Programación Avanzada IIC2233 2023-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Joaquín Tagle - Francisca Cattan

Anuncios

Jueves 28 de septiembre 2023

- 1. Repasaremos dos temas:
 - a. Serialización
 - b. Excepciones
- Hoy tenemos la cuarta
 actividad, se entrega el miércoles
 11 oct 2023 a las 20:00hrs.
- 3. ¡Se viene la semana de receso!
- 4. El 17 de octubre es el *midterm*

Manejo de bytes

Manejo de bytes

I/O: Forma de interactuar con un programa.

En el contexto de archivos, todo archivo se guarda en un computador como *bytes*.

Un programa es capaz de leer al nivel más mínimo de un archivo y manipularlo para generar más información.

Formato de almacenamiento de información de más bajo nivel.







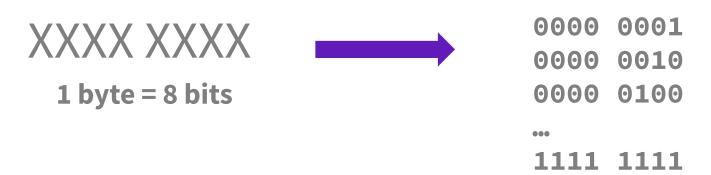
- Entero entre 0 y 255 (2**8 1)
- Hexadecimal entre 0 y FF
- Un literal (a, b, ...)



Formato de almacenamiento de información de más bajo nivel.

En Python los bytes se representan con el objeto de tipo bytes.

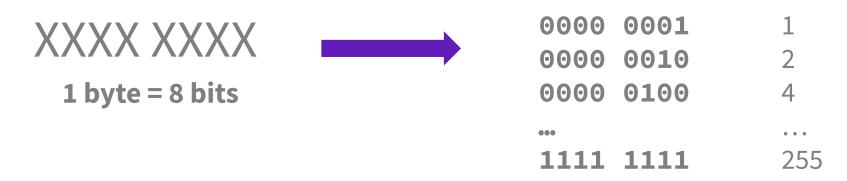
```
my_bytes = b"\x63\x6c\x69\x63\x68\xe9"
print(my_bytes) # b'clich\xe9'
```



Formato de almacenamiento de información de más bajo nivel.

En Python los bytes se representan con el objeto de tipo bytes.

```
my_bytes = b"\x63\x6c\x69\x63\x68\xe9"
print(my bytes) # b'clich\xe9'
```



Formato de almacenamiento de información de más bajo nivel.

En Python los bytes se representan con el objeto de tipo bytes.

```
my_bytes = b"\x63\x6c\x69\x63\x68\xe9"
print(my_bytes) # b'clich\xe9'
```





 0000
 0001

 0000
 0010

 0000
 0100

... 1111 1111 Caracteres ASCII



Formato de almacenamiento de información de más bajo nivel.

En Python los bytes se representan con el objeto de tipo bytes.

```
my_bytes = b"\x63\x6c\x69\x63\x68\xe9"
print(my_bytes) # b'clich\xe9'
```

XXXX XXXX		0000	0001	01
		0000	0010	02
1 byte = 8 bits		0000	0100	04
		•••		• • •
		1111	1111	FF

Bytearray

- Forma de hacer mutable nuestros *bytes*
- Arreglos (listas) de bytes.

```
ba = bytearray(b'\x15\xa3')
ba[0]
                             # 21
ba[1]
                              # 163
ba[0:1] = b' \x44'
ba
                             # bytearray(b'\x44\xa3')
len(ba)
                             # 2
max(ba)
                             # 163
ba[::-1]
                              # bytearray(b'\xa3\x44')
ba.zfill(4)
                             # bytearray(b'00\x44\xa3')
bytearray(b'\x00\x00') + ba # bytearray(b'\x00\x00\x44\xa3')
```

Bytearray

- Forma de hacer mutable nuestros *bytes*
- Arreglos (listas) de bytes.

```
ba = bytearray(b'\x15\xa3')
ba[0]
                             # 21
                                      • ojo que
                                      bytearray(b'0') != bytearray(b'\x00')
                             # 163
ba[1]
ba[0:1] = b' \x44'
                             # bytear ord(bytearray(b'0'))
ba
                                      ord(bytearray(b'\x00')) = 0
len(ba)
max(ba)
                             # 163
ba[::-1]
                             # bytearray(b'\xa3\x44')
ba.zfill(4)
                             # bytearray(b'00\x44\xa3')
bytearray(b'\x00\x00') + ba # bytearray(b'\x00\x00\x44\xa3')
```

Serialización

Serialización de objetos

La serialización consiste en tener una **manera particular de guardar** *bytes*, de manera que estos puedan ser interpretados de manera inconfundible (por el mismo programa, otro programa o humanos).

En Python utilizamos dos módulos para hacer esto:

- pickle: Formato de Python, eficiente en almacenamiento, pero no es leíble y puede ser inseguro al deserializar.
- **json:** Formato interoperable y leíble, pero ineficiente en almacenamiento.

