I/O: STRINGS, BYTES Y ARCHIVOS

AYUDANTÍA 11

STRINGS

- Caracteres Unicode
- Métodos para manipular/analizar strings
- Opciones de formato

STRINGS

- Caracteres Unicode
- Métodos para manipular/analizar strings
- Opciones de formato

STRINGS

- Caracteres Unicode
- Métodos para manipular/analizar strings
- Opciones de formato

.isalpha()
.isdigit()
.startswith()
.endswith()
.find()
.split()
.strip()
.
.
.

STRINGS

- Caracteres Unicode
- Métodos para manipular/analizar strings
- Opciones de formato

.format() ofrece muchas opciones!!!

OPCIONES DE FORMATO

```
'{2}, {0}, {2}, {1}'.format('a', 'b', 'c')

'A: {a}, A: {b}, A: {c}'.format(**{'a':134, 'c':'hola', 'b': [1,2]})

'{:>40}'.format('hola')

"{0:.8s}{1: ^9d}    ${2: <8.2f}${3: >7.2f}".format(a, b, c, d)
```

OPCIONES DE FORMATO

```
'{2}, {0}, {2}, {1}'.format('a', 'b', 'c')

'A: {a}, A: {b}, A: {c}'.format(**{'a':134, 'c':'hola', 'b': [1,2]})

'{:>40}'.format('hola')

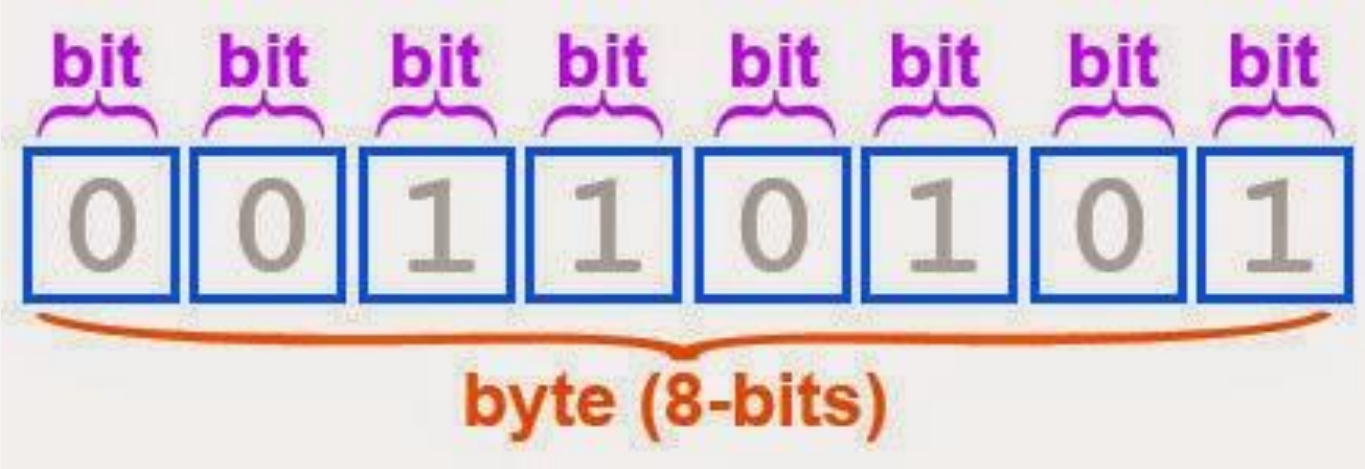
"{0:.8s}{1: ^9d}    ${2: <8.2f}${3: >7.2f}".format(a, b, c, d)
```

Mas ejemplos en…



BYTES

Byte: secuencia de 8 bits.
Bit (**Bi**nary Digi**t**): dígito que puede tomar el valor 0 o 1.



BYTES

Representaciones de números:

DECIMAL (Base 10):

0 1 2 3 4 5 6 7 8 9

$$135d$$
$$= 1 \times 10^2 + 3 \times 10^1 + 5 \times 10^0 = 100 + 30 + 5 = 135$$

BINARIO (Base 2):

0 1

$$10000111b$$
$$= 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$
$$= 128 + 0 + 0 + 0 + 0 + 4 + 2 + 1 = 135$$

HEXADECIMAL (Base 16):

0 1 2 3 4 5 6 7
8 9 *A B C D E F*

$$87h$$
$$= 8 \times 16^1 + 7 \times 16^0$$
$$= 128 + 7 = 135$$

BYTES

Representaciones de números:

Máximo valor con 1 Byte:

$$\underbrace{11111111}b = 255d$$

$$FFh = 255d$$

1 Byte (8 bits) se puede representar con 2 dígitos en Hexadecimal

BYTES

¿Cómo los usamos en Python?

