I/O, serialización y RegEx

Semana 11 - Jueves 14 de noviembre de 2019

Video de la Clase

https://youtu.be/lwiw68FsFVA

- Les recomendamos ver este video para que puedan revisar las slides con la explicación del equipo docente de ellas.

Input/Output

- Manejo de Strings
- Bytes

Strings

"hola"

- Secuencia inmutable de caracteres

list(s)

```
list("hola")
# h, o, l, a
```

- Secuencia mutable de caracteres

Bytes

```
b"al\xc3\xb3"
```

- Secuencia inmutable de bytes

bytearray(b)

```
bytearray(b"al\xc3\xb3")
# 97, 108, 195, 179
```

- Secuencia mutable de bytes

```python
# Como sabemos, los strings son una secuencia
# inmutable de caracteres, pero tienen métodos
# muy útiles
mi_string = "Programando en IIC2233"
print(mi_string.isalpha())
print(mi_string[-4:].isdigit())
print(mi_string.startswith("Durmiendo"))

print(f"Estado: {mi_string}")
print("Estado: {}".format(mi_string))

# También podemos trabajar con la representación de
# un objeto como bytes
mis_bytes = b"\x63\x6c\x69\x63\x68\xe9"
print(mis_bytes)
print(mis_bytes.decode("latin-1"))     # cliché
mi_string_como_bytes = mi_string.encode("latin-1")
print(mi_string_como_bytes)
```

```
# Los bytearrays son arreglos de bytes, y se
# comportan como una lista: son mutables
mi_bytearray = bytearray(b"Estoy en IIC2233")

# Puedo indexar y usar notación de slices
print(mi_bytearray[0])
>> 69 # bytes([69]).decode("ascii") == "E"
mi_bytearray[2:4] = b"\x15\xa3"
print(mi_bytearray[1:5])
>> bytearray(b"s\x15\xa3y")
# bytearray(b'Es\x15\xa3y en IIC2233')

# Puedo agregar bytes.
mi_bytearray.extend(b" 2019-1")
print(mi_bytearray)
>> bytearray(b"Es\x15\xa3y en IIC2233 2019-1")
```

Leer y escribir archivos

Context Manager

- `r := read`
- `w := write`
- `a := append`

- `t := text`
- `b := binary`

```
# Cuando intentamos abrir un archivo, podemos
# encontrarnos con un error algo raro...
with open(path, "r") as archivo:
    archivo.read()

Traceback (most recent call last):
  File "...\programa.py", line 2, in <module>
    archivo.read()
  ...
UnicodeDecodeError: 'charmap' codec can't decode
byte 0x81 in position 178: character maps to
<undefined>
```

```python
# Esto lo podemos arreglar incorporando el
# encoding en que está el archivo
with open(path, "r", encoding="utf-8") as archivo:
    archivo.read()

# Esto se debe a que Python intentó abrir el
# archivo con un encoding por defecto, pero un byte
# en particular no es conocido por ese encoding
# (no sabe cómo interpretar 0x81).
# Normalmente, esto se debe al uso de tildes (o ñ)
```

```python
# Queremos leer los bytes de un archivo encriptado,
# y para recuperar el archivo original tenemos que
# armar grupos de 8 bytes e invertirlos

# Al leer el archivo como bytes no usamos encoding,
# estamos trabajando con los bytes directamente
with open(path_archivo, "rb") as archivo:

    # Leemos todos los bytes y los usamos como lista
    original = bytearray(archivo.read())

    # Hacemos un bytearray para la nueva versión
    modificado = bytearray()

    # Ahora podemos hacer el procesamiento
    for i in range(0, len(original), 8):
        segmento = original[i:i+8] # Agrupamos 8
        segmento = segmento[::-1] # Invertimos
        modificado.extend(segmento)
```

Serialización

- Guardar objetos en un formato que pueda ser guardado y reconstruido
- Módulos `pickle` y `json`

`pickle`

- Ventajas
- Desventajas
- ¿Cómo se modifica la forma de serializar?

