INSTITUTO INFNET

CARLOS HENRIQUE DOS SANTOS RODRIGUES JUNIOR

DESENVOLVIMENTO PYTHON PARA REDES E SISTEMAS OPERACIONAIS

AT

Rio de Janeiro, novembro

2020

CARLOS HENRIQUE DOS SANTOS RODRIGUES JUNIOR

DESENVOLVIMENTO PYTHON PARA REDES E SISTEMAS OPERACIONAIS

AT

Trabalho apresentado no curso de graduação do Instituto Infnet.

Professor: Cassius

Rio de Janeiro, novembro

2020

Sumário

1. Escreva um programa em Python que:

a. Obtenha a lista de processos executando no momento, considerando que o processo pode deixar de existir enquanto seu programa manipula suas informações;
b. Imprima o nome do processo e seu PID
c. Imprima também o percentual de uso de CPU e de uso de memória

```
# Código fonte questão 1 AT - PYTHON
import psutil, threading, time

from psutil import Process

cache = {
    'processes': []
}

class Process:
    def __init__(self, pid, name, use_cpu, use_memory):
        self.pid = pid
        self.name = name
        self.use_cpu = use_cpu
        self.use_memory = use_memory

    def to_map(self):

        return {
                'pid': self.pid
            ,   'name': self.name
            ,   'use_cpu': self.use_cpu
            ,   'use_memory': self.use_memory
        }

def get_processo():
    pids = psutil.pids()

    collected_processes = []

    for pid in pids:
        try:
            name = psutil.Process(pid).name()
            memory = format(psutil.Process(pid).memory_percent(), '.2f')
            cpu = psutil.Process(pid).cpu_percent()
            process = Process(pid, name, cpu, memory)
            collected_processes.append(process)
```

```
        except:
            pass

    cache['processes'].clear()
    cache['processes'].append(collected_processes)

class ThreadProcess(threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.id = threadID
        self.name_process = name
        self.counter = counter

    def run(self):
        while True:
            get_processo()

t_process = ThreadProcess(1, 'GET-PROCESS', 1)
t_process.start()

processes = ''

while processes == '':
    try:
        processes = cache['processes'][len(cache['processes']) - 1]
    except:
        print('[LOADING]\n')
        time.sleep(10)

for p in processes:
    print(p.to_map(), '\n')
```

```
{'pid': 13476, 'name': 'RadeonSoftware.exe', 'use_cpu': 0.0, 'use_memory': '0.18'}

{'pid': 13672, 'name': 'Code.exe', 'use_cpu': 0.0, 'use_memory': '0.72'}

{'pid': 13972, 'name': 'svchost.exe', 'use_cpu': 0.0, 'use_memory': '0.06'}

{'pid': 14056, 'name': 'AMDRSServ.exe', 'use_cpu': 0.0, 'use_memory': '0.58'}

{'pid': 14128, 'name': 'Code.exe', 'use_cpu': 0.0, 'use_memory': '0.26'}

{'pid': 14156, 'name': 'amdow.exe', 'use_cpu': 0.0, 'use_memory': '0.01'}

PS C:\Users\Carlos\Desktop\at - python>
```

Figure 1 Evidência questão 1

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.**

```python
# Código fonte questão 2 AT – PYTHON

import subprocess

arquivo = input('INFORME O NOME DO ARQUIVO') +'.txt'

subprocess.Popen(['notepad', arquivo])
```

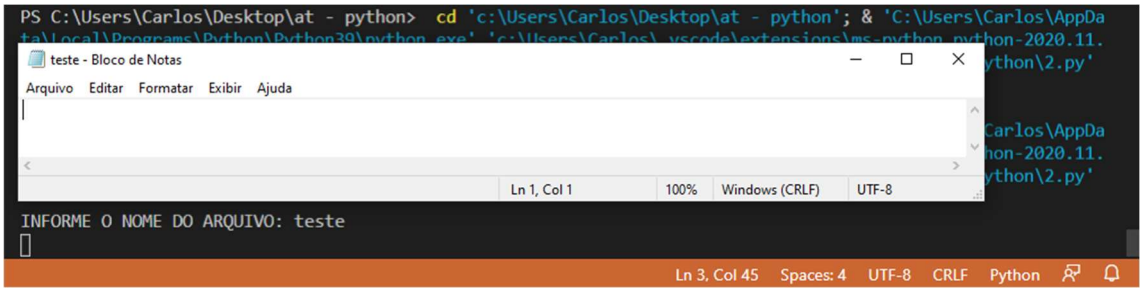


Figure 2 - Evidência questão 2

3.Escreva um programa em Python que:

