

Lab 2 – Base 64 y XOR

Competencias a desarrollar

- Implementar el uso de funciones para XOR
- Implementar el uso de funciones para Base64
- Implementar el uso de funciones para Binario
- Identifica los requisitos necesarios para unificar 2 imagenes
- Implementar el uso de la librería Pillow en python para manejo de imagenes
- Implementar el uso de matrices de bits para imagenes

source ASCII (if <128)	M								a								n															
source octets	77 (0x4d)								97 (0x61)								110 (0x6e)															
Bit pattern	0	1	0	0	1	1	0	1	0	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0								
Index	19								22								5								46							
Base64-encoded	T								W								F								u							
encoded octets	84 (0x54)								87 (0x57)								70 (0x46)								117 (0x75)							

Problemas a resolver

En este laboratorio implementaremos funciones que convierten cadenas a bits y a Base 64. Además, la investigación de una propiedad estadística de la función XOR.

1. Implementar una función para convertir una cadena de caracteres a bits. Por cada carácter de la cadena encontrar la representación en bytes (8 bits) del valor ASCII de dicho carácter. La función debe devolver la concatenación de todos los bits de la cadena.

- a. Muestre 2 ejemplos sencillos de convertir cadenas a bytes

```
2 ejemplos sencillos de convertir cadenas a bytes
1. sol
2. hola

ejemplo1 = string_to_bits("sol")
print(ejemplo1)

ejemplo2 = string_to_bits("hola")
print(ejemplo2)

011100110110111101101100
01101000011011110110110001

ascii -> octeto -> binario

s -> 115 -> 01110011
o -> 111 -> 01101111
l -> 108 -> 01101100

h -> 104 -> 01101000
o -> 111 -> 01101111
l -> 108 -> 01101100
a -> 97 -> 01100001
```

2. Implementar una función para convertir una cadena de bytes a caracteres. Por cada grupo de 8 bits encontrar su representante correspondiente en ASCII. La función debe devolver el texto correspondiente.

- a. Muestre 2 ejemplos sencillos de convertir bytes a cadena

```
Ejemplos:

1. 011010000110000101110100
2. 011000110110000101110100

ejemplo1 = bits_to_string("011010000110000101110100")
print(ejemplo1)

ejemplo2 = bits_to_string("011000110110000101110100")
print(ejemplo2)

hat
cat

binario -> octeto -> ascii

01101000 -> 104 -> h
01100001 -> 97 -> a
01110100 -> 116 -> t

01100011 -> 99 -> c
01100001 -> 97 -> a
01110100 -> 116 -> t
```

3. Implementar funciones que permitan convertir una cadena de caracteres a Base64, para esto utilizar la conversión manual (texto a binario, binario a código UNICODE).
 - a. Mostrar 2 ejemplos sencillos de convertir una cadena a base 64.

```
Ejemplos:

1. sol
2. hola

texto_a_base64("sol")
✓ 0.0s
'c29s'

texto_a_base64("hola")
✓ 0.0s
'aG9sYQ=='

ascii -> octeto -> binario

• sol:
s -> 115 -> 01110011
o -> 111 -> 01101111
l -> 108 -> 01101100

011100 110110 111101 101100
indices: 28 54 61 44
Caracteres en base64: c29s

• hola:
h -> 104 -> 01101000
o -> 111 -> 01101111
l -> 108 -> 01101100
a -> 97 -> 01100001

011010 000110 111101 101100 011000 010000
indices: 26 6 61 44 24 16 = =
Caracteres en base64: aG9sYQ==
```

4. Implementar funciones que permitan convertir una cadena de base 64 a su texto correspondiente para esto utilizar la conversión manual (texto UNICODE a binario , binario a Codigos ASCII).

- a. Mostrar 2 ejemplos sencillos de convertir una cadena de base64 a su texto correspondiente.

```
Ejemplos:

1. Y2F0
2. cmFw

bits_to_string(base64_a_binario("Y2F0"))
✓ 0.0s
'cat'

bits_to_string(base64_a_binario("cmFw"))
✓ 0.0s
'rap'

base64 -> binario -> octeto -> ascii

• Y2F0 ->
Y -> 24 -> 011000
2 -> 54 -> 110110
F -> 5 -> 000101
0 -> 52 -> 110100
binario: 011000110110000101110100
ascii:
c <- 143 <- 01100011
a <- 141 <- 01100001
t <- 164 <- 01110100
r/. cat

• cmFw ->
c -> 99 -> 011100
m -> 38 -> 100110
F -> 5 -> 000101
w -> 119 -> 110000
binario: 011100100110000101110000
ascii:
r <- 162 <- 01110010
a <- 141 <- 01100001
p <- 160 <- 01110000
r/. rap
```