

# Projeto de bloco: Arquitetura de Computadores, Sistemas Operacionais e Redes.



Projeto de Bloco da disciplina Projeto de Bloco: Arquitetura de Computadores, Sistemas Operacionais e Redes.



**Aluno:** Bruno Fernandes

**Email:** bruno.fernandes@al.infnet.edu.br

**Prof:** Alcione Dolavale

## Relatório:

Para acessar o repositório deste trabalho clique abaixo:

Bwendel26/PB\_INFNET\_Python\_redes\_SO

Um software cliente-servidor em Python que explore conceitos de arquitetura de redes, arquitetura de computadores e/ou de sistemas operacionais, acompanhado de relatório explicativo.

[https://github.com/Bwendel26/PB\\_INFNET\\_Python\\_redes\\_SO/tree/master/tp2](https://github.com/Bwendel26/PB_INFNET_Python_redes_SO/tree/master/tp2)



A screenshot of a GitHub repository page. The repository is named "Bwendel26/PB\_INFNET\_Python\_redes\_SO" and has 2 branches and 0 tags. The main branch is selected. The repository description is "Projeto do bloco de Arquitetura de Computadores, Sistemas Operacionais e Redes com Python da INFNET." The repository contains a file tree with folders .idea, imgs, tp2, tp3, tp4, tp5, tp6, tp7, tp8, tp9, and files Bruno\_Fernandes\_PB\_TP3.zip, README.md, and Relatorio.pdf. The right sidebar shows the "About" section with the project description, a "Readme" link, and sections for "Releases" (No releases published), "Packages" (No packages published), and "Languages" (Python 100.0%).

## Ferramentas utilizadas:

Linguagem de programação: Python (versão 3.9)

IDEs: PyCharm Community Edition 64 bits | Visual Studio Code

### Bibliotecas utilizadas:

- Pygame

Pygame Front Page - pygame v2.0.0.dev25 documentation

Basic information about pygame: what it is, who is involved, and where to find it. Steps needed to compile pygame on several platforms. Also help on finding and installing prebuilt binaries for


 <https://www.pygame.org/docs/>



- psutil

psutil documentation - psutil 5.7.4 documentation

psutil (python system and process utilities) is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in Python. It is useful mainly for system monitoring, profiling, limiting process resources and the management of

 <https://psutil.readthedocs.io/en/latest/>

- platform

platform - Access to underlying platform's identifying data - Python 3.9.0 documentation

Source code: Lib/platform.py Note Specific platforms listed alphabetically, with Linux included in the Unix section. Queries the given executable (defaults to the Python interpreter binary) for various architecture information. Returns a tuple which contain information about the bit architecture and the

 <https://docs.python.org/3/library/platform.html>

- cpuinfo

workhorsy/py-cpuinfo

Py-cpuinfo gets CPU info with pure Python. Py-cpuinfo should work without any extra programs or libraries, beyond what your OS provides. It does not require any

 <https://github.com/workhorsy/py-cpuinfo>



- OS

#### os - Miscellaneous operating system interfaces - Python 3.9.2 documentation

This module provides a portable way of using operating system dependent functionality. If you just want to read or write a file see , if you want to manipulate paths, see the module, and if you want to read all the lines in all the files on the command line see the module.

 <https://docs.python.org/pt-br/3/library/os.html>

- socket

#### socket - Low-level networking interface - Python 3.9.2 documentation

Source code: Lib/socket.py This module provides access to the BSD socket interface. It is available on all modern Unix systems, Windows, MacOS, and probably additional platforms. Note Some behavior may be platform dependent, since calls are made to the operating system socket APIs.

 <https://docs.python.org/3/library/socket.html>

- tkinter

#### tkinter - Python interface to Tcl/Tk - Python 3.9.2 documentation

Source code: Lib/tkinter/\_\_init\_\_.py The package ("Tk interface") is the standard Python interface to the Tk GUI toolkit. Both Tk and are available on most Unix platforms, as well as on Windows systems. (Tk itself is not part of Python; it is maintained at ActiveState.)

 <https://docs.python.org/3/library/tkinter.html>

- datetime

#### datetime - Basic date and time types - Python 3.9.2 documentation

Only one concrete class, the class, is supplied by the module. The class can represent simple timezones with fixed offsets from UTC, such as UTC itself or North American EST and EDT timezones. Supporting timezones at deeper levels of detail is up to the application.

 <https://docs.python.org/3/library/datetime.html>

- sched

### **sched - Event scheduler - Python 3.9.2 documentation**

Source code: Lib/sched.py The module defines a class which implements a general purpose event scheduler: The class defines a generic interface to scheduling events. It needs two functions to actually deal with the "outside world" - timefunc should be callable without arguments, and return a

 <https://docs.python.org/3/library/sched.html>

- **time**

### **time - Time access and conversions - Python 3.9.2 documentation**

This module provides various time-related functions. For related functionality, see also the and modules. Although this module is always available, not all functions are available on all platforms. Most of the functions defined in this module call platform C library functions with the same name.

 <https://docs.python.org/3/library/time.html>

- **subprocess**


### **subprocess - Subprocess management - Python 3.9.2 documentation**

Source code: Lib/subprocess.py The module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. This module intends to replace several older modules and functions: Information about how the module can be used to replace these modules

 <https://docs.python.org/3/library/subprocess.html>

- **pickle**

### **pickle - Python object serialization - Python 3.9.2 documentation**

 <https://docs.python.org/3/library/pickle.html>

- **nmap**

### **python-nmap**

This is a python class to use nmap and access scan results from python3

 <https://pypi.org/project/python-nmap/>

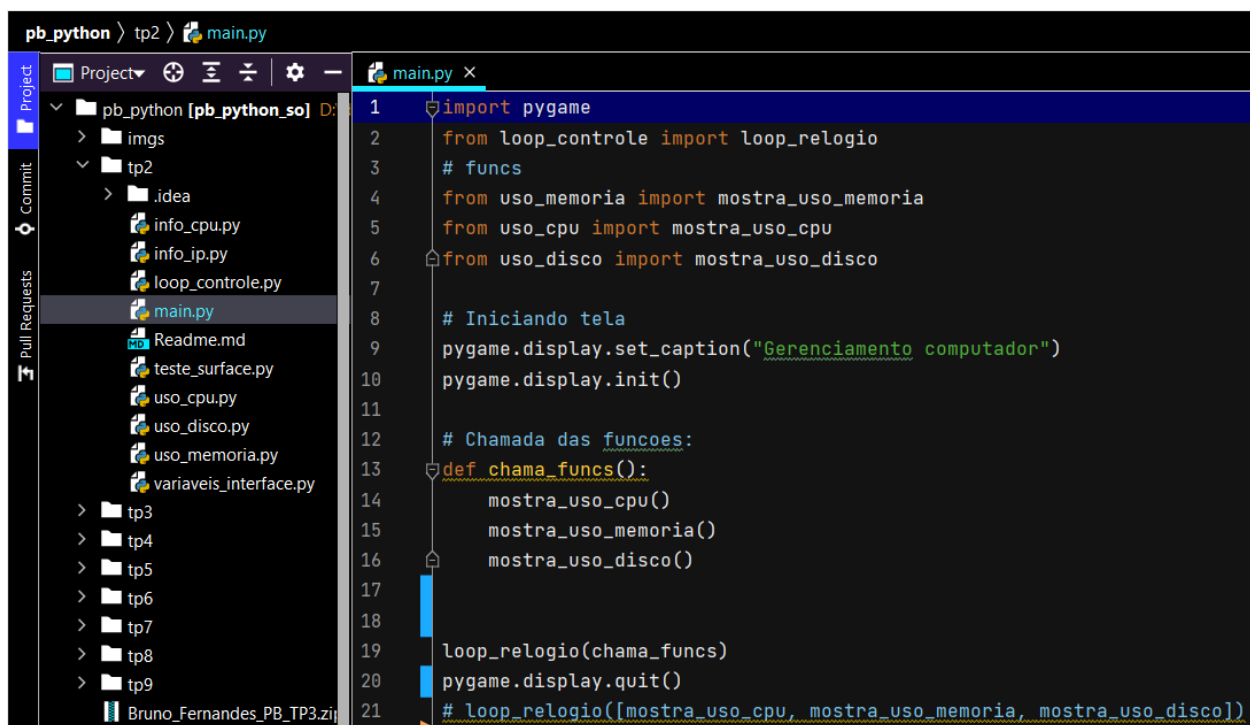


## Objetivo:

Um aplicativo simples de apresentação textual do monitoramento e análise de computadores em rede.

No TP2 o objetivo foi introduzir a prática do Projeto de Bloco, introduzindo o Pygame para disposição do Front-end da aplicação, trazendo informações básicas sobre o sistema.

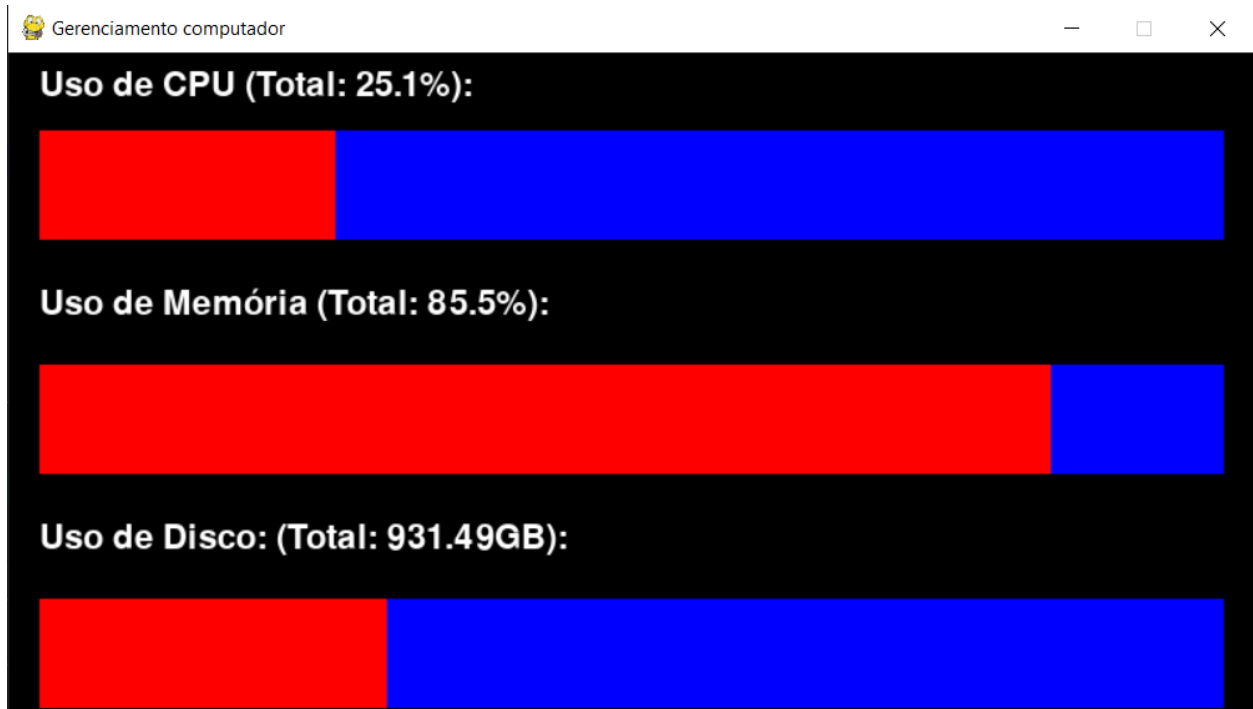
A estrutura do TP2 foi a seguinte:



The screenshot shows a code editor with a project structure on the left and the code for `main.py` on the right. The project structure includes folders `pb_python` and `tp2`, with `main.py` selected. The code in `main.py` is as follows:

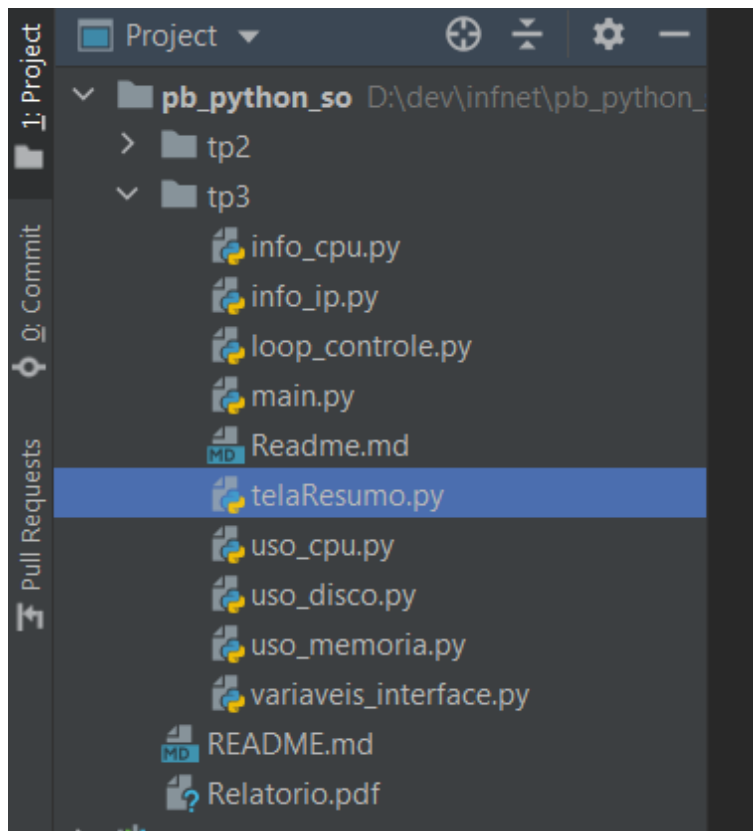
```
1 import pygame
2 from loop_controle import loop_relogio
3 # funcs
4 from uso_memoria import mostra_uso_memoria
5 from uso_cpu import mostra_uso_cpu
6 from uso_disco import mostra_uso_disco
7
8 # Iniciando tela
9 pygame.display.set_caption("Gerenciamento computador")
10 pygame.display.init()
11
12 # Chamada das funcoes:
13 def chama_funcs():
14     mostra_uso_cpu()
15     mostra_uso_memoria()
16     mostra_uso_disco()
17
18
19 loop_relogio(chama_funcs)
20 pygame.display.quit()
21 # loop_relogio([mostra_uso_cpu, mostra_uso_memoria, mostra_uso_disco])
```

Resultado ao rodar o `main.py`, chama as funcionalidades que serão apresentadas:



No TP3 o objetivo foi continuar a montagem de um pequeno sistema desktop de monitoramento dos componentes principais do computador. Foi codificado uma série de algoritmos, na linguagem Python, que utilizaram bibliotecas da linguagem para monitoramento do computador e para criar a interface que mostra os gráficos e informações a serem apresentadas pelo usuário.