a. 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.
b. Ordene descentemente esta estrutura pelo valor da quantidade de bytes ocupada em disco
c. Gere um arquivo texto com os valores desta estrutura ordenada

```
# Código fonte questão 3 AT - PYTHON
import os

class Archive:
    def __init__(self, name, size):
        self.name = name
        self.size = size

    def to_map(self):
        return {
            'name': self.name
        ,   'size': self.size
    }


def get_arquives(directory):
    archives = []

    archives_aux = os.listdir(directory)

    for archive in archives_aux:
        path = str(directory + archive)
        if os.path.isfile(path):
            size = os.stat(path).st_size
            archive = Archive(path, size)
            archives.append(archive)

    archives.sort(key=lambda x:x.size)

    return archives


directory = input('Inform The Directory')
result = ''

if os.path.exists(directory):
    response = get_arquives(directory)
    response.reverse()
```

```
    result = response

elif directory == '':
    response = get_arquives(os.getcwd())
    response.reverse()
    result = response
else:
    print('Not found directory')


if result != '':
    file_aux = open('at-python.txt', 'w')

    for r in result:
        file_aux.writelines(str(r.to_map()) + '\n')

    file_aux.close()
```

```
Inform The DirectoryC:/
PS C:\Users\Carlos\Desktop\at - python>
```

Figura 1 Evidência solicitação diretório

at-python - Bloco de Notas

Arquivo Editar Formatar Exibir Ajuda

```
{'name': 'C:/hiberfil.sys', 'size': 13723168768}
{'name': 'C:/pagefile.sys', 'size': 5100273664}
{'name': 'C:/swapfile.sys', 'size': 16777216}
{'name': 'C:/bootmgr', 'size': 413738}
{'name': 'C:/DumpStack.log.tmp', 'size': 8192}
{'name': 'C:/DumpStack.log', 'size': 8192}
{'name': 'C:/BOOTNXT', 'size': 1}
{'name': 'C:/Novo Documento de Texto.txt', 'size': 0}
```

Figura 2 Evidência log arquivos

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

```python
import os

file = open('at-python.txt', 'r')

lines = file.readlines()

for line in lines:
    print(line[::-1])
```

```
}4663720015 :'ezis' ,'sys.elifegap/:C' :'eman'{

}61277761 :'ezis' ,'sys.elifpaws/:C' :'eman'{

}837314 :'ezis' ,'rgmtoob/:C' :'eman'{

}2918 :'ezis' ,'pmt.gol.kcatSpmuD/:C' :'eman'{

}2918 :'ezis' ,'gol.kcatSpmuD/:C' :'eman'{

}1 :'ezis' ,'TXNTOOB/:C' :'eman'{

}0 :'ezis' ,'txt.otxeT ed otnemucoD ovoN/:C' :'eman'{
PS C:\Users\Carlos\Desktop\at - python>
```

Figura 3 Evidência print console

5. Escreva um programa em python que leia dois arquivos, a.txt e b.txt, como a seguir e realize a soma

```python
file_a = open('a.txt', 'r')
file_b = open('b.txt', 'r')

line_a = str(file_a.read())
columns_a = line_a.split(' ')
len_columns_a = len(columns_a)

line_b = str(file_b.read())
columns_b = line_b.split(' ')
len_columns_b = len(columns_b)

def add_zero(vet, max_len):
    len_aux = len(vet)

    for v in vet:
        if len_aux < max_len:
            vet.append(0)
            len_aux = len(vet)
        else:
            break
    return vet
if len_columns_a < len_columns_b:
    columns_a = add_zero(columns_a, len_columns_b)
elif len_columns_b < len_columns_a:
    columns_b = add_zero(columns_b, len_columns_a)
sums = []
max = len(columns_a)

for n in range(0, max):
    position_a = int(columns_a[n])
    position_b = int(columns_b[n])

    sum = position_a + position_b
    sums.append(sum)

print(sums)
```

```
[20, 71, -85, 56, -14, -13, 72, 29, 61, 9]
PS C:\Users\Carlos\Desktop\at - python>
```

