RELATÓRIO DE ESTRUTURA DE ARQUIVOS

**ARCHIVER - Pyver**

André L. R. Estevam - 166348

Kaulitz Guimarães - 188530

Mayara Ramos - 184517

Pedro Artico - 185545

# 

# Sumário

[Sumário 2](#_Toc498717188)

[Breve descrição 3](#_Toc498717189)

[Resolução do problema 3](#_Toc498717190)

[Estrutura do archive (exemplo) 3](#_Toc498717191)

[Atendimento dos casos de uso 4](#_Toc498717192)

[C1. Criar um archive com base em uma lista de arquivos informados. 4](#_Toc498717193)

[Iniciando o programa com a criação do archiver com o nome teste 4](#_Toc498717194)

[Código : 4](#_Toc498717195)

[C2. Listar os nomes dos arquivos armazenados em um archive. 6](#_Toc498717196)

[Listando os arquivos que estão no archiver pelo comando list e listando todos os arquivos, inclusive os deletados pelo comando listall: 6](#_Toc498717197)

[Código: 6](#_Toc498717198)

[C3. Extrair um arquivo de um archive, dado o nome do arquivo (sem remover esse arquivo dentro do archive). 8](#_Toc498717199)

[Extraindo um arquivo no archiver pelo comando un <nome\_do\_arquivo> 8](#_Toc498717200)

[Código: 8](#_Toc498717201)

[C4. Inserir um arquivo em um archive já criado. 10](#_Toc498717202)

[Inserindo um arquivo no archiver pelo comando ar <nome\_do\_arquivo> 10](#_Toc498717203)

[Código: 10](#_Toc498717204)

[C5. Remover um arquivo de um archive, dado o nome do arquivo 14](#_Toc498717205)

[Removendo um arquivo pelo comando rm <nome\_do\_arquivo> 14](#_Toc498717206)

[Código: 14](#_Toc498717207)

[C6. Menu do programa: 15](#_Toc498717208)

[Menu do programa : 15](#_Toc498717209)

[Código: 15](#_Toc498717210)

[C7. Recuperando um arquivo deletado: 20](#_Toc498717211)

[Recuperando um arquivo do archiver pelo comando rec <nome\_do\_arquivo> 20](#_Toc498717212)

[Código 20](#_Toc498717213)

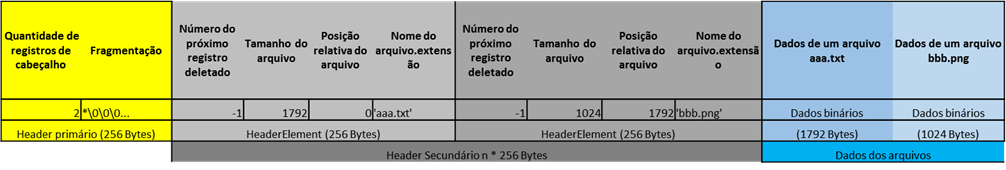
# Breve descrição

A linguagem de programação utilizada para a codificação foi Python. Todos os Casos de Uso foram cumpridos, e, além disso, como extra, foi implementado: Um menu e a opção de recuperar um arquivo deletado. (vide C6 e C7)

O tamanho dos registros de cabeçalho será mantido independentemente da plataforma utilizada. O código foi feito de forma que os registros não fiquem em dois clusters ao mesmo tempo, para evitar diversos acessos a disco, prejudicando o desempenho do código.

