81-CRIPT-codificacion

March 4, 2018

Limpiar un texto de caracteres indeseados

```
In [1]: alfb = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" #Caracteres permitidos

In [2]: texto = "Through the use of abstraction and logical reasoning, mathematics developed fi
```

La codificación ASCII representa cada caracter mediante un entero entre 0 y 255. Como $256 = 2^8$, cada uno de esos enteros se representa en binario mediante 8 bits (ceros o unos), que es igual a 1 byte. Entonces un texto de N caracteres ocupa en memoria N bytes. La función *ord* nos da el entero, en ASCII, que corresponde a un caracter ($ord('A') = 65$), mientras que *chr* es su función inversa.

```
In [3]: print map(ord,[x for x in alfb])

[65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88

In [4]: print map(chr,map(ord,[x for x in alfb]))  #Comprobamos que chr es la inversa de ord

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S'

In [5]: from string import * ##Hace falta para que funcione join
        def limpiar(texto,alfb):
            '''Dejamos en el texto solo los caracteres de alf. '''
            L = [x.capitalize() for x in list(texto)] #Cambiamos el texto a mayusculas
            L1 = [car for car in L if car in alfb]
            C1 = join(L1,sep = "")
            return C1

In [6]: limpiar(texto,alfb)

Out[6]: 'THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULATI
```

Codificación más adaptada al tamaño del alfabeto

En ocasiones, en lugar de usar la codificación ASCII queremos codificar los caracteres, suponiendo que tenemos 26 como en $alfb$, mediante enteros del 0 al 25. Para eso debemos redefinr las funciones *ord* y *chr*:

```
In [7]: L_alfb = list(alfb)
```

```
In [8]: def ord2(c):
            return L_alfb.index(c)

In [9]: def chr2(n):
            return L_alfb[n]

In [10]: print map(ord2,[x for x in alfb])

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]


In [11]: print map(chr2,map(ord2,[x for x in alfb]))

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S'
```

Una codificación binaria

En este ejercicio, que se repite y se utiliza en la hoja "87-CRIPT-mhellman", codificamos cada letra mediante un entero entre 0 y 25, y luego cada entero como una palabra de 5 bits.

```
In [12]: alfb = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"

In [13]: L_alfb = list(alfb)

In [14]: texto = "Through the use of abstraction and logical reasoning, mathematics developed

In [15]: def ord2(c):
             return L_alfb.index(c)

In [16]: def chr2(n):
             return L_alfb[n]

In [17]: print map(ord2,[x for x in alfb])

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]


In [18]: (7).digits(base=2,padto=5)

Out[18]: [1, 1, 1, 0, 0]
```

Digits escribe los dígitos de izquierda a derecha. Como nos interesa el orden inverso en la función *cadena*, cuarta línea, invertimos el orden.

```
In [19]: def cadena(L):
             '''Convertimos una lista en cadena de caracteres'''
             C = ''
             L.reverse()
             for item in L:
                 C = C+str(item)
             return C
```

```
In [20]: def bin_2(n):
             '''Dado un entero <32 lo convertimos en una cadena de 5 bits'''
             return cadena(ZZ(n).digits(base=2,padto=5))

In [21]: bin_2(7)

Out[21]: '00111'

In [22]: print map(chr2,map(ord2,[x for x in alfb]))

['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S'


In [23]: def limpiar(texto,alfb):
             L = map(ord,[x.capitalize() for x in list(texto)])
             L1 = [item for item in L if item in map(ord,[x for x in alfb])]
             C1 = join(map(chr,L1),sep = "")
             return C1

In [24]: texto2 = limpiar(texto,alfb);texto2

Out[24]: 'THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULA

In [25]: texto_cod0 = map(ord2,list(texto2));print texto_cod0 #Cambiamos el texto por una list

[19, 7, 17, 14, 20, 6, 7, 19, 7, 4, 20, 18, 4, 14, 5, 0, 1, 18, 19, 17, 0, 2, 19, 8, 14, 13, 0


In [26]: texto_cod1 = map(bin_2,texto_cod0);print texto_cod1 #Codificamos cada uno de esos ent

['10011', '00111', '10001', '01110', '10100', '00110', '00111', '10011', '00111', '00100', '10


In [27]: texto_cod3 = join(texto_cod1,sep='');print texto_cod3 #Texto codificado como una pala

10011001111000101110101000011000111100110011100100101001001000100011100010100000000011001010011
```

La función que codifica sería, resumiendo algunas de las celdas anteriores:

```
In [28]: def cod(texto):
             L = map(ord2,list(texto))
             L1 = map(bin_2,L)
             return join(L1,sep='')
```

Para descodificar:

```
In [29]: ZZ('10011',base=2)

Out[29]: 19
```

```
In [30]: def descod(C):
             C1 = ''
             while C != '':
                 C1 = C1+chr2(ZZ(C[:5],base=2))
                 C = C[5:]
             return C1

In [31]: descod(texto_cod3)

Out[31]: 'THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULA

In [32]: descod(cod(texto2)) #Parecen funciones inversas

Out[32]: 'THROUGHTHEUSEOFABSTRACTIONANDLOGICALREASONINGMATHEMATICSDEVELOPEDFROMCOUNTINGCALCULA
```