****

**ESTI - Escola Superior da Tecnologia da Informação**EDC - Graduação em Engenharia de Computação  
Desenvolvimento Python para Redes e Sistemas Operacionais  
Assessment

Aluno: Eloy Francisco Barbosa  
Professor: Cassius Figueiredo  
Data: 09/02/2019

Sumário

[**1.** 3](#_Toc5666045)

[**2.** 4](#_Toc5666046)

[**3.** 5](#_Toc5666047)

[**4.** 7](#_Toc5666048)

[**5.** 8](#_Toc5666049)

[**6.** 9](#_Toc5666050)

[**7.** 13](#_Toc5666051)

[**8.** 16](#_Toc5666052)

[**9.** 21](#_Toc5666053)

**1.**

**Escreva um programa em Python que:**

1. **obtenha a lista de processos executando no momento, considerando que o processo pode deixar de existir enquanto seu programa manipula suas informações;**
2. **imprima o nome do processo e seu PID;**
3. **imprima também o percentual de uso de CPU e de uso de memória.**

import psutil

print("{:10} {:30} {:10} {:10}".format("PID", "Nome", "Memoria", "Cpu"))

for pid in psutil.pids():

try:

processo = {}

p = psutil.Process(pid)

pid = pid

nome = p.name()

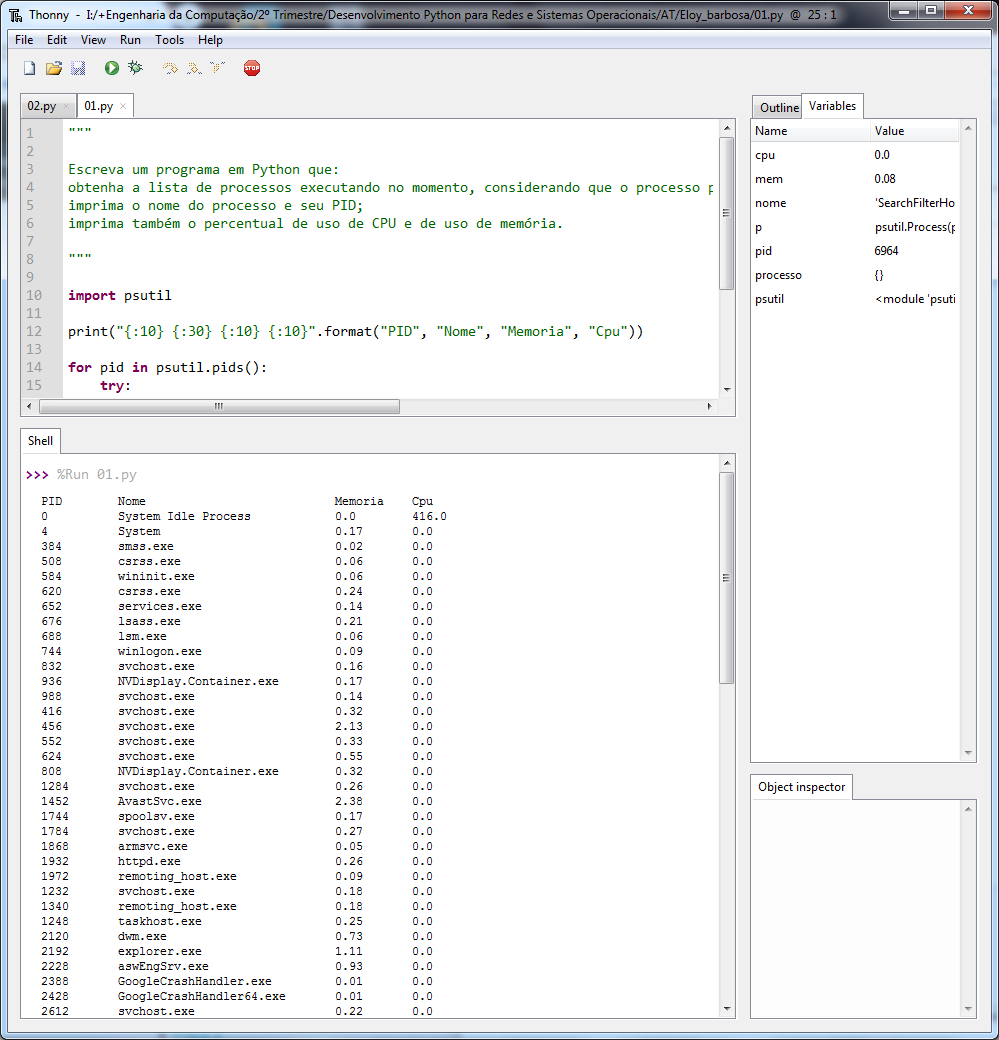
mem = round(p.memory\_percent(), 2)

cpu = p.cpu\_percent(interval=0.01)

print("{:<10} {:30} {:<10} {:<10}".format(pid, nome, mem, cpu))

except:

pass



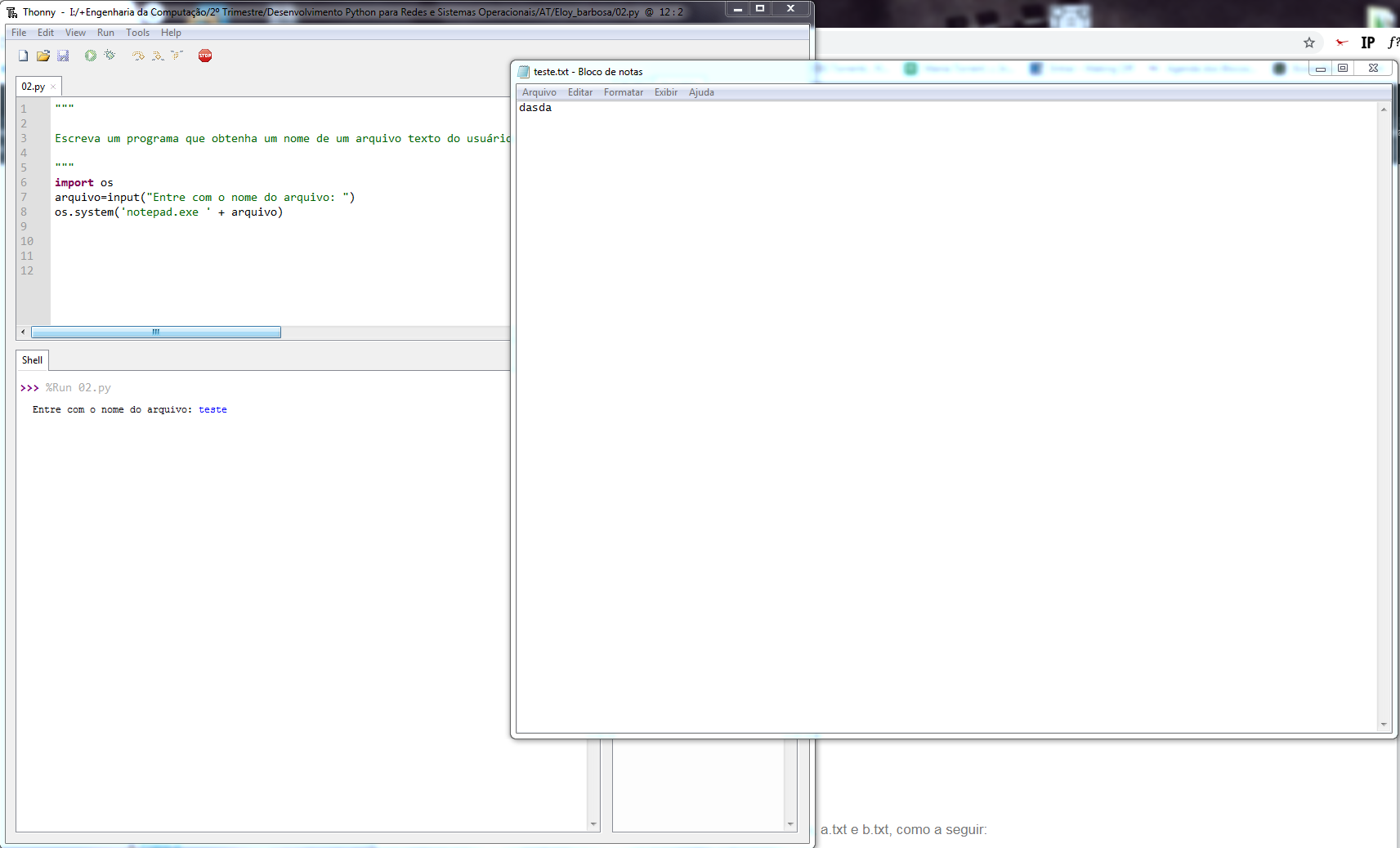
**2.**

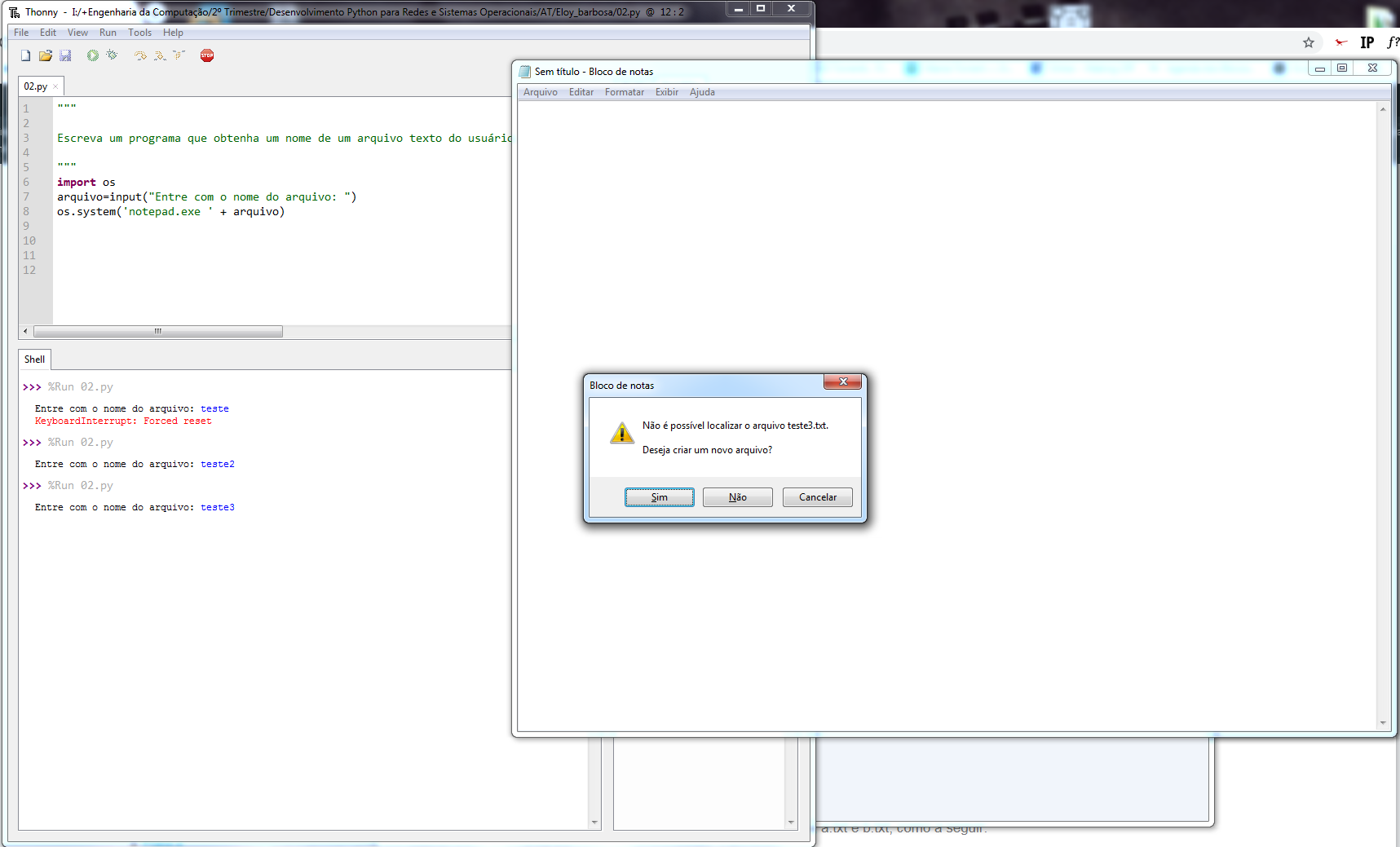
**Escreva um programa que obtenha um nome de um arquivo texto do usuário e crie um processo para executar o programa do sistema Windows bloco de notas (notepad) para abrir o arquivo.**

import os

arquivo=input("Entre com o nome do arquivo: ")

os.system('notepad.exe ' + arquivo)





**3.**

**Escreva um programa em Python que:**

1. **Gere uma estrutura que armazena o nome dos arquivos em um determinado diretório e a quantidade de bytes que eles ocupam em disco. Obtenha o nome do diretório do usuário.**
2. **Ordene decrescentemente esta estrutura pelo valor da quantidade de bytes ocupada em disco (pode usar as funções sort ou sorted);**
3. **gere um arquivo texto com os valores desta estrutura ordenados.**

