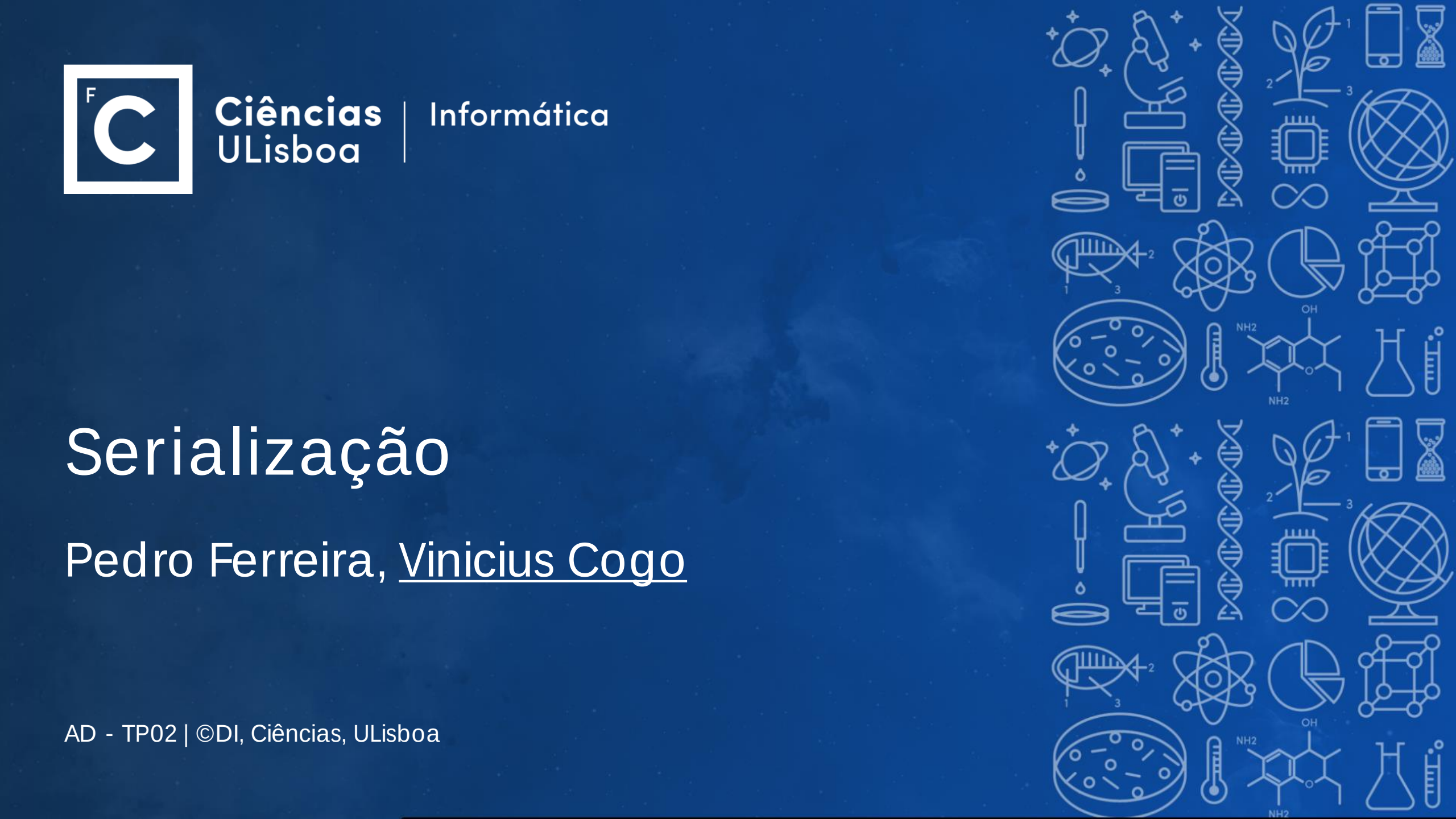


Ciências ULisboa | Informática

Serialização

Pedro Ferreira, Vinicius Cogo

Tópicos

1. Serialização
2. Serialização de Valores Primitivos em Python
3. Serialização de Objetos em Python
4. Cliente-Servidor em Python com Serialização