# Resolução do problema

## Estrutura do *archive* (exemplo)

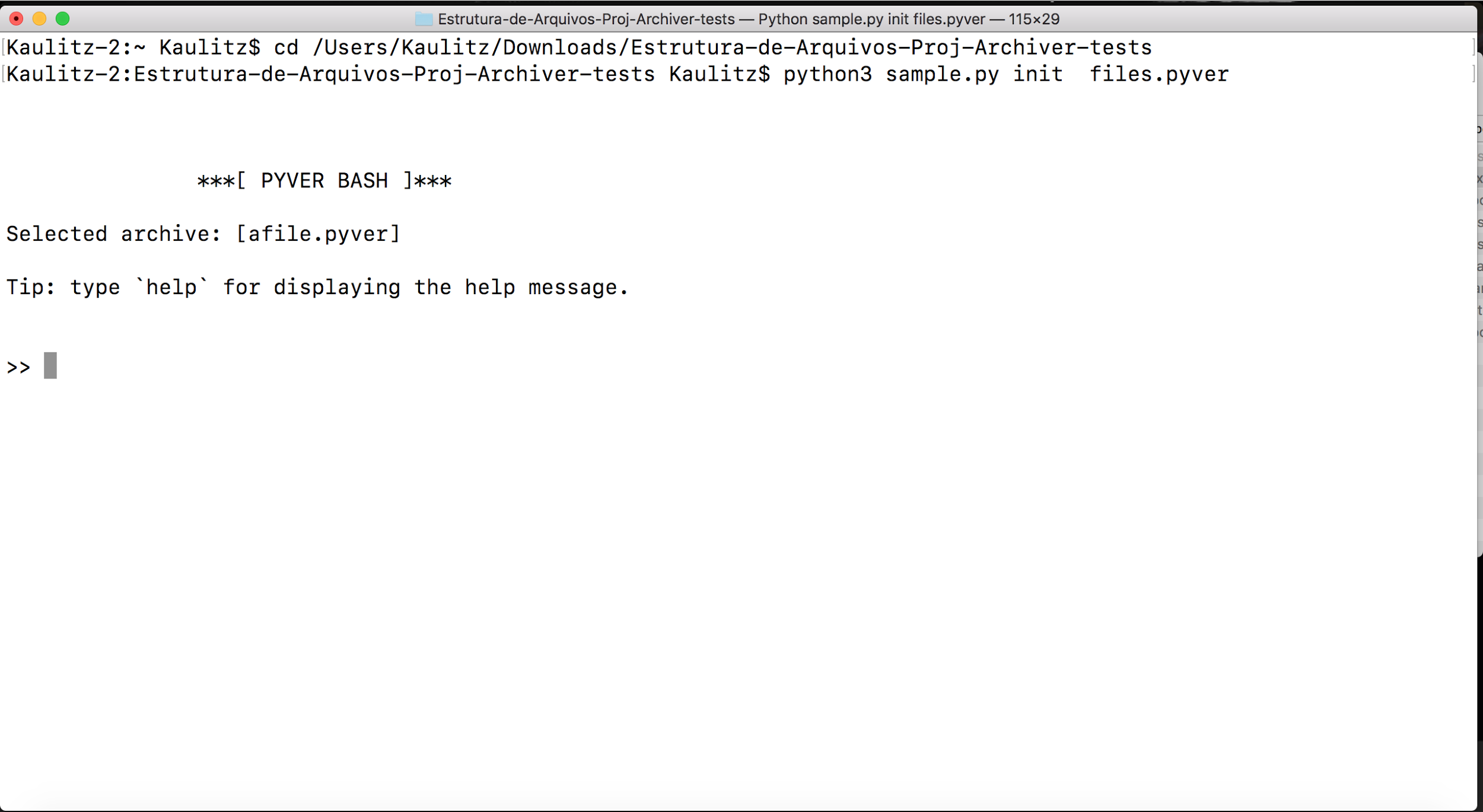


* **Parte 1: Header primário:** auxilia na leitura do cabeçalho secundário
  + Tamanho: fixo (256 Bytes)
  + Dados armazenados
    - Int: quantidade de registros no cabeçalho
* **Parte 2: Header secundário:** possui elementos que identificam os arquivos dentro do *archive*
  + Tamanho: variável (n \* 256 Bytes)
  + Dados armazenados
    - **Elemento do cabeçalho:** identifica unicamente um arquivo do archive
      * Tamanho: fixo (256 Bytes)
      * Dados armazenados:
        + Int: RRN (Número Relativo de Registro) do próximo registro deletado
        + long long: tamanho do *file* armazenado
        + long long: indicador da posição do *file* no *archive*
        + char[]: nome do *file*
* **Parte 3: Dados dos *files:*** dados dos arquivos armazenados colocados de forma sequencial e sem separadores
  + Tamanho: variável
  + Dados armazenados:
    - Dados de arquivos em formato binário

# Atendimento dos casos de uso

## C1. Criar um archive com base em uma lista de arquivos informados.

### Iniciando o programa com a criação do archiver com o nome teste



Uso do programa: no terminal (necessidade de ter o Python3 instalado)

Iniciar um novo archive:

python pyver.py init myarchivename

Abrir um archive já criado:

python pyver.py open myarchivename.pyver

O archive selecionado será aberto no bash do pyver, a mensagem de ajuda é exibida com o comando help

### Código :

class Packer:

"""

Manages read and write operations into the archive

"""

def \_\_init\_\_(self, archiveName):

self.archiveName = archiveName

self.headerPat = '=i q q ' + str(maxfnameSize) + 's'

self.bytesPerCluster = 4096

def initFile(self):

"""

Initiates a new archive

:return: nothing

"""

try:

file = open(self.archiveName, 'wb')

packedData = struct.pack('=I' + str(256 - sizeOfUnsignedInteger) + 's', 0, b'\*')

file.write(packedData)

file.close()

except:

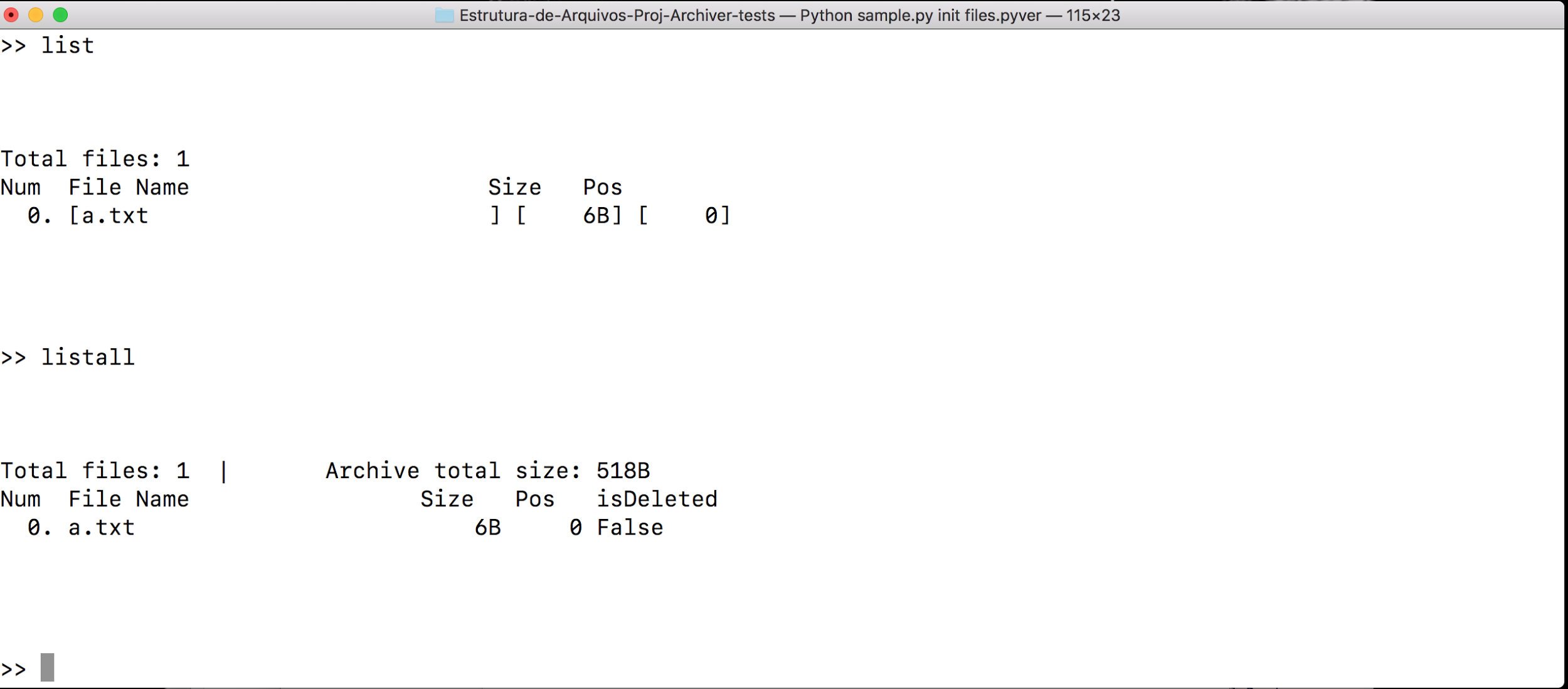
print('Error initializing the archive')

input()

exit(-1)