**import os**

**import time**

**arquivos =[]**

**caminho = input('Informe o caminho de um diretório: ')**

**print()**

**try:**

**for nome in (os.listdir(caminho)):**

**arquivo =[ nome, int(os.stat(caminho + nome).st\_size)]**

**arquivos.append(arquivo)**

**arquivos.sort(key=lambda o: o[1], reverse=True)**

**print('A relação do dos arquivos desse diretório em ordem decrescente pelo tamanho em bytes é a seguinte: \n\n')**

**print('{:<30}'.format('Nome'), end='')**

**print('{:<30}'.format('Tamanho em bytes'), end='\n\n')**

**for x in range (len(arquivos)):**

**print('{:<30}'.format(arquivos[x][0]), end='')**

**print('{:<30}'.format(arquivos[x][1]), end='\n')**

**except (FileNotFoundError):**

**print('O caminho informado não existe!')**

**now = ('Questão 03 - ' + time.strftime('%y%d%m%H%m%S') + ".txt")**

**txt = open(now, "w")**

**txt.write('{:<30}'.format('Nome'))**

**txt.write('{:<30}'.format('Tamanho em bytes'))**

**txt.write('\n')**

**for x in range (len(arquivos)):**

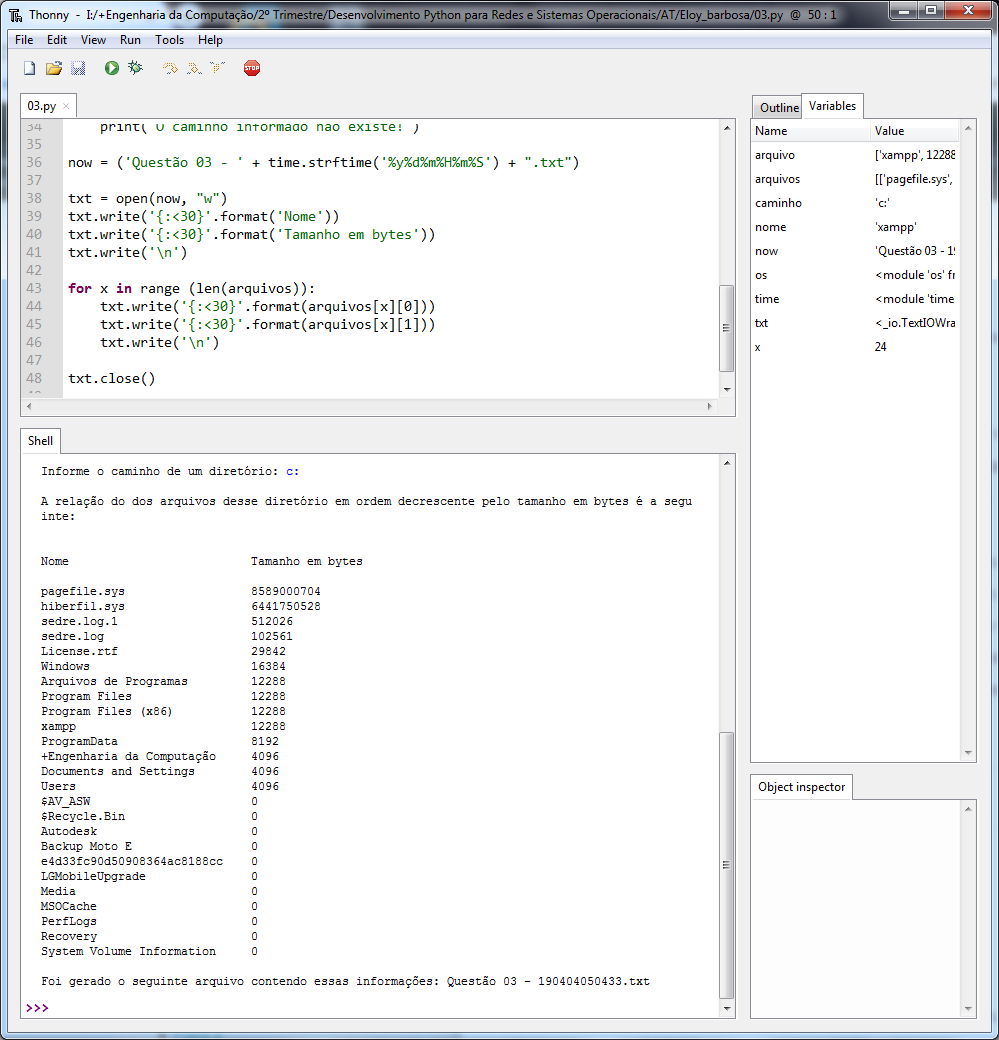
**txt.write('{:<30}'.format(arquivos[x][0]))**

**txt.write('{:<30}'.format(arquivos[x][1]))**

**txt.write('\n')**

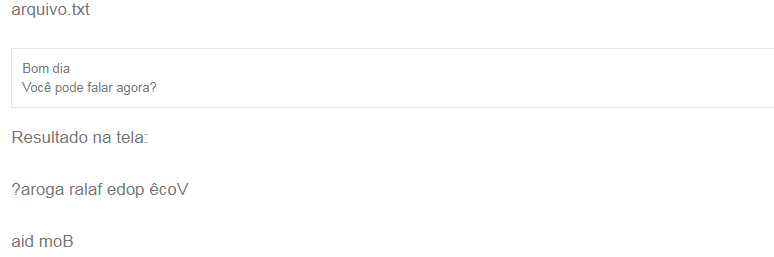
**txt.close()**

**print('\nFoi gerado o seguinte arquivo contendo essas informações: {} '.format(now))**



**4.**

**Escreva um programa em Python que leia um arquivo texto e apresente na tela o seu conteúdo reverso. Exemplo:**



try:

arq = open("arquivo.txt", "r")

lista = []

for linha in arq:

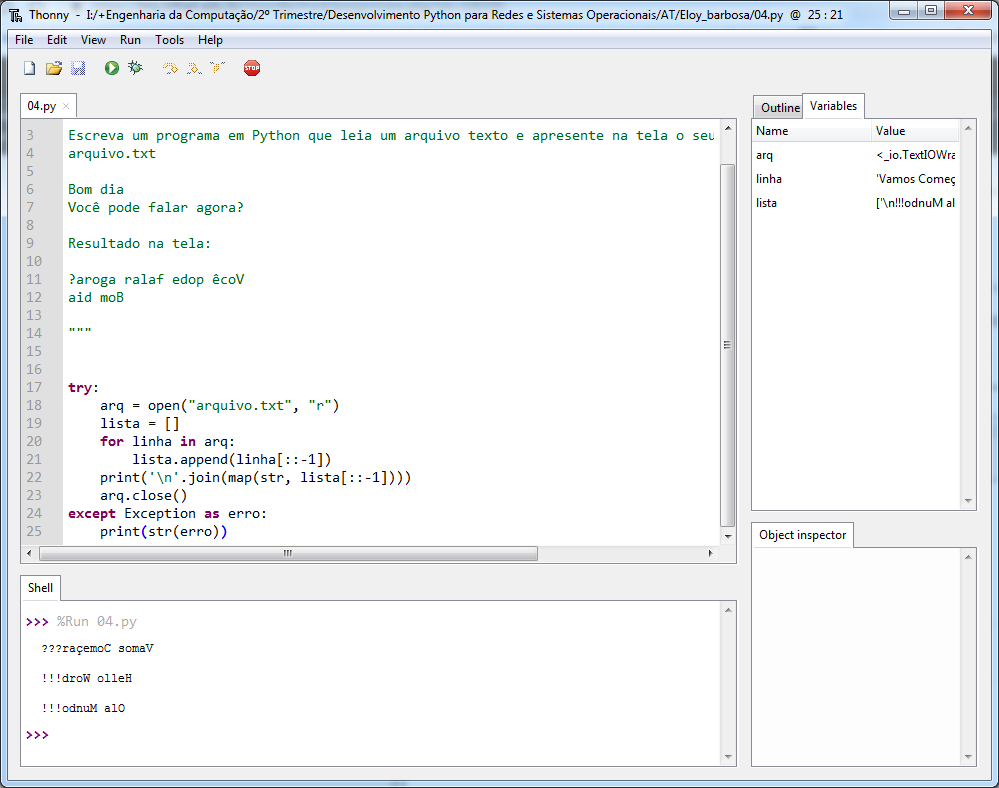
lista.append(linha[::-1])

print('\n'.join(map(str, lista[::-1])))

arq.close()

except Exception as erro:

print(str(erro))



**5.**

**Escreva um programa em Python que leia dois arquivos, a.txt e b.txt, como a seguir:**

**a.txt**

**1 15 -42 33 -7 -2 39 8**

**b.txt**

**19 56 -43 23 -7 -11 33 21 61 9**

**Seu programa deve somar elemento por elemento de cada arquivo e imprimir o resultado na tela.**

**Isto é, o primeiro elemento de a.txt deve ser somado ao primeiro elemento de b.txt, segundo elemento de a.txt deve ser somado ao segundo elemento de b.txt, e assim sucessivamente.**

**Caso um arquivo tenha mais elementos que o outro, os elementos que sobrarem do maior devem ser somados a zero.**

try:

a = open("a.txt", "r")

b = open("b.txt", "r")

lista\_a = []

lista\_b = []

resultado = []

for x in a:

lista\_a = x.split(" ")

for y in b:

lista\_b = y.split(" ")

print('a.txt : ', ' '.join(map(str, lista\_a)))

print('b.txt : ', ' '.join(map(str, lista\_b)))

A=len(lista\_a)

B=len(lista\_b)

if A < B:

ab= B-A

for x in range (ab):

lista\_a.append('0')

else:

ba=A-B

for y in range (ba):

lista\_b.append('0')

for i in range (len(lista\_a)):

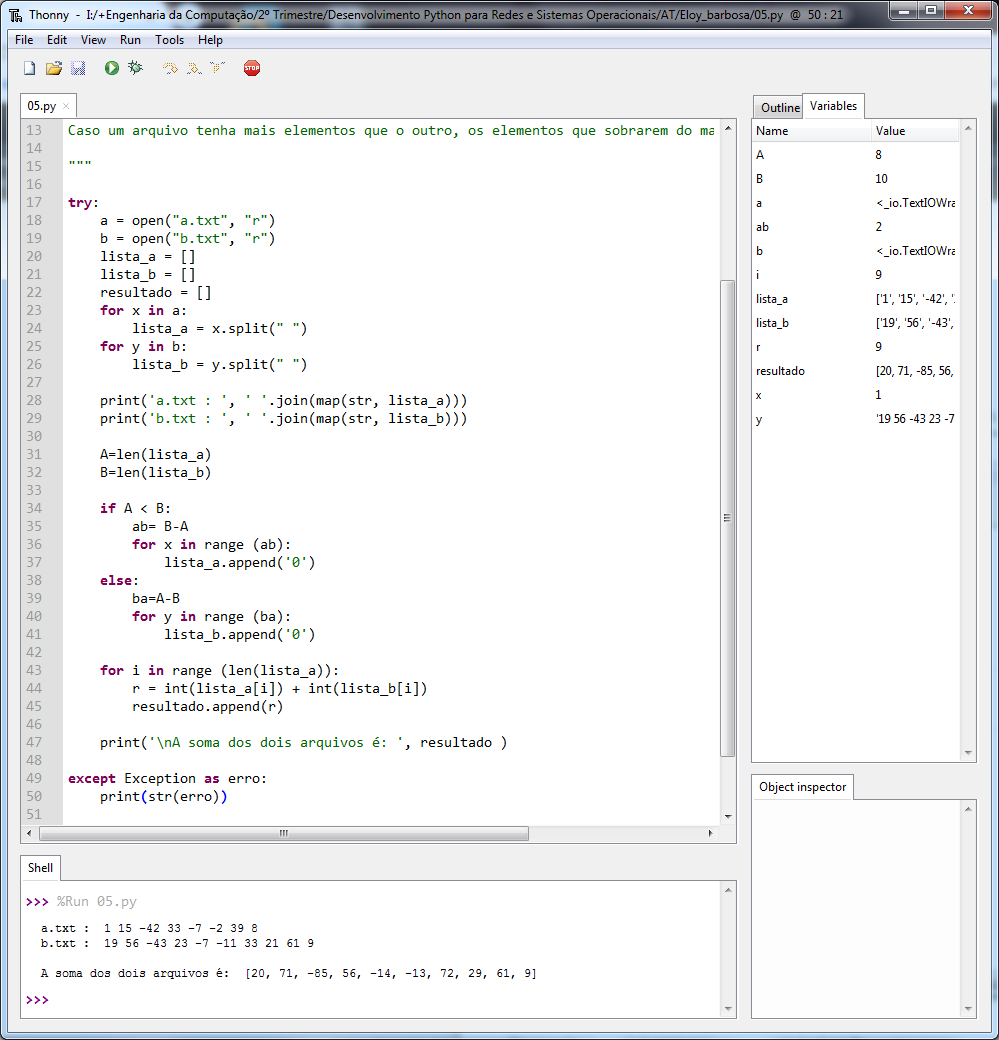
r = int(lista\_a[i]) + int(lista\_b[i])

resultado.append(r)

print('\nA soma dos dois arquivos é: ', resultado )

except Exception as erro:

print(str(erro))



**6.**

**Escreva um programa cliente e servidor sobre TCP em Python em que:**

1. **O cliente envia para o servidor o nome de um diretório e recebe a lista de arquivos (apenas arquivos) existente nele.**
2. **O servidor recebe a requisição do cliente, captura o nome dos arquivos no diretório em questão e envia a resposta ao cliente de volta.**

#Cliente

import socket

import os

import pickle

socket\_cliente = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

arquivo = input('Informe o diretório: ')

print()

socket\_cliente.connect((socket.gethostname(), 6666))

socket\_cliente.send(arquivo.encode('utf-8'))

mensagem = socket\_cliente.recv(10000)

