Programación Avanzada
IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattan

Anuncios

Jueves 10 de octubre 2024

1. Hoy tenemos la **quinta (y última)** actividad, se entrega a las 23:59 hrs.
2. Hagan push de su T3 👀.

Hablemos de la ETC

Evaluación Temprana de Cursos

Positivos

- ✅ Modalidad *Flipped Classroom*
- ✅ Subir actividades junto con contenidos
- ✅ Explicaciones en clase didácticos con ejemplos reales
- ✅ Ambiente del curso

Negativos

- ❌ Modalidad *Flipped Classroom*
- ❌ Dificultades con los *tests*
- ❌ Extensión enunciados
- ❌ Carga académica/salto de dificultad desde IIC1103

Compromisos

- 👍 Estudiar con anticipación
- 👍 Empezar las tareas/actividades con tiempo
- 👍 Hacer preguntas (presencial/online)
- 👍 Participar en clase

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 y manipular directamente los *bytes* que representan un archivo.

Bytes

Formato de almacenamiento de información de más bajo nivel.

1 bit = valor 0 o 1

XXXX XXXX

1 byte = 8 bits

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

Tabla de conversión

Bytes

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

1 byte = 8 bits

```
0000 0001
0000 0010
0000 0100
    ⋮
1111 1111
```

Bytes

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

1 byte = 8 bits

```
0000 0001     1
0000 0010  =  2
0000 0100     4
     ⋮        ⋮
1111 1111     255
```

Bytes

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

1 byte = 8 bits

0000	0001	=	01
0000	0010		02
0000	0100		04
⋮			⋮
1111	1111		FF

Bytes

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

1 byte = 8 bits

```
0000 0001
0000 0010
0000 0100
    ⋮
1111 1111
```

Caracteres ASCII

Tabla ASCII

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]
ba[1]
ba[0:1] = b'\x44'
ba
len(ba)
max(ba)
ba[::-1]
ba.zfill(4)
bytearray(b'\x00\x00') + ba
```

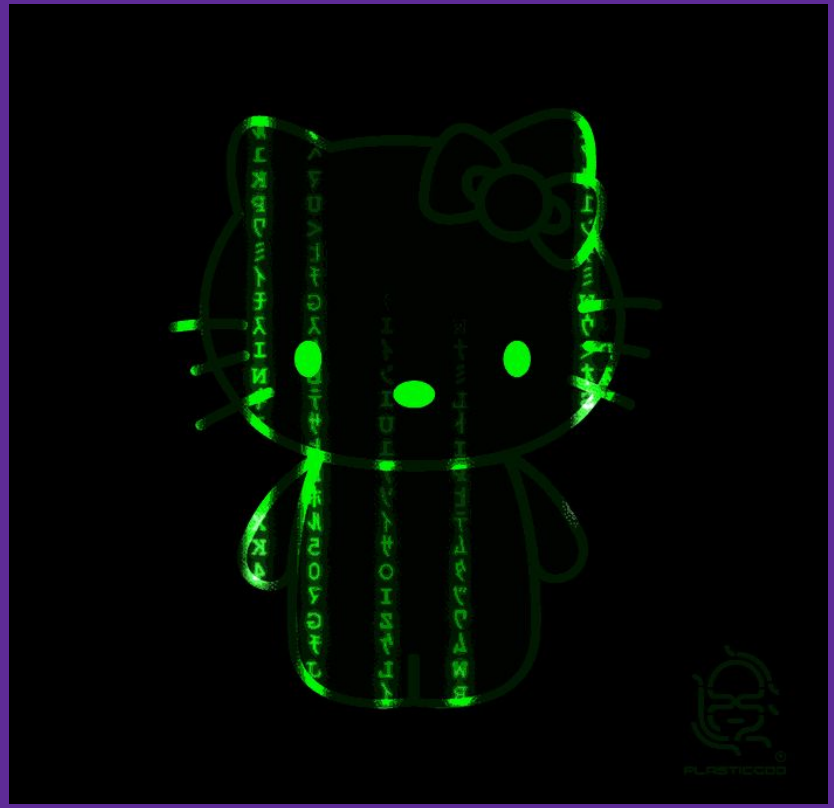
Ojo que... 👀

```
bytearray(b'0') != bytearray(b'\x00')

ord(bytearray(b'0')) => 48
ord(bytearray(b'\x00')) => 0
```

```
# bytearray(b'\xa3\x44')
# bytearray(b'00\x44\xa3')
# 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.