## C2. Listar os nomes dos arquivos armazenados em um archive.

### Listando os arquivos que estão no archiver pelo comando list e listando todos os arquivos, inclusive os deletados pelo comando listall:



### Código:

def describeNonDeletedFiles(self):

"""

Prints a description of the non-deleted files

:return: nothing

"""

totalFiles = self.pk.elemsInHeader()

headers = self.pk.retrieveHeader()

count = 0

print('\n\n')

print('Total files: {:}'.format(totalFiles))

print('{:4s} {:30s} {:6s} {:5s}'.format('Num', 'File Name', 'Size', 'Pos'))

for header in headers:

if header.nextdeleted < 0:

print('{:3}. [{:30s}] [{:5}B] [{:5}]'

.format(

count,

str(header.fnameStr[0:30]),

header.fsize,

header.fPos,

)

)

count += 1

print('\n')

def getUserInput(self):

return str(input('\n\n>> ')).split(' ')

def describeAllFiles(self):

"""

Prints a description of all the files in the archive

:return: nothing

"""

totalFiles = self.pk.elemsInHeader()

headers = self.pk.retrieveHeader()

count = 0

print('\n\n')

totalSize = os.path.getsize(self.archiveName)

print('Total files: {}\t|\tArchive total size: {}B'.format(totalFiles, totalSize))

print('{:4s} {:25s} {:6s} {:5s} {:5s}'.format('Num', 'File Name', 'Size', 'Pos', 'isDeleted'))

for header in headers:

print('{:3}. {:25s} {:5}B {:5} {:5s}'

.format(

count,

str(header.fnameStr[0:30]),

header.fsize,

header.fPos,

str(header.nextdeleted > -1)

)

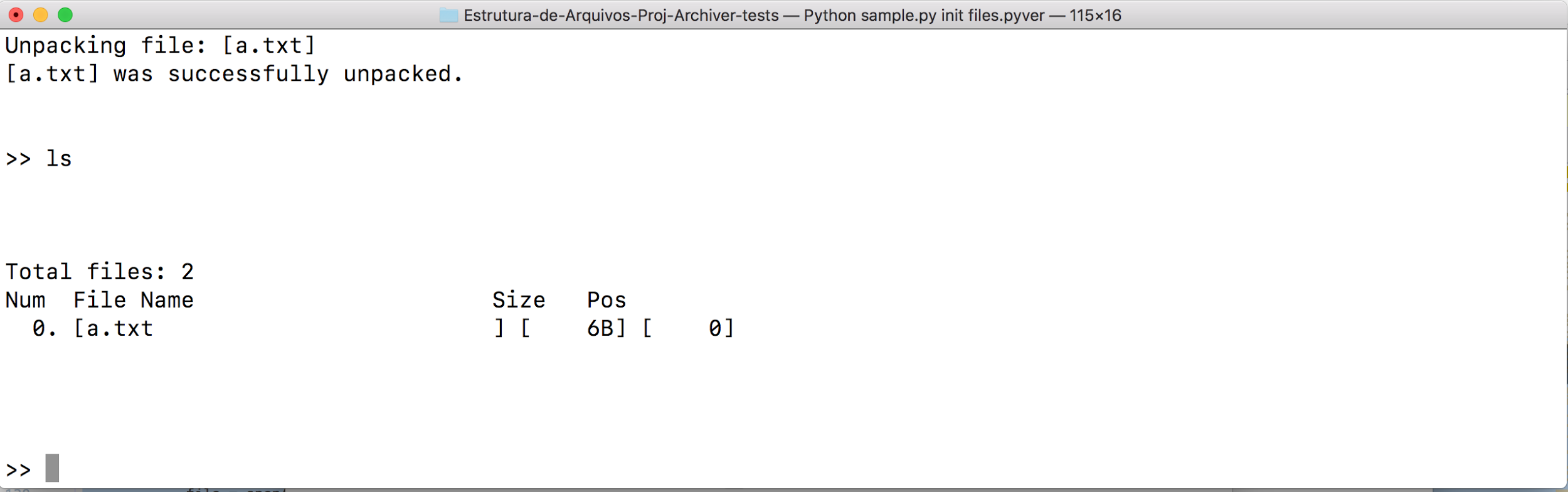
)

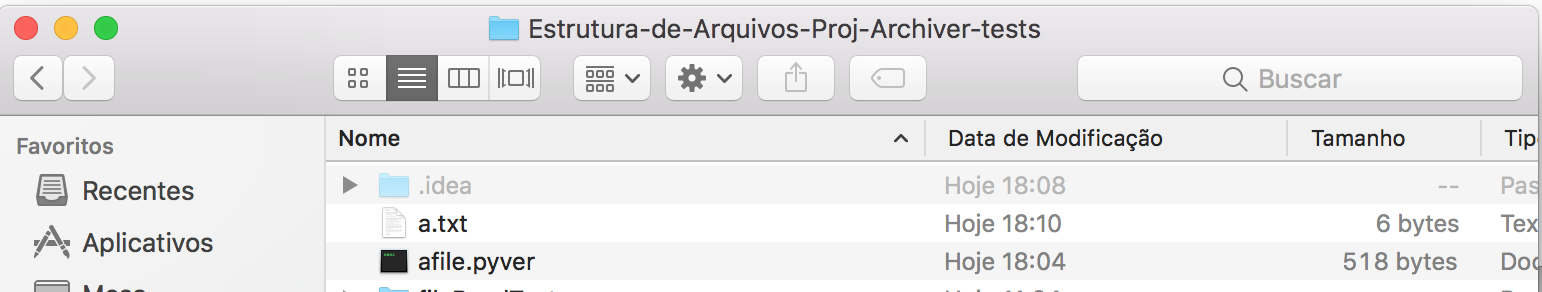
count += 1

print('\n')