1. Serialização

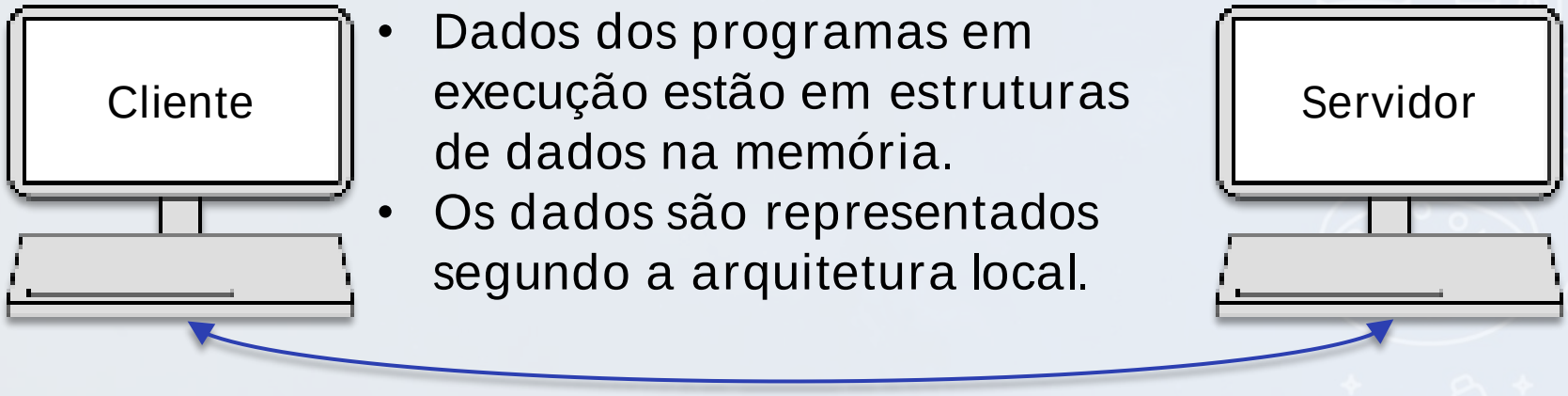
- O método `sock.sendall` transmite apenas bytes

- Na aula anterior (cliente-servidor):
  - Cliente: `sock.sendall(mensagem.encode('utf-8'))`
  - Servidor: `sock.sendall(mensagem.decode('utf-8'))`

- Os métodos **`encode`** e **`decode`** pertencem à classe string
- Convertem de (resp. para) string para (resp. de) bytes

- Porém, como converter um objeto qualquer para/de bytes?

1. Serialização

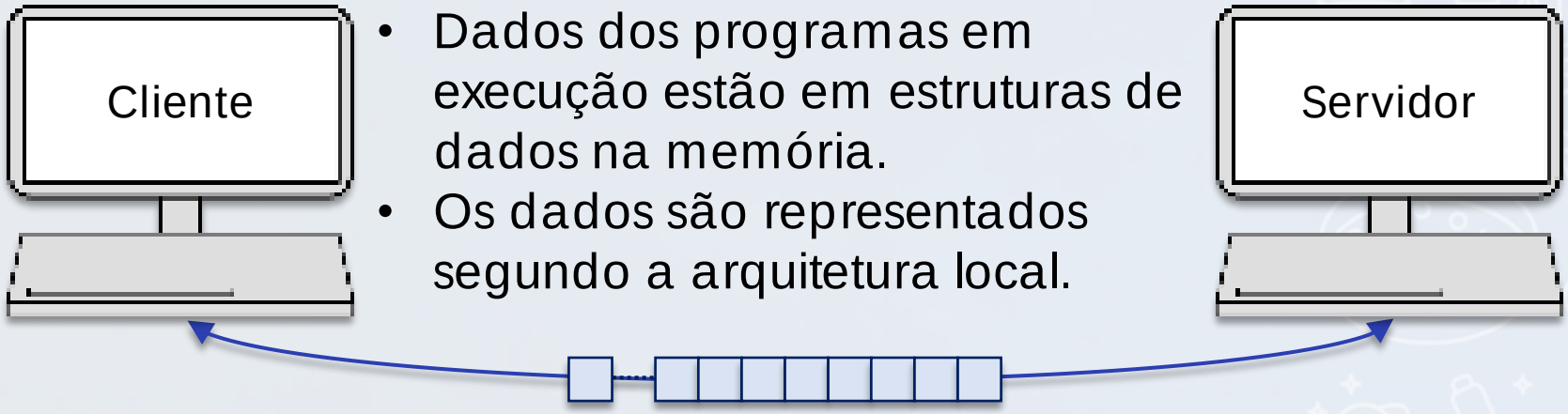


- Dados dos programas em execução estão em estruturas de dados na memória.
- Os dados são representados segundo a arquitetura local.