Serialización de objetos

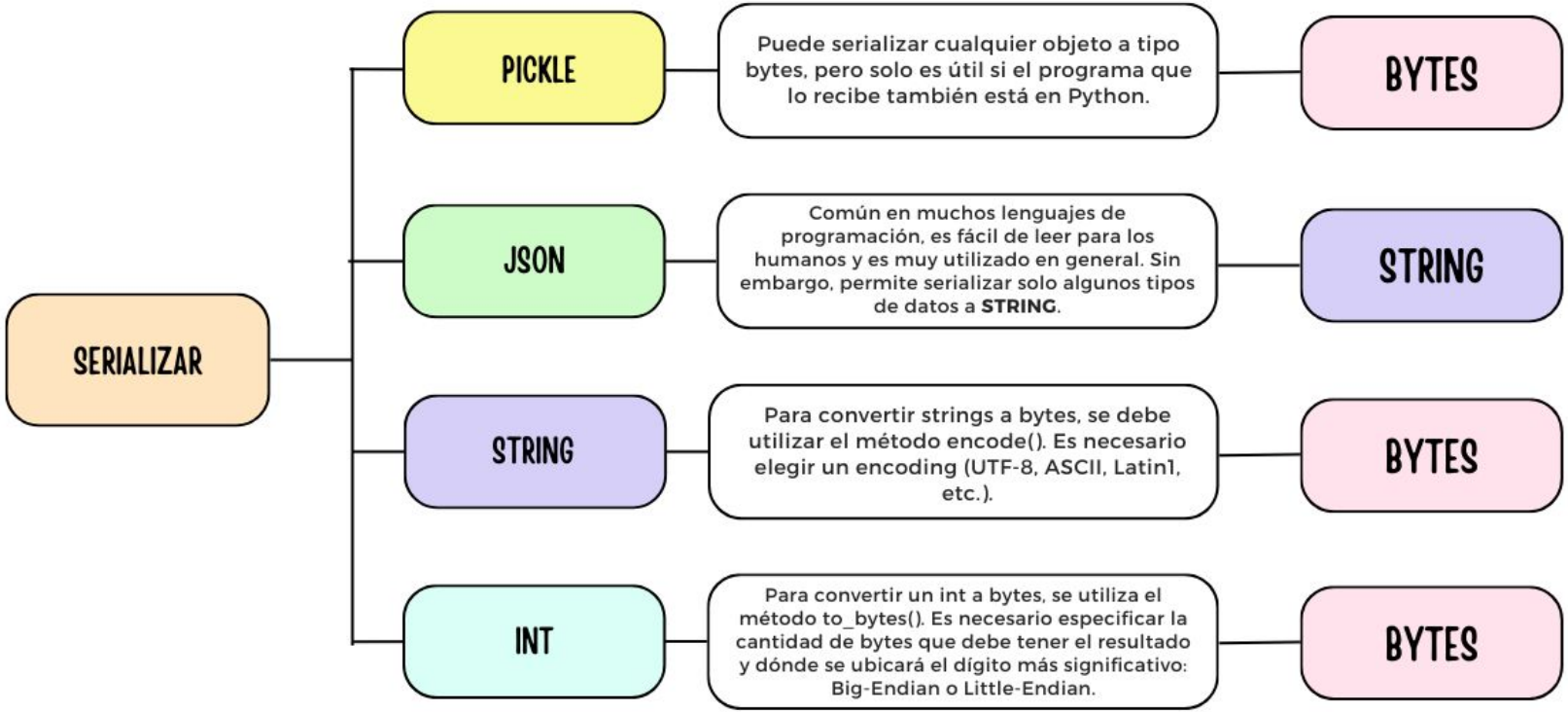


Foto de resumen creada por Diego Toledo (2024-2)

Con *strings*: dumps y loads

```python
import pickle

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

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

```
> b'\x80\x04\x95\x14\x00\x00\x00[...]'
> <class 'bytes'>
> ('a', 1, 3, rulo)
```

```python
import json

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

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

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

Con archivos: dump y load

```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
```

```python
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
```

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):
        # Serializamos instancias
        diccionario = {
            "nombre": obj.nombre,
            # ...
        }
        return diccionario

instancia = Persona(...)
json_string = json.dumps(
    instancia,
    cls=PersonaEncoder,
)
```