pickle

```
b'\x80\x03}q\x00(X\n\x00\x00\x00first_nameq\x01X\x06\x
00\x00\x00Alexisq\x02X\t\x00\x00\x00last_nameq\x03X\x0
8\x00\x00\x00S\xc3\xa1nchezq\x04X\t\x00\x00\x00birthda
teq\x05X\n\x00\x00\x001988-12-19q\x06X\x06\x00\x00\x00
heightq\x07G?\xfb\n=p\xa3\xd7\nX\x04\x00\x00\x00clubq\
x08}q\t(X\x04\x00\x00\x00nameq\nX\x0f\x00\x00\x00Inter
de
MIl\xc3\xa1nq\x0bX\x07\x00\x00\x00foundedq\x0cMt\x07uX
\x07\x00\x00\x00aliasesq\r]q\x0e(X\x0f\x00\x00\x00Ni\x
c3\xb1o Maravillaq\x0fX\x15\x00\x00\x00La Ardilla de
Atacamaq\x10eX\n\x00\x00\x00girlfriendq\x11NX\x07\x00\
x00\x00injuredq\x12\x88u.'
```

JSON

- Ventajas
- Desventajas
- ¿Cómo se modifica la forma de serializar?

JSON

```
{
    "first_name": "Alexis",
    "last_name": "Sánchez",
    "birthdate": "1988-12-19",
    "height": 1.69,
    "club": {
        "name": "Inter de MIlán",
        "founded": 1908
    },
    "aliases": [
        "Niño Maravilla",
        "La Ardilla de Atacama"
    ],
    "girlfriend": null,
    "injured": true
}
```

Métodos importantes

- dump, load
- dumps, loads

pickle

```python
import pickle

tupla = ("a", 1, 3, "hola")
serializacion = pickle.dumps(tupla)

print(serializacion)
print(type(serializacion))
print(pickle.loads(serializacion))
```

```
>
b'\x80\x03(X\x01\x00\x00\x00aq\x00K\x01K\x03X\x04\x00\x00\x00holaq\x
01tq\x02.'
> <class 'bytes'>
> ('a', 1, 3, 'hola')
```

JSON

```
import json

tupla = ("a", 1, 3, "hola")
serializacion = json.dumps(tupla)

print(serializacion)
print(type(serializacion))
print(json.loads(serializacion))


> ["a", 1, 3, "hola"]
> <class 'str'>
> ['a', 1, 3, 'hola']
```

pickle

```python
import pickle

lista = [1, 2, 3, 7, 8, 3]

with open("mi_lista.bin", 'wb') as file:
    pickle.dump(lista, file)

with open("mi_lista.bin", 'rb') as file:
    lista_cargada = pickle.load(file)

print(f"¿Las listas son iguales? {lista == lista_cargada}")
print(f"¿Las listas son el mismo objeto? {lista is lista_cargada}")
```

```
> ¿Las listas son iguales? True
> ¿Las listas son el mismo objeto? False
```

JSON

```
import json

lista = [1, 2, 3, 7, 8, 3]

with open("mi_lista.bin", 'w') as file:
    json.dump(lista, file)

with open("mi_lista.bin", 'r') as file:
    lista_cargada = json.load(file)

print(f"¿Las listas son iguales? {lista == lista_cargada}")
print(f"¿Las listas son el mismo objeto? {lista is lista_cargada}")

> ¿Las listas son iguales? True
> ¿Las listas son el mismo objeto? False
```

RegEx

- Secuencia de caracteres
- Patrón de búsqueda
- Búsqueda o reemplazo de patrones en *strings*

Meta-caracteres

- Cualquier carácter **.**
- Inicio/término **^ , $**
- Cardinalidad **? , + , ***
- Conjunto de opciones **[ ]**
- Repeticiones **{ }**
- Opciones a o b **|**
- Definir grupo **( )**
- Escape ****

`search` vs. `match`

- `search`

  Verifica si **algún substring** cumple el patrón

- `match`

  Verifica si un **prefijo** cumple el patrón

RegEx

```python
import re

texto = "Mi cursos favoritos son IIC2233 e IIC1105, y los menos favoritos son FIS1533 e IIC321."

for curso in re.findall("IIC\d{4}", texto):
    # Encuentra todas las coincidencias y retorna una lista
    print(f"Se nombró al curso {curso}")
```

```
> Se nombró al curso IIC2233
> Se nombró al curso IIC1105
```

Actividad

1. En el *syllabus*, vayan a la carpeta “Actividades” y descarguen el enunciado de la actividad 09 (AC09). https://github.com/IIC2233/Syllabus
2. Trabajen **en forma individual** hasta las **20:00**.
3. Recuerden hacer *commit* y *push* cada cierto tiempo.