pessoa = [‘Hatem Mostafa’, **18, 01, 1972, 33**]

pessoa = [‘Hatem Mostafa’, **18, 01, 1972, 33**]

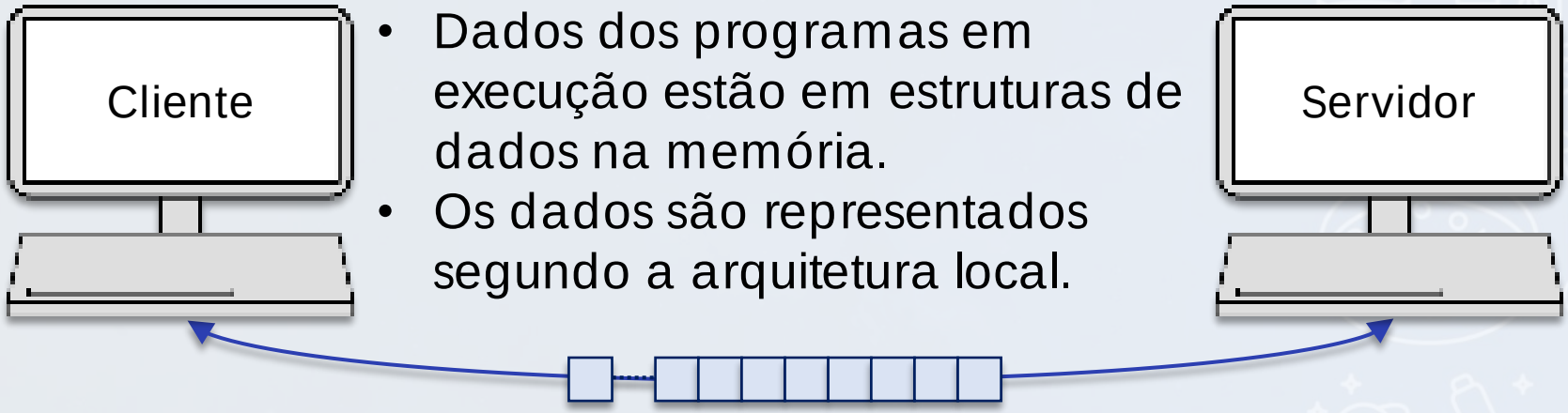
1. Serialização



- Dados dos programas em execução estão em estruturas de dados na memória.
- Os dados são representados segundo a arquitetura local.

Dados nas sockets são sequências de bytes!

1. Serialização



- Dados dos programas em execução estão em estruturas de dados na memória.
- Os dados são representados segundo a arquitetura local.

Dados nas sockets são sequências de bytes!

big-endian

representação em vírgula flutuante

little-endian

códigos de representação de carateres: UTF-**8, ASCII, Unicode, ...**

1. Serialização

```
pessoa = [‘Hatem Mostafa’, 18, 01, 1972, 33]
```

MARSHALLING (Serialização)

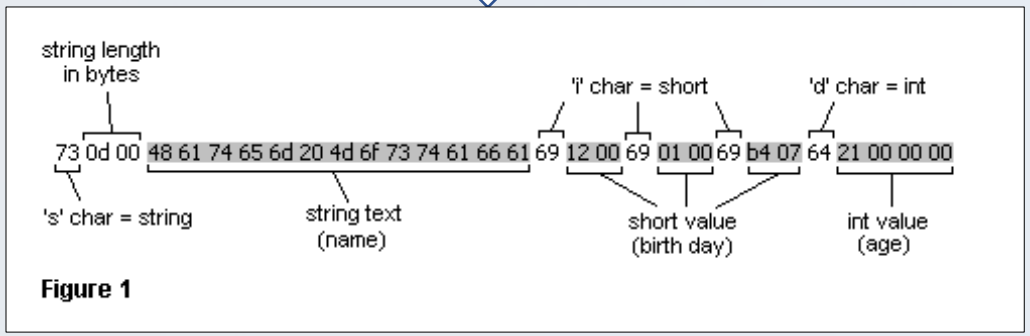


Figure 1

(Transmissão)

UNMARSHALLING (Desserialização)

```
pessoa = [‘Hatem Mostafa’, 18, 01, 1972, 33]
```

1. Serialização

SERIALIZAÇÃO

Conversão de dados estruturados e valores primitivos numa representação externa, segundo uma sequência de bytes.

DESSERIALIZAÇÃO

Partindo da representação externa, geração de valores primitivos da arquitetura local e reconstrução das estruturas de dados originais.