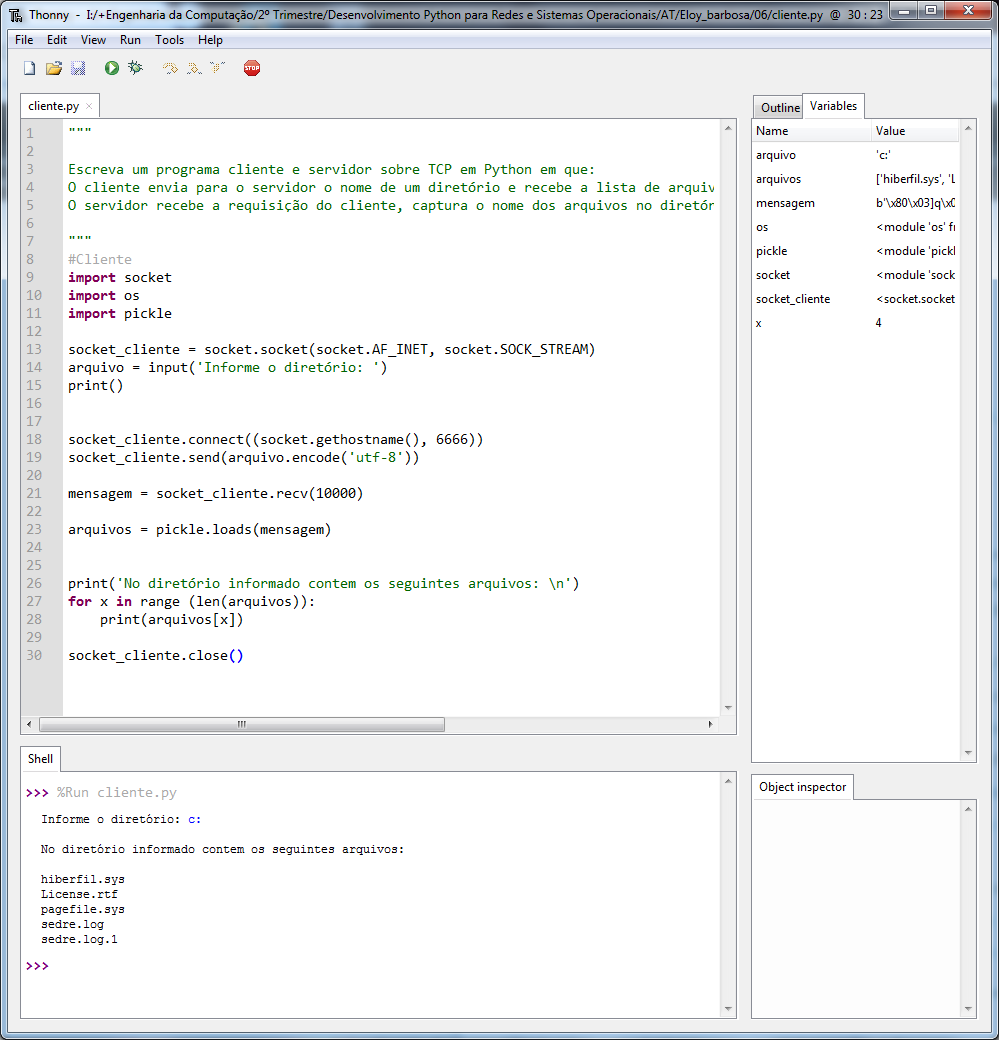
arquivos = pickle.loads(mensagem)

print('No diretório informado contem os seguintes arquivos: \n')

for x in range (len(arquivos)):

print(arquivos[x])

socket\_cliente.close()



#Servidor

import socket

import os

import pickle

socket\_servidor = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

host = socket.gethostname()

porta = 6666

socket\_servidor.bind((host, porta))

socket\_servidor.listen()

print("Servidor de nome", host, "esperando conexão na porta", porta)

while True:

(socket\_cliente,addr) = socket\_servidor.accept()

print("Conectado a:", str(addr))

print()

arquivo = socket\_cliente.recv(1024)

print ('O cliente informou que quer a lista de arquivos (somente arquivos) do seguinte diretório: ', arquivo.decode('utf-8'))

print()

files=[]

if os.path.exists(arquivo):

arquivos = os.listdir(arquivo)

for i in range (len(arquivos)):

if os.path.isfile(arquivo + arquivos[i]) == True:

files.append(arquivos[i].decode('utf-8'))

print('No diretório informado contem os seguintes arquivos: ', files)

print()

resposta = pickle.dumps(files)

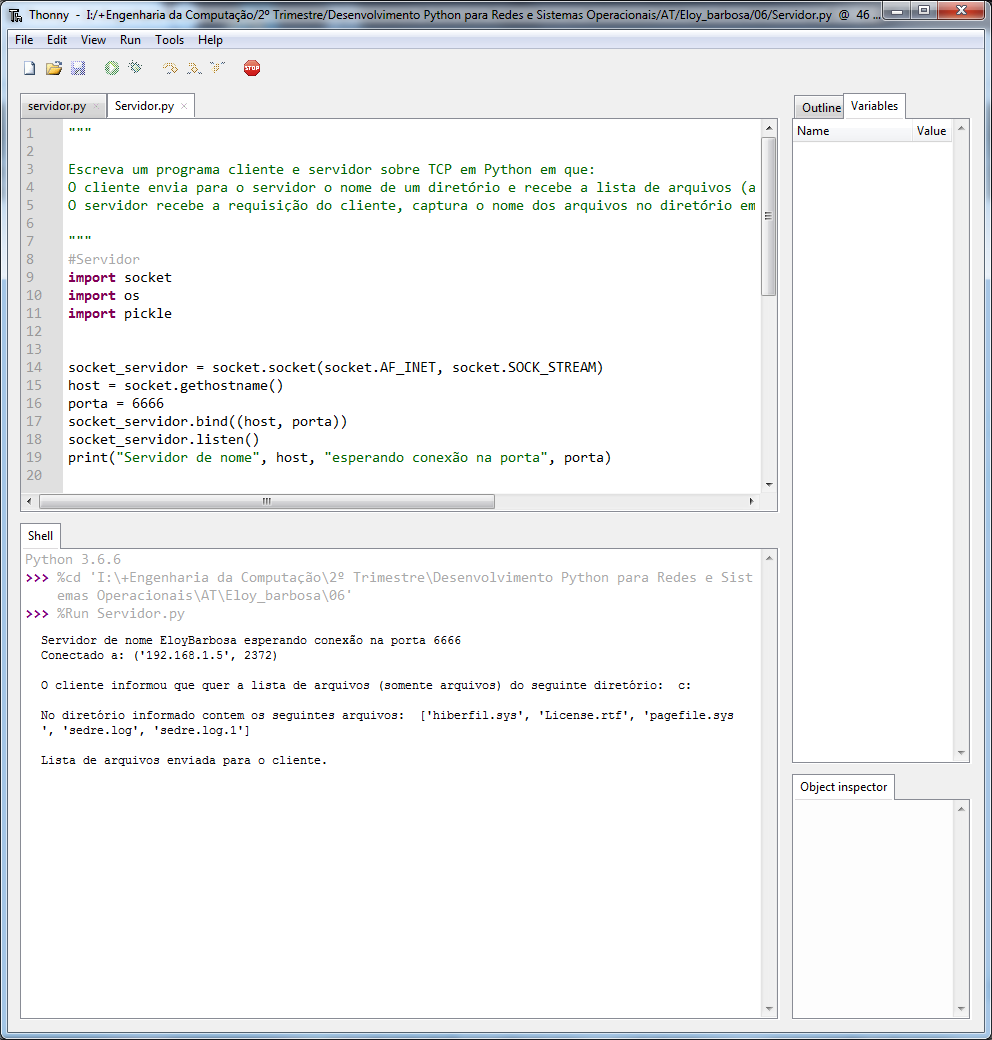
socket\_cliente.send(resposta)