Figura 4 Evidência questão 5

6. Escreva um programa cliente e servidor sobre TCP em Python em que

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

```
#server
import os, socket, pickle

class Archive:
    def __init__(self, name, size):
        self.name = name
        self.size = size

    def to_map(self):
        return {
            'name': self.name
        ,   'size': self.size
    }

def get_arquives(directory):
    archives = []
    archives_aux = os.listdir(directory)
    for archive in archives_aux:
        path = str(directory + archive)
        if os.path.isfile(path):
            size = os.stat(path).st_size
            archive = Archive(path, size)
            archives.append(archive.to_map())
    return archives

socket_server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

host_name = socket.gethostname()
server_port = 8080

socket_server.bind((host_name, server_port))
socket_server.listen()

print(host_name, 'waiting connection in port ', server_port)

while True:
    (socket_client, addr) = socket_server.accept()
```

```
    print('server ', host_name, ' connected with ', addr)
    request = socket_client.recv(1024)
    request_decoded = request.decode('utf-8')
    response = get_arquives(request_decoded)
    response_encoded = pickle.dumps(response)
    socket_client.send(response_encoded)
    socket_client.close()
    break
```

```
#client
import os, socket, pickle

socket_client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    socket_client.connect((socket.gethostname(), 8080))
    directory = input('put the path')
    socket_client.send(directory.encode('utf-8'))
    response = socket_client.recv(1024)
    response_decoded = pickle.loads(response)
    print(response_decoded)
    socket_client.close()

except  Exception as erro:
    print(str(erro))
```

```
put the pathC:/
[{'name': 'C:/bootmgr', 'size': 413738}, {'name': 'C:/BOOTNXT', 'size': 1}, {'name': 'C:/DumpStack.log', 'size': 8192}, {'name': 'C:/DumpStack.log
.tmp', 'size': 8192}, {'name': 'C:/hiberfil.sys', 'size': 13723168768}, {'name': 'C:/Novo Documento de Texto.txt', 'size': 0}, {'name': 'C:/pagefi
le.sys', 'size': 5100273664}, {'name': 'C:/swapfile.sys', 'size': 16777216}]
PS C:\Users\Carlos\Desktop\at - python>
```

Figura 5 Evidência cliente

```
esktop\at - python\6 - server.py
DESKTOP-ELTSB8Q waiting connection in port  8080
server  DESKTOP-ELTSB8Q  connected with  ('192.168.0.12', 51414)
PS C:\Users\Carlos\Desktop\at - python>
```

Figura 6 Evidência Server

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

a. 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.
b. 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.

```python
#[cliente]
import pickle, time, socket

socket_client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
host_name = socket.gethostname()

destiny = (host_name, 8080)

try:
    for i in range(0, 5):
        socket_client.sendto(' '.encode('utf-8'), destiny)
        ok = False
        bytes = []
        try:
            bytes = socket_client.recv(1024)
            response = pickle.loads(bytes)
            print(response)
            ok = True
        except:
            print('falhou ao obter, tentativa: ', i + 1)

        if ok:
            break
        time.sleep(10)
except Exception as erro:
    print('erro', str(erro))
```

```python
#[servidor]
import psutil, socket
import pickle, threading
import time
socket_server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
host_name = socket.gethostname()
socket_server.bind((host_name, 8080))
print('Aguardando conexão na porta: ', host_name,':', 8080)
cache = {
    'memory': []
}
class Memory:
    def __init__(self, total, used):
        self.total = total
        self.used = used
    def to_map(self):
        return {
            'total': self.total,
            'used': self.used
        }
def collect_memory_information(memory):
    total_memory = memory.total
    used_memory = memory.used
    collected_memory = Memory(total_memory, used_memory)
    cache['memory'].append(collected_memory)
def get_last_info_of_memory():
    last = len(cache['memory']) - 1
    response = cache['memory'][last]
    return response
class ThreadMemoria(threading.Thread):
    def __init__(self, threadID, name, counter):
        threading.Thread.__init__(self)
        self.threadID = threadID
        self.name = name
        self.counter = counter
    def run(self):
        while True:
            print ("starting: " + self.name)
            memory = psutil.virtual_memory()
            collect_memory_information(memory)
            print ("sleeping...")
            time.sleep(10)

thread = ThreadMemoria(1, 'ThreadMemoria', 1)
thread.start()

while True:
```

```
    response = None
    (request, client) = socket_server.recvfrom(1024)
    if request.decode('utf-8') == 'close':
        break
    if len(cache['memory']) > 0:
        response = get_last_info_of_memory()
        bytes_response = pickle.dumps(response.to_map())
        socket_server.sendto(bytes_response, client)
socket_server.close()
```

```
Aguardando conexão na porta:  DESKTOP-ELTSB8Q : 8080
starting: ThreadMemoria
sleeping...
starting: ThreadMemoria
sleeping...
PS C:\Users\Carlos\Desktop\at - python>
```