1. Serialização

- Abordagens generalistas
  - Disponíveis para várias linguagens.
    Permitem SDs heterogéneos quanto às linguagens de desenvolvimento
  - Exemplos:
    - Implementação manual: int = 4 bytes em little-endian (e.g., módulo *struct* in Python)
    - JSON, XML, YAML: Formatos baseados em texto, praticamente para qualquer linguagem
    - CORBA CDR: Ada, C/C++, Java, Lisp, Python, Ruby, ...
    - Google Protocol Buffers: Java, C++, Python, JavaNano, Ruby
    - Apache Thrift: C, C++, Go, Java, JavaScript, PHP, Python, ...
- Abordagens específicas
  - Exclusivas das linguagens de programação
  - Exemplos:
    - Python: *pickle* module
    - Java: Java Object serialization

2. Serialização de Valores Primitivos em Python

- Módulo *struct – Interpret bytes as packed binary data*: serialização/desserialização de valores primitivos em Python

Para serializar um valor primitivo para uma *sequência de bytes*

```
val = 321
val_bytes = struct.pack('i',val) #i=formato
sock.sendall(val_bytes)
```

Para desserializar um valor primitivo de uma *sequência de bytes*

```
val_bytes = sock.recv(4)
val = struct.unpack('i',val_bytes)
```

3. Serialização de Objetos em Python

- Módulo *pickle – Python Object Serialization*: serialização/desserialização de objetos Python

- Serialização/desserialização = *pickling/unpickling*

- Dois módulos: pickle e cPickle
  - Interfaces idênticas
  - As sequências de bytes produzidas e os objetos reconstruídos, são iguais
  - cPickle é implementado na linguagem C
    - Muito mais eficiente
    - Não suporta subclasses das classes que define

3. Serialização de Objetos em Python

- Formato dos dados serializados é específico do Python

- Existem 3 versões do protocolo:
  - 0: é baseada em texto legível.
    Ocupa mais memória. É usada por omissão.
  - 1: formato binário (versões anteriores a 2.3)
  - 2: formato binário mais eficiente (pós 2.3)

- Versões 0 e 1 são compatíveis
  com versões de Python anteriores à 2.3

3. Serialização de Objetos em Python

Para serializar um objeto numa *string*

```
obj_string = pickle.dumps(objeto, protocolo)
```

protocolo: opcional, 0 por omissão. -1 para utilizar a versão mais recente.

Para desserializar um objeto de uma *string*

```
objeto = pickle.loads(obj_string)
```

3. Serialização de Objetos em Python

Para serializar um objeto num ficheiro

```
pickle.dump(objeto, ficheiro, protocolo)
```

ficheiro: objecto *file* do Python, ou objecto com método `write()` que aceite uma *string*

Para desserializar um objeto de um ficheiro

```
objeto = pickle.load(ficheiro)
```

4. Cliente-Servidor em Python com Serialização

Cliente revisitado, usando serialização

```
import socket as s, pickle as p, struct

… # Criação da socket de ligação com o servidor

msg = ['Hatem Mostafa', 18, 01, 1972, 33] # lista
msg_bytes = pickle.dumps(msg)
msg_size_bytes = struct.pack('i',len(msg_bytes))

conn_sock.sendall(msg_size_bytes)
conn_sock.sendall(msg_bytes)
…
resp_size_bytes = conn_sock.recv(4)
resp_size = struct.unpack('i',resp_size_bytes)[0]

resp_bytes = conn_sock.recv(resp_size)
msg = pickle.loads(resp_bytes)

…
```

4. Cliente-Servidor em Python com Serialização

Servidor revisitado, usando serialização

```
import socket as s, pickle as p, struct

… # Criação da socket de escuta e ligação com o cliente

req_size_bytes = conn_sock.recv(4)
req_size = struct.unpack('i’,req_size_bytes)[0]

req_bytes = conn_sock.recv(req_size)
req = pickle.loads(req_bytes)

…
resp_bytes = pickle.dumps(resp)
resp_size_bytes = struct.pack('i',len(resp_bytes))

conn_sock.sendall(resp_size_bytes)
conn_sock.sendall(resp_bytes)
…
```

Bibliografia

- Brandon Rhodes and John Goerzen. Foundations of Python Network Programing, second edition, Apress.

- Python online documentation: struct — Interpret bytes as packed binary data. (https://docs.python.org/3/library/struct.html)

- Python online documentation: pickle — Python object Serialization. (https://docs.python.org/3/library/pickle.html)

- Python online documentation: socket — Low-level networking interface. (https://docs.python.org/3/library/socket.html)