print('Lista de arquivos enviada para o cliente.')

else:

pass

socket\_cliente.close()



**7.**

**Escreva um programa cliente e servidor sobre UDP em Python que:**

1. **O cliente envia para o servidor o pedido de obtenção da quantidade total e disponível de memória no servidor e espera receber a resposta durante 5s. Caso passem os 5s, faça seu programa cliente tentar novamente mais 5 vezes (ainda esperando 5s a resposta) antes de desistir.**
2. **O servidor repetidamente recebe a requisição do cliente, captura a informação da quantidade total e disponível de memória há no servidor e envia a resposta ao cliente de volta.**

#Cliente

import socket

import time

print('Nesta atividade vamos nos conectar a um servidor via UDP, coletar as informações de quantidade total e disponivel de memória no servidor .\n')

UDPClient = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

UDPClient.settimeout(5)

ServerName = socket.gethostname()

Port = 6666

print('Solicitando dados... ')

tentativas = 0

for i in range(5):

print("Tentativa ", (i+1), "...")

tentativas += 1

msg = 'y'

UDPClient.sendto(msg.encode('utf-8'), (ServerName, Port))

try:

(msg, client) = UDPClient.recvfrom(1024)

print(msg.decode('utf-8'))

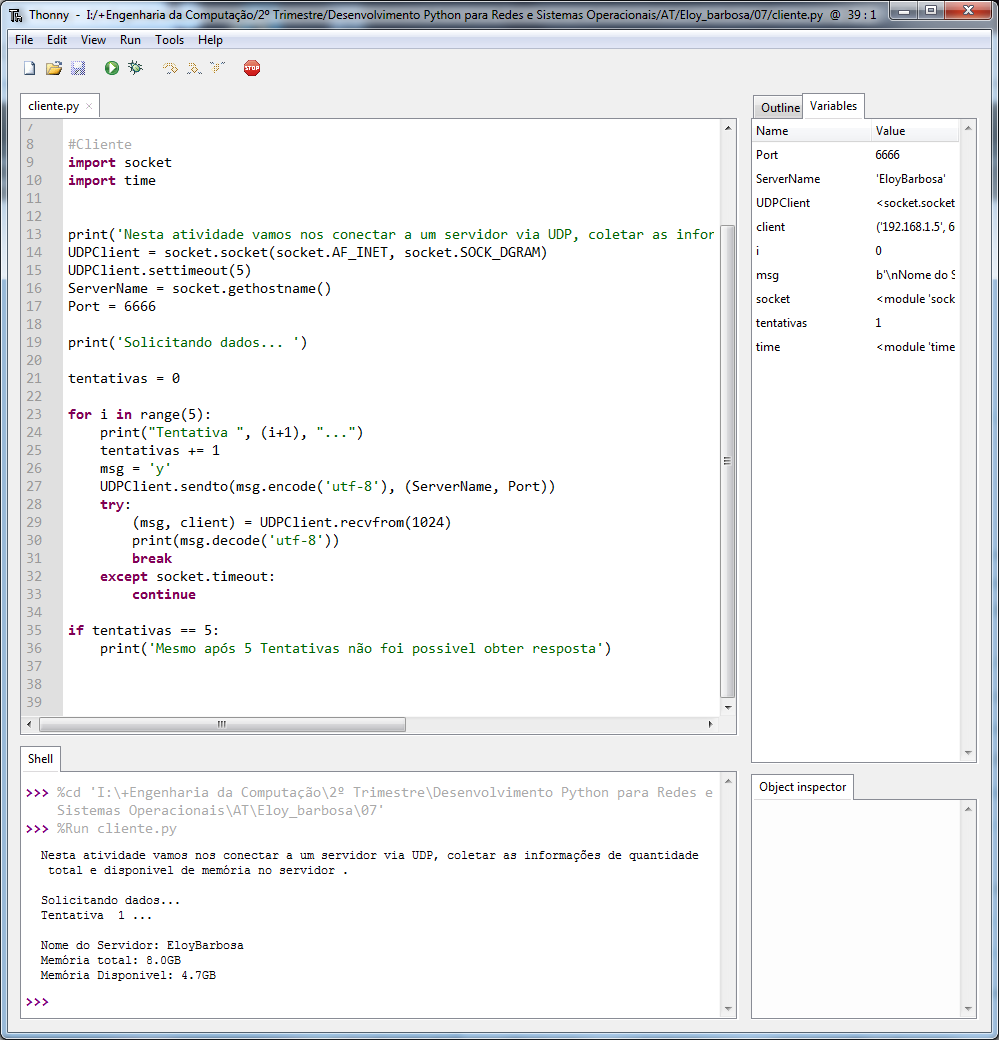
break

except socket.timeout:

continue

if tentativas == 5:

print('Mesmo após 5 Tentativas não foi possivel obter resposta')



#Servidor

import socket

import psutil

UDPServer = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

ServerName = socket.gethostname()

Port = 6666

UDPServer.bind((ServerName, Port))

memoria\_total = round(psutil.virtual\_memory().total / (1024 \* 1024 \* 1024), 1)

memoria\_disponivel = round(psutil.virtual\_memory().free / (1024 \* 1024 \* 1024), 1)

try:

print('Aguardando requisição na porta', Port, '...')

