el siguiente primo, y entonces tachamos los siguientes múltiplos de 5 (25, 30, 35, ...). Repetimos este proceso hasta que alcanzamos el primer número en la lista que no está tachado y cuyo cuadrado es mayor que n. Todos los números que quedan en la lista sin tachar son los primos entre 2 y n.

Escribe un programa que utilice este algoritmo de criba para encontrar todos los números primos entre 2 y n.

Solución

```
def no_nulo(l,i):
    """ó
    Funcin que devuelve el índice del primer valor de la lista l
    que es distinto de cero.

    @type l: list
    @param l: una lista de únmeros
    @type i: int
    @param i: un índice de la lista a partir del cual buscar valores no nulos
    @return: el menor índice j>=i tal que l[i]!=0, si tal índice no existe devuielve
    """
    while (i < len(l)) and (l[i] == 0):
            i += 1
    return i

def tacha(l,i,step):
    """
    Ponea cero los elementos de la lista l a partir del índice i
    saltando de step en step, es decir, pone a cero los valores
    correspondientes a los índices i+step, i+step*2, i+step*3...
    @type l: list
    @param l: lista
    @type i: int
    @param i: índice de la lista
    que esten a distancia úmltiplos de step
    @type step: int
    @param step: incremento
    """
    j = i + step
    while j < len(l):
        l[j] = 0
```

```
        j = j + step

def criba(l):
    """
    Recibe una lista de únmeros consecutivos comenzando en 2 y pone a cero
    los valores de la lista que no corresponden a únmeros primos. Utiliza
    el émtodo conocido como Criba de óEratstenes.
    @type l: list
    @param l: lista de enteros consecutivos comenzando en 2
    """
    i = 0;
    while i < len(l):
        j = no_nulo(l,i)
        step = l[j]
        tacha(l,j,step)
        i = j+1

def main():
    l = range(2,200000)
    criba(l)
    for a in l:
        if a!=0:
            print "%d," % a,
    print

if __name__=="__main__":
    main()
```

▷ 16. Descomposición en factores primos

Seguro que has realizado alguna vez la tediosa tarea de descomponer un número en factores primos:

12	2	85	5	56	2	110	2
6	2	17	17	28	2	55	5
3	3	1		14	2	11	11
1				7	7	1	
				1			

¿Podríamos hacer un programa que fuése capaz de hacer estas descomposiciones? Seguro que sí. Escribe un programa que reciba como entrada un número entero positivo y calcule la descomposición del mismo en factores primos.

Ejemplo Un ejemplo de ejecución podría ser el siguiente:

```
Numero a descomponer? -1
El número ha de ser positivo
Numero a descomponer? 12
Factores primos: 2 2 3
Otro número (s/n)? s

Numero a descomponer? 85
Factores primos: 5 17
Otro número (s/n)? s

Numero a descomponer? 56
Factores primos: 2 2 2 7
```

```
Otro número (s/n)? s

Numero a descomponer? 110
Factores primos: 2 5 11
Otro número (s/n)? n
```

Solución

```
def factors(n):
    """
    This function prints the list of factors of n
    @type n: int
    @param n: n>1
    @rtype: List
    """
    fct = 2
    factors = []
    while n>1:
        if n % fct == 0:
            factors.append(fct)
            n = n/fct
        else:
            fct += 1
    return factors

def factors_exp(n):
    """
    This function returns the list of factors of n,
    together with their exponents
    @type n: int
    @param n: n>1
    @rtype: List
    """
    fct = 2
    factors = []
    while n>1:
        if n % fct == 0:
            exp = 1
            n = n/fct
            while n % fct ==0:
                exp += 1
                n = n/fct
            factors.append([fct,exp])
        else:
            fct += 1
    return factors

def main():
    cont = "s"
    while cont=="s":
        n = int(raw_input("Dame un únmero: "))
        if n>1:
            print "La lista de factores es: ", factors(n)
            print "La lista de factores es: ", factors_exp(n)
        else:
            print "El únmero debe ser mayor que 1"
```

```
cont = raw_input("Otro [s/N]")
```

▷ 17. Evaluación de polinomios

Deseamos desarrollar un programa que evalúe polinomios de una variable real, dados en dos líneas de la entrada estándar: en la primera, se dará el valor de la variable; en la segunda, la lista de coeficientes, no vacía. Se pide escribir dos versiones de este programa ateniéndose a lo que se describe seguidamente.

Órdenes crecientes En esta primera versión, se asume que los coeficientes del polinomio tienen pesos *crecientes*:

$$a_0 + a_1x^1 + \ldots + a_{n-1}x^{n-1} + a_nx^n$$

(El grado n del polinomio no se conoce *a priori*.)

Órdenes decrecientes En esta segunda versión, se asume que los coeficientes del polinomio tienen pesos *decrecientes*:

$$a_0x^n + a_1x^{n-1} + \ldots + a_{n-1}x + a_n$$

Pista: Tenemos que evaluar los datos que recibimos en una sola pasada, ya que sólo podemos leer una vez la entrada estándar. Por tanto, en principio no podemos usar la potencia x^n, puesto que no conocemos aún dicha n. La regla de Horner, debida al matemático inglés Willian George Horner (1786–1837), permite calcular un polinomio de grado n de forma acumulativa:

$$((\ldots((a_0x + a_1)x + a_2)x + \ldots)x + a_{n-1})x + a_n.$$

Teniendo en cuenta que en particular n podría ser 0.

Pista: Para facilitar la lectura en todos los casos de los coeficientes, se puede suponer que la secuencia de coeficienes finaliza con un coeficiente especial.

`ULT=....;`

Solución

Órdenes crecientes La idea del programa consiste en ir acumulando en la variable `poli` el valor del polinomio, de manera que en la vuelta i-ésima se tenga

$$\texttt{poli} = a_0 + a_1x^1 + \ldots + a_ix^i.$$

Para evitar hallar cada vez la potencia $\texttt{poli} = x^i$, puede actualizarse esta variable en cada vuelta haciendo `pot = pot * x`.

Todo ello se logra así: antes de entrar en el bucle (vuelta 0), se establece el valor inicial de *poli* con el término independiente (a_0), y el de *pot* a 1. Además, el paso de la vuelta i a la $i+1$-ésima cambia

$$\{pot = x^i\}\ \texttt{pot := pot*x}\ \{pot = x^{i+1}\}$$

y

$$\{poli = a_0 + a_1x^1 + \ldots + a_ix^i\}$$

por

$$\{poli = a_0 + a_1x^1 + \ldots + a_{i+1}x^{i+1}\}$$

En resumen, el siguiente programa consigue el resultado descrito:

```
ULT=3141592
x=float(raw_input("Valor de la variable:"))
acum = 0.0
potencia = 1.0
coeficiente=float(raw_input("coeficiente:"))
while coeficiente != ULT:
    acum += potencia*coeficiente
    potencia *= x
    coeficiente=float(raw_input("coeficiente:"))
print acum
```

Órdenes decrecientes En este caso, el programa es más sencillo que el anterior, porque no se necesita mantener las potencias x^i. Ahora basta con que cada vuelta cambie

$$(\ldots((a_0x + a_1)x + a_2)x + \ldots)x + a_i$$

por

$$((\ldots((a_0x + a_1)x + a_2)x + \ldots)x + a_i)x + a_{i+1}$$

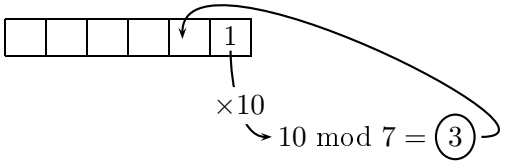
lo que se tiene fácilmente así:

```
ULT=3141592
x=float(raw_input("Valor de la variable:"))
acum = 0.0
potencia = 1.0
coeficiente=float(raw_input("coeficiente:"))
while coeficiente != ULT:
    acum += potencia*coeficiente
    potencia *= x
    coeficiente=float(raw_input("coeficiente:"))
print acum
```

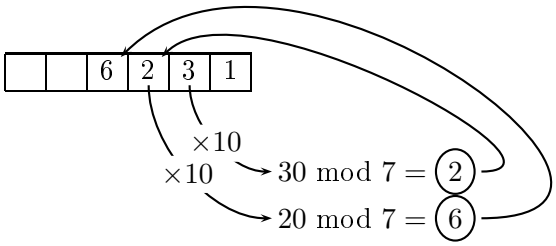
▷ 18. Criterios de divisibilidad

Tablas de módulos Dado un número k de una cifra, se denomina *tabla de módulos* a la tabla construida de la siguiente forma: en el extremo de la tabla ponemos un 1, el mecanismo general consiste en tomar la última entrada multiplicarla por diez y calcular el módulo de la división por el número k; este resultado pasa a ser una nueva entrada.

Supongamos que queremos calcular la tabla de módulos del número 7, comenzamos poniendo 1 en el extremo. Para calcular el siguiente número multiplicamos el 1 por 10 y ponemos el resto del producto dividido entre 7 en la segunda posición de la tabla:



Repetimos el mismo proceso comenzando con el último número de la tabla, es decir, lo multiplicamos por diez y calculamos el resto de dividir por 7; este número ocupa la siguiente casilla:



Repitiendo el proceso obtenemos la tabla siguiente:

5	4	6	2	3	1

Se termina de calcular la tabla de módulos cuando se obtiene el primer factor repetido, que no se incluye en la tabla. En el caso de la tabla del 7, si realizamos el cálculo de la siguiente entrada para el 5, obtenemos que $50 \bmod 7 = 1$ que es la primera entrada de la tabla.

Escribe un programa que, dado un número entre 2 y 9, genere y guarde en un array (suficientemente grande) la tabla de módulos de dicho número. Sólo tienen que calcularse las entradas de la tabla necesarias.

Divisibilidad La tabla de módulos del 7 puede utilizarse para saber si un número es múltiplo de 7 con sólo inspeccionar repetidamente sus cifras.

Si queremos saber si un número (pongamos 21756) es múltiplo de 7, y tenemos calculada la tabla de módulos correspondiente, basta con que vayamos calculando ascendentemente las cifras del número haciendo al mismo tiempo la siguiente operación:

5	4	6	2	3	1
	×	×	×	×	×
	2	1	7	5	6
	‖	‖	‖	‖	‖
	8	6	14	15	6

El número 21756 es múltiplo de 7 si y sólo si el resultado obtenido en 8+6+14+15+6 = 49 lo es. Averigüémoslo: para saber si 49 es múltiplo de 7 repetimos el proceso,

5	4	6	2	3	1
				×	×
				4	9
				‖	‖
				12	9

y resulta que ahora necesitamos saber si 21 lo es: damos un último paso y obtenemos,

5	4	6	2	3	1
				×	×
				2	1
				‖	‖
				6	1

cuya suma es 7, lo que confirma que 21756 es múltiplo de 7. Si, por el contrario, al final del proceso hubiésemos obtenido un número de una sola cifra diferente de 7, sabríamos que el número inicial no era múltiplo de 7.

Escribe un programa que, dado un número n presumiblemente grande, calcule si dicho n es múltiplo de 7 utilizando la tabla de módulos del 7.

Pista: Aunque el número tenga más cifras que elementos tiene la tabla de módulos, el algoritmo puede aplicarse igualmente. Si tenemos un número que tiene más cifras que entradas en la tabla, entonces tenemos que *"extender"* la tabla tantas veces como sea necesario. Si consideramos la tabla de módulos del 7,

5	4	6	2	3	1

y el número 2398765541172 que tiene 13 cifras, entonces necesitamos *"pegar"* tres tablas para poder abarcar todas sus cifras:

5	4	6	2	3	1	5	4	6	2	3	1	5	4	6	2	3	1
					2	3	9	8	7	6	5	5	4	1	1	7	2

Aunque realmente no hace falta tener 3 copias físicas; ten en cuenta que la posición séptima en ese array extendido se corresponde con la primera posición en el array original.

Notas bibliográficas El libro en el que Pascal publicó estos resultados es [Pas65]. El libro [Cha99] es mucho más fácil de conseguir y en él se puede encontrar el extracto de la obra anterior de Pascal en la que se describe el mecanismo que presenta el ejercicio.

Un poco de historia Etienne Pascal, padre de Blaise Pascal, fue matemático. Etienne mantuvo a Blaise apartado de los libros de matemáticas, ya que prefería que su hijo se dedicase a otras cosas. A pesar de eso, el talento de Blaise para la geometría era tal que hizo recapacitar a su padre y, cuando sólo tenía doce años de edad, lo inició en las matemáticas con los *Elementos* de Euclides [Euc91, Euc94, Euc96]. A los 17 años de edad Blaise escribió y publicó *Essay pour le coniques*.

El cálculo algorítmico siempre interesó a Blaise Pascal. A los 18 años diseñó y construyó una de las primeras calculadoras mecánicas, *La Pascalina*, que sumaba y restaba.

Solución

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# $Id$
#

def table(x):
    """
    This function computes the divisibility table for x. Example:
    table(7) -> [1,3,2,6,4,5]
    @type x: int
    """

    t=[1]
    foundRepetition = False
    while not foundRepetition:
        new = t[-1]*10 % x #t[-1] is the last element in the list
        if new in t:
            foundRepetition = True
        else:
```

```
            t.append(new)
    return t

def isMultiple_7(n):
    """
    This function finds out if the number 7 is divisible by 7 by
    using its divisibility table.
    @type n: int
    @param n: n>1
    @rtype: boolean
    """
    table = [1,3,2,6,4,5]
    while n>10:
        aux = n # we need to keep the
                # old value of n to compute the new one
        n = 0
        pos = 0 # the digit we are considering now
        while aux>0:
            d = aux%10
            n = n + d*table[pos % len(t)]
            pos +=1
            aux = aux/10
    return n==7
```

Solución

```
"""
Criterios de divisibilidad
"""

def belongs_to(elem, lst):
    """
    This functions check if the element is in the list
    @type lst: list
    @rtype: boolean
    """
    i = 0
    while i < len(lst) and lst[i] != elem:
        i += 1
    return i<len(lst)

def modules(n):
   """
   computes the modules table for integeger n.
   The table must be read backwards.
   @type n: int
   @param n: 2<=n<10
   @rtype: list

   Example: n=7 -> [1, 3, 2, 6, 4, 5]
   """

   vector = [1]
   pos = 0
   cont = True
   while cont:
```

```
        new = vector[pos]*10 % n
        if not belongs_to(new, vector):
            vector.append(new)
        else:
            cont = False
        pos += 1

    return vector

def reduction(vector, number):
    """
    performs a reduction step of number using the modules of vector
    @type vecter: list of int
    @type number: int
    @rtype: int

    Example:
    vector = [1, 3, 2, 6, 4, 5]
    number = 2398765541172
    5  4  6  2  3  1  5  4  6  2  3  1  5  4  6  2  3  1
                  2  3  9  8  7  6  5  5  4  1  1  7  2
                  2+15+36+48+14+18 +5+25+16 +6 +2+31 +2 = 220
    """

    tam = len(vector)
    cif = 0
    acu = 0
    while (number > 0) :
        acu = acu + vector[cif % tam]* (number % 10 )
        number = number / 10
        cif +=  1
    return acu

def divisible(number, factor):
    """
    tests if number is divisible by factor by using its module table.
    @type number: int
    @type factor: int
    @rtype: boolean
    """

    vector = modules(factor)
    while (number>9) :
        number = reduction(vector, number)
    return number == factor

def divisible7(number):
    """
    tests if number is divisible by 7
    """
    return divisible(number, 7)
```

Referencias

[Cha99] Jean-Luc Chabert, editor. *A History of Algorithms: From the Pebble to the Microchip.* Springer, 1999.

[Euc91] Euclides. *Elementos. Libros I-IV.* Gredos, 1991.

[Euc94] Euclides. *Elementos. Libros V-IX.* Gredos, 1994.

[Euc96] Euclides. *Elementos. Libros X-XIII.* Gredos, 1996.

[Pas65] Blaise Pascal. Des caractères de divisibilité des nombres de déduits de la somme de leurs chiffres, oeuvres complètes, 1665.