I/O, serialización y RegEx

Semana 11 - Jueves 14 de noviembre de 2019

Cierre

Diagrama de flujo de AC

Daniar

Serialización JSON

```
“docent.json” -> class Docengelion
class DocengelionEncoder
```

Enzohor

Serialización `pickle`

```
“pilotos.magi” -> class Piloto
def __setstate__(self, state)
```

Pintosar

Manejo de *bytes*

```
“EVA.xdc” -> “Docengelion.bmp”
def reparar_comunicacion(ruta)
```

I/O, serialización y RegEx

La información digital se almacena como *bytes*, y hay muchas formas de guardar, cargar y alterar información.

Aprendemos un poco de cómo se guardan efectivamente los archivos en un computador: ¡son solo un montón de *bytes*!

Además, conocimos distintas maneras de guardar y cargar información, mediante serialización.

Serialización JSON (Daniar)

La idea era cargar instancias de Docengelion que provienen de un archivo JSON, y luego volver a guardar esas instancias pero con atributos alterados mediante un JSONEncoder.

```
# Cargando desde archivo JSON a instancias Docengelion
def recibir_eva(ruta):
    with open(ruta, 'r') as archivo:
        json_evas = json.load(archivo)
    return [Docengelion(**dict_eva) for dict_eva in json_evas]

# Cargando desde instancia a JSON
def reparar_eva(docengelion):
    nombre_archivo = f'Unidad-{docengelion.modelo}.json'
    ruta = os.path('Daniar', nombre_archivo)
    with open(ruta, 'w') as archivo:
        json.dump(docengelion, archivo, cls=DocengelionEncoder)
```

Serialización `pickle` (Enzohor)

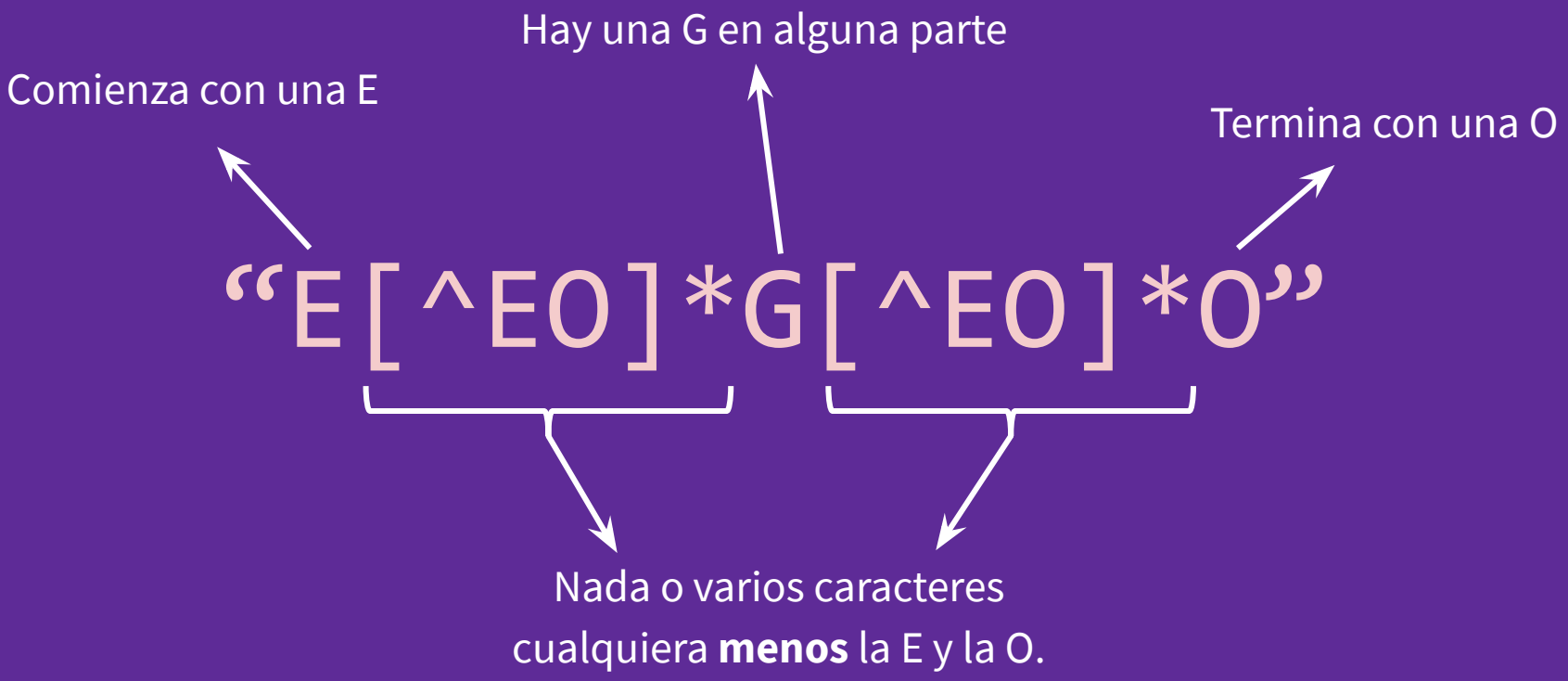
La idea era cargar instancias de `Piloto` que provienen de un archivo previamente serializado mediante `pickle`. La carga también debía arreglar un atributo recibido en la serialización.

```
# Cargando desde archivo serializado a instancias Piloto
def cargar_almas(ruta):
    with open(ruta, ‘rb’) as archivo:
        lista_pilotos = pickle.load(archivo)
    return lista_pilotos

# Override de método para cargar los atributos recibidos
class Piloto:
    ...
    def __setstate__(self, estado):
        nuevo_estado = aumentar_sincronizacion(estado)
        self.__dict__ = estado
```

