

**INSTITUTO INFNET**  
**ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO**



Teste de Performance 3

**Projeto de Bloco**

**Marcelo da Silva Oliveira**

**Prof.: Alcione Dolavale**

Rio de Janeiro, 2021

# Relatório

Um aplicativo simples de apresentação gráfica do monitoramento e análise do computador. Ele foi implementado em Python usando módulos como psutil (para capturar dados do sistema computacional) e Pygame (para exibir graficamente os dados).

Três bibliotecas do Python foram utilizadas para o desenvolvimento da aplicação, sendo elas *psutil*, *pygame* e *platform*.

## Psutil

É uma biblioteca de plataforma cruzada para recuperar informações sobre os processos em execução e a utilização do sistema (CPU, memória, discos, rede, sensores) em Python. É útil principalmente para monitoramento de sistema, criação de perfil, limitação de recursos de processo e gerenciamento de processos em execução.

## Pygame

Pygame é um conjunto de módulos Python projetados para escrever videogames. Isso permite que se crie jogos completos e programas multimídia na linguagem python. Foi utilizada nesta aplicação para exibição das informações recuperadas do sistema.

## Platform

Biblioteca utilizada para buscar informações sobre a arquitetura do processador, como o nome e modelo. Além disso, também é possível encontrar informações sobre o sistema operacional.

## CPUINFO

Biblioteca utilizada para buscar algumas informações sobre o processador.

Primeiramente foi criado uma interface gráfica para mostrar o uso da memória, da CPU com os dados do processador, do uso de disco e IP do computador.

Eles serão apresentados como uma barra indicativa de quanto está em uso e quanto está livre, exceto o IP do computador e os dados do processador. Estes serão apresentados apenas com textos.

Para criar a interface gráfica, inicialmente foi gerado uma janela e uma estrutura de repetição até que o botão de sair seja pressionado.

```
import pygame, psutil, platform

# Iniciando a janela principal
largura_tela = 800
altura_tela = 690
tela = pygame.display.set_mode((largura_tela, altura_tela))
pygame.display.set_caption("Monitoramento e análise do computador")
pygame.display.init()
vermelho = (255,0,0)
azul = (0,0,255)
verde = (0,255,0)
preto = (0,0,0)
branco = (255,255,255)
s1 = pygame.surface.Surface((largura_tela, altura_tela/3))
s2 = pygame.surface.Surface((largura_tela, altura_tela/3))
s3 = pygame.surface.Surface((largura_tela, altura_tela/3))
```

Criou-se uma janela de largura 800 e altura 690 com o título “Monitoramento e análise do computador”. Além disso, definimos as cores vermelho, azul, preto, branco e verde para utilizarmos nos próximos passos e utilizamos o *pygame surface* para criar sub janelas dentro da janela principal para colocar as análises do computador.

### Barra de Uso da memória

Após a criação da janela foi criado uma função chamada “mostra\_uso\_memoria” para mostrar um gráfico com o uso da memória do computador.

```
pygame.font.init()
font = pygame.font.Font(None, 22)

def mostra_uso_memoria():
    mem = psutil.virtual_memory()
    perc = mem.percent
    larg = largura_tela - 2*20
    tela.fill(preto)
    pygame.draw.rect(s1, azul, (20, 60, larg, 70))
    tela.blit(s1, (0, 0))
    larg = larg*mem.percent/100
    pygame.draw.rect(s1, vermelho, (20, 60, larg, 70))
    tela.blit(s1, (0, 0))
    total = round(mem.total/(1024*1024*1024),2)
    texto_barra = "Uso de Memória (Total: " + str(total) + "GB): " + str(perc) + "%"
    text = font.render(texto_barra, 1, branco)
    tela.blit(text, (20, 10))
```

Essa função mostra o valor da memória total, com a porcentagem escrita e um gráfico com retângulos vermelho e azul para descrever a memória utilizada e a memória livre, respectivamente.

Note que as informações da memória foram captadas através da função “psutil.virtual\_memory()”.

A exibição da barra de uso da memória ficou da seguinte forma:



## **Barra de uso detalhado da CPU e informação detalhada da plataforma de processamento**

Após a criação da barra de uso da memória, foi utilizado a mesma estrutura de código para criar a barra de uso detalhado da CPU, detalhando a porcentagem de uso de cada core(núcleo). Além disso, foi adicionado também informações mais detalhadas da CPU como, o nome do processador, arquitetura e a palavra(bits) através do “cpuinfo”. As informações de núcleos físicos e lógicos e a frequência são retiradas do “psutil”.

Os núcleos do processador são divididos em núcleos físicos e lógicos. Os núcleos físicos são hardwares que estão dentro da CPU. Os núcleos lógicos são as habilidades de um núcleo físico fazer duas ou mais atividades simultaneamente.

Os processadores podem possuir arquitetura CISC e RISC. Processador de arquitetura CISC executa uma instruções mais complexas. Com isso, o tempo de processamento será maior afetando, portanto, a capacidade de processamento. Para executar uma só instrução, um processador CISC pode exigir vários ciclos de relógio. Já os processadores de arquitetura RISC executam instruções reduzidas, ou seja, ele quebra a instrução em várias menores e mais simples e todas assumem um tamanho padrão. Cada uma dessas instruções têm as características necessárias para que possa ser executada em apenas um ciclo de relógio.

Além da arquitetura, o “cpuinfo” também indica o tamanho da palavra. A palavra é utilizada para indicar a unidade de transferência e processamento de um computador. As palavras são múltiplos de 1 byte, sendo que os microprocessadores geralmente utilizam 32bits – 4 bytes como tamanho da palavra (já existem projetos e

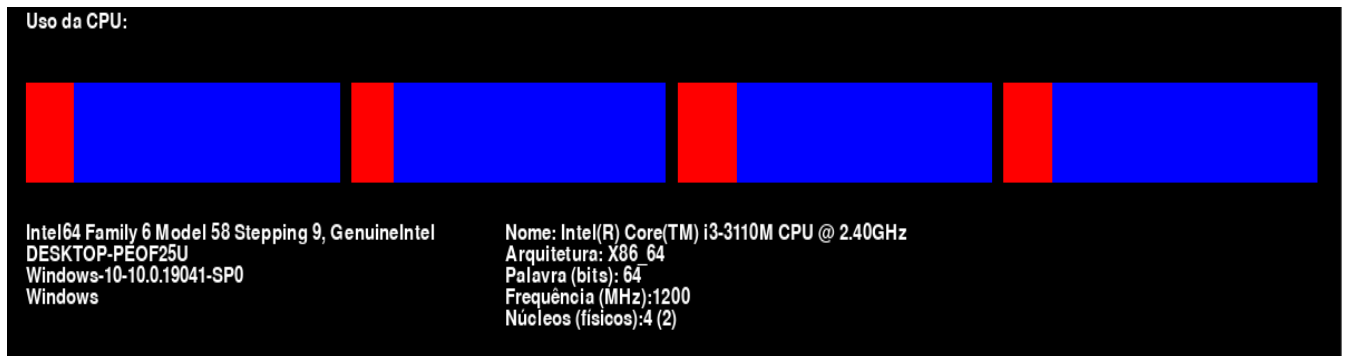
microprocessadores que utilizam palavras de 64 bits, porém estes microprocessadores ainda não se popularizaram).

Para capturar a porcentagem de uso de processamento de cada núcleo do processador, foi usado a mesma função utilizada na TP2(`psutil.cpu_percent`), mas adicionando um parâmetro a ela.

```
def mostra_uso_cpu(s, l_cpu_percent):
    num_cpu = len(l_cpu_percent)
    x = y = 10
    desl = 10
    larg = (s.get_width()-2*y - (num_cpu+1)*desl)/num_cpu
    d = x + desl
    for i in l_cpu_percent:
        pygame.draw.rect(s, azul, (d, y, larg, 70))
        pygame.draw.rect(s, vermelho, (d, y, (i/100)*larg, 70))
        d = d + larg + desl
    tela.blit(s3, (0, 6*altura_tela/10))
    texto_barra_2 = "Uso da CPU:"
    text2 = font.render(texto_barra_2, 1, branco)
    tela.blit(text2, (20, 6*altura_tela/10 - 40))
    texto_proc = str(platform.processor())
    textp = font.render(texto_proc, 1, branco)
    tela.blit(textp, (20, 7*altura_tela/10 + 40))
    texto_node = str(platform.node())
    textn = font.render(texto_node, 1, branco)
    tela.blit(textn, (20, 7*altura_tela/10 + 55))
    texto_plat = str(platform.platform())
    textpl = font.render(texto_plat, 1, branco)
    tela.blit(textpl, (20, 7*altura_tela/10 + 70))
    texto_sistema = str(platform.system())
    texts = font.render(texto_sistema, 1, branco)
    tela.blit(texts, (20, 7*altura_tela/10 + 85))
    texto_nome = "Nome: " + str(info_cpu["brand_raw"])
    textnm = font.render(texto_nome, 1, branco)
    tela.blit(textnm, (450, 7*altura_tela/10 + 40))
    texto_arq = "Arquitetura: " + str(info_cpu["arch"])
    textar = font.render(texto_arq, 1, branco)
    tela.blit(textar, (450, 7*altura_tela/10 + 55))
    texto_palavra = "Palavra (bits): " + str(info_cpu["bits"])
    textpa = font.render(texto_palavra, 1, branco)
    tela.blit(textpa, (450, 7*altura_tela/10 + 70))
    texto_freq = "Frequência (MHz):" + str(round(psutil.cpu_freq().current))
    textfr = font.render(texto_freq, 1, branco)
    tela.blit(textfr, (450, 7*altura_tela/10 + 85))
```

Note que, a estrutura de código da função “mostra\_uso\_CPU” foram colocados informações detalhadas do processamento através da biblioteca “platform”, foram colocados, também, informações detalhadas da CPU através das bibliotecas “cpuinfo”(nome, arquitetura e palavra) e “psutil”(frequência, núcleos lógicos e físicos).

A barra de uso da CPU com a informação detalhada do processador ficou da seguinte forma:



## Barra de uso de Disco

Para criar a barra de uso de Disco, também foi utilizado a mesma estrutura de código do uso da memória e da CPU, porém com as informações de uso do mesmo.

```
def mostra_uso_disco():
    disco = psutil.disk_usage('.')
    larg = largura_tela - 2*20
    pygame.draw.rect(s3, azul, (20, 50, larg, 70))
    tela.blit(s3, (0, 6*altura_tela/10 - 50))
    larg = larg*disco.percent/100
    pygame.draw.rect(s3, vermelho, (20, 50, larg, 70))
    tela.blit(s3, (0, 6*altura_tela/10 - 50))
    total = round(disco.total/(1024*1024*1024), 2)
    texto_barra_3 = "Uso de Disco: (Total: " + str(total) + "GB): " + str(disco.percent) + "