## C3. Extrair um arquivo de um archive, dado o nome do arquivo (sem remover esse arquivo dentro do archive).

### Extraindo um arquivo no archiver pelo comando un <nome\_do\_arquivo>





### Código:

Como decisão de aumento de performance foi definido que a leitura do *file* a partir do *archive* será feita de 4KB em 4KB, evitando ter de ler o mesmo cluster duas vezes ou trazer todo o arquivo para a memória.

def unpack(self, headerElem):

"""

Unpacks a file from the archive using its name (and directories)

:param headerElem: a instance of HeaderElement that describes the file to be unpacked

:return: True only if the file is marked as not deleted

"""

if headerElem.nextdeleted >= 0:

return False

relativePos = headerElem.fPos

absolutePos = Util(self.archiveName).relativeToAbsolutePosition(relativePos)

fsize = headerElem.fsize

fname = headerElem.fnameStr

if '\\' in fname:

os.makedirs(os.path.dirname(fname), exist\_ok=True)

bytesPerLoop = self.bytesPerCluster

loopCount = 0

try:

with open(self.archiveName, 'rb') as archive, open(fname, 'wb') as outFile:

archive.seek(absolutePos)

while True:

data = archive.read(bytesPerLoop) # read 4K

if data == b'': # end of file reached

break

outFile.write(data)

loopCount += 1

print('Read\Written: [{:^10} of {:^10}] [{:^3}%]'.format(

str(bytesPerLoop \* loopCount) + ' B',

str(fsize) + ' B',

'100' if (bytesPerLoop \* loopCount) \* 100 // fsize > 100 else (

bytesPerLoop \* loopCount) \* 100 // fsize

, end='\r'))

return True

except Exception:

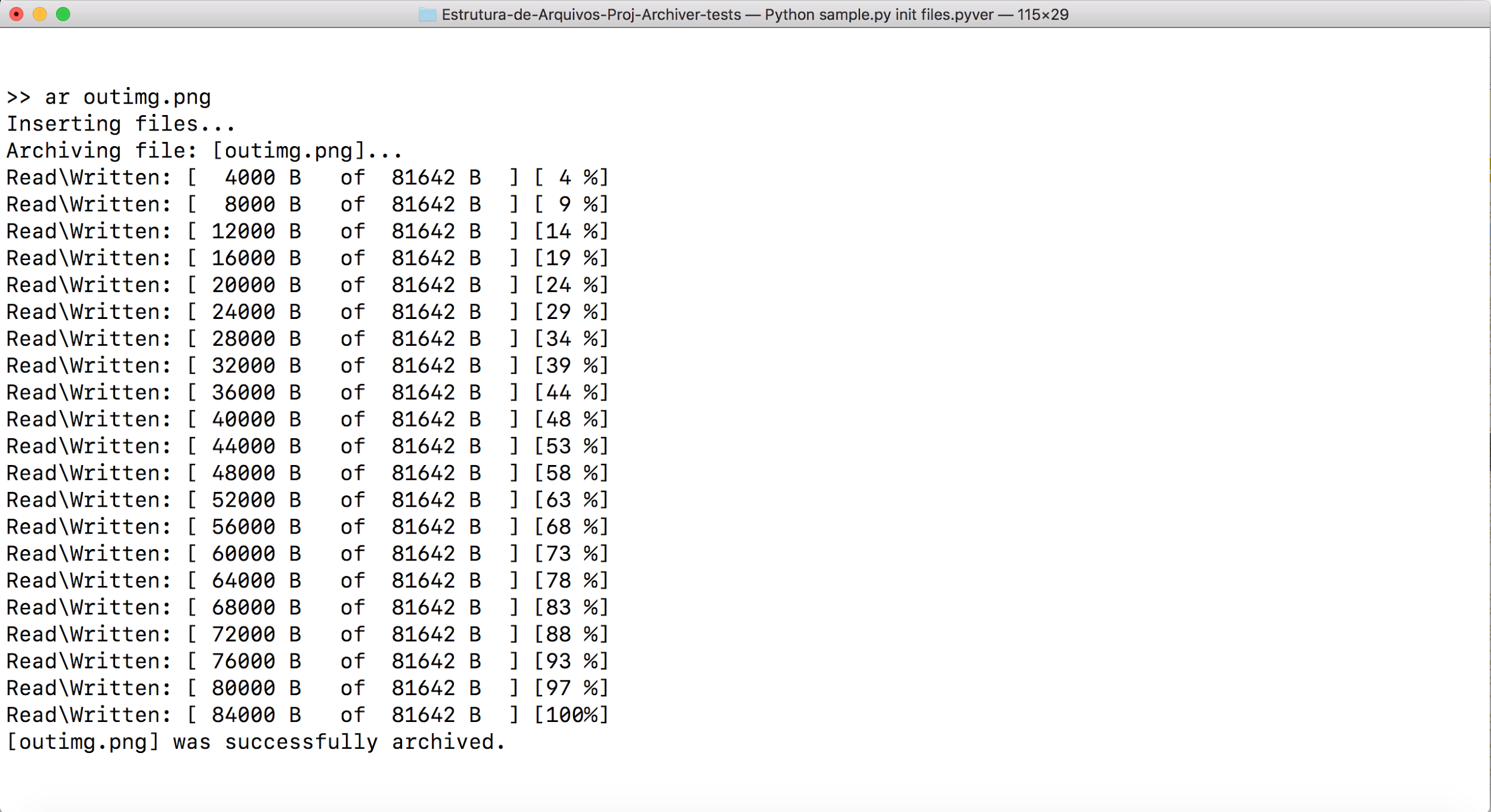
raise Exception

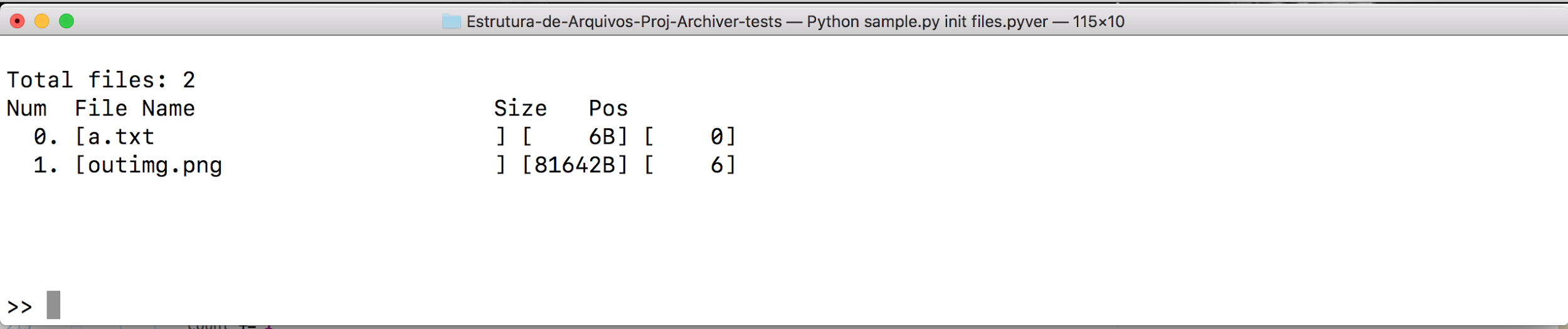
## 

## 

## C4. Inserir um arquivo em um archive já criado.

### Inserindo um arquivo no archiver pelo comando ar <nome\_do\_arquivo>





### Código:

Tanto o tamanho primeiro cabeçalho quanto o tamanho de cada elemento do segundo cabeçalho serão limitados a 256 bytes para evitar ter de acessar dois clusters para ler apenas um registro, por exemplo.

A quantidade de caracteres para os nomes de arquivos varia de acordo com a plataforma. Como cada elemento do cabeçalho secundário foi definido com o tamanho fixo de 256 Bytes, com cada elemento do cabeçalho contendo:

* Um campo int para armazenar o RRN do próximo registro deletado (pode ser utilizado para implementar uma estratégia de reaproveitamento de espaços como *worst-fit* mas não foi implementada na prática);
* Um campo long long para armazenar o tamanho do arquivo inserido no *archiver;*
* Um campo long long para armazenar a posição de início do arquivo dentro do *archive;*
* Um campo char[] para armazenar o nome do arquivo.

Como o tamanho de cada tipo de variável muda de acordo com a plataforma (já que para gravar os registros de cabeçalho no arquivo foi utilizado o módulo struct que imita os structs da linguagem de programação C [[Documentação da biblioteca struct do Python](https://docs.python.org/2/library/struct.html#struct-alignment)]) o tamanho máximo do tamanho do arquivo pode ser calculado com:

A leitura de um *file* e a gravação no archive é feita de 4KB em 4KB (evita acessar o mesmo custer duas vezes para pegar partes diferentes do registro).

class Util:

"""

This is a utility class

"""

def \_\_init\_\_(self, archiveName):

self.archiveName = archiveName

self.pk = Packer(archiveName)

def fsize(self, fname):

"""

Calculate a size of a file by its name or path

:param fname: the name or path of the file

:return: a number representing the quantity of bytes in the given file or -1 if the file cannot be found

"""

if os.path.exists(fname):

return os.path.getsize(fname)

else:

return -1

def relativeToAbsolutePosition(self, relativePos):

"""

Converts a relative position in the archive to the absolute position

:param relativePos: a relative position in the archive

:return: a absolute position in the archive

"""

headerElems = self.pk.elemsInHeader()

return relativePos + miniHeaderSize + (elemSize \* headerElems)

def absoluteToRelativePosition(self, absolutePos):

"""

Converts a absolute position in the archive to the relative position

:param absolutePos: a absolute position in the archive to the absolute position

:return: a relative position in the archive

"""

headerElems = self.pk.elemsInHeader()

return absolutePos - (miniHeaderSize + (elemSize \* headerElems))

def fileExists(self, fname):

"""

Returns true only if the file exists

:param fname: a name or path of a file

:return: True only if the file exists (could be found by the OS)

"""

return os.path.exists(fname)

def filesExists(self, files):

"""

Prints a message if a file of the given list does not exist

:param files: a list of filenames or path

:return: nothing

"""

for file in files:

if not self.fileExists(file):

print('File does not exists: {}'.format(file))

def packAtTheEnd(self, fname):

"""

Packs a file in the end of the archive

:param fname: the name or path to the file

:return: an instance of the HeaderElement that describes the inserted file (must be inserted as well)

"""

fsize = Util(self.archiveName).fsize(fname)

if(fsize <= 0):

return -1

arSize = Util(self.archiveName).fsize(self.archiveName)

pos = Util(self.archiveName).absoluteToRelativePosition(arSize)

rsp = HeaderElement(fsize, pos, fname, -1)

bytesPerLoop = self.bytesPerCluster

loopCount = 0

try:

with open(self.archiveName, 'ab') as archive, open(fname, 'rb') as file:

while True:

data = file.read(bytesPerLoop) # read 4K

if data == b'': # end of file reached

break

archive.write(data)

loopCount += 1

print('Read\Written: [{:^10} of {:^10}] [{:^3}%]'.format(

str(bytesPerLoop \* loopCount) + ' B',

str(fsize) + ' B',

'100' if (bytesPerLoop \* loopCount) \* 100 // fsize > 100 else (

bytesPerLoop \* loopCount) \* 100 // fsize

, end='\r'))

return rsp

except Exception:

raise Exception

def appendToHeader2(self, headElem):

"""