(*Bonus:* Usar Regex en Enzohor)

La función `aumentar_sincronizacion` debía alterar un atributo con cierto patrón, que podía encontrarse con una expresión regular.

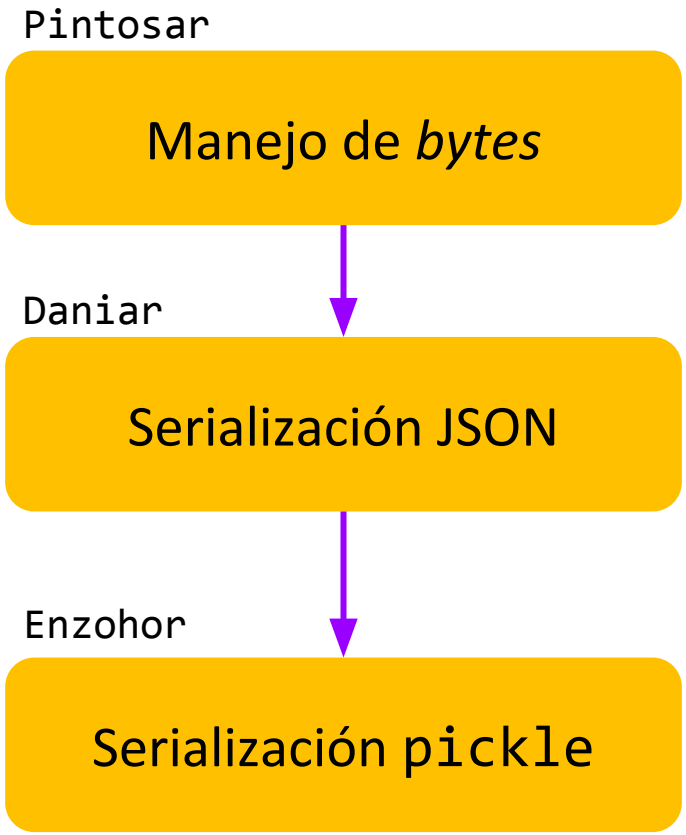


Manejo de *bytes* (Pintosar)

La idea era cargar bytes corrompidos de un archivo y seguir cierto algoritmo para alterar los bytes chunk por chunk para obtener la cadena de bytes original. ¡Esta última luego debía guardarse para verse como una imagen!

```
def reparar_comunicacion(ruta):
    # Se cargan los bytes corrompidos
    with open(ruta, 'rb') as archivo:
        bytes_corrompidos = archivo.read()
    bytes_limpios = bytearray()
    for indice in range(0, len(bytes_corrompidos), 16):
        chunk = bytes_corrompidos[indice : indice + 16]
        ...
        # Se procesa el chunk y se agregan a bytes_limpios
        ...
    # Se guardan los bytes procesados
    with open('Docengelion.bmp', 'wb') as archivo:
        archivo.write(bytes_limpios)
```

Diagrama para el *secreto* 👀



Crear la imagen y ver que modelo dice:

“EVA.xdc” -> “Docengelion.bmp”

De los archivos JSON creados, buscar aquel con el modelo encontrado y ver su `nucleo`:

“docent.json” -> “Unidad-**modelo**.json”

Busca el `alma` arreglado del Piloto “Shinji Gonzalez”

“pilotos.magi” -> `class Piloto`

¡Concatena el `nucleo` y `alma` y habrás encontrado el secreto!