A estrutura do TP3 foi a seguinte:



Como podemos ver os arquivos foram divididos um para cada funcionalidade, e o arquivo main.py (principal) chama todas as funções que executam o programa por um todo. Como podemos ver temos funções para cada chamada (uso\_cpu.py, uso\_memoria.py, uso\_disco, info\_cpu, etc...).

Temos o script main.py que faz a chamada das funções e variáveis dentro dos demais arquivos e consequentemente executa:



```
main.py x
1 import pygame
2 from loop_controle import loop_relogio
3
4 import variaveis_interface as int_vars
5 # funcs
6 from uso_memoria import mostra_uso_memoria
7 from uso_cpu import cpu_tela
8 from uso_disco import mostra_uso_disco
9 from telaResumo import main
10 |
11 # Iniciando tela
12 pygame.display.set_caption("Gerenciamento computador")
13 pygame.display.init()
14 int_vars.tela.fill(int_vars.PRETO)
15
16 loop_relogio([cpu_tela, mostra_uso_memoria, mostra_uso_disco], main)
17
18 pygame.display.quit()
```

Logo no início fazemos o import da função loop\_relogio, do arquivo loop\_controle.py, que faz o controle das chamadas das funções dentro de um timer:

```
loop_controle.py x
2
3 def loop_relogio(funcoes, tela_resumo):
4     # cria relogio
5     clock = pygame.time.Clock()
6     cont = 60
7
8     terminou = False
9     func_atual = 0 #controle de funcao apresentada
10
11     while not terminou:
12         # Checar os eventos do mouse aqui:
13         for event in pygame.event.get():
14             if event.type == pygame.QUIT:
15                 terminou = True
16
17             if event.type == pygame.KEYDOWN:
18                 if event.key == pygame.K_RIGHT:
19                     func_atual += 1
20                     if func_atual >= len(funcoes):
21                         func_atual = 0
22                     funcoes[func_atual]()
23                 if event.key == pygame.K_LEFT:
24                     func_atual -= 1
25                     if func_atual < 0:
26                         func_atual = len(funcoes) - 1
27                     funcoes[func_atual]()
28
29                 # CHAMADA DA TELA DE RESUMO:
30                 if event.key == pygame.K_SPACE:
31                     func_atual = 10
32                     if func_atual == 10:
33                         tela_resumo()
34
35         # atualiza desenho
36         if cont == 60:
37             if func_atual == 10:
38                 tela_resumo()
39             else:
40                 funcoes[func_atual]()
41             # for i in range(len(funcoes)):
42             #     funcoes[i]()
43
44             cont = 0
45         # Atualiza o desenho na tela
46         pygame.display.update()
47         # 60 frames por segundo
48         clock.tick(60)
49         cont += 1
50
51     loop_relogio()
```

Podemos observar que essa função faz todo o controle do tempo, chamada das funções e controle de cada evento do pygame, seja no momento de sair do

programa ou ao utilizar setas ou barra de espaço para fazer a navegação dentre as telas que fazem o monitoramento do computador dentro do software.

A função `loop_relogio` recebe como parâmetros uma lista de funções para serem chamadas e uma função separada para chamada da tela de resumo.

Dentro do código temos um arquivo Python específico para as variáveis gerais utilizadas no sistema, segue a imagem:

```
variaveis_interface.py x
1  import pygame
2
3  AZUL = (0, 0, 255)
4  VERMELHO = (255, 0, 0)
5  PRETO = (0, 0, 0)
6  BRANCO = (255, 255, 255)
7  CINZA = (100, 100, 100)
8  dimensoes = [800, 600]
9
10 #Iniciando tela
11 tela_largura = dimensoes[0]
12 tela_altura = dimensoes[1]
13 tela = pygame.display.set_mode((tela_largura, tela_altura))
14 # fonte
15 pygame.font.init()
16 font = pygame.font.Font(None, 32)
17
18 #Surfaces:
19 s1 = pygame.surface.Surface((tela_largura, tela_altura / 3))
20 s2 = pygame.surface.Surface((tela_largura, tela_altura / 3))
21 s3 = pygame.surface.Surface((tela_largura, tela_altura / 3))
22
23 #Surfaces_resumo
24 sr1 = pygame.surface.Surface((tela_largura, tela_altura / 5))
25 sr2 = pygame.surface.Surface((tela_largura, tela_altura / 5))
26 sr3 = pygame.surface.Surface((tela_largura, tela_altura / 5))
27 sr4 = pygame.surface.Surface((tela_largura, tela_altura / 5))
28 sr5 = pygame.surface.Surface((tela_largura, tela_altura / 5))
29
```

Dentro do sistemas temos um arquivo para mostrar de cada funcionalidade;

- Temos os arquivos de controle da cpu:

```

1 import psutil
2 import cpuinfo
3 import pygame
4 from info_cpu import mostra_info_cpu
5 import variaveis_interface as int_vars
6
7 #vars
8 surface1 = int_vars.s1
9 surface2 = int_vars.s2
10 lista_percentual_cpus = psutil.cpu_percent(1, percpu=True)
11
12 #funçs
13 def percentual_cpu():
14     """
15     Função que retorna o percentual de uso da CPU
16     do computador.
17     :param tempo: em segundos para retornar o percentual.
18     :return: float percentual-usado
19     """
20     percentual_usado = psutil.cpu_percent(interval=0)
21
22     return percentual_usado
23
24
25 def mostra_uso_cpu():
26     capacidade = percentual_cpu()
27     larg = int_vars.tela_largura - 2*20
28     int_vars.tela.fill(int_vars.PRETO)
29     pygame.draw.rect(surface1, int_vars.AZUL, (20, 50, larg, 70))
30     larg = larg*capacidade/100
31     pygame.draw.rect(surface1, int_vars.VERMELHO, (20, 50, larg, 70))
32     texto_barra = "Uso de CPU (Total: " + str(capacidade) + "%):"
33     text = int_vars.font.render(texto_barra, 1, int_vars.BRANCO)
34     int_vars.tela.blit(surface1, (0, 0)) # setando divisao tela
35     int_vars.tela.blit(text, (20, 10))
36
37
38 def mostra_uso_de_cada_cpu(s, l_cpu_percent):
39     s.fill(int_vars.CINZA)
40     int_vars.tela.blit(s, (0, int_vars.tela_altura / 3))
41
42     num_cpu = len(l_cpu_percent)

```

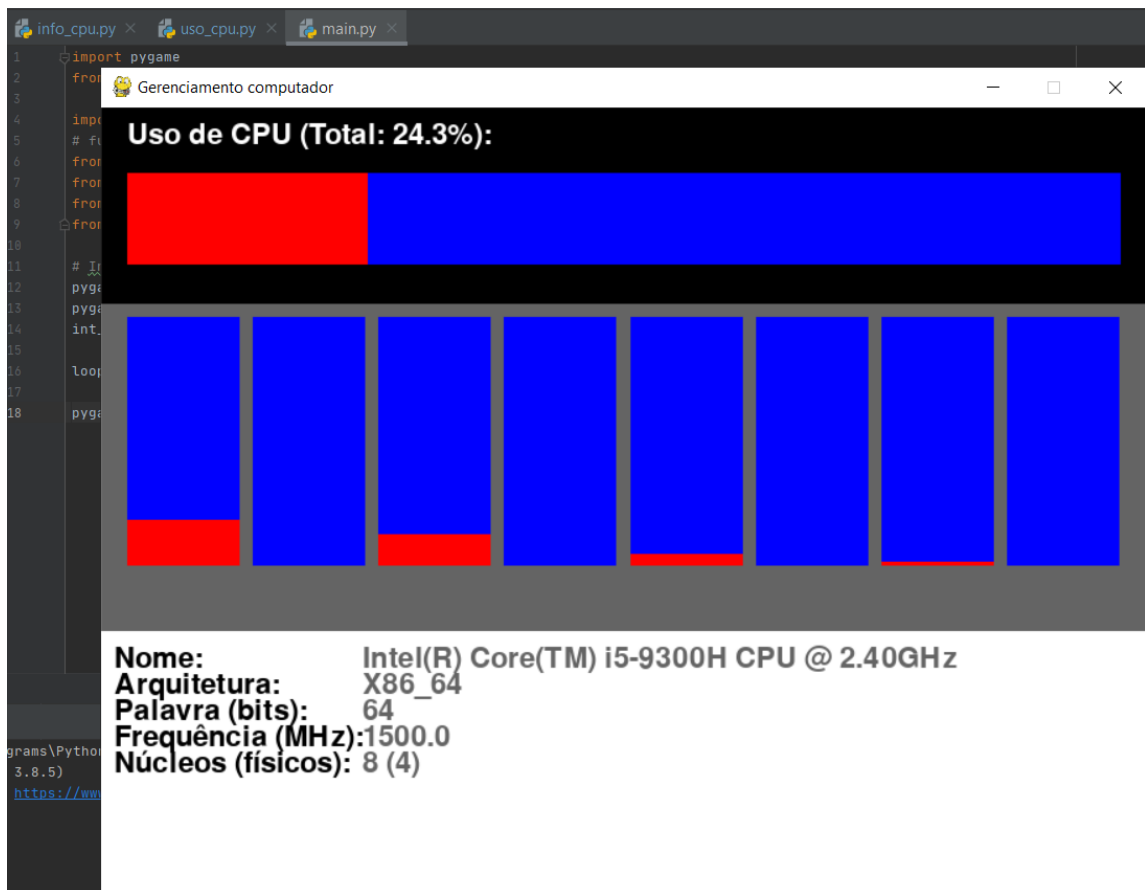
percentual\_cpu()

Nesse arquivo temos as funções que mostram o uso da cpu e uma para cada um de seus núcleos, fazendo a plotagem com pygame, em barras que mostram o total de uso.