(msg, client) = UDPServer.recvfrom(1024)

print('Requisição aceita')

if msg.decode() == 'y':

UDPServer.sendto('\nNome do Servidor: {} \nMemória total: {}GB \nMemória Disponivel: {}GB'.format(ServerName, memoria\_total, memoria\_disponivel).encode('utf-8'), client)

print('Informações enviadas com sucesso')

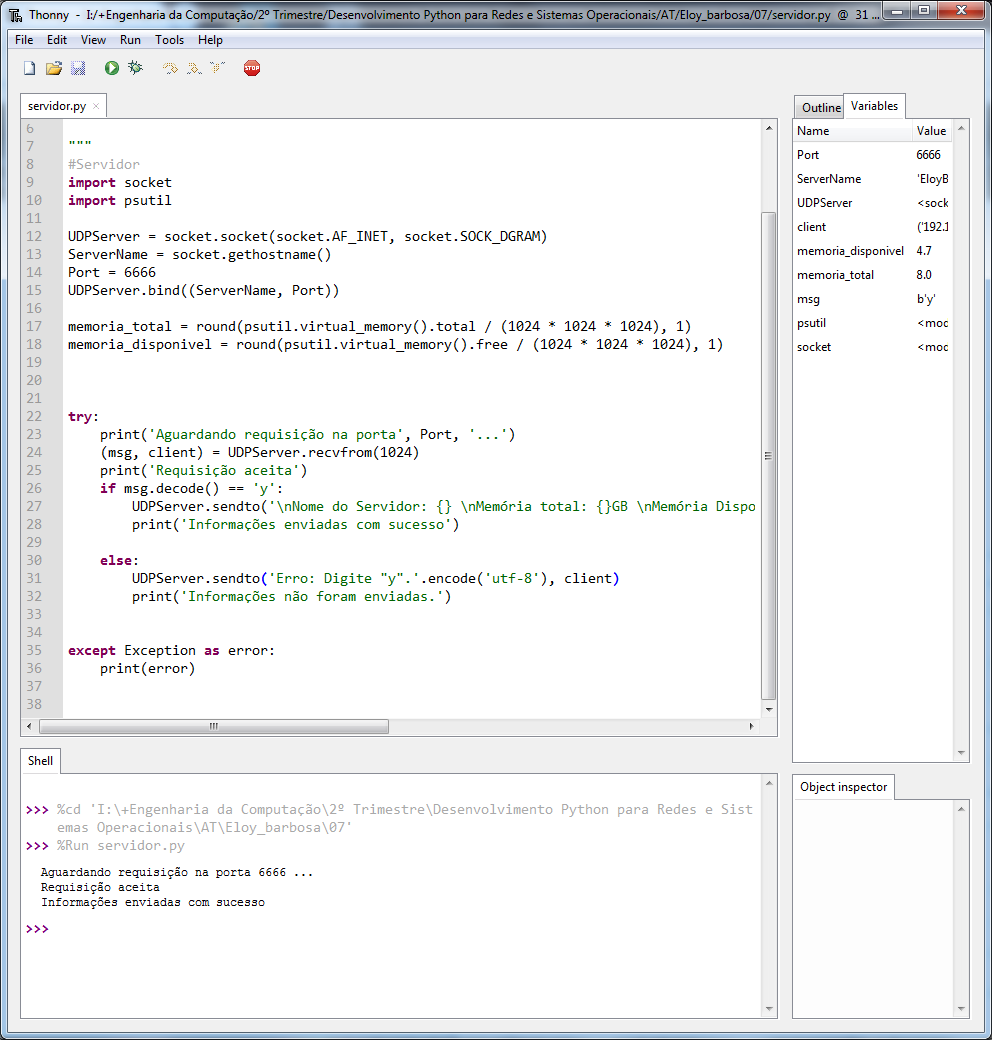
else:

UDPServer.sendto('Erro: Digite "y".'.encode('utf-8'), client)

print('Informações não foram enviadas.')

except Exception as error:

print(error)



**8.**

**Escreva 3 programas em Python que resolva o seguinte problema:**

**Dado um vetor A de tamanho N com apenas números inteiros positivos, calcule o fatorial de cada um deles e armazene o resultado em um vetor B.**

**Para calcular o fatorial, utilize a seguinte função:**

****

**Os modos de desenvolver seu programa devem ser:**

1. **sequencialmente (sem concorrência);**
2. **usando o módulo threading com 4 threads;**
3. **usando o módulo multiprocessing com 4 processos.**

a.  
import time

import random

resultado = []

def fatorial(n):

fat = n

for i in range(n-1,1,-1):

fat = fat \* i

#return fat

resultado.append(fat)

t\_inicio = float(time.time())

listaFatorial = []

def criaLista(n):

for i in range(n):

k = random.randint(1,10)

listaFatorial.append(k)

criaLista(1000000)

for i in listaFatorial:

fatorial(i)

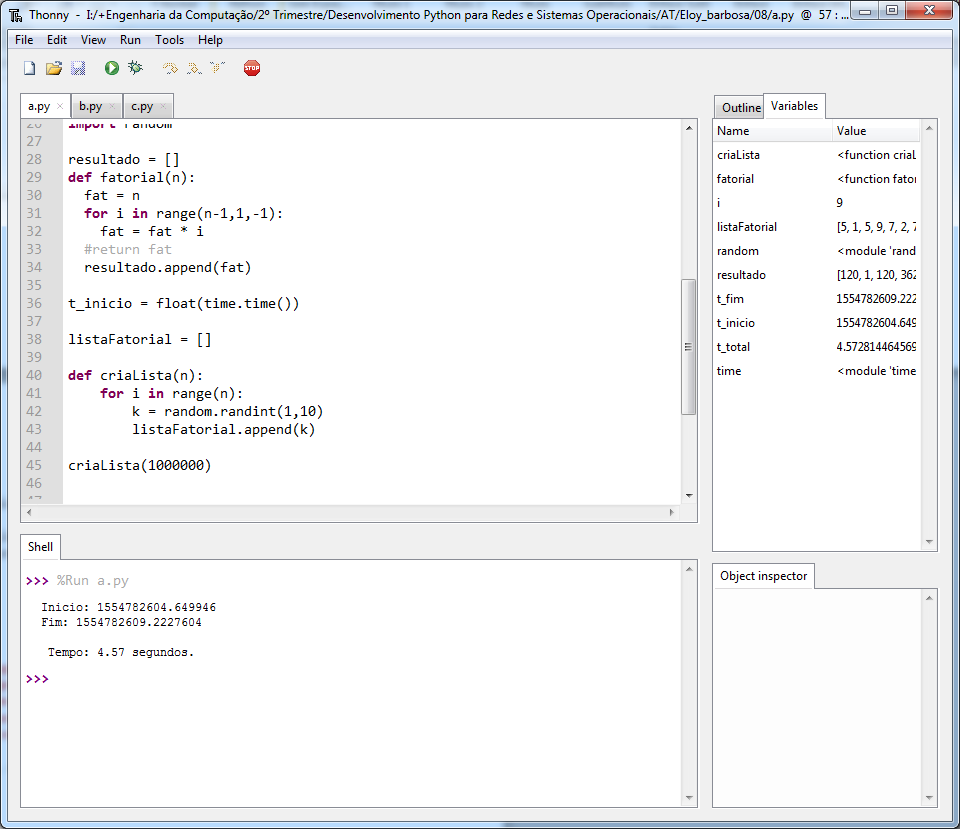
t\_fim = float(time.time())

print('Inicio:',t\_inicio)

print('Fim:',t\_fim)

t\_total = t\_fim - t\_inicio

print('\n','Tempo:',round(t\_total,2),'segundos.')