Es parecido a usar strings!

```
mi_byte = b“Creando bytes”
```

Solo basta con anteceder el string con una ‘b’.

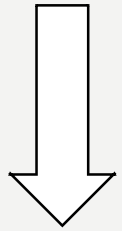
BYTEARRAYS

Arreglo de bytes individuales.

- Mutables
- Indexables
- Extendibles

BYTES V/S BYTEARRAYS

	Strings	Listas
Ejemplo	“hola”	[“h”, “o”, “l”, “a”]
Mutable	NO	SI
Indexable	SI	SI



Análogamente

	Byte	Bytearray
Ejemplo	b“hola”	bytearray(b”hola”)
Mutable	NO	SI
Indexable	SI	SI

ARCHIVOS

```
file = open(“archivo.txt”, “r”)
# Hacer cosas con file
datos = file.read()
file.close()
```

```
with open(“archivo.txt”, “r”) as file:
    # Hacer cosas con file
    datos = file.read()
```

ARCHIVOS

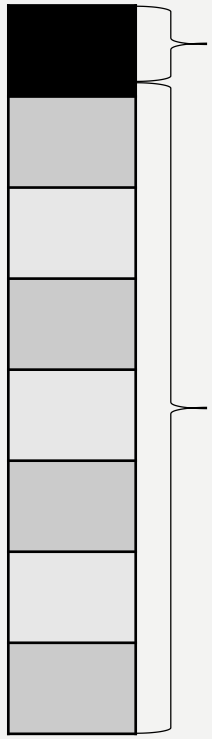
```
open("archivo.txt", "rb")
```

Opciones de apertura de archivos:

"w", "a", "r", "rb", "wb", …

ARCHIVOS

Estructuras de archivos:



HEADER (Metadatos): Permite saber qué encontraremos dentro del archivo

CHUNKS (Datos) Contiene la información del archivo

ARCHIVOS

Ejemplo PNG:

0	1	2	3	4	5	6	7	8	9	10	11	12	…	n	n+1	n+2	n+k	…
137	80	78	71	13	10	26	10	Chunk							Chunk			

Metadatos

Datos

Además, cada Chunk sigue la siguiente estructura:

Largo de la info del Chunk	Tipo de Chunk	Información	CRC
4 bytes	4 bytes	Largo	4 bytes

EJERCICIO 1

- Entendiendo Bytes / Bytearrays:

¿ Qué ocurre en el siguiente código?

```
a = bytearray(b‘hola’)
print(a)
print(a[0])
a[0] = 104
print(a)
b = b‘hola’
print(b)
print(b[0])
b[0] = 104
print(b)
```

EJERCICIO 2

- La Corporación de Inteligencia Internacional (IIC por su sigla en inglés) ha activado el protocolo 2233, por lo cual has recibido un mensaje de suma importancia. Sin embargo, para evitar que el mensaje llegue a las manos equivocadas, este ha sido enviado en dos archivos que han sido modificados. Para poder obtener el contenido del mensaje, tu misión es unir la información de los dos archivos en uno, siguiendo las instrucciones de la IIC.
- Para la primera mitad del archivo, se debe usar el archivo llamado `confidencial1.iic2233.` A cada byte del archivo se le debe restar 2233, y luego a ese número se le debe aplicar la operación Mod_{256} ($\%256$).
- Para la segunda mitad del archivo, se debe usar el archivo llamado `confidencial2.iic2233.` En esta sección el protocolo indica que cada chunk del archivo es del tamaño de la posición del chunk. Es decir, el primer chunk es de tamaño 1, el segundo es de tamaño 2, el tercero de tamaño 3, y así sucesivamente. El protocolo indica que todos los chunks de tamaño impar están correctos, mientras que los chunks de tamaño par están invertidos (estos deben invertirse nuevamente).
- Una vez hecho esto, el protocolo indica que los primeros 8 bytes del archivo deben ser ingresados a mano, y que el archivo corresponde a un .png, con lo cual estos deben ser los bytes de metadatos.
- Finalmente, se debe concatenar los bytes de metadatos, con los bytes del archivo 1 y del archivo 2, para luego escribir estos datos en un archivo con extensión .png