Logo em seguida temos um arquivo para capturar e apresentar as informações sobre a cpu e sistema no geral.

```
info_cpu.py x uso_cpu.py x main.py x
4 import pygame
5 import cpuinfo
6
7 info = cpuinfo.get_cpu_info()
8 surface3 = int_vars.s3
9
10 def mostra_texto(s1, nome, chave, pos_y):
11     text = int_vars.font.render(nome, True, int_vars.PRETO)
12     s1.blit(text, (10, pos_y))
13     if chave == "freq":
14         s = str(round(psutil.cpu_freq().current, 2))
15     elif chave == "nucleos":
16         s = str(psutil.cpu_count())
17         s = s + " (" + str(psutil.cpu_count(logical=False)) + ")"
18     else:
19         s = str(info[chave])
20     text = int_vars.font.render(s, True, int_vars.CINZA)
21     s1.blit(text, (200, pos_y))
22
23 def mostra_info_cpu():
24     # Mostra as informações de CPU escolhidas:
25     surface3.fill(int_vars.BRANCO)
26     mostra_texto(surface3, "Nome:", "brand_raw", 10)
27     mostra_texto(surface3, "Arquitetura:", "arch", 30)
28     mostra_texto(surface3, "Palavra (bits):", "bits", 50)
29     mostra_texto(surface3, "Frequência (MHz):", "freq", 70)
30     mostra_texto(surface3, "Núcleos (físicos):", "nucleos", 90)
31     int_vars.tela.blit(surface3, (0, 2 * int_vars.tela_altura / 3))
```

E o resultado da chamada da função main quando chamamos a tela de cpu é o seguinte:



- Temos, também, o arquivo de controle da memória principal (RAM):

```

1  import psutil
2  import pygame
3  import variaveis_interface as int_vars
4  #variaveis
5  memoria = psutil.virtual_memory()
6
7  #funcs
8  def percentual_memoria():
9      """
10     Função que retorna o percentual de memória RAM
11     utilizado por um computador.
12     :return: float percentual_usado
13     """
14
15     mem = psutil.virtual_memory().percent
16     return mem
17
18     surface1 = int_vars.s1
19
20     def mostra_uso_memoria():
21         mem = percentual_memoria()
22         larg = int_vars.tela_largura - 2 * 20
23         int_vars.tela.fill(int_vars.PRETO)
24         pygame.draw.rect(surface1, int_vars.AZUL, (20, 50, larg, 70))
25         larg = larg * mem / 100
26         pygame.draw.rect(surface1, int_vars.VERMELHO, (20, 50, larg, 70))
27         texto_barra = "Uso de Memória (Total: " + str(mem) + "%):"
28         text = int_vars.font.render(texto_barra, 1, int_vars.BRANCO)
29         int_vars.tela.blit(surface1, (0, 0)) # setando divisao tela
30         int_vars.tela.blit(text, (20, 10))

```

temos uma função que extrai a informação sobre a porcentagem de uso da memória e a segunda função usa a biblioteca pygame para mostrar a barra de uso da memória.

- Função que mostra os discos e uso do disco em percentual:

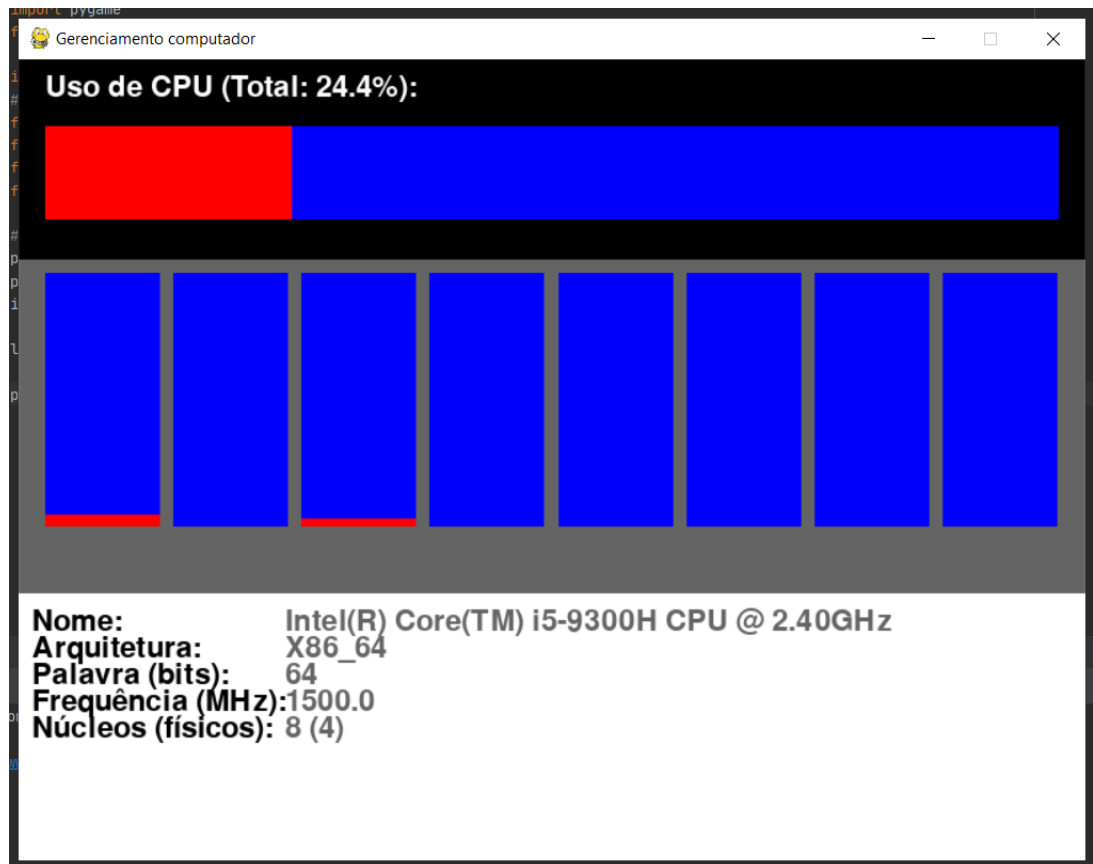


```
main.py x uso_disco.py x
1 import psutil
2 import pygame
3 import variaveis_interface as int_vars
4
5 surface1 = int_vars.s1
6
7 def mostra_uso_disco():
8     # disk = str(disk)
9     disco = psutil.disk_usage('.')
10    larg = int_vars.tela_largura - 2 * 20
11    int_vars.tela.fill(int_vars.PRETO)
12    pygame.draw.rect(surface1, int_vars.AZUL, (20, 50, larg, 70))
13    larg = larg * disco.percent / 100
14    pygame.draw.rect(surface1, int_vars.VERMELHO, (20, 50, larg, 70))
15    total = round(disco.total / (1024 * 1024 * 1024), 2)
16    texto_barra = "Uso de Disco: (Total: " + str(total) + "GB):"
17    text = int_vars.font.render(texto_barra, 1, int_vars.BRANCO)
18    int_vars.tela.blit(surface1, (0, 0)) #setando divisao tela
19    int_vars.tela.blit(text, (20, 10)) # setando posicao do text
20
21 # print(psutil.disk_partitions())
22 # Modificar para iterar sobre cada disco do pc
23 # e consequentemente criar uma surface para cada um.
```

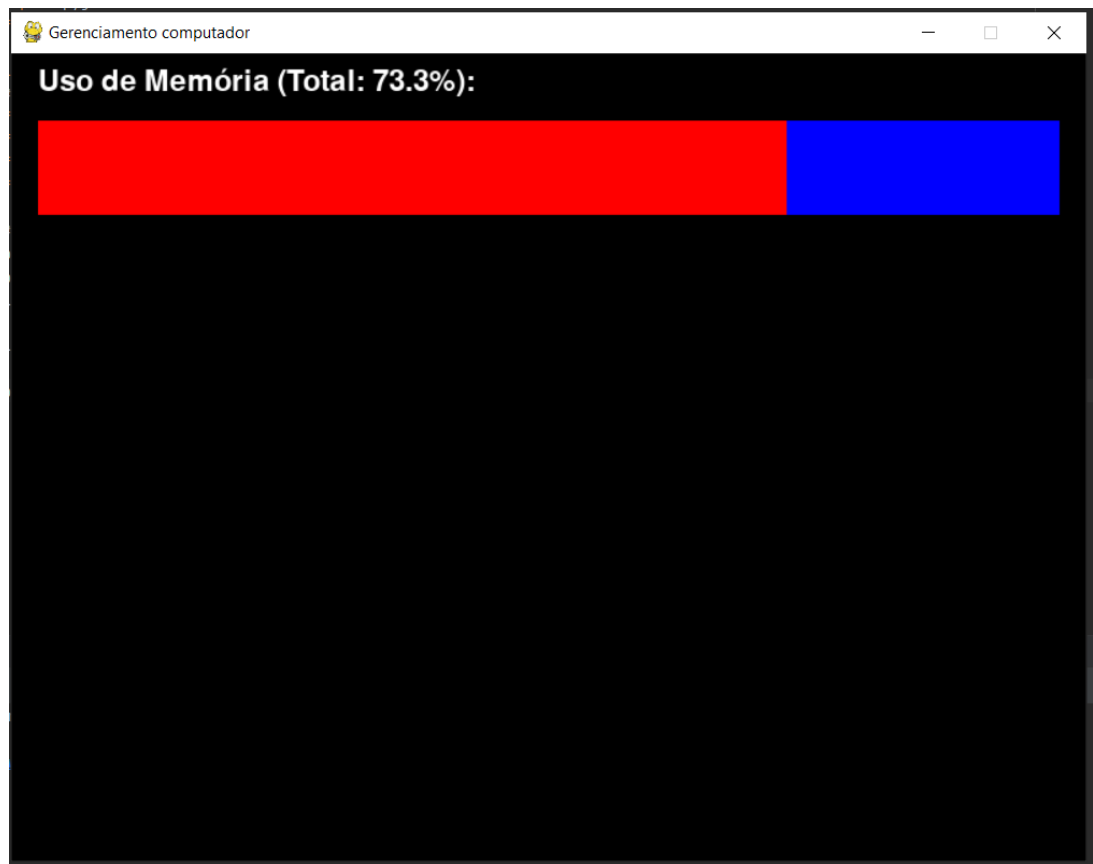
Temos uma função que pega, diretamente da biblioteca psutil, o percentual da memória utilizada e após mostra na interface do pygame.

Como resultado final temos a seguinte interface controlada por pelas setas laterais:

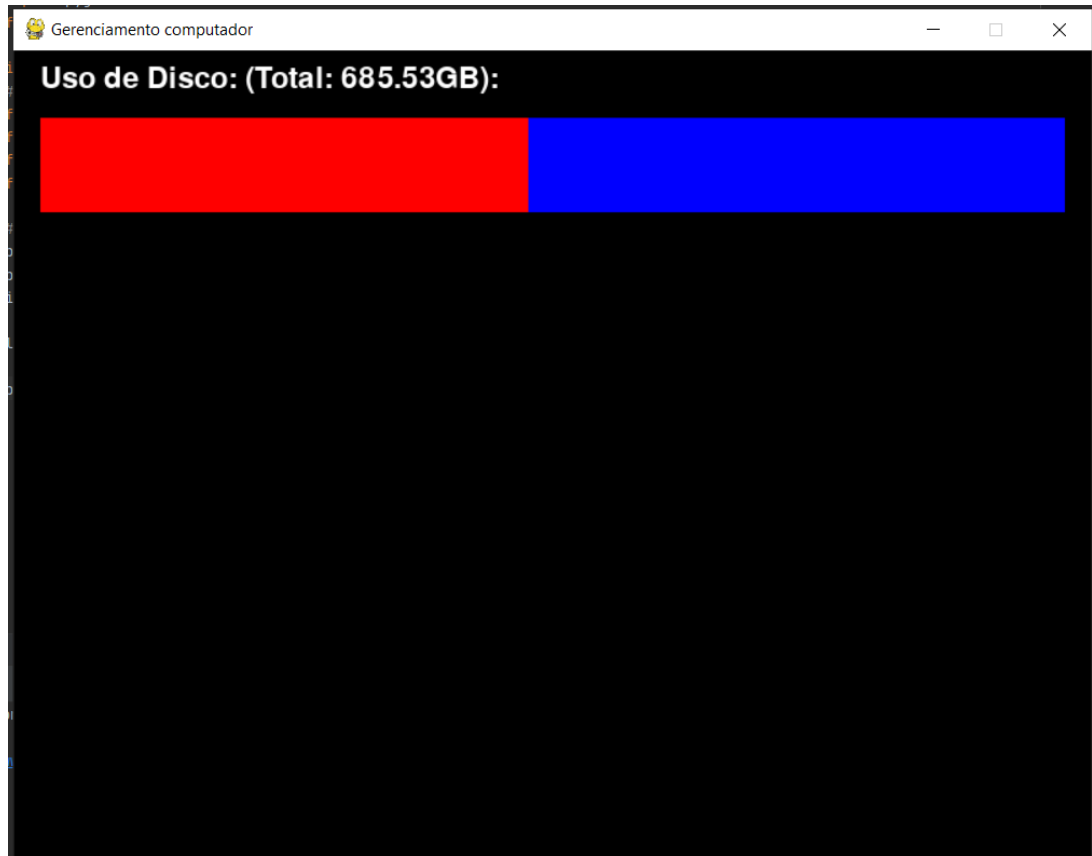
Tela 1



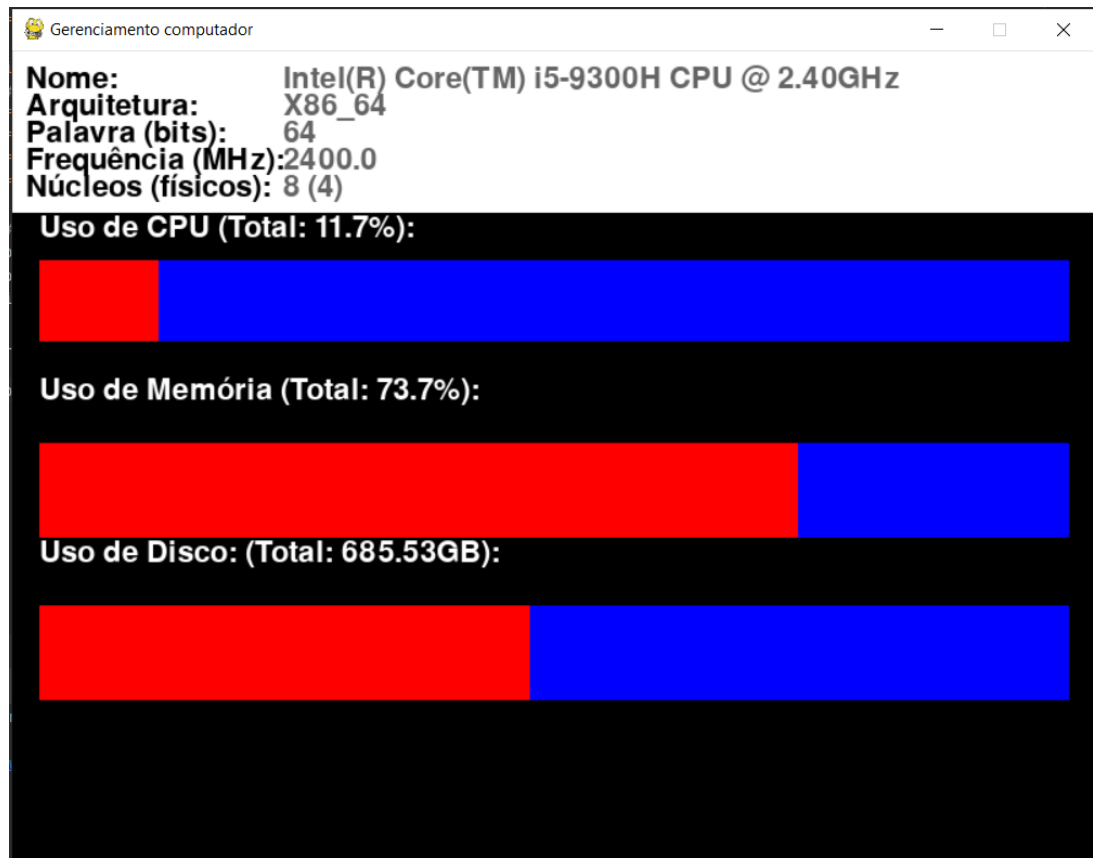
Tela 2



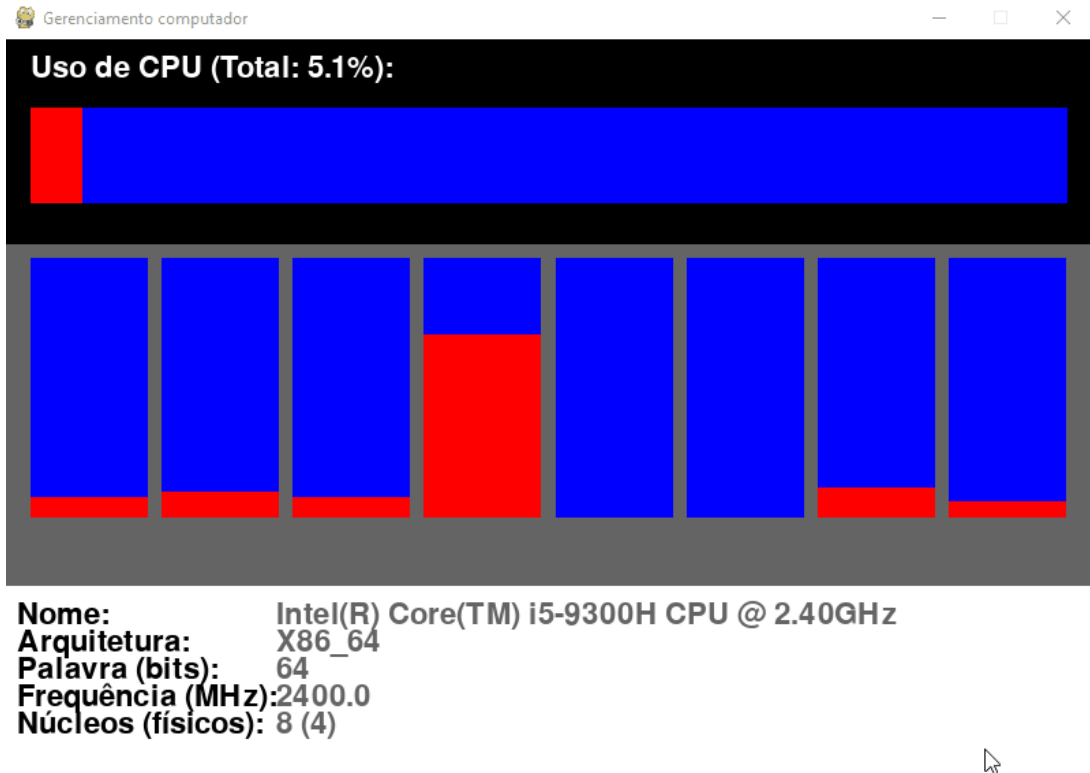
Tela 3



Tela 4 (Resumo)

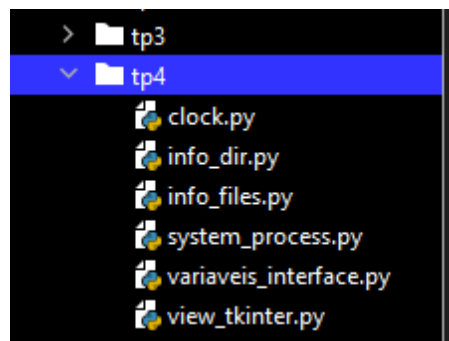


Apresentação do TP3 rodando:



No TP4 tivemos continuidade em extração de informações do sistema, que consiste no objetivo do projeto como um todo. Utilizamos a biblioteca `datetime` para extrair as datas de modificação de processos, arquivos e diretórios.

A estrutura do TP4 foi a seguinte:



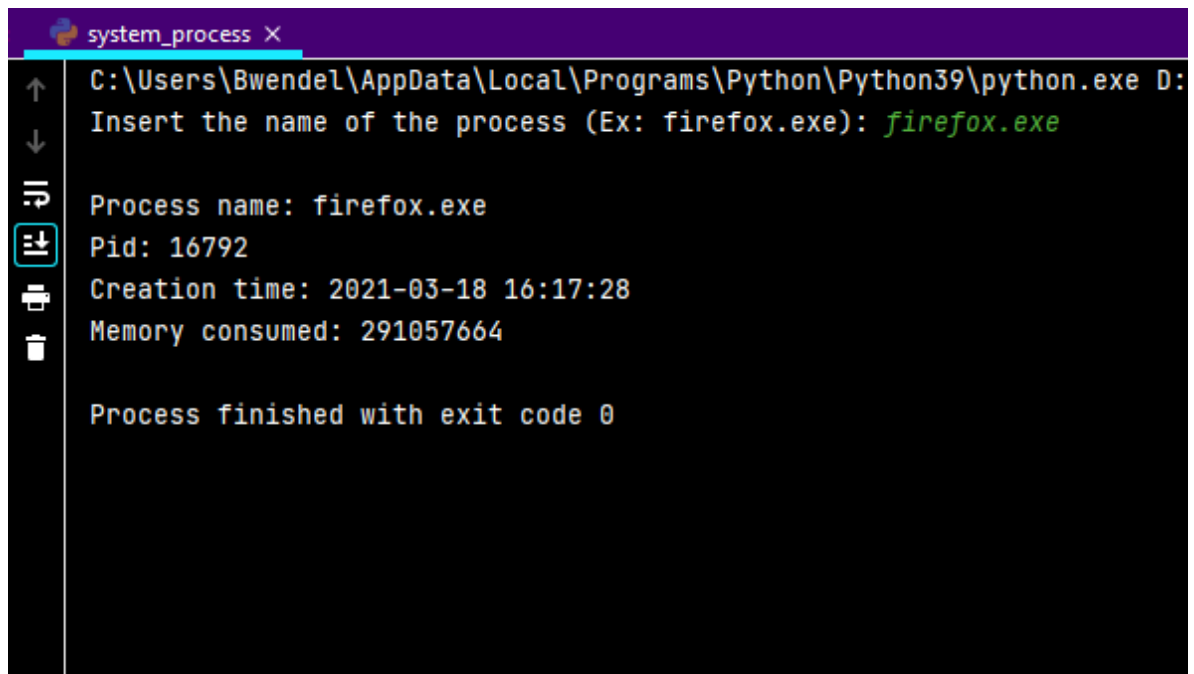
- O código em `system_process.py` tem como objetivo trazer informações sobre um processo que está em execução na máquina

```

system_process.py X
1  """
2  Criar uma ou mais funções que retornem ou apresentem
3  informações sobre processos do sistema. As informações
4  podem ser: PID, nome do executável, consumo de processamento,
5  consumo de memória, entre outras disponíveis no módulo 'psutil' de Python.
6  """
7  import psutil
8  import datetime
9  # import pygame
10 # import variaveis_interface as iv
11 # from clock import clock_pygame as clock
12 |
13 def process_info(p_name):
14     pid = False
15     name = p_name
16     c_time = 0
17     p_mem = 0
18
19     for procs in psutil.process_iter(["pid", "name", "username"]):
20         if str(procs.name()) == name:
21             c_time = datetime.datetime.fromtimestamp(procs.create_time()).strftime("%Y-%m-%d %H:%M:%S")
22             pid = str(procs.pid)
23             p_mem = procs.memory_info()
24         if pid:
25             return "\nProcess name: " + name + \
26                 "\nPid: " + pid + \
27                 "\nCreation time: " + c_time + \
28                 "\nMemory consumed: " + str(p_mem.rss)
29
30     else:
31         return "This process do not exist."
32
33
34 e_info = str(input("Insert the name of the process (Ex: firefox.exe): "))
35 print(process_info(e_info))
36
37 """

```

Tem como retorno:



```
system_process X
C:\Users\Bwendel\AppData\Local\Programs\Python\Python39\python.exe D:
Insert the name of the process (Ex: firefox.exe): firefox.exe

Process name: firefox.exe
Pid: 16792
Creation time: 2021-03-18 16:17:28
Memory consumed: 291057664

Process finished with exit code 0
```

Em seguida temos os códigos referentes a informações sobre um arquivo e um diretório do sistema:



```
view_tkinter.py × info_files.py × info_dir.py ×
1 """
2 Criar uma ou mais funções que retornem ou apresentem
3 informações sobre diretórios e arquivos. Tais informações
4 podem ser qualquer uma que você achar relevante disponível
5 no módulo 'os' e 'psutil' de Python, como nome, tamanho,
6 localização, data de criação, data de modificação, tipo, etc.
7 """
8 import os
9 import os.path
10 import time
11
12
13 def file_info(file):
14     path = ".\\"
15     os.chdir(path)
16     current_path = os.getcwd()
17     status = os.stat(file)
18     file_size = os.path.getsize(file)
19     created_time = time.ctime(status.st_ctime)
20     mod_time = time.ctime(status.st_mtime)
21     name_type = str(file).split(".")
22
23     final = "File name: " + name_type[0] + \
24         "\nType: " + name_type[1] + \
25         "\nLocation: " + str(current_path) + \
26         "\nSize: " + str(file_size) + \
27         "bytes.\nCreated at: " + created_time + \
28         "\nLast modification: " + mod_time
29
30     return final
```

Rodando:

```
File name: README  
Type: md  
Location: D:\dev\infnet\bloco_python\pb_python\tp4  
Size: 988bytes.  
Created at: Thu Mar 18 20:10:02 2021  
Last modification: Thu Mar 18 20:10:46 2021
```

```
view_tkinter.py × info_dir.py ×
1 """
2     Criar uma ou mais funções que retornem ou apresentem
3     informações sobre diretórios e arquivos. Tais informações
4     podem ser qualquer uma que você achar relevante disponível
5     no módulo 'os' e 'psutil' de Python, como nome, tamanho,
6     localização, data de criação, data de modificação, tipo, etc.
7 """
8 import os
9 import os.path
10 import time
11
12
13 def dir_info(dir_path):
14     os.chdir(dir_path)
15     current_path = os.getcwd()
16     status = os.stat(".\\")
17     created_time = time.ctime(status.st_ctime)
18     mod_time = time.ctime(status.st_mtime)
19
20     path_list = current_path.split("\\")
21     length_path_list = len(path_list) - 1
22     name = path_list[length_path_list]
23
24     final = "Current path: " + str(current_path) + \
25             "\nPath name: " + name + \
26             "\nCreated at: " + str(created_time) + \
27             "\nLast modification date: " + str(mod_time)
28
29     return final
30
31
32 print(dir_info("./"))
```

Rodando:

```
Current path:D:\dev\infnet\bloco_python\pb_python\tp4
Path name: tp4
Created at: Tue Mar  2 12:05:33 2021
Last modification date: Thu Mar 18 20:10:59 2021

Process finished with exit code 0
```

Em seguida foi solicitado o uso de uma GUI (Interface gráfica de usuário), não necessariamente sendo o Pygame, portanto entramos no desafio de usar o TKinter, bem famoso na comunidade do python, com o objetivo de retornar as informações nos dois códigos anteriores.

O código foi o seguinte:

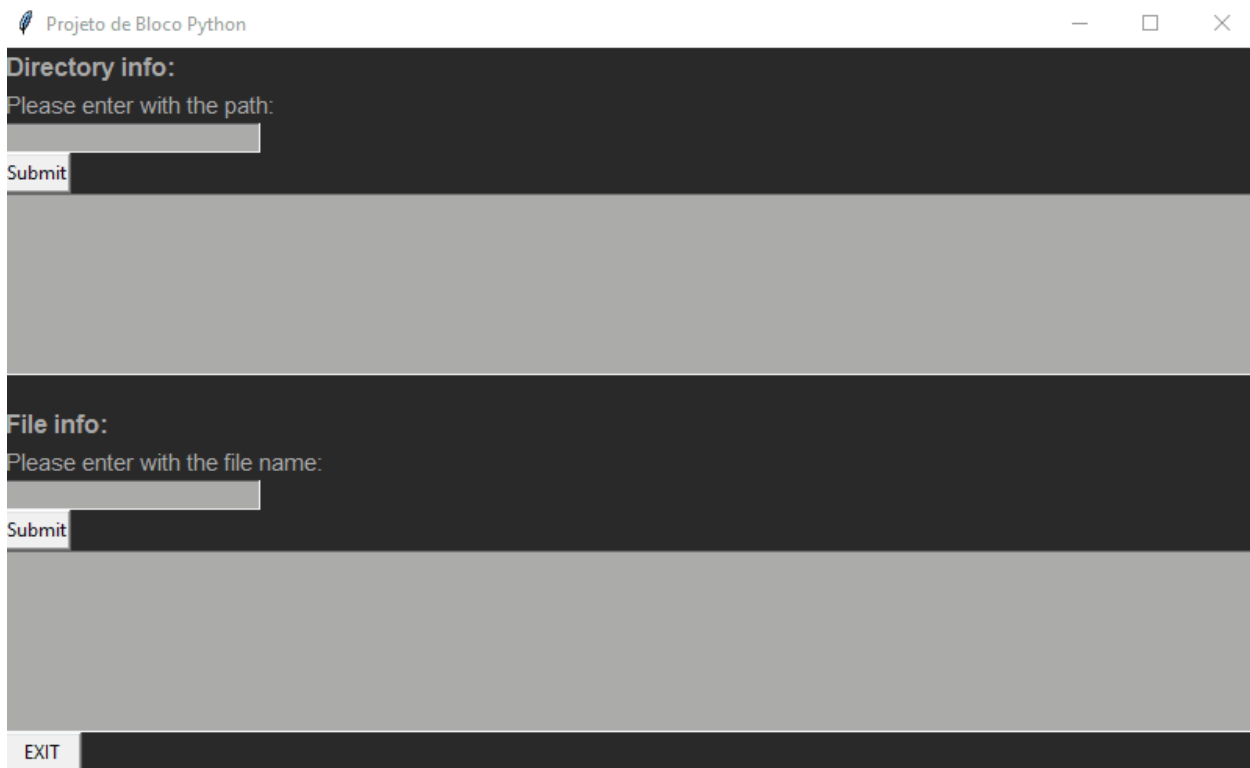
```

view_tkinter.py X
tp4 > view_tkinter.py > ...
6 from tkinter import *
7 from info_dir import dir_info
8 from info_files import file_info
9
10 # Vars
11 black = "#292929"
12 white = "#ababab"
13 font1 = "none 12 bold"
14 font2 = "none 11"
15
16
17 # Funcs
18 def click(out, func):
19     """
20     This func collect the text from the entry box
21     and execute the function by clicking in the button.
22     :return: None
23     """
24     entered_txt = textentry.get()
25     out.delete(0.0, END)
26     try:
27         info = func(str(entered_txt))
28     except:
29         info = "Sorry, do you passed a wrong parameter"
30
31     out.insert(END, info)
32
33
34 def close_window():
35     window.destroy()
36     exit()
37
38
39 # Screen set
40 window = Tk()
41 window.title("Projeto de Bloco Python")
42 window.configure(background=black)
43 # Content
44 # Dir:
45 Label(window, text="Directory info:", bg=black, fg=white, font=font1).grid(row=0, column=0, sticky=W)
46 Label(window, text="Please enter with the path:", bg=black, fg=white, font=font2).grid(row=1, column=0, sticky=W)
47 textentry = Entry(window, width=27, bg=white)
48 textentry.grid(row=2, column=0, sticky=W)
49 output_dir = Text(window, width=100, height=7, wrap=WORD, background=white)
50 output_dir.grid(row=4, column=0, sticky=W)
51 Button(window, text="Submit", width=5, command=lambda: click(output_dir, dir_info)).grid(row=3, column=0, sticky=W)
52 # File:
53 Label(window, text="\nFile info:", bg=black, fg=white, font=font1).grid(row=5, column=0, sticky=W)
54 Label(window, text="Please enter with the file name:", bg=black, fg=white, font=font2).grid(row=6, column=0, sticky=W)
55 textentry = Entry(window, width=27, bg=white)
56 textentry.grid(row=7, column=0, sticky=W)
57 output_file = Text(window, width=100, height=7, wrap=WORD, background=white)
58 output_file.grid(row=9, column=0, sticky=W)
59 Button(window, text="Submit", width=5, command=lambda: click(output_file, file_info)).grid(row=8, column=0, sticky=W)
60 # Exit
61 Button(window, text="EXIT", width=6, command=close_window).grid(row=10, column=0, sticky=W)
62 # Run the main loop
63 window.mainloop()

```

Nota: Aqui tivemos o problema do Tkinter, por algum motivo não retornar as informações sobre diretórios, independente do diretório usado ou ambiente.

No teste usamos o diretório atual como exemplo("./").



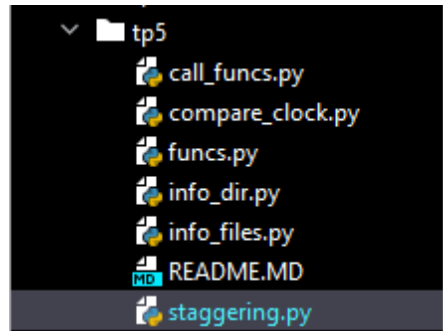
Sobre o erro apresentado acima, podemos observar, no exemplo abaixo, que a função que faz o retorno das informações sobre diretórios está funcionando e está retornando no mesmo formato que a de arquivos;

```
Current path:D:\dev\infnet\bloco_python\pb_python\tp4
Path name: tp4
Created at: Tue Mar  2 12:05:33 2021
Last modification date: Thu Mar 18 20:10:59 2021

Process finished with exit code 0
```

No TP5 introduzimos a utilização do módulo sched, que faz chamadas escalonadas do código conforme solicitado, utilizamos ele com objetivo de retornar funções, já antes feitas, de forma que o código executasse em um determinado tempo.

A estrutura do TP5 foi a seguinte:



Logo de início temos o código apresentado em staggering.py, que é a resolução da primeira questão do TP, com o seguinte código:

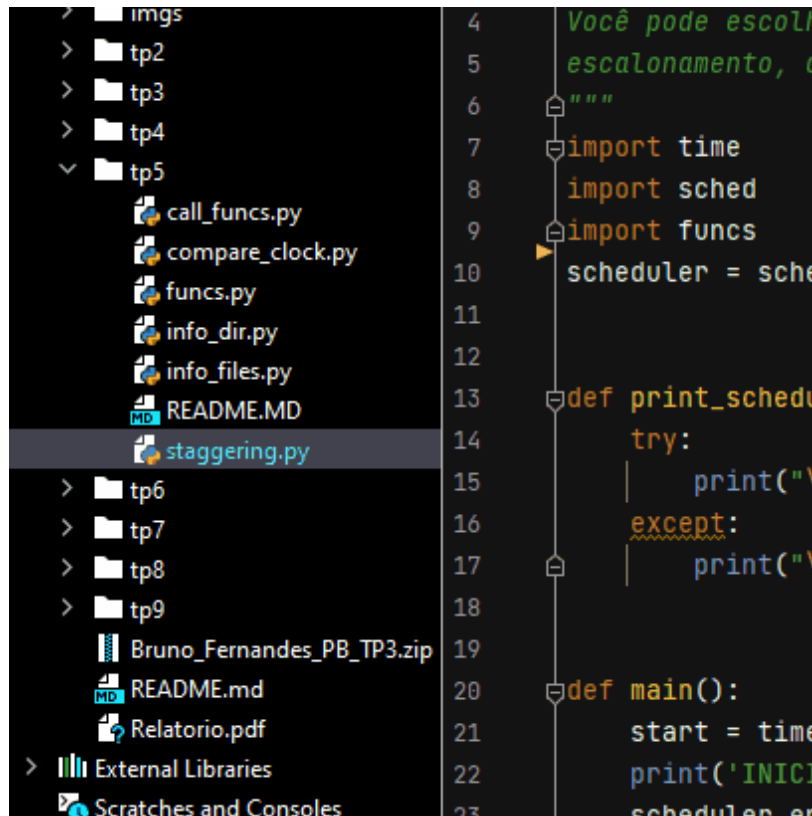
```
staggering.py x funcs.py x
1 """
2 Realizar um escalonamento das chamadas das funções com o módulo 'sched'
3 e medir o tempo total utilizado por cada chamada com o módulo 'time'.
4 Você pode escolher com quais funções do seu projeto realizar o
5 escalonamento, deixando indicado no relatório.
6 """
7 import time
8 import sched
9 import funcs
10 scheduler = sched.scheduler(time.time, time.sleep)
11
12
13 def print_scheduled(func, name=None):
14     try:
15         print("\nInfo: ", func(name), " ")
16     except:
17         print("\nInfo: ", func())
18
19
20 def main():
21     start = time.time()
22     print('INICIO:', time.ctime())
23     scheduler.enter(2, 1, print_scheduled, (funcs.percentual_cpu,))
24     scheduler.enter(3, 1, print_scheduled, (funcs.percentual_memoria,))
25     scheduler.enter(4, 1, print_scheduled, (funcs.informacao_ip,))
26     scheduler.enter(5, 1, print_scheduled, (funcs.info_bateria,))
27     print('CHAMADAS ESCALONADAS:')
28
29     scheduler.run()
30
31     finish = time.time()
32
33     print("\nFIM: " + time.ctime())
34
35     total_secs = abs(finish - start)
36     print(f"Tempo total: {total_secs:.0f} segundos")
37
38
39 main()
```

As funções chamadas pelo arquivo estão no funcs.py;



```
funcs.py ×
1  import psutil
2
3  def percentual_cpu():
4      """
5          This func uses the psutil lib to get and
6          return the cpu used percent.
7          :return: float percentual_usado
8          """
9      percentual_usado = psutil.cpu_percent(interval=0)
10     return "CPU: " + str(percentual_usado) + "%"
11
12
13     def percentual_memoria():
14         """
15             This func use the psutil lib to get the memory
16             used percent.
17             :return: float percentual_usado
18             """
19
20         mem = psutil.virtual_memory().percent
21         return "RAM: " + str(mem) + "%"
22
23
24     def info_battery():
25         battery = psutil.sensors_battery()
26         return "Bateria: " + str(battery.percent) + "%"
27
28
29     def informacao_ip():
30         """
31             This func return the IP computer's address.
32             :return ip:
33             """
34
35         dic_interface = psutil.net_if_addrs()
36         ip = dic_interface["Ethernet"][1].address
37         return "IP: " + str(ip)
```

Ao final retornamos o tempo total da execução do programa:



```
4  Você pode escolh
5  escalonamento, d
6  """
7  import time
8  import sched
9  import funcs
10 scheduler = sche
11
12
13 def print_schedu
14     try:
15         print("V
16     except:
17         print("V
18
19
20 def main():
21     start = time
22     print('INICI
23     scheduler = er
```

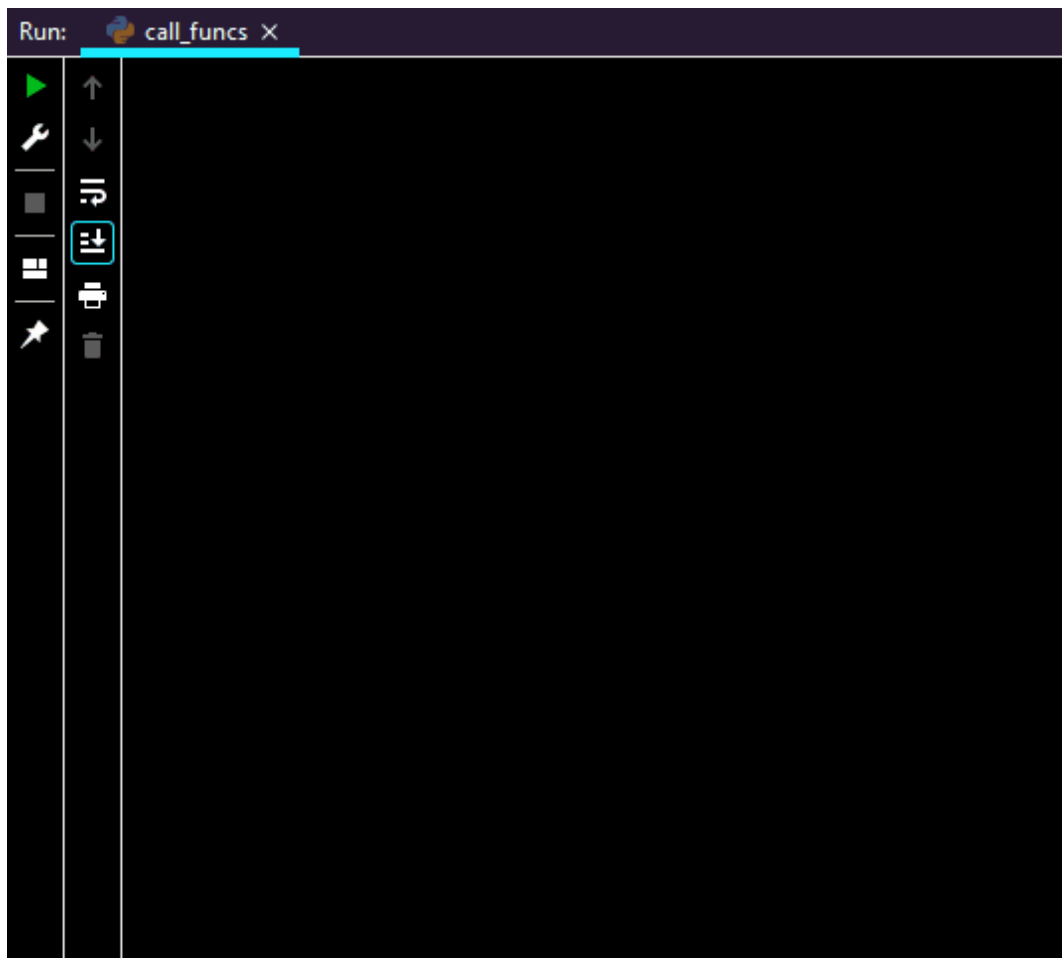
Como podemos observar, temos os tempos inicial e final da execução, com o tempo total calculado.

Em seguida utilizamos esse módulo para chamar demais funções, anteriores:

```
call_funcs.py x
1  """
2  Utilizar o módulo 'sched' para chamar as funções criadas
3  no TP4 que retornam as informações sobre diretórios e arquivos.
4  """
5  from info_dir import dir_info
6  from info_files import file_info
7  import sched
8  import time
9
10 scheduler = sched.scheduler(time.time, time.sleep)
11
12 dir = str(input("Entre com o caminho do diretório (Ex: ../../): "))
13 file = str(input("Entre com o nome do arquivo (Ex: call_funcs.py): "))
14
15
16 def print_scheduled(func, name):
17     print("\nInformações sobre o arquivo/diretório escolhidos:\n", func(name))
18
19
20 print('INICIO:', time.ctime())
21 scheduler.enter(2, 1, print_scheduled, (dir_info, dir,))
22 scheduler.enter(3, 1, print_scheduled, (file_info, file,))
23 print('CHAMADAS ESCALONADAS:')
24
25 scheduler.run()
```

As funções utilizadas chamadas estão nos arquivos info\_files.py e info\_dir.py, já apresentadas no TP4.

Execução do código:



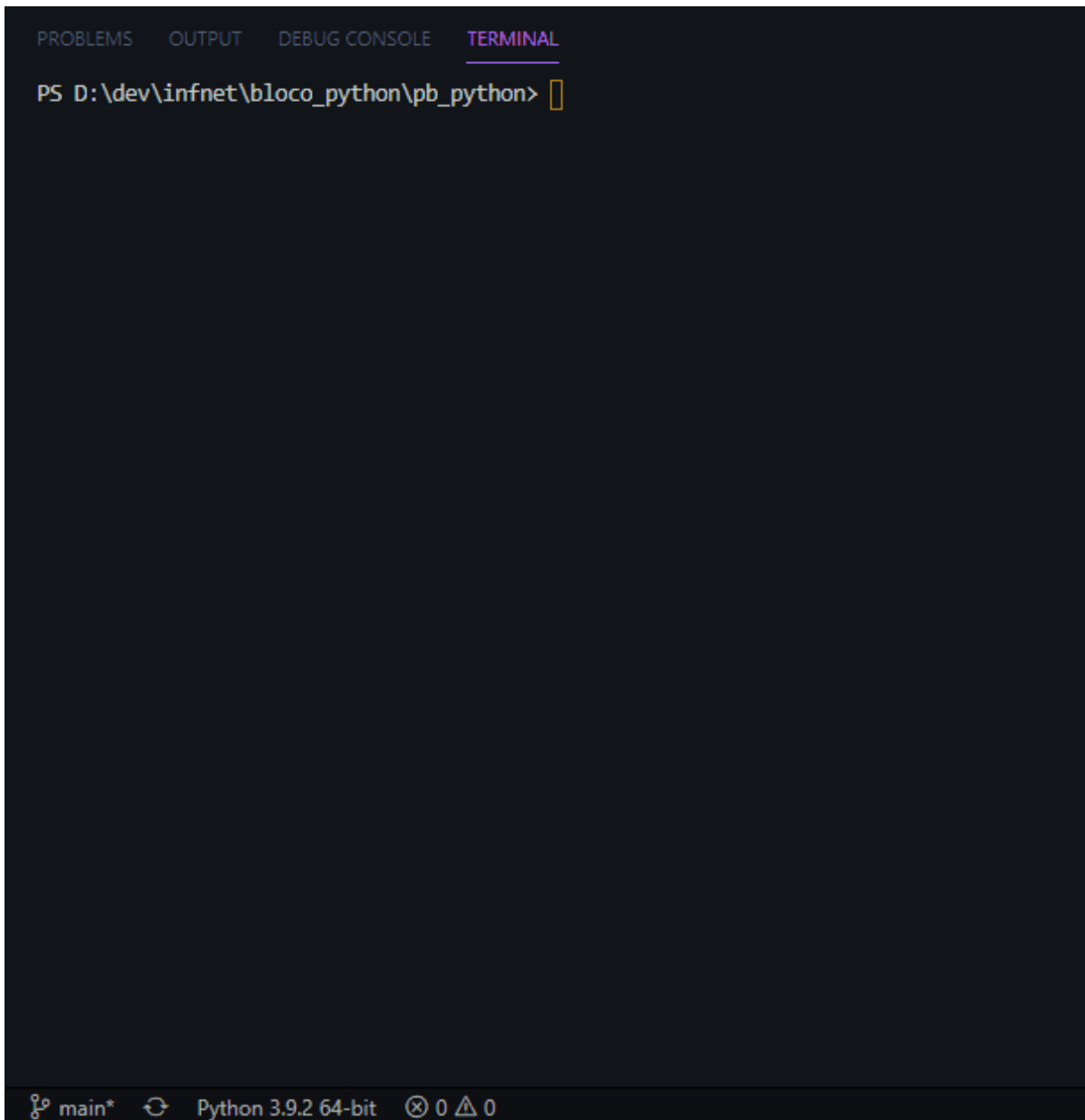
A seguir tivemos que fazer um programa que retornasse de forma escalonada os clocks da máquina em tempos diferentes, segue o código:

```

compare_clock.py X
tp5 > compare_clock.py
1  """
2  Dentro do escalonamento realizado na questão anterior, realizar
3  uma comparação dos tempos obtidos com a quantidade total de
4  clocks utilizados pela CPU para a realização dessas mesmas chamadas.
5
6  - Indique a diferença de tempo real e tempo do clock do computador.
7  - Indique o que acontece com essa diferença quando insere um tempo
8    de espera, como por exemplo utilizando o 'time.sleep' dentro de alguma chamada
9  """
10
11 import psutil
12 import time
13 import sched
14
15 scheduler = sched.scheduler(time.time, time.sleep)
16
17 print("Frequência max da CPU: " + str(psutil.cpu_freq().max) + "GHz.")
18
19
20 def print_scheduled(func, name=None):
21     try:
22         print("\nInfo: ", func(), " ")
23     except:
24         print("\nInfo: ", func())
25
26
27 def cpu_current_freq():
28     return psutil.cpu_freq().current
29
30
31 def main():
32     start = time.time()
33     print('INICIO:', time.ctime())
34     scheduler.enter(2, 1, print_scheduled, (cpu_current_freq,))
35     scheduler.enter(3, 1, print_scheduled, (cpu_current_freq,))
36     scheduler.enter(4, 1, print_scheduled, (cpu_current_freq,))
37     scheduler.enter(5, 1, print_scheduled, (cpu_current_freq,))
38     scheduler.enter(6, 1, print_scheduled, (cpu_current_freq,))
39     scheduler.enter(7, 1, print_scheduled, (cpu_current_freq,))
40     scheduler.enter(8, 1, print_scheduled, (cpu_current_freq,))
41     scheduler.enter(9, 1, print_scheduled, (cpu_current_freq,))
42     print('CHAMADAS ESCALONADAS:')
43
44     scheduler.run()
45
46     finish = time.time()
47
48     print("\nFIM: " + time.ctime())
49
50     total_secs = abs(finish - start)
51     print(f"Tempo total: {total_secs:.0f} segundos")
52
53
54 main()

```

Segue com o código acima rodando:



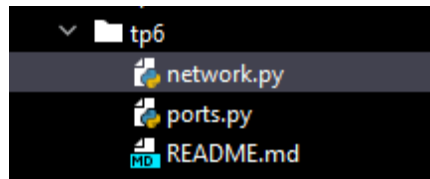
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
PS D:\dev\infnet\bloco_python\pb_python> 
```

Nota: Aparentemente o trecho de código (`psutil.cpu_freq().current`) que usa o módulo do `psutil`, para retornar a frequência atual, está retornando de forma estática.

Entrando no TP6, a estrutura do código foi bem enxuta, no geral tivemos que codificar o retorno de informações sobre a rede, trazendo informações sobre hosts

em um determinado IP, máquinas pertencentes a uma SUB-REDE e informações sobre as portas que estão rodando naquele endereço, utilizamos aqui os módulos os, subprocess, platform e o nmap pra fazer a visualização das portas;

Estrutura:

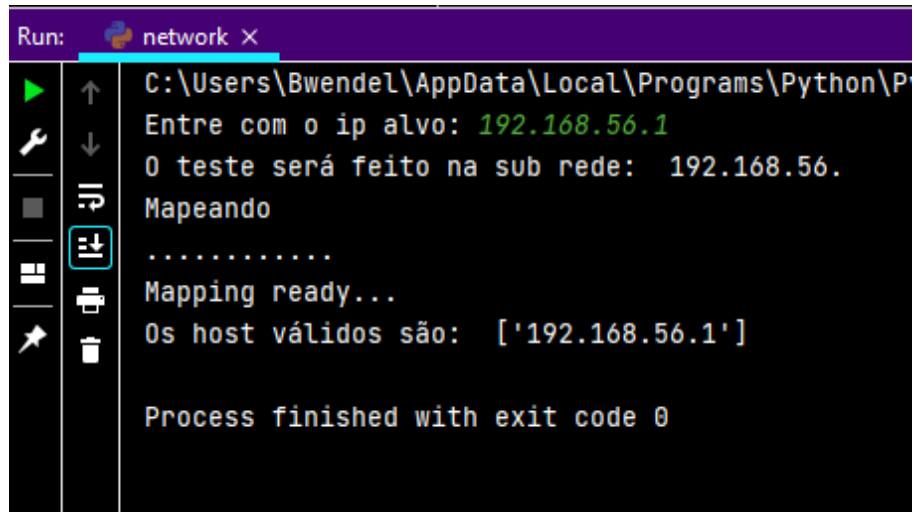


Código presente em network.py, com objetivo de apresentar informações sobre as máquinas em uma sub-rede:

```
network.py X
tp6 > network.py > ...
1  """
2  Criar uma ou mais funções que retornem ou apresentem informações
3  sobre as máquinas pertencentes à sub-rede do IP específico.
4  """
5  import os
6  import subprocess
7  import platform
8
9  def ping_code(hostname):
10     """
11     Uses the ping of the OS.
12     ('-c 5'): Says that Linux that have to send 5 packates.
13     ('-W 3'): Says that the Linux have to wait 3
14     miliseconds for an answer. that func return the ping response
15     """
16     plataforma = platform.system()
17     args = []
18     if plataforma == "Windows":
19         args = ["ping", "-n", "1", "-l", "1", "-w", "100", hostname]
20     else:
21         args = ['ping', '-c', '1', '-W', '1', hostname]
22
23     ret_cod = subprocess.call(args, stdout=open(os.devnull, 'w'), stderr=open(os.devnull, 'w'))
24     return ret_cod
25
26
27 def verify_hosts(base_ip):
28     """
29     verify all hosts with the base ip between 1 and 255 returns a
30     list with all hosts that had 0 responses(active).
31     """
32     print("Mapeando\r")
33     host_validos = []
34     return_codes = dict()
35     for i in range(1, 255):
36
37         return_codes[base_ip + '{0}'.format(i)] = ping_code(base_ip + '{0}'.format(i))
38         if i % 20 == 0:
39             print(".", end="")
40         if return_codes[base_ip + '{0}'.format(i)] == 0:
41             host_validos.append(base_ip + '{0}'.format(i))
42     print("\nMapping ready...")
43
44     return host_validos
45
46
47 # calls
48 ip_string = input("Entre com o ip alvo: ")
49 ip_lista = ip_string.split('.')
50 base_ip = ".".join(ip_lista[0:3]) + '.'
51 print("O teste será feito na sub rede: ", base_ip)
52 print("Os host válidos são: ", verify_hosts(base_ip))
```

Segue a execução do código:





```
Run: network X
C:\Users\Bwendel\AppData\Local\Programs\Python\Python39\python.exe
Entre com o ip alvo: 192.168.56.1
O teste será feito na sub rede: 192.168.56.
Mapeando
.....
Mapping ready...
Os host válidos são: ['192.168.56.1']

Process finished with exit code 0
```

Nota: nesse caso o teste foi feito propositalmente em um IP que teria apenas um host, pela questão do tempo de execução desse código ser grande.

O código referente as portas é o seguinte:

```

ports.py X
tp6 > ports.py > ...
1  """
2  Criar uma ou mais funções que retornem ou apresentem
3  informações sobre as portas dos diferentes IPs obtidos nessa sub rede.
4  """
5  import os
6  import subprocess
7  import platform
8  import nmap
9
10 nmScan = nmap.PortScanner()
11
12
13 def scan_host(host):
14
15     nmScan.scan(host)
16     print(nmScan[host].hostname())
17     for proto in nmScan[host].all_protocols():
18         print('-----')
19         print('Protocolo : %s' % proto)
20
21         lport = nmScan[host][proto].keys()
22         #lport.sort()
23         for port in lport:
24             print ('Porta: %s\t Estado: %s' % (port, nmScan[host][proto][port]['state']))
25
26
27 def ping_code(hostname):
28     """
29     Uses the ping of the OS.
30     ('-c 5'): Says that linux that have to send 5 packates.
31     ('-W 3'): Says that the linux have to wait 3
32     miliseconds for an answer. that func return the ping response
33     """
34
35     plataforma = platform.system()
36     args = []
37     if plataforma == "Windows":
38         args = ["ping", "-n", "1", "-l", "1", "-w", "100", hostname]
39
40     else:
41         args = ['ping', '-c', '1', '-W', '1', hostname]
42
43     ret_cod = subprocess.call(args, stdout=open(os.devnull, 'w'), stderr=open(os.devnull, 'w'))
44     return ret_cod
45

```

```

47 def verify_hosts(base_ip):
48     """
49     verify all hosts with the base ip between 1 and 255 returns a
50     list with all hosts that had 0 responses(active).
51     """
52     print("Mapeando\r")
53     host_validos = []
54     return_codes = dict()
55     for i in range(1, 255):
56
57         return_codes[base_ip + '{0}'.format(i)] = ping_code(base_ip + '{0}'.format(i))
58         if i % 20 == 0:
59             print(".", end="")
60         if return_codes[base_ip + '{0}'.format(i)] == 0:
61             host_validos.append(base_ip + '{0}'.format(i))
62     print("\nMapping ready...")
63
64     return host_validos
65
66
67 def main():
68     for item in host_validos:
69         print("Verificando", item)
70         print(scan_host(item))
71         print("\n")
72
73
74
75 # MAIN
76 ip_string = input("Entre com o ip alvo: ")
77 ip_lista = ip_string.split(".")
78 base_ip = ".".join(ip_lista[0:3]) + "."
79 print("O teste será feito na sub-rede: ", base_ip, "\n")
80 host_validos = verify_hosts(base_ip)
81 main()

```

Segue a execução do código:

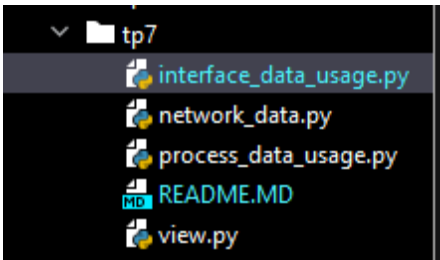
```
Entre com o ip alvo: 192.168.56.1
O teste será feito na sub-rede: 192.168.56.

Mapeando
.....
Mapping ready...
Verificando 192.168.56.1

-----
Protocolo : tcp
Porta: 135      Estado: open
Porta: 139      Estado: open
Porta: 443      Estado: open
Porta: 445      Estado: open
Porta: 902      Estado: open
Porta: 912      Estado: open
None
```

Seguindo para o TP7, usamos o módulo socket para requisitar informações sobre pacotes e bytes transferidos na rede, também utilizamos o módulo OS para trazer informações sobre Gateway e Subnet da máquina.

A estrutura do TP7 foi a seguinte:



Inicialmente temos o código de network\_data.py que retorna informações sobre IP, Gateway e Subnet Mask, segue o código:

```

tp7 > network_data.py > ...
1  """
2  Crie uma ou mais funções que retornem ou apresentem as seguintes
3  informações de redes: IP, gateway, máscara de subrede.
4  """
5  # import psutil
6  import socket
7  import os
8
9
10 def ip():
11     hostname = socket.gethostname()
12     ip_address = socket.gethostbyname(hostname)
13     return ip_address
14
15
16 def default_gateway():
17     if os.name == "posix":
18         dgw = os.system('ip r | grep default | awk {"print $3"}')
19         return dgw
20     if os.name == "nt":
21         dgw = os.system('ipconfig | findstr /i "Gateway"')
22         return dgw
23
24
25 def subnet_mask():
26     if os.name == "posix":
27         sm = os.system('ip r | grep default | awk {"print $3"}')
28         return sm
29     if os.name == "nt":
30         sm = os.system('ipconfig | findstr /i "Subnet"')
31         return sm
32
33
34 #MAIN
35 #ARRUMAR FORMATAÇÃO
36 def main():
37     print("Dados da rede: ")
38     default_gateways = default_gateway()
39     subnet_masks = subnet_mask()
40     print("\nIP:", ip())
41     print("\n" + "-" * 35)
42

```

O código utiliza o socket para pegar a informação sobre o IP da máquina e o os para fazer uma chamada do comando "ipconfig" e iterar sobre ele para pegar informação sobre Gateway e Subnet. Ao rodar o código, temos o seguinte resultado:

```
Dados da rede:
  Default Gateway . . . . . :
  Default Gateway . . . . . : fe80::be2e:48ff:fed3:db97%10
  Default Gateway . . . . . :
  Default Gateway . . . . . :
  Subnet Mask . . . . . : 255.255.255.0
  Subnet Mask . . . . . : 255.255.255.0
  Subnet Mask . . . . . : 255.255.0.0
  Subnet Mask . . . . . : 255.255.0.0

IP: 192.168.0.11

-----

Process finished with exit code 0
```

Em seguida tivemos os códigos `process_data_usage.py` e `interface_data_usage.py`, eles retornam informações sobre o uso de dados por processos e por interfaces, consecutivamente;

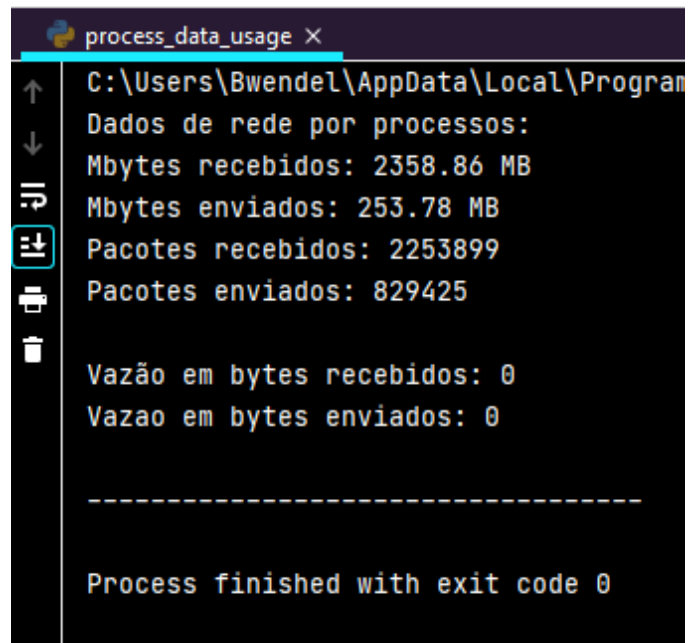
Processos:

```

tp7 > process_data_usage.py > ...
1  """
2  Crie uma ou mais funções que retornem ou apresentem as seguintes
3  informações de redes: Uso de dados de rede por processos.
4  """
5
6  import psutil
7  import time
8
9  net_counters = psutil.net_io_counters()
10
11  def data_io_mbytes():
12      """
13      Return data usage (sent and received)
14      in MegaBytes.
15      """
16      rv = net_counters.bytes_recv / (1024 ** 2)
17      st = net_counters.bytes_sent / (1024 ** 2)
18      return rv, st
19
20
21  def data_io_packets():
22      """
23      Return data usage (sent and received)
24      in packets.
25      """
26      rv = net_counters.packets_recv
27      st = net_counters.packets_sent
28      return rv, st
29
30
31  #MAIN
32  def main():
33      print("Dados de rede por processos: ")
34      bytes_rv = net_counters.bytes_recv
35      bytes_st = net_counters.bytes_sent
36      time.sleep(1)
37      vazao_r = net_counters.bytes_recv - bytes_rv
38      vazao_s = net_counters.bytes_sent - bytes_st
39
40      print("Mbytes recebidos:", round(data_io_mbytes()[0], 2), "MB")
41      print("Mbytes enviados:", round(data_io_mbytes()[1], 2), "MB")
42      print("Pacotes recebidos:", data_io_packets()[0])
43      print("Pacotes enviados:", data_io_packets()[1], "\n")
44      print("Vazão em bytes recebidos:", vazao_r)
45      print("Vazao em bytes enviados:", vazao_s)
46      print("\n" + "-" * 35)
47

```

Rodando:



```
process_data_usage X
C:\Users\Bwendel\AppData\Local\Program
Dados de rede por processos:
Mbytes recebidos: 2358.86 MB
Mbytes enviados: 253.78 MB
Pacotes recebidos: 2253899
Pacotes enviados: 829425

Vazão em bytes recebidos: 0
Vazao em bytes enviados: 0

-----

Process finished with exit code 0
```

Interfaces:

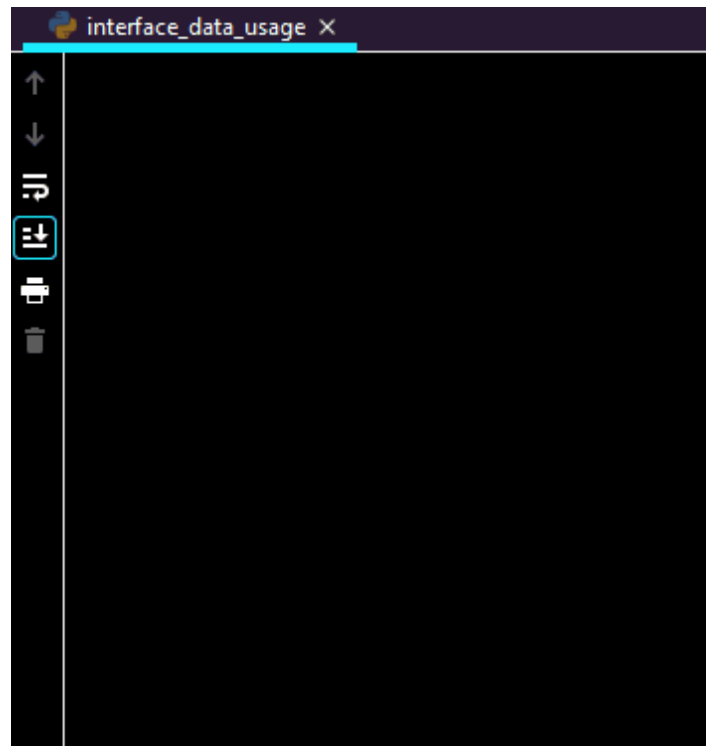


```

tp7 > interface_data_usage.py > main
1  """
2  Crie uma ou mais funções que retornem ou apresentem as seguintes
3  informações de redes: Uso de dados de rede por interface.
4  """
5  import psutil
6  import time
7
8  io_status = psutil.net_io_counters(pernic=True)
9  nomes = []
10 for inf in io_status:
11     nomes.append(str(inf))
12
13
14 def data_io_mbytes(index):
15     """
16     Return data usage (sent and received)
17     in MegaBytes.
18     """
19     rv = io_status[index].bytes_recv / (1024 ** 2)
20     st = io_status[index].bytes_sent / (1024 ** 2)
21     return rv, st
22
23
24 def data_io_packets(index):
25     """
26     Return data usage (sent and received)
27     in packets.
28     """
29     rv = io_status[index].packets_recv
30     st = io_status[index].packets_sent
31     return rv, st
32
33
34 #MAIN
35 def main():
36     print("Dados de rede por interface: ")
37
38     for interface in nomes:
39         bytes_rv = io_status[interface].bytes_recv
40         bytes_st = io_status[interface].bytes_sent
41         time.sleep(1)
42         vazao_r = io_status[interface].bytes_recv - bytes_rv
43         vazao_s = io_status[interface].bytes_sent - bytes_st
44
45         print(interface + ":")
46         print("Mbytes recebidos:", round(data_io_mbytes(interface)[0], 2), "MB")
47         print("Mbytes enviados:", round(data_io_mbytes(interface)[1], 2), "MB")
48         print("Pacotes recebidos:", data_io_packets(interface)[0])
49         print("Pacotes enviados:", data_io_packets(interface)[1])
50         print("Vazão em bytes recebidos:", vazao_r)
51         print("Vazao em bytes enviados:", vazao_s)
52         print("\n" + "-" * 25)
53     print("\n" + "-" * 35)
54
55

```

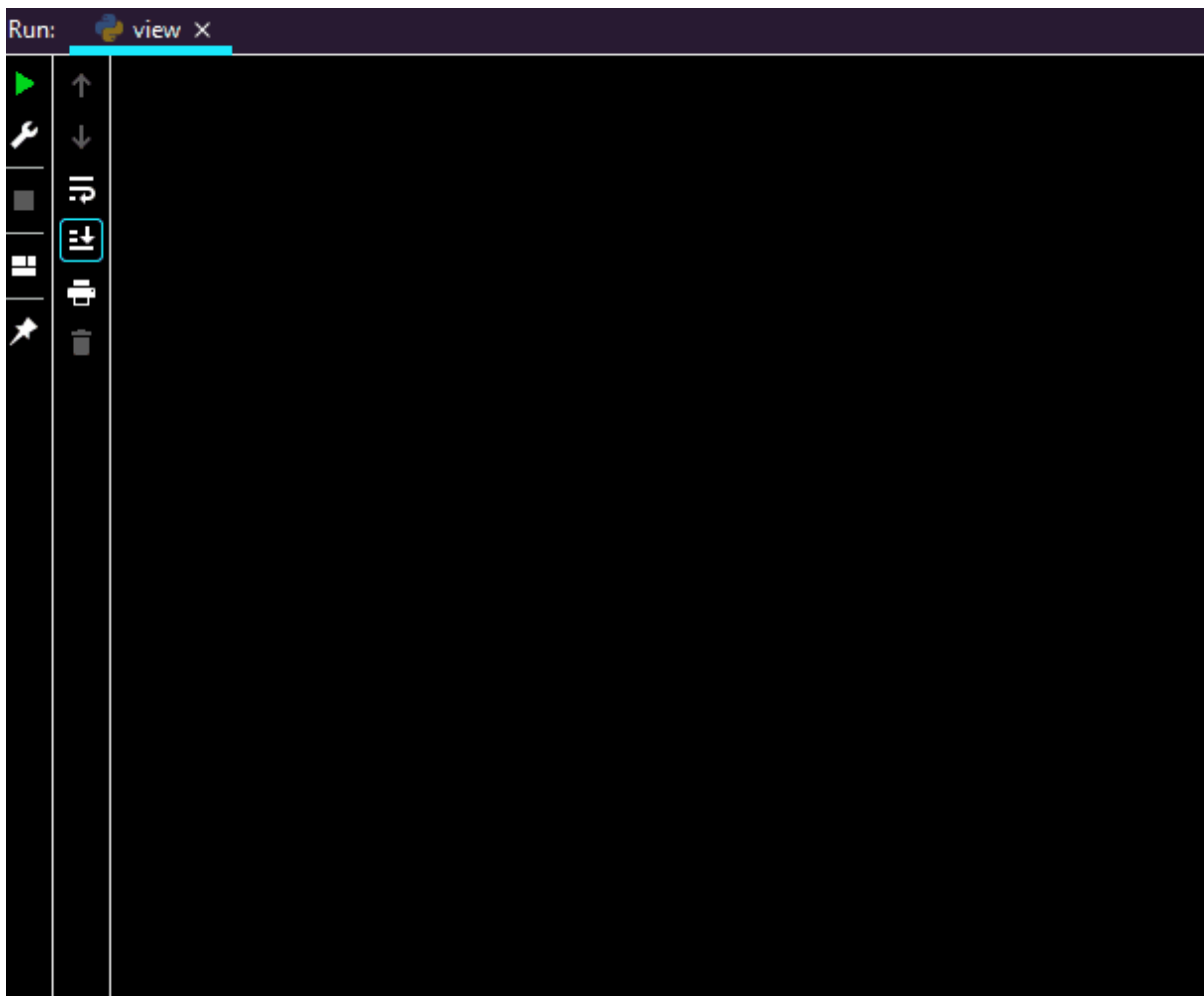
Rodando:



Ao final, como pedido, fizemos um código (view.py) que retorna todas as informações das funções anteriores de maneira conjunta:

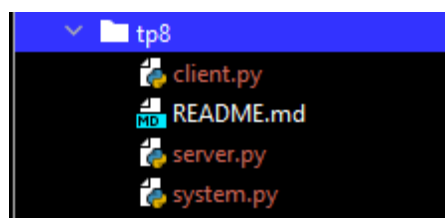
```
view.py X
1  """
2      Use as funções em seu programa para mostrar o resultado.
3      O resultado pode ser em texto formatado impresso na tela
4      ou gráfico, usando o Pygame.
5  """
6
7  import network_data as nd
8  import interface_data_usage as idu
9  import process_data_usage as pdu
10
11
12  def call_funcs():
13      nd.main()
14      idu.main()
15      pdu.main()
16
```

Rodando:



A seguir entramos no TP8, onde utilizamos, principalmente, os módulos socket e pickle para levantar uma aplicação servidor e uma cliente, o pickle faz a transição dos dados, enxergados pelo python, para bits ao subir os dados para o servidor e no momento que o cliente recebe esses dados, ele retorna esses bits em informações que o python reconhece.

Estrutura:



Inicialmente temos a aplicação servidor:

```
server.py X
tp8 > server.py > ...
1  import socket
2  import psutil
3  import pickle
4  from system import System_Info as _sys
5
6  # SERVER
7  localIP = socket.gethostname()
8  localPort = 9991
9  bufferSize = 1024
10
11 msgFromServer = "Olá cliente UDP"
12 bytesToSend = str.encode(msgFromServer)
13
14 disk_info = psutil.disk_partitions(all=False)
15 normal = _sys().memory()[0]
16 swap = _sys().memory()[1]
17 cpu = _sys().cpu_info()
18 message = dict()
19 message['cpu'] = cpu
20 message['memory'] = {'normal':normal}
21 message['disk'] = disk_info
22 bytes_to_send = pickle.dumps(message)
23
24 # Create a datagram socket
25 UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
26
27 # Bind to address and ip
28 UDPServerSocket.bind((localIP, localPort))
29
30 print("UDP server up and listening")
31
32 # Listen for incoming datagrams
33 while(True):
34     bytesAddressPair = UDPServerSocket.recvfrom(bufferSize)
35     message = bytesAddressPair[0].decode('ascii')
36     address = bytesAddressPair[1]
37
38     clientMsg = "Message from Client:{}".format(message)
39     clientIP = "Client IP Address:{}".format(address)
40
41     print(clientMsg)
42     print(clientIP)
43
44     # Sending a reply to client
45     UDPServerSocket.sendto(bytes_to_send, address)
```

Essa aplicação importa a função de informações do sistema do arquivo system.py

```
system.py
tp8 > system.py > System_Info > memory
1  import os
2  import cpuinfo
3  import psutil
4  import subprocess
5  import time
6  from psutil._common import bytes2human
7
8  class System_Info():
9      """ This class retrieves some types of system information. """
10     def __init__(self):
11         """ Constructor. """
12         self.cpu_info = cpuinfo.get_cpu_info()
13         self.cpu_count_phisycal = psutil.cpu_count(logical=False)
14         self.cpu_count_logical = psutil.cpu_count(logical=True)
15         self.cpu_percent = psutil.cpu_percent(interval=None, percpu=True)
16         self.disk_usage = psutil.disk_usage('/')
17         self.network = psutil.net_if_addrs()
18
19     def memory(self):
20         """ This function returns a dict of memory informations. """
21
22         def format_info(info):
23             dict_info = dict()
24             for name in info._fields:
25                 value = getattr(info, name)
26                 if name != 'percent':
27                     value = bytes2human(value)
28                 dict_info[name.capitalize()] = value
29             return dict_info
30
31         memory = format_info(psutil.virtual_memory())
32         swap = format_info(psutil.swap_memory())
33         return memory, swap
34
35     def _disk_usage(self):
36         """ This function returns a dict of disk partitions and your usages. """
37         usage_dict = dict()
38         for part in psutil.disk_partitions(all=False):
39             if os.name == 'nt':
40                 if 'cdrom' in part.opts or part.fstype == '':
41                     continue
42             usage = psutil.disk_usage(part.mountpoint)
43             usage_dict[part.device] = [bytes2human(usage.total),
44                                       bytes2human(usage.used),
45                                       bytes2human(usage.free),
46                                       int(usage.percent),
47                                       part.fstype,
48                                       part.mountpoint]
49         return usage_dict
```

Iniciamos o servidor, logo em seguida a aplicação cliente entra em ação, segue o código do cliente (client.py):

```
client.py X
tp8 > client.py > ...
1 import os
2 import time
3 import socket
4 import psutil
5 import pickle
6 from psutil._common import bytes2human
7 # CLIENT
8 msgFromClient = " Hello UDP Server"
9 bytesToSend = msgFromClient.encode('ascii')
10 serverAddressPort = (socket.gethostname(), 9991)
11 bufferSize = 1024
12 # Create a UDP socket at client side
13 UDPClientSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
14 # Send to server using created UDP socket
15 UDPClientSocket.sendto(bytesToSend, serverAddressPort)
16 try:
17     msgFromServer = UDPClientSocket.recvfrom(bufferSize)
18     _msg = pickle.loads(msgFromServer[0])
19     print(_msg)
20     for k, v in _msg.items():
21         if k == "cpu":
22             print('===' * 25, 'Informações da CPU'.center(75), '===' * 25, sep='\n')
23             for info_type, info in v.items():
24                 if info_type != 'cpus':
25                     print('{}: {}'.format(info_type, info))
26                 else:
27                     pass
28             print('---' * 25)
29             time.sleep(3)
30         elif k == "disk":
31             print('===' * 25, 'Informações do disco'.center(75), '===' * 25, sep='\n')
32             templ = "%-17s %8s %8s %8s %5s%% %9s %s"
33             print(templ % ("Device", "Total", "Used", "Free", "Use ", "Type",
34                             "Mount"))
35             for part in v:
36                 if os.name == 'nt':
37                     if 'cdrom' in part.opts or part.fstype == '':
38                         continue
39                 usage = psutil.disk_usage(part.mountpoint)
40                 print(templ % (
41                     part.device,
42                     bytes2human(usage.total),
43                     bytes2human(usage.used),
44                     bytes2human(usage.free),
45                     int(usage.percent),
46                     part.fstype,
47                     part.mountpoint))
48             print('---' * 25)
49             time.sleep(3)
50         elif k == "memory":
51             print('===' * 25, 'Informações de memória'.center(75), '===' * 25, sep='\n')
52             for mem, values in v.items():
53                 for info_type, _info in values.items():
54                     print('{}: {}'.format(info_type, _info))
55             print('---' * 25)
56
57 except Exception as erro:
58     print(str(erro))
```



Ao levantar o servidor e iniciar a aplicação cliente temos as seguintes informações retornadas:

```
=====
                        Informações da CPU
=====
Name: : Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz
Architecture: : X86_64
Bits: : 64
Frequency: : 2.4 Ghz
Physical Cores: : 4
Logical Cores: : 8
-----
=====
                        Informações de memória
=====
Total: 5.9G
Available: 594.0M
Percent: 90.1
Used: 5.3G
Free: 594.0M
-----
=====
                        Informações do disco
=====
Device          Total      Used      Free  Use %    Type  Mount
C:\             476.3G    153.9G    322.4G   32%     NTFS  C:\
D:\             931.5G    273.7G    657.8G   29%     NTFS  D:\
-----

Process finished with exit code 0
```

Informações referentes a máquina que gerou todo o código.

Ao final publicamos a apresentação e a aplicação toda no moodle da INFNET.