b.

import time

import random

import threading

def fatorial(n):

fat = n

for i in range(n - 1, 1, -1):

fat = fat \* i

return (fat)

def listaFatorial(lista, inicio, fim):

for i in range(inicio, fim):

bLista.append(fatorial(lista[i]))

t\_inicio = float(time.time())

N = 1000000

aLista = []

bLista = []

for i in range(N):

aLista.append(random.randint(0, 10))

Nthreads = 4

lista\_threads = []

for i in range(Nthreads):

ini = i \* int(N / Nthreads)

fim = (i + 1) \* int(N / Nthreads)

t = threading.Thread(target=listaFatorial, args=(aLista, ini, fim))

t.start()

lista\_threads.append(t)

for t in lista\_threads:

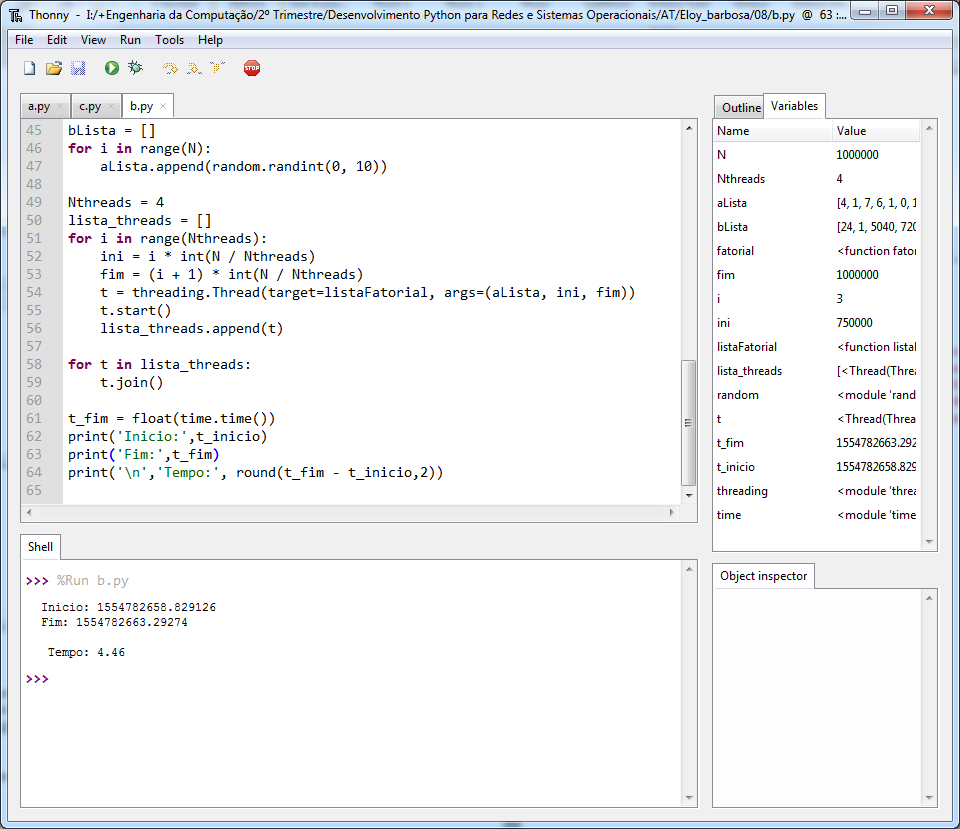
t.join()

t\_fim = float(time.time())

print('Inicio:',t\_inicio)

print('Fim:',t\_fim)

print('\n','Tempo:', round(t\_fim - t\_inicio,2))



c.

import multiprocessing

import random

import time

def Factorial(n):

fat = n

for i in range(n - 1, 1, -1):

fat = fat \* i

return fat

def Main():

N = 1000000

t\_inicio = float(time.time())

aLista = []

bLista = []

for i in range(N):

aLista.append(random.randint(0, 10))

NProc = 4

q\_entrada = multiprocessing.Queue()

q\_saida = multiprocessing.Queue()

lista\_proc = []

for i in range(NProc):

ini = i \* int(N / NProc)

fim = (i + 1) \* int(N / NProc)

q\_entrada.put(aLista[ini:fim])

p = multiprocessing.Process(target=ListFat, args=(q\_entrada, q\_saida))

p.start()

lista\_proc.append(p)

for i in range(0, NProc):

bLista = q\_saida.get(timeout=10)

for p in lista\_proc:

p.join()

t\_fim = float(time.time())

print('Inicio:',t\_inicio)

print('Fim:',t\_fim)

print('\n', 'Tempo :', round(t\_fim - t\_inicio, 2))

input('Digite enter para sair...')

def fatorial(n):

fat = n

for i in range(n - 1, 1, -1):

fat = fat \* i

return (fat)

def ListFat(q1, q2):

l1 = q1.get()

l2 = []

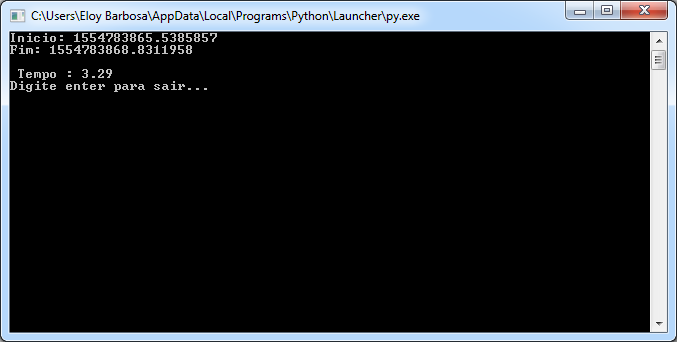
for i in l1:

l2.append(fatorial(l1[i]))

q2.put(l2)

if \_\_name\_\_ == '\_\_main\_\_':

Main()



**9.**

**Teste todos os 3 programas da questão 8, capture os tempos de execução deles e compare-os, explicando os resultados de tempos. Varie o valor de N em 1.000.000, 5000.000, 10.000.000 (ou escolha números maiores ou melhores de acordo com a velocidade de processamento do computador utilizado para testes).**

Tempo de execução:

Sequencial: 4.57

Threading: 4.46

Multiprocessing: 3.29

Diante dos testes feitos, ficou notável que o calculo do Fatorial de 1000000 foi mais rápido com o método de Multiprocessing.