Cómo se ven pickle y JSON

```
b'\x80\x03}q\x00(X\n\x00\x00\x00first nameq\
x01X\x06\x00\x000\x00Alexisq\x02X\t\x00\x00\x
001ast nameq\x03X\x08\x00\x000\x005\xc3\xa1nc
hezq\x04X\t\x00\x00\x00birthdateq\x05X\n\x00
x00\x001988-12-19q\x06X\x06\x00\x00\x000\x000
htq\x07G?\xfb\n=p\xa3\xd7\nX\x04\x00\x00\x00
clubq\x08\q\t(X\x04\x00\x00\x00nameq\nX\x0f\
x00\x00\x00Inter de
MI1\xc3\xa1nq\x0bX\x07\x00\x00\x00foundedq\x
\x0f\x00\x00\x00Ni\xc3\xb1o
Maravillag\x0fX\x15\x00\x00\x00La Ardilla de
Atacamag\x10eX\n\x00\x00\x00girlfriendg\x11N
X\x07\x00\x00\x00injuredq\x12\x88u.'
```

```
"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
```

Con strings: dumps y loads

```
import pickle
                                             import json
tupla = ("a", 1, 3, "hola")
                                            tupla = ("a", 1, 3, "hola")
serializacion = pickle.dumps(tupla)
                                             serializacion = json.dumps(tupla)
print(serializacion)
                                            print(serializacion)
print(type(serializacion))
                                            print(type(serializacion))
print(pickle.loads(serializacion))
                                            print(json.loads(serializacion))
> b' \times 80 \times 03(X \times 01 \times 00 \times 00 \times 00 = [...]' > ["a", 1, 3, "hola"]
> <class 'bytes'>
                                             > <class 'str'>
                                             > ['a', 1, 3, 'hola']
> ('a', 1, 3, 'hola')
```

Con archivos: dump y load

```
import json
import pickle
                                              lista = [1, 2, 3, 7, 8, 3]
lista = [1, 2, 3, 7, 8, 3]
                                              with open("mi lista.bin", 'w') as file:
with open("mi lista.bin", 'wb') as file:
                                                  json.dump(lista, file)
   pickle.dump(lista, file)
                                              with open("mi lista.bin", 'r') as file:
with open("mi lista.bin", 'rb') as file:
                                                  lista cargada = ison.load(file)
    lista cargada = pickle.load(file)
                                              print(f"¿Las listas son iguales? {lista ==
print(f"¿Las listas son iguales? {lista ==
                                              lista cargada}")
lista cargada}")
                                              print(f"¿Las listas son el mismo objeto?
print(f"¿Las listas son el mismo objeto?
                                              {lista is lista cargada}")
{lista is lista cargada}")
                                              > ¿Las listas son iguales? True
> ¿Las listas son iguales? True
                                              > ¿Las listas son el mismo objeto? False
> ¿Las listas son el mismo objeto? False
```

Personalización en pickle: set y get state

```
class Persona:
   # ...
   def getstate (self):
       a serializar = self.__dict__.copy()
       # Lo que retornemos será serializado por pickle
       return a serializar
   def setstate (self, state):
       # self. dict_ contendrá los atributos deserializados
       self. dict = state
```

... y en json: JSONEncoder y object_hook

```
class PersonaEncoder(json.JSONEncoder):
    def default(self, obj):
                                         def hook persona(diccionario):
        # Serializamos instancias
                                             # Recibe objetos de JSON
        diccionario = {
                                             # Podemos retornar lo que queramos
            "nombre": obj.nombre,
                                              instancia = Persona(**diccionario)
            # ...
                                              return instancia
        return diccionario
                                         json string = ...
instancia = Persona(...)
                                          instancia = json.loads(
json string = json.dumps(
                                             json string,
    instancia,
                                              object hook=hook persona,
    cls=PersonaEncoder,
```

Excepciones

Mensajes de error

Hasta ahora nos hemos encontrado con mensajes de error al realizar ciertas operaciones no permitidas o utilizar métodos de forma incorrecta.

Excepciones Built-in

BaseException

SyntaxError

IndentationError

EOFError

NameError

ZeroDivisionError

IndexError

KeyError

AttributeError

TypeError

ValueError

Cada una tendrá una forma distinta de **capturar**, tratar y **manejar** la excepción.

<u>... y mas en la</u> documentación



raise

Dada cierta condición, podríamos diseñar el **levantar** un tipo de excepción particular y añadir un mensaje adicional que informe al usuario sobre el error. Estas excepciones **interrumpen el flujo** del programa.

> AttributeError: El largo del mensaje es menor a 10

raise

Dada cierta condición, podríamos diseñar el **levantar** un tipo de excepción particular y añadir un mensaje adicional que informe al usuario sobre el error. Estas excepciones **interrumpen el flujo** del programa.

> AttributeError: El largo del mensaje es menor a 10

raise

Dada cierta condición, podríamos diseñar el **levantar** un tipo de excepción particular y añadir un mensaje adicional que informe al usuario sobre el error. Estas excepciones **interrumpen el flujo** del programa.

> ValueError: El largo del mensaje es menor a 10

try y except

> ValueError: Error al leer archivo JSON.

Si una excepción fue levantada durante la ejecución, podemos **atraparla** y manejarla. Lo que queremos **intentar** se encapsula dentro del bloque **try:** try: # Intentaremos leer un archivo JSON que no existe with open("archivo json.json", "r") as json_file: data = json.load(json file) except FileNotFoundError: # Atrapamos FileNotFoundError y levantamos otra excepcion raise ValueError ("Error al leer archivo JSON.") > FileNotFoundError: [Errno 2] No such file or directory: 'archivo json.json' > During handling of the above exception, another exception occurred:

try y except

También podemos asignar la instancia del objeto error a una variable, y usar sus atributos o métodos. Dependiendo del manejo del error, **podemos continuar la ejecución del código.**

```
try:
   # Intentaremos leer un archivo JSON que no existe
   with open("archivo json.json", "r") as json file:
       data = json.load(json file)
except FileNotFoundError as e:
    # Imprimimos un mensaje y el código continúa
    print(f"Error {e. class . name } al leer archivo JSON: {e.filename}")
print("...sigamos")
> Error FileNotFoundError al leer archivo JSON: archivo json.json
> ...sigamos
```

Programación Avanzada IIC2233 2023-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Joaquín Tagle - Francisca Cattan

Comentarios Actividad 4 > Tomen agua