MessagePack

Alunos:

- Guilherme da Silva Medeiros
- Rafael Teles Espindola

1 - Escolher uma tecnologia para especificação e codificação de tipos de dados

- O tipo de codificação de dados escolhido foi: **MessagePack**

2 - Especificar as mensagens usando essa tecnologia

-

O protocolo TFTP define cinco tipos de mensagens:

- **RRQ**
  - String: opcode1
  - String: filename
- **WRQ**
  - String: opcode2
  - String: filename
- **DATA**
  - String: opcode3
  - int: block
  - String: mode
- **ACK**
  - String: opcode4
  - int: block

- **Error**

A nova versão do TFTP precisará também destas novas mensagens:

- **LIST**: fazer listagem de uma pasta. Seu formato deve ser: *opcode* (int 16 bits com valor 10), *caminho* (string)
    - Resposta de **LIST**: contém a listagem da pasta. Seu formato é dado por opcode (int 16 bits com valor 11) e lista de *Elementos*. Cada *Elemento* é um valor de um destes dois tipos:
        - *Arquivo*: representa um arquivo, e é formado por: nome (string), tamanho(int 32 bits)
        - *Pasta*: representa uma pasta, sendo formado por: nome(string)
    - Resposta de **LIST** pode também ser uma mensagem **Error**
- **MKDIR**: cria uma pasta. Seu formato deve ser: *opcode* (int 16 bits com valor 12), *caminho* (string)
    - Resposta de **MKDIR**: deve ser uma mensagem **Error**, com *ErrorCode*:
        - *0*: se sucesso, e assim *ErrMsg* deve ser vazia
        - *demais valores*: um código de erro, com *ErrMsg* contendo uma breve descrição
- **MOVE**: renomeia ou remove arquivos. Seu formato deve ser: *opcode* (int 16 bits com valor 13), *nome_original* (string), *novo_nome* (string)
    - Se *novo_nome* for vazio, o arquivo deve ser removido
    - Resposta de **MOVE**: deve ser uma mensagem **Error**, com *ErrorCode*:
    - *0*: se sucesso, e assim *ErrMsg* deve ser vazia*demais valores*: um código de erro, com *ErrMsg* contendo uma breve descrição

2.1 - Tutorial como realizar a especificação, e as mensagens especificadas, com detalhamento suficiente para que possa ser reproduzido

Com base nesse exemplo conseguimos dentro do dicionário criar qualquer tipo de conjunto de variáveis.

Neste exemplo temos a definição de “data” sendo uma lista, string e até mesmo outro dicionário.

```
import msgpack

# Define data
data = {
    "a list": [1, 42, 3.141, 1337, "help"],
    "a string": "bla",
    "another dict": {"foo": "bar", "key": "value", "the answer": 42},
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(data)
    outfile.write(packed)

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data == data_loaded)
```

2.2 - Vídeo tutorial

Em anexo

3 - Escrever dois programas de teste

Exemplo RRQ Codificador

```
import msgpack

# Define data
rrq = {
    "opcode": 1,
    "fileaname": "File/to/path",
    "mode": "netascii",
}

# Write msgpack file
with open("data.msgpack", "wb") as outfile:
    packed = msgpack.packb(rrq)
    outfile.write(packed)
```

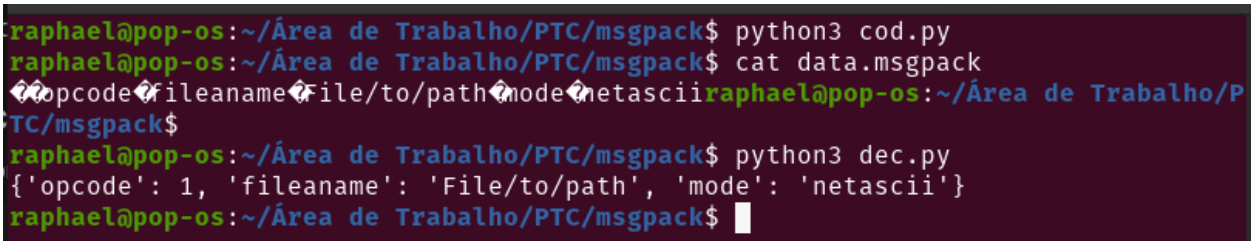
Exemplo WRQ Decodificador

```
import msgpack

# Read msgpack file
with open("data.msgpack", "rb") as data_file:
    byte_data = data_file.read()

data_loaded = msgpack.unpackb(byte_data)
print(data_loaded)
```

Exemplo em uso:



https://s3-us-west-2.amazonaws.com/secure.notion-static.com/03990300-b792-480e-8433-545c87e5f242/codificadores_decodificadores.zip