Criptograf?a 1 - Alejandro Santorum (17-02-2018)

February 17, 2018

EJERCICIO 6 - Comenzamos creando dos diccionarios, uno que a partir de las letras te devuelva los números; y otro que a partir del número te devuelva su letra.

```
In [80]: diccLetras = {'a':0, 'b':1, 'c':2, 'd':3, 'e':4, 'f':5, 'g':6, 'h':7, 'i':8, 'j':9, 'l
```

```
In [81]: diccNum = {0:'a', 1:'b', 2:'c', 3:'d', 4:'e', 5:'f', 6:'g',7:'h', 8:'i', 9:'j', 10: 'l
```

```
In [82]: def autoclave_cod(T, clave):
             L = list(T)
             L[0] = diccNum[(diccLetras[clave]+diccLetras[L[0]])%27]
             for i in xrange (1, len(L)):
                 L[i] = diccNum[(diccLetras[T[i-1]]+diccLetras[L[i]])%27]
             S = ''.join(L)
             return S
```

```
In [83]: string = "atacaremos por el lado derecho a las tres de la tarde"
         S = autoclave_cod(string, 'z')
         S
```

```
Out[83]: 'zttccrvq froceqdpkkldrnchvvgjvn  klsrsjvwrchdkl struh'
```

Vemos que la codificación es bastante buena, no se tiene ni idea de lo que quiere decir.

```
In [84]: def autoclave_decod(T, clave):
             L = list(T)
             L[0] = diccNum[(diccLetras[L[0]]-diccLetras[clave])%27]
             for i in xrange (1, len(L)):
                 L[i] = diccNum[(diccLetras[L[i]]-diccLetras[L[i-1]])%27]
             S = ''.join(L)
             return S
```

```
In [85]: SD = autoclave_decod(S, 'z')
         SD
```

```
Out[85]: 'atacaremos por el lado derecho a las tres de la tarde'
```

La decodificación funciona a la perfección.

EJERCICIO 8 - Vamos a programar una función que busca esos números primeros cuyo producto no se pueda factorizar en menos de 30 minutos (en la máquina que se utilice la función). Como es lógico, no se va a comprobar si correcto funcionamiento, pero el código, es lo suficientemente simple como para ver si está bien o mal.

```
In [87]: from time import time

         def search_unfactable_product():
             m = 1000

             while(exe_time < 1800): #30 minutos = 1800 segundos

                 p1 = nth_prime(m)
                 p2 = nth_prime(m+1)

                 ini_time = time() #devuelve el tiempo en segundos
                 n = factor(p1*p2)
                 fin_time = time()

                 exe_time = fin_time - ini_time
                 m += 1

             return p1, p2

In [ ]:
```