Appends a header element to the archive

:param headElem: an instance of HeaderElement

:return: nothing

"""

headerElemsCount = self.elemsInHeader()

tup = (headElem.nextdeleted, headElem.fPos, headElem.fsize, headElem.fname)

st = struct.Struct(self.headerPat)

packedData = st.pack(\*tup)

self.updtHeaderElemAmount(self.elemsInHeader(), 1)

try:

file = open(self.archiveName, 'rb')

data = file.read()

file.close()

bkPos = miniHeaderSize + headerElemsCount \* elemSize

lst = [data[0:bkPos], packedData, data[bkPos:]]

newData = b''

for dt in lst:

newData += dt

file = open(self.archiveName, 'wb')

file.write(newData)

file.close()

except Exception:

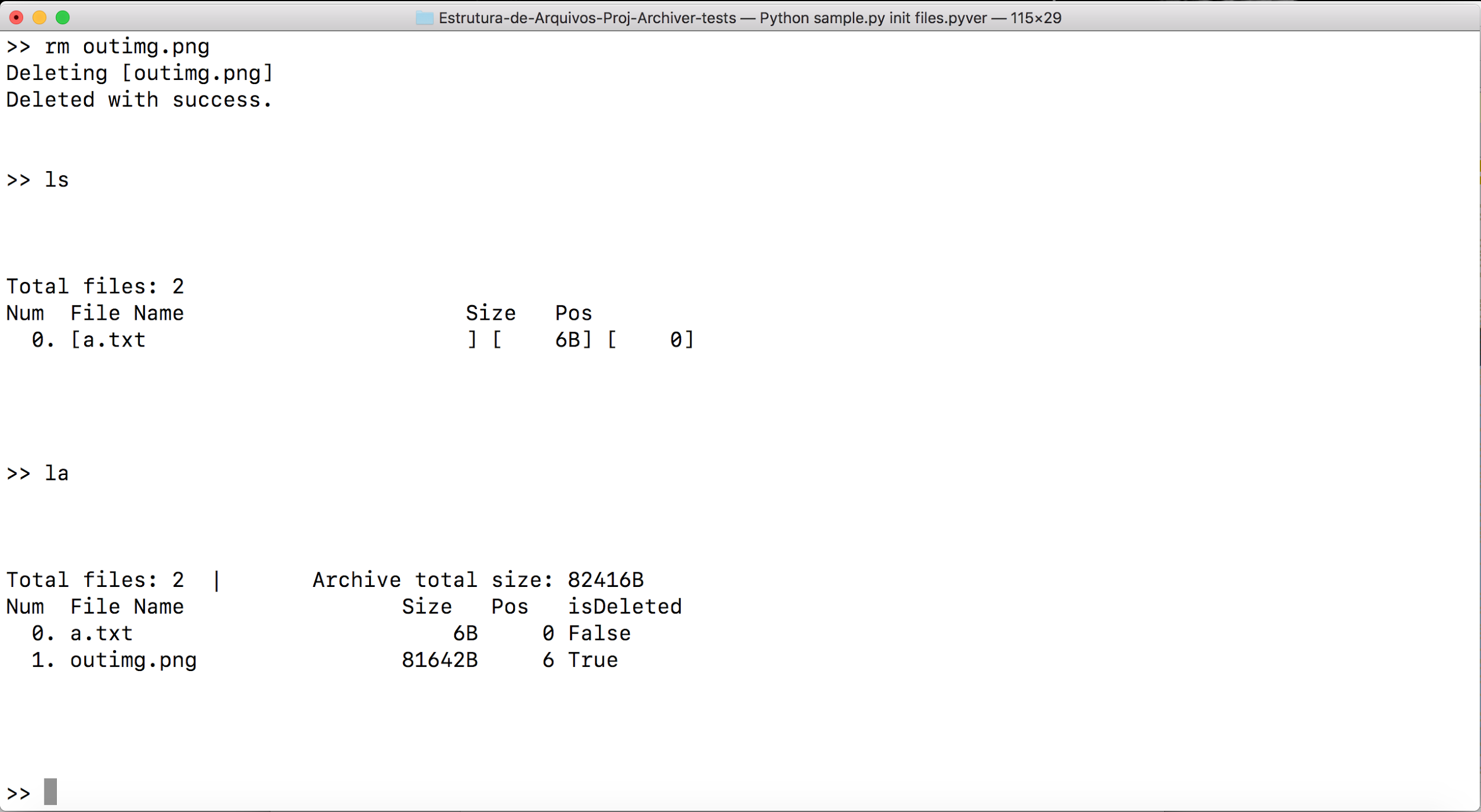
print('Error at appending an element to the header.')

input()

exit(-1)

## C5. Remover um arquivo de um archive, dado o nome do arquivo

### Removendo um arquivo pelo comando rm <nome\_do\_arquivo>



## Código:

def delFile(self, fname):

"""

Marks a file as deleted

:param fname: the name of the file to be marked as deleted in the archive

:return: True only if the filename exists in the archive

"""

headers = self.retrieveHeader()

count = 0

for header in headers:

if header.fnameStr == fname:

header.nextdeleted = count

self.updateHeader(header, count)