```
def hook_persona(diccionario):
    # Recibe objetos de JSON
    # Podemos retornar lo que queramos
    instancia = Persona(**diccionario)
    return instancia

json_string = ...
instancia = json.loads(
    json_string,
    object_hook=hook_persona,
)
```

Posibles errores de serialización

_pickle.UnpicklingError

```
data = pickle.dumps('gato')
print(data)
fragmento = data[:4]
try:
    pickle.loads(fragmento)
except pickle.UnpicklingError as e:
    print("Formato inválido:", e)
```

JSONDecodeError

```
json_data = '''
    { "nombre": "Cachirulo", "edad": 4,
    '''
try:
    data = json.loads(json_data)
    print(data)
except json.JSONDecodeError as e:
    print("Formato inválido:", e)
```

Ambos errores se deben a que la librería de serialización usada no considera el valor de la variable `data` como un formato válido para ser des-serializado.

Pregunta de Evaluación Escrita

Tema: Serialización (Examen 2024-2)

14. Respecto a la codificación en *bytes*, es **correcto** afirmar que:

A) Solo puede realizarse utilizando JSON y pickle.
B) Al codificar un *string* de largo n, siempre se obtendrán n *bytes*.
C) No usar el mismo *encoding* para codificar y decodificar siempre levanta una excepción.
D) Un mismo *byte* puede codificar diferentes caracteres.
E) Existen *bytes* que no tienen una representación numérica.