Figure 3 - Evidência servidor

```
ms-python.python-2020.11.371526539\pythonFiles\lib\python
cliente.py'
{'total': 34307923968, 'used': 7515508736}
PS C:\Users\Carlos\Desktop\at - python>
```

Figure 4 - Evidência response

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.

a.

```
PS C:\Users\Carlos\Desktop\at - python>  cd 'c:\Users\Ca
t - python\8 - threading.py'
4037913
PS C:\Users\Carlos\Desktop\at - python>
```

```python
import random
vet_a = []
def fatorial(n):
  fat = n
  for i in range(n-1, 1, -1):
    fat = fat * i
  return(fat)
resultado = 0
for n in range(0, 1000000):
  vet_a.append(random.randint(0, 10))
for t in vet_a:
  result = fatorial(t)
  resultado = resultado + result
print(resultado)
```

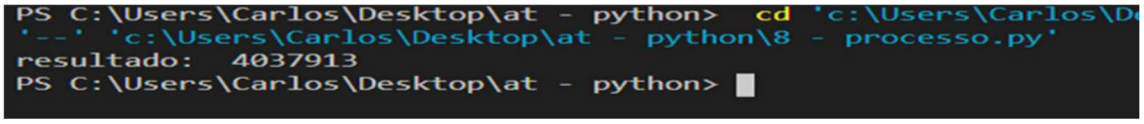
b.

```
import threading, time, random

def fatorial(n):
  fat = n
  for i in range(n-1,1,-1):
    fat = fat * i
  return(fat)

def to_list(function, veta, vetb):
  for e in veta:
    vetb.append(function(e))
veta = []
for n in range(0, 1000000):
  veta.append(random.randint(0, 10))
vetb = []
n = 500
size = len(veta)
inicio = time.time()
threading0 = threading.Thread(target = to_list, args = (fa-
torial, veta[0:int(size / 4)], vetb))
threading0.start()
threading1 = threading.Thread(target = to_list, args = (fa-
torial, veta[int(size / 4): int(size / 2)], vetb))
threading1.start()
threading2 = threading.Thread(target = to_list, args = (fa-
torial, veta[int(size / 2) : int(size * (3 / 4))], vetb))
threading2.start()
threading3 = threading.Thread(target = to_list, args = (fa-
torial, veta[int(size * (3 / 4)) : int(size)], vetb))
threading3.start()
threading0.join()
threading1.join()
threading2.join()
threading3.join()
resultado = 0
for n in vetb:
  resultado = resultado + n
print(resultado)
```

```
4037913
PS C:\Users\Carlos\Desktop\at - python>
```

c.

```
import time, multiprocessing, random
def fatorial(n):
  fat = n
  for i in range(n-1,1,-1):
    fat = fat * i
  return(fat)
def to_list(funcao, vetor_a, q):
    vetor_b = []
    for n in vetor_a:
        vetor_b.append(funcao(n))
    q.put(vetor_b)

if __name__ == "__main__":
    veta = []
    vetb = []
    n = 5000
    for n in range(0, 1000000):
        veta.append(random.randint(0, 10))
    queue = multiprocessing.Queue()
    tamanho = len(veta)
    time_inicio = time.time()
    p0 = multiprocessing.Process(target = to_list, args = (fato-
rial, veta[0:int(tamanho / 4)], queue))
    p0.start()
    p1 = multiprocessing.Process(target = to_list, args = (fato-
rial, veta[int(tamanho / 4): int(tamanho / 2)], queue))
    p1.start()
    p2 = multiprocessing.Process(target = to_list, args = (fato-
rial, veta[int(tamanho / 2) : int(tamanho * (3 / 4))], queue))
    p2.start()
    p3 = multiprocessing.Process(target = to_list, args = (fato-
rial, veta[int(tamanho * (3 / 4)) : int(tamanho)], queue))
    p3.start()
    while len(veta) != len(vetb):
        while queue.empty() is False:
            vetb += queue.get()
    p0.join()
    p1.join()
    p2.join()
    p3.join()
    time_fim = time.time()
    sum = 0
    for b in vetb:
        sum = sum + b
    print('resultado: ', sum)
```