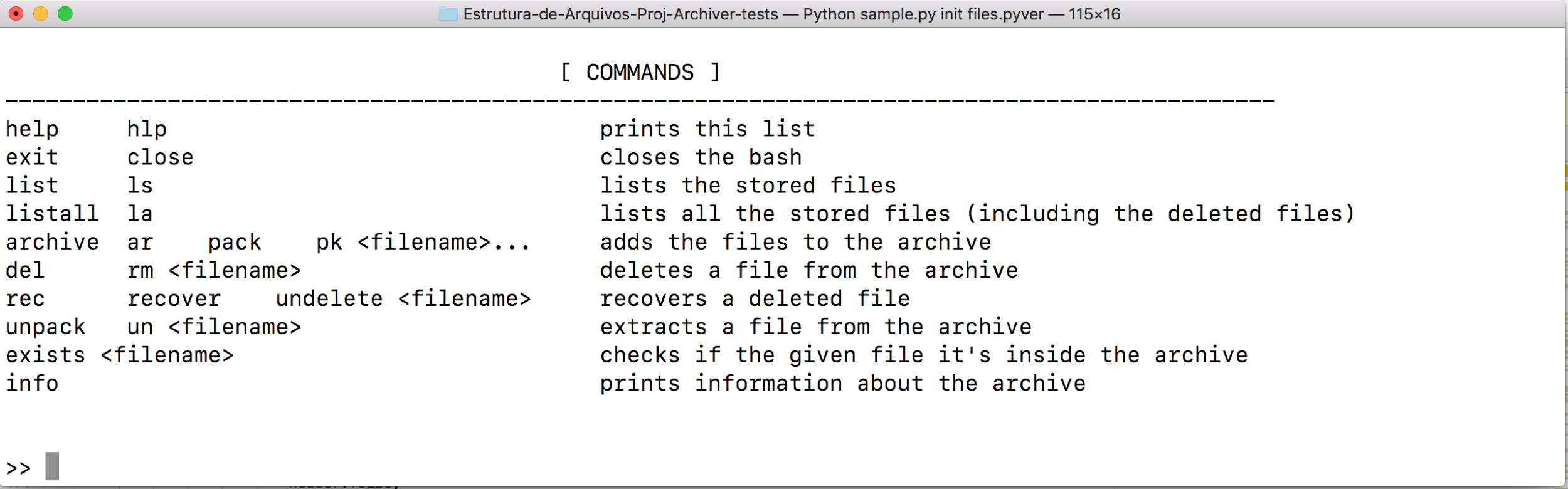
return True

count += 1

return False

## C6. Menu do programa:

### Menu do programa :



### Código:

class TextInterface:

"""

This class generates the text interface and comunicate with the classes that manage the archive

"""

def \_\_init\_\_(self):

self.archiveName = 'defaultArchiver.pyver'

if len(sys.argv) == 1:

print('You have not entered any commands')

print('Try: `open <archivename>.pyver` to open a already initiated archive')

print('Try: `init <archivename>` to initializes a new archive')

input()

exit(0)

elif len(sys.argv) == 3:

auxArchiveName = sys.argv[2]

self.archiveName = auxArchiveName

if sys.argv[1] == 'init':

self.archiveName += '.pyver'

p = Packer(self.archiveName)

p.initFile()

self.pk = Packer(self.archiveName)

self.util = Util(self.archiveName)

print('\n\n\n{:^50}\n'.format('\*\*\*[ PYVER BASH ]\*\*\*'))

print('Selected archive: [{}]'.format(self.archiveName))

print('\nTip: type `help` for displaying the help message.')

while True:

inpt = self.getUserInput()

inpt[0] = inpt[0].lower()

if len(inpt) == 1:

inpt.append('')

## EXIT

if inpt[0] == 'exit' or inpt[0] == 'close':

exit(0)

## INFO

elif inpt[0] == 'info':

info = self.pk.getInfo()

for line in info:

print('{:30}:{:>20}'.format(line[0], line[1]))

## SHOW HELP

elif inpt[0] == 'help' or inpt[0] == 'hlp':

self.printHelp()

## FILE IS IN THE ARCHIVE

elif inpt[0] == 'exists':

pk = self.pk

try:

if pk.getHeaderByfname(inpt[1]):

rsp = '[ Yes ]'

print(rsp)

else:

rsp = '[ No ]'

print(rsp)

except:

print('An error occurred while looking for the file [{}] in the archive\'s header.'.format(inpt[1]))

## LIST FILES

elif inpt[0] == 'list' or inpt[0] == 'ls':

self.describeNonDeletedFiles()

## LIST ALL FILES

elif inpt[0] == 'listall' or inpt[0] == 'la':

self.describeAllFiles()

## DELETE FILE

elif inpt[0] == 'del' or inpt[0] == 'rm' or inpt[0] == 'remove':

print('Deleting [{}]'.format(inpt[1]))

if self.pk.delFile(inpt[1]):

print('Deleted with success.')

else:

print('An error occurred, the file could not be deleted.')

## RECOVERY FILE

elif inpt[0] == 'rec' or inpt[0] == 'recover' or inpt[0] == 'undelete':

print('Recovering [{}]'.format(inpt[1]))

if self.pk.recFile(inpt[1]):

print('Reovered with success.')

else:

print('An error occurred, the file could not be recovered.')

## ARCHIVE FILE

elif inpt[0] == 'archive' or inpt[0] == 'ar' or inpt[0] == 'pack' or inpt[0] == 'pk' or inpt[0] == 'append' or inpt[0] == 'add' or inpt[0] == 'arc':

print('Inserting files...')

for i in range(1, len(inpt)):

try:

print("Archiving file: [{}]...".format(inpt[i]))

insetedHead = self.pk.packAtTheEnd(inpt[i])

self.pk.appendToHeader2(insetedHead)

print("[{}] was successfully archived.".format(inpt[i]))

except Exception:

print('Error: File {} could not be archived.'.format(inpt[i]))

## UNPACK FIILE

elif inpt[0] == 'unpack' or inpt[0] == 'un':

try:

print('Unpacking file: [{}]'.format(inpt[1]))

headElem = self.pk.getHeaderByfname(inpt[1])

self.pk.unpack(headElem)

print("[{}] was successfully unpacked.".format(inpt[1]))

except Exception:

print('Error: File {} could not be unpacked.'.format(inpt[1]))

else:

print('Unknown command, try using:')

self.printHelp()

def printHelp(self):

"""

Print the help message

:return: nothing

"""

print('\n{:^94}'.format('[ COMMANDS ]'))

print('-'\*94)

print("{:42} {:50}".format('help hlp', 'prints this list'))

print("{:42} {:50}".format('exit close', 'closes the bash'))

print("{:42} {:50}".format('list ls', 'lists the stored files'))

print("{:42} {:50}".format('listall la', 'lists all the stored files (including the deleted files)'))

print("{:42} {:50}".format('archive ar pack pk <filename>...', 'adds the files to the archive'))

print("{:42} {:50}".format('del rm <filename>', 'deletes a file from the archive'))

print("{:42} {:50}".format('rec recover undelete <filename>', 'recovers a deleted file'))

print("{:42} {:50}".format('unpack un <filename>', 'extracts a file from the archive'))

print("{:42} {:50}".format('exists <filename>', 'checks if the given file it\'s inside the archive'))

print("{:42} {:50}".format('info', 'prints information about the archive'))

def describeAllFiles(self):

