31_ESTRD_ejercicios_sol

October 12, 2017

Ejercicio 1
Dada la lista $L = [3, 5, 6, 8, 10, 12]$, se pide:
averiguar la posición del número 8;
cambiar el valor 8 por 9, sin reescribir toda la lista;
intercambiar 5 y 9;
intercambiar cada valor por el que ocupa su posición simétrica, es decir, el primero con el último, el segundo con el penúltimo, ...
crear la lista resultante de concatenar a la original, el resultado del apartado anterior.

```
In [2]: L=[3,5,6,8,10,12]
        L_copia=list(L) ### Interesa guardar una copia para el apartado e.
        #a
        j=L.index(8)
        j
```

```
Out[2]: 3
```

```
In [4]: #b
        L[j]=9
        L
```

```
Out[4]: [3, 5, 6, 9, 10, 12]
```

```
In [5]: #c
        j,k=L.index(5),L.index(9)
        L[j],L[k]=9,5
        L
```

```
Out[5]: [3, 9, 6, 5, 10, 12]
```

```
In [6]: #d
        L.reverse()
        L
```

```
Out[6]: [12, 10, 5, 6, 9, 3]
```

```
In [10]: #e
         LL2=L_copia+L
         LL2
```

```
Out[10]: [3, 5, 6, 8, 10, 12, 12, 10, 5, 6, 9, 3]
```

Ejercicio 2

La orden prime_range(100,1001) genera la lista de los números primos entre 100 y 1000. Se pide, siendoprimos=prime_range(100,1001):

averiguar el primo que ocupa la posición central en primos;

averiguar la posición, en primos, de 331 y 631;

extraer, conociendo las posiciones anteriores, la sublista de primos entre 331 y 631 (ambos incluidos);

extraer una sublista de primos que olvide los dos centrales de cada cuatro;

(**) extraer una sublista de primos que olvide el tercero de cada tres.

```
In [27]: primos=prime_range(100,1001)

Out[27]:

In [29]: #2.a
         L=len(primos)
         a=primos[L//2]
         a, len(primos[:L//2])==len(primos[L//2+1:])

Out[29]: (509, True)

In [30]: #2.b
         p1,p2=331,631
         a1,a2=primos.index(p1),primos.index(p2)
         a1,a2

Out[30]: (41, 89)

In [31]: #2.c
         primos[a1+1:a2] ### Si se entiende sin los extremos

Out[31]: [337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433,
```

Para el apartado d podemos entender que se pide:

recorrer la lista y decidir conforme al criterio SíNONOSíNONOSíNONO..., o con 0's (para el NO) y 1's (para el Sí): 1001001001... En este caso basta saltar de 3 en 3 y la notación slice nos resuelve.

o dividir en bloques de 4 y quitar los dos centrales; el dibujo es distinto 1001 1001 1001 .... En este caso, basta tomar los restos de la división de cada índice por 4, y quedarnos con los que dan resto 0 o 4.

```
In [32]: #2.d 100100100100...
         print 'Una muestra: ', primos[:8:3], primos[:8]
         print 'Todos:'; primos[::3]

Out[32]: Una muestra:  [101, 109, 131] [101, 103, 107, 109, 113, 127, 131, 137]
         Todos:
         [101, 109, 131, 149, 163, 179, 193, 211, 229, 241, 263, 277, 293, 313, 337, 353, 373,
```

```
In [58]: #2.d 1001 1001 1001 ...
         [primos[j] for j in range(len(primos)) if (j%4==0) or (j%4==3)]

Out[58]: [101, 109, 113, 137, 139, 157, 163, 179, 181, 197, 199, 227, 229, 241, 251, 269, 271,
```

Para olvidar el tercero de cada tres, hay que trabajar algo más, pues hemos de quedarnos con dos de cada tres. Se pueden pegar todas las sublistas que nos interesan, lo que conlleva un bucle for. A estas alturas es mejor quedarse con los índices múltiplos congruentes con 0 o 1 módulo 3:

```
In [35]: #2.e
         [primos[j] for j in range(len(primos)) if j%3!=2]

Out[35]: [101, 103, 109, 113, 131, 137, 149, 151, 163, 167, 179, 181, 193, 197, 211, 223, 229,
```

Ejercicio 3

Considérese el n úmero 1000! (factorial(1000)):

£en cuántos ceros acaba?

£Se encuentra el número 666 entre sus subcadenas? En caso afirmativo, localizar (encontrar los índices de) todas las apariciones.