Pregunta de Evaluación Escrita

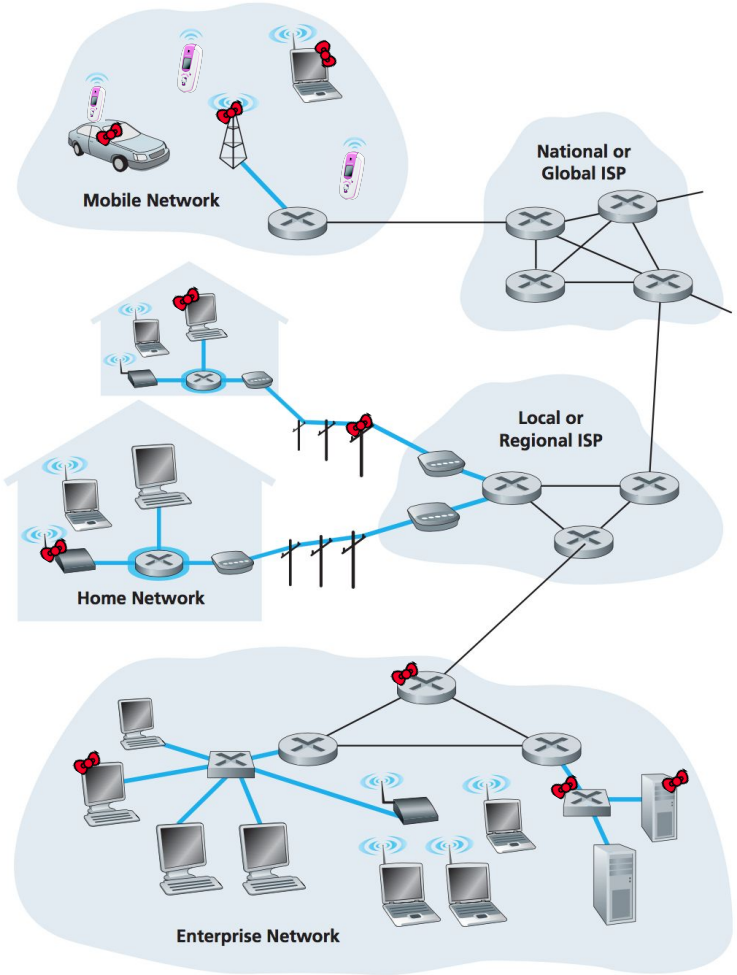
Tema: Serialización (Examen 2024-2)

14. Respecto a la codificación en *bytes*, es **correcto** afirmar que:

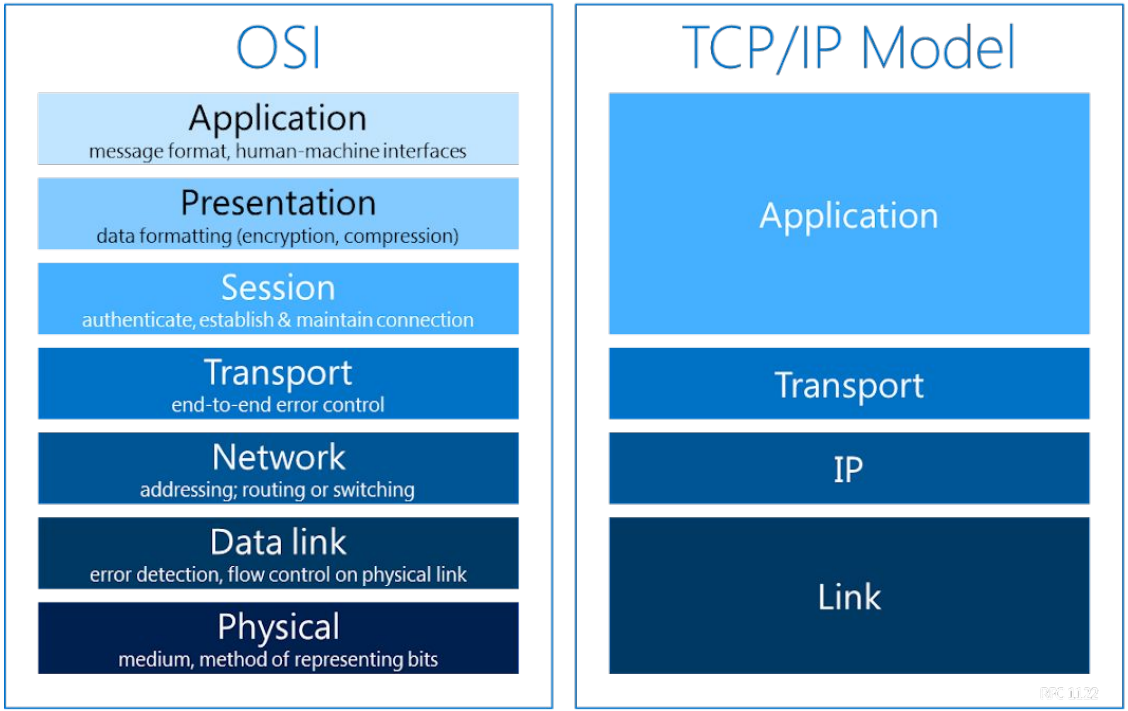
A) Solo puede realizarse utilizando JSON y pickle.
B) Al codificar un *string* de largo n, siempre se obtendrán n *bytes*.
C) No usar el mismo *encoding* para codificar y decodificar siempre levanta una excepción.
D) Un mismo *byte* puede codificar diferentes caracteres.
E) Existen *bytes* que no tienen una representación numérica.

Networking

Redes


Mobile Network
National or
Global ISP
Local or
Regional ISP
Home Network
Enterprise Network

Encapsulamiento



Puertos e IPs

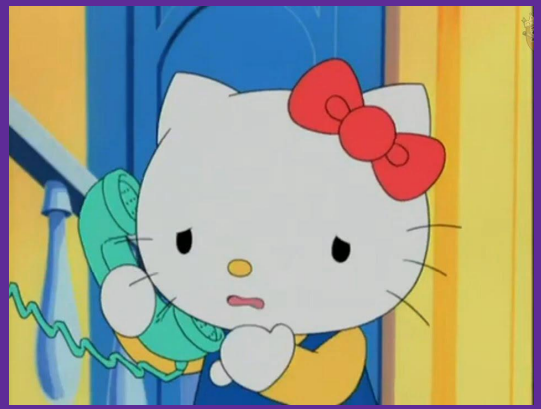
- **IP:** Identifica al computador
- **Puerto:** Identifica a la aplicación

Protocolos de transporte
TCP (*Transmission Control Protocol)*

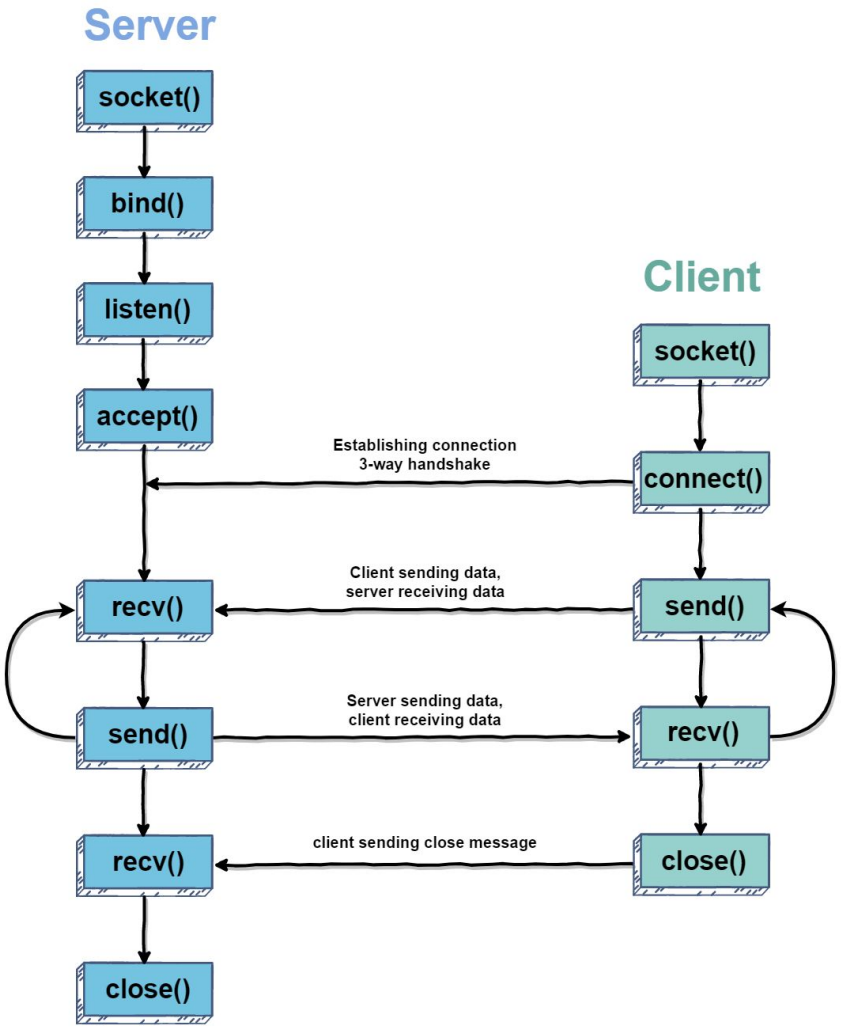
- Orientado a conexión.
  - Requiere de *handshake* (establecimiento de conexión) antes de transferir datos.
- Verifica que todos los paquetes que se envían sean recibidos por el destinatario.
- Lo anterior hace que sea más lento por *overhead.*
  - Otros protocolos pueden ser más rápidos, pero no necesariamente aseguran que toda la información se envíe correctamente.
- Reserva *buffers* en *sender* y en *receiver.*