"""

Prints a description of all the files in the archive

:return: nothing

"""

totalFiles = self.pk.elemsInHeader()

headers = self.pk.retrieveHeader()

count = 0

print('\n\n')

totalSize = os.path.getsize(self.archiveName)

print('Total files: {}\t|\tArchive total size: {}B'.format(totalFiles, totalSize))

print('{:4s} {:25s} {:6s} {:5s} {:5s}'.format('Num', 'File Name', 'Size', 'Pos', 'isDeleted'))

for header in headers:

print('{:3}. {:25s} {:5}B {:5} {:5s}'

.format(

count,

str(header.fnameStr[0:30]),

header.fsize,

header.fPos,

str(header.nextdeleted > -1)

)

)

count += 1

print('\n')

# Prints a description of the non-deleted files

def describeNonDeletedFiles(self):

"""

Prints a description of the non-deleted files

:return: nothing

"""

totalFiles = self.pk.elemsInHeader()

headers = self.pk.retrieveHeader()

count = 0

print('\n\n')

print('Total files: {:}'.format(totalFiles))

print('{:4s} {:30s} {:6s} {:5s}'.format('Num', 'File Name', 'Size', 'Pos'))

for header in headers:

if header.nextdeleted < 0:

print('{:3}. [{:30s}] [{:5}B] [{:5}]'

.format(

count,

str(header.fnameStr[0:30]),

header.fsize,

header.fPos,

)

)

count += 1

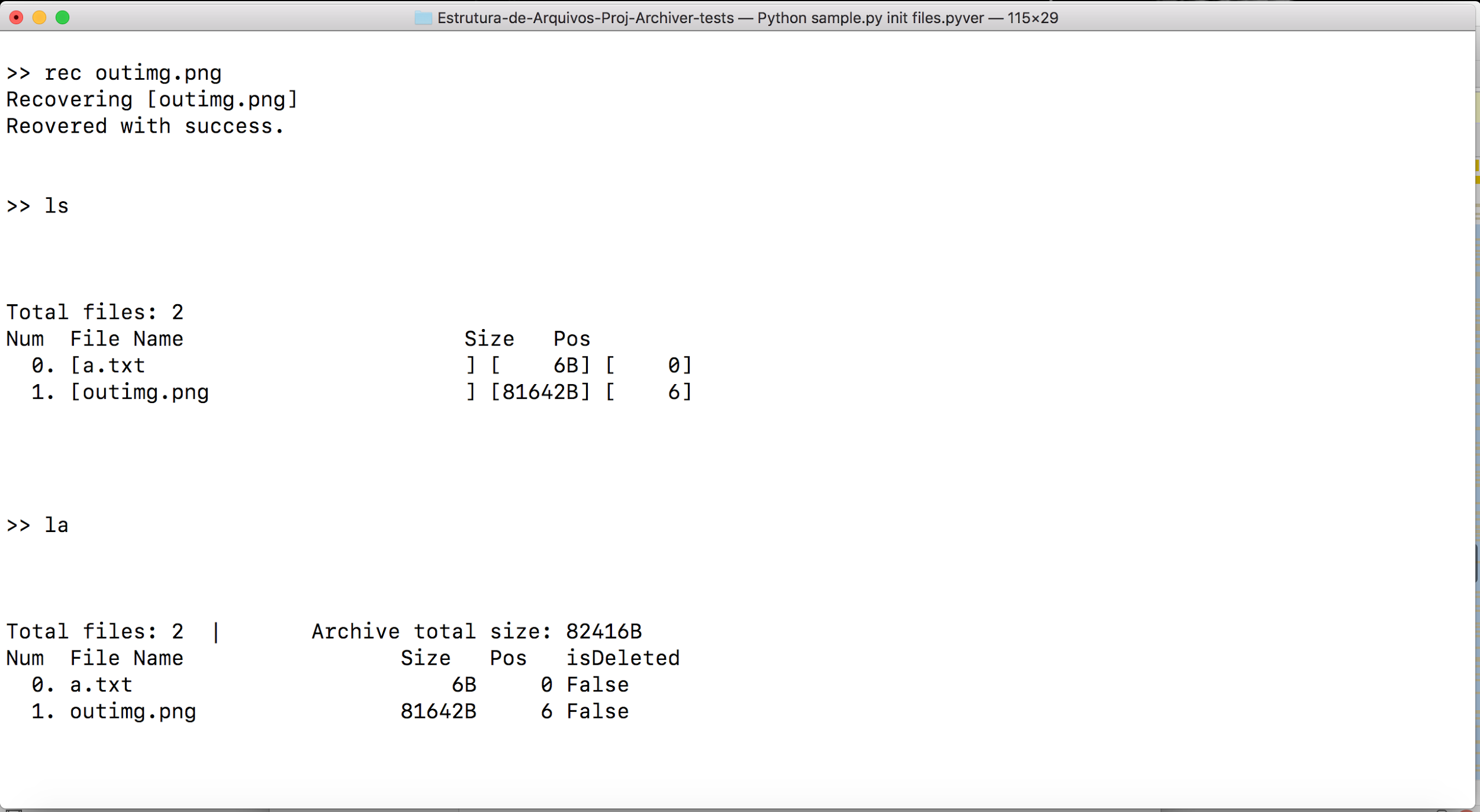
print('\n')

def getUserInput(self):

return str(input('\n\n>> ')).split(' ')

## C7. Recuperando um arquivo deletado:

### Recuperando um arquivo do archiver pelo comando rec <nome\_do\_arquivo>



### Código

def recFile(self, fname):

"""

Marks a file as not deleted

:param fname: the name of the file to be marked as not deleted in the archive

:return: True only if the filename exists in the archive

"""

headers = self.retrieveHeader()

count = 0

for header in headers:

if header.fnameStr == fname:

header.nextdeleted = -1

self.updateHeader(header, count)

return True

count += 1

return False