Encontrar la subcadena má larga de doses consecutivos. Mostrarla con los dos dígitos que la rodean.

```
In [61]:

Out[61]:
```

El primer apartado se puede resolver numéricamente localizando los múltiplos de 5, 5^2, 5^3, ... que hay en los números del 2 al 1000. La potencia máxima es la parte entera del logaritmo en base 5 de 1000. Así, el siguiente cuadro nos calcula el número de ceros.

```
In [63]: %%time
         #3.a
         k=floor(log(10^3,base=5)) ### floor(x) devuelve la parte entera de x
         sum([floor(10^3/5^j) for j in [1..k]])

Out[63]: 249
         CPU time: 0.00 s,  Wall time: 0.00 s
```

Pero vamos a resolverlo de otras maneras, menos eficientes, pero que ayudan a ilustrar los métodos y funciones visitados en este capítulo.

```
In [65]: %%time
         #3.a Factorizando un entero (tremendo)
         [q for p,q in factor(factorial(10^3)) if p==5]

Out[65]: [249]
         CPU time: 0.01 s,  Wall time: 0.01 s
```

```
In [12]: %%time
         #3.a Utilizando una cadena de caracteres
         numeraco=str(factorial(10^3))
         m=5 ### Probamos buscar m ceros
         ceros='0'*m ### m=5 ceros
         j=numeraco.index(ceros)
         print numeraco[j-1:j+m+1]
         len(numeraco[j:])

Out[12]: 2000000
         249
         CPU time: 0.00 s,  Wall time: 0.00 s
```

Una última opción es tomar la lista de dígitos y localizar el último dígito no cero. Para utilizar .index(), damos la vuelta a la lista.

```
In [66]: %%time
         #3.a Buscando en la lista de dígitos el último no cero
         a=list(str(factorial(10^3)))
         a.reverse()
         min([a.index(j) for j in list('123456789')])

Out[66]: 249
         CPU time: 0.00 s,  Wall time: 0.00 s

In [15]: #3.b
         numeraco.count('666')

Out[15]: 2

In [37]: a1=numeraco.index('666')
         a2=numeraco[a1+1:].index('666')
         numeraco[a1:a1+3]; numeraco[a1+1+a2:a1+1+a2+3]

Out[37]: '666'
         '666'

In [69]: #3.c
         a=[numeraco.find('2'*j) for j in [1..10]]
         print a
         numdoses=a.index(-1)
         print numdoses
         comienzo=a[numdoses-1]
         final=comienzo+numdoses-1
         print comienzo,final
         print numeraco[comienzo-1:final+2]

Out[69]: [2, 453, 695, 1662, 1662, -1, -1, -1, -1, -1]
         5
         1662 1666
         0222221
```

Ejercicio 4

Si se aplica la función sum() a una lista numérica, nos devuelve la suma de todos sus elementos. En particular, la composición sum(k.digits()), para k un variable entera, nos devuelve la suma de sus dígitos (en base 10).

Calcular, con la composición sum(k.digits()), la suma de los dígitos del número k=factorial(1000).

Calcular la misma suma sin utilizar el método .digits().Sugerencia: considerar la cadena digitos='0123456789', en la que digitos[0]='0', digitos[1]='1', ..., digitos[9]='9'.Sumar los elementos de la lista

$$[j(\text{ veces que aparece } j \text{ en } 1000!) : \text{ con } j = 1,2,3,4,5,6,7,8,9] .$$

```
In [39]: #4.a
         k=factorial(1000)
         sum(k.digits())

Out[39]: 10539

In [40]: #4.b
         numeraco=str(k)
         digitos='0123456789'
         ### Recorriendo índices
         sum([j*numeraco.count(digitos[j]) for j in range(len(digitos))])

Out[40]: 10539

In [42]: ## Recorriendo caracteres
         sum([digitos.index(j)*numeraco.count(j) for j in digitos])

Out[42]: 10539
```

Ejercicio 5

A partir de la cadena de caracteres

```
In [48]: texto='Omnes homines, qui sese student praestare ceteris animalibus, summa ope niti de

Out[48]:
```

extraer la lista de los caracteres del alfabeto utilizados, sin repeticiones, sin distinguir mayúsculas de minúsculas y ordenada alfabéticamente.

Sugerencia: La composición list(set()) aplicada a una lista, genera una lista con los elementos de la original sin repeticiones, £por qué?

```
In [46]: letras=set(texto.lower())
         signos=' ,.'
         letras=list(letras.difference(signos))
         letras.sort()
         letras

Out[46]: ['a', 'b', 'c', 'd', 'e', 'f', 'h', 'i', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't',
```

```
In [50]: frecuencias=dict([(letra,texto.count(letra)) for letra in letras])

Out[50]:

In [51]: #Ejemplo-> contabilidad de vocales
         [(vocal,frecuencias[vocal]) for vocal in 'aeiou']

Out[51]: [('a', 15), ('e', 22), ('i', 16), ('o', 7), ('u', 11)]
```

Ejercicio 6. Máximo común divisor

Sin utilizar los métodos .divisors() ni la función max(), elabora código que, a partir de dos números a y b, calcule:

El conjunto $\text{Div}_a = \{k \in \mathbb{N} : k|a\}$

El conjunto $\text{Div}_b = \{k \in \mathbb{N} : k|b\}$

El conjunto $\text{Div}_{a,b} = \{k \in \mathbb{N} : k|a \text{ y } k|b\}$.