- Algunos casos de uso: Navegación web, emails, transferencia de archivos.

Existen más protocolos, pero en este curso nos enfocaremos en TCP

Arquitectura Cliente - Servidor y *Sockets*



Flujo de comunicación con *sockets* entre Cliente y Servidor



Pregunta de Evaluación Escrita

Tema: Networking (Examen 2023-2)

4. Según los contenidos vistos en el curso, ¿en cuál o cuáles de estos casos es recomendable el uso de TCP?

   I. Subida de cambios de un repositorio local a uno remoto en Github.
   II. Una videollamada entre dos o más participantes.
   III. Edición de documentos de texto en la nube entre dos o más personas, como Word Online o Google Docs.

   A) Solo I
   B) Solo II
   C) I y II
   D) I y III
   E) I, II y III

Pregunta de Evaluación Escrita

Tema: Networking (Examen 2023-2)

4. Según los contenidos vistos en el curso, ¿en cuál o cuáles de estos casos es recomendable el uso de TCP?

   I. Subida de cambios de un repositorio local a uno remoto en Github.
   II. Una videollamada entre dos o más participantes.
   III. Edición de documentos de texto en la nube entre dos o más personas, como Word Online o Google Docs.

   A) Solo I
   B) Solo II
   C) I y II
   D) I y III
   E) I, II y III

Programación Avanzada
IIC2233 2024-2

Hernán Valdivieso - Daniela Concha - Francisca Ibarra - Lucas Van Sint Jan - Francisca Cattan