Ayudantía Examen 2

Contenidos

- Simulación
- Threads
- Serialización
- Networking

Simulación

¿Cómo enfrentar un problema de simulación?

1. Definir *entorno:* contiene tiempo universal
2. Definir *variables:* sus atributos de tiempo se calculan en función del tiempo universal
3. Analizar todos los eventos y dependencias
4. Programar...

Simulación

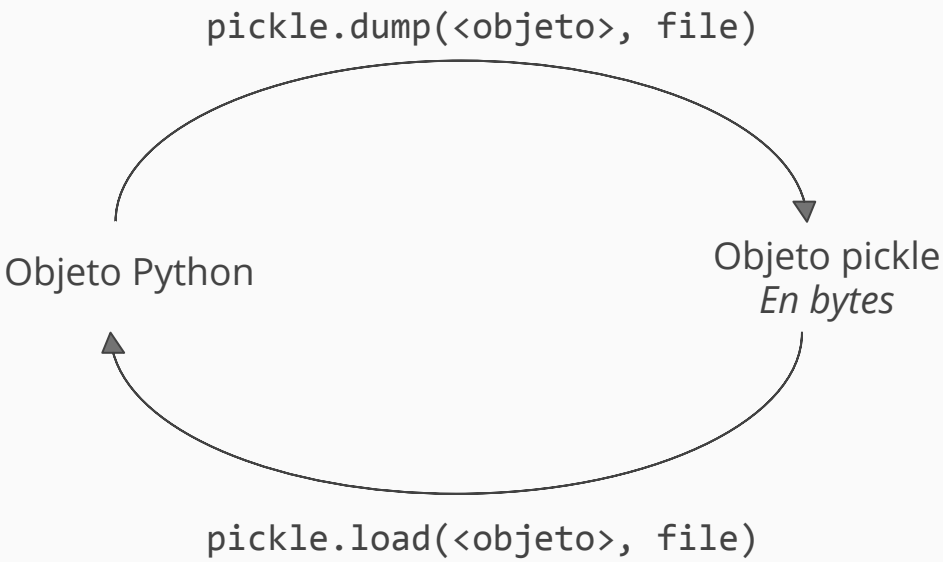
MIENTRAS la lista de eventos no esté vacía y el tiempo de simulación no termine:

1. tomar un evento desde el principio de la lista de eventos
2. avanzar el tiempo de simulación al tiempo del evento
3. simular el evento

Threads

- Join: hace que el programa principal se detenga hasta que termine el thread
- Locks: para coordinar threads que quieren acceder a un mismo recurso
- Daemon=true: para que termine el thread cuando se acaba el programa principal

Serialización I/O



Serialización I/O

```
class Persona:
    def __init__(self, nombre, edad):
        self.nombre = nombre
        self.edad = edad
        self.mensaje = "No pasa nada"

    def __getstate__(self):
        nueva = self.__dict__.copy()
        nueva.update({"mensaje" : "Me están serializando!!"})
        return nueva

    def __setstate__(self, state):
        print("Objeto recién des-serializado, seteando su estado...\n")
        state.update({"nombre" : state["nombre"] + " deserializado"})
        self.__dict__ = state
```

Serialización I/O

Serialización I/O

```
class PersonaEncoder(json.JSONEncoder):
       def default(self, obj):
            # Creamos una serialización personalizada para el tipo de objeto Persona
            if isinstance(obj, Persona):
                return {'Persona_id': obj.idn,
                        'nombre': obj.nombre,
                        'edad': obj.edad,
                        'estado_civil': obj.estado_civil,
                        'fecha_nac' : datetime.now().year - obj.edad
            return super().default(obj)


p1 = Persona()
json_string = json.dumps(p1, cls = PersonaEncoder)
```

Servidor

```
class Servidor:


    def __init__(self, usuario):
        self.usuario = usuario
        self.host = '127.0.0.1'
        self.port = 3490
        self.s_servidor = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.s_servidor.bind((self.host, self.port))
        self.s_servidor.listen(1)
        self.cliente = None
        self.aceptar()
```

Cliente

```
class Cliente:


    def __init__(self, usuario):
        self.usuario = usuario
        self.host = '127.0.0.1'
        self.port = 3490
        self.s_cliente = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            self.s_cliente.connect((self.host, self.port))
            recibidor = threading.Thread(target=self.recibir_mensajes, args=())
            recibidor.daemon = True
            recibidor.start()
        except socket.error:
            print("No fue posible realizar la conexión")
            sys.exit()
```

Servidor

Cliente

```
def aceptar(self):
    cliente_nuevo, address = self.s_servidor.accept()
    self.cliente = cliente_nuevo
    thread_cliente = threading.Thread(target=self.recibir_mensajes, args=())
    thread_cliente.daemon = True
    thread_cliente.start()
```

Networking

Servidor

```
def recibir_mensajes(self):
        while True:
            data = self.cliente.recv(2048)
            mensaje = data.decode('utf-8')
            print(mensaje)

    def enviar(self, mensaje):
        msj_final = self.usuario + ": " + mensaje
        self.cliente.send(msj_final.encode('utf-8'))
```

Cliente

```
def recibir_mensajes(self):
        while True:
            data = self.s_cliente.recv(1024)
            print(data.decode('utf-8'))


    def enviar(self, mensaje):
        msj_final = self.usuario + ": " + mensaje
        self.s_cliente.send(msj_final.encode('utf-8'))
```

Ejercicio 2

La sociedad secreta de hackers Anonymouse ha desarrollado un nuevo tipo de virus que está causando estragos en la Interném, el virus fue bautizado como “Exmn-IIC2233”. Ahora necesitamos sus habilidades expertas para analizar el comportamiento de este virus y ver si existe alguna posible solución. Se sabe que el virus se puede generar aleatoriamente al momento en que un cliente decide enviar un mensaje a otra persona(el mensaje se transforma en virus) con una probabilidad de 70% (está bien hecho el virus), cuando éste llega a otro cliente, el sistema de antivirus permite que el mensaje se cheque a sí mismo para ver si es un virus o no, si lo es, el mensaje tiene un 25% de probabilidades de autodestruirse, si falla el cliente no puede volver a recibir mensajes y automáticamente se comenzarán a reenviar periodicamente virus a otros clientes (podemos apreciar la alta tecnología illuminati que permite que estas cosas pasen). Por simplicidad puede asumir que los mensajes los redirecciona el Servidor al azar entre los clientes y que todos los clientes se conectan automáticamente. Lo importante son cómo están modeladas las clases