PS C:\Users\Carlos\Desktop\at - python> cd 'c:\Users\Carlos\D
'--' 'c:\Users\Carlos\Desktop\at - python\8 - processo.py'
resultado: 4037913
PS C:\Users\Carlos\Desktop\at - python>

9.

```
import time, threading, multiprocessing, random

veta = []

for n in range(0, 1000000):
    veta.append(random.randint(0, 10))

inicio_sequencial = time.time()

def fatorial(n):
  fat = n
  for i in range(n-1, 1, -1):
    fat = fat * i
  return(fat)

resultado = 0
for t in veta:
  result = fatorial(t)
  resultado = resultado + result

fim_sequencial = time.time()

print('inicio medicao sequencial: ', inicio_sequencial)
print('fim medicao sequencial: ', fim_sequencial)
print('tempo gasto: ', fim_sequencial - inicio_sequencial)

################################################################################
def to_list(function, veta, vetb):
  for e in veta:
    vetb.append(function(e))

vetb = []
n = 500

size = len(veta)
inicio = time.time()

inicio_threading = time.time()

threading0 = threading.Thread(target = to_list, args = (fa-
torial, veta[0:int(size / 4)], vetb))
threading0.start()

threading1 = threading.Thread(target = to_list, args = (fa-
torial, veta[int(size / 4): int(size / 2)], vetb))
threading1.start()
```

```
threading2 = threading.Thread(target = to_list, args = (fa-
torial, veta[int(size / 2) : int(size * (3 / 4))], vetb))
threading2.start()

threading3 = threading.Thread(target = to_list, args = (fa-
torial, veta[int(size * (3 / 4)) : int(size)], vetb))
threading3.start()

threading0.join()
threading1.join()
threading2.join()
threading3.join()

fim_threading = time.time()

print('\n')

print('inicio medicao threading: ', inicio_threading)
print('fim medicao threading: ', fim_threading)
print('tempo gasto: ', fim_threading - inicio_threading)

################################################################################

def to_list_(funcao, vetor_a, q):
    vetor_b = []
    for n in vetor_a:
        vetor_b.append(funcao(n))
    q.put(vetor_b)

processing_inicio = time.time()

if __name__ == "__main__":
    veta = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    vetb = []
    n = 5000
    queue = multiprocessing.Queue()
    tamanho = len(veta)
    p0 = multiprocessing.Process(target = to_list_, args = (fato-
rial, veta[0:int(tamanho / 4)], queue))
    p0.start()

    p1 = multiprocessing.Process(target = to_list_, args = (fato-
rial, veta[int(tamanho / 4): int(tamanho / 2)], queue))
    p1.start()

    p2 = multiprocessing.Process(target = to_list_, args = (fato-
rial, veta[int(tamanho / 2) : int(tamanho * (3 / 4))], queue))
    p2.start()
```

```
    p3 = multiprocessing.Process(target = to_list_, args = (fato-
rial, veta[int(tamanho * (3 / 4)) : int(tamanho)], queue))
    p3.start()

    while len(veta) != len(vetb):
        while queue.empty() is False:
            vetb += queue.get()

    p0.join()
    p1.join()
    p2.join()
    p3.join()

processing_fim = time.time()

print('\n')

tempo = float(processing_fim - processing_inicio)

print('inicio medicao processing: ', processing_inicio)
print('fim medicao processing: ', processing_fim)
print('tempo gasto: ', tempo )
```

```
inicio medicao sequencial:  1607467616.8930037
fim medicao sequencial:  1607467621.580003
tempo gasto:  4.686999320983887
```

Figura 7 - Evidência sequêncial

```
inicio medicao threading:  1607467631.3800044
fim medicao threading:  1607467636.0950027
tempo gasto:  4.714998245239258
```

Figura 8 - Evidência threading

```
inicio medicao processing:  1607467636.0980024
fim medicao processing:  1607467636.0980024
tempo gasto:  0.0
```

Figura 9 - Evidência processing