Una vez se tenga el conjunto de los divisores comunes, encontrar el mayor de ellos.

```
In [16]: a,b=2^3*5^3*7^2,7^3*2^4 ###Números para probar
         Div_a=set([k for k in srange(1,a+1) if a%k==0])
         Div_b=set([k for k in srange(1,b+1) if b%k==0])
         Div_ab=Div_a.intersection(Div_b)
         lDiv_ab=list(Div_ab) ### Pasado a lista...
         lDiv_ab.sort() ### ...y ordenados (por defecto es de menor a mayor), interesa el últi
         lDiv_ab[-1]

Out[16]: 392
```

Ejercicio 7. Mínimo común múltiplo

El mínimo comú múltiplo, m, de dos números, a y b, es menor o igual que su producto: $m \leq a \cdot b$. Sin utilizar la función min(), elabora código que, a partir de dos números a y b, calcule:

El conjunto $\text{Mult}_{a(b)} = \{k \in \mathbb{N} : a|k \text{ y } k \leq ab\}$

El conjunto $\text{Mult}_{b(a)} = \{k \in \mathbb{N} : b|k \text{ y } k \leq ab\}$

El conjunto $\text{Mult}_{a,b} = \{k \in \mathbb{N} : a|k,\ b|k \text{ y } k \leq ab\}$

Una vez se tenga este subconjunto de los múltiplos comunes, encontrar el menor de ellos.

```
In [52]: a,b=2^3*5^3*7^2,7^3*2^4 ###Números para probar
         Mult_a=set([a*j for j in range(1,b+1)])
         Mult_b=set([b*j for j in range(1,a+1)])
         Mult_ab=Mult_a.intersection(Mult_b)
         lMult_ab=list(Mult_ab)
         lMult_ab.sort()
         lMult_ab[0]

Out[52]: 686000

In [18]: ##Comprobación sencilla
         lMult_ab[0]*lDiv_ab[-1]==a*b

Out[18]: True
```

Ejercicio 8

Dada una lista de números enteros, construye un conjunto con los factores primos de todos los números de la lista.

Indicación: Usa list(k.factor()) para obtener los factores primos.

Nota: Mejor que la indicación propuesta, usamos la versión con la función factor(): list(factor(k)). El único problema con la propuesta es que los números k a los que aplicar el método .factor() deberían ser enteros de sage, pero los que generamos con randint() no lo son. Bastaría multiplicarlos por 1.

```
In [57]:
```

```
Out[57]:
```

```
In [55]: nums=[randint(1,100) for j in xrange(8)] ### Una lista de enteros al azar entre 1 y 1
         factores=set()
         for k in nums:
             factores.update([p for p,q in list(factor(k))])  ##como no son enteros de sage, u
         factores
```

```
Out[55]: set([2, 3, 5, 7, 11, 13, 47])
```

```
In [70]: ##Utilizando la indicación del enunciado. Obsérvese la multiplicación en randint(1,10
         nums=[randint(1,100)*1 for j in xrange(8)] ### Una lista de enteros al azar entre 1 y
         factores=set()
         for k in nums:
             factores.update([p for p,q in list(k.factor())])  ##como no son enteros de sage,
         factores
```

```
Out[70]: set([2, 3, 5, 7, 13, 17, 67])
```

Ejercicio 9

Almacena en un diccionario los pares letra:frecuencia resultantes de hacer el análisis de frecuencias de la cadena del siguiente texto.

Utilizar el diccionario para averiguar la frecuencia de cada una de las cinco vocales.

```
In [19]: Hamlet='''To be, or not to be: that is the question: Whether tis nobler in the mind t
```

```
Out[19]:
```

```
In [21]: caracteres=list(set(Hamlet))
         caracteres
```

```
Out[21]: ['\x80', '\x93', '\x99', '!', ' ', ',', '.', ';', ':', '?', 'A', 'B', 'D', 'F', 'I',
```

```
In [22]: veces=[(j,Hamlet.count(j)) for j in caracteres]
         frecuencias=dict(veces)
         [(vocal,frecuencias[vocal]) for vocal in 'aeiou']
```

```
Out[22]: [('a', 82), ('e', 146), ('i', 56), ('o', 97), ('u', 41)]
```

Ejercicio 10

A partir del diccionario de frecuencias del ejercicio anterior, construir una lista de pares (frecuencia,letra) y ordenarla, en orden creciente de frecuencias.

```
In [23]: lfrecs=[(frecuencias[letra],letra) for letra in caracteres]
         lfrecs.sort()
         lfrecs

Out[23]: [(1, 'D'), (1, 'I'), (1, 'M'), (1, 'S'), (1, '\x93'), (2, '!'), (2, 'B'), (2, 'F'), (

In [26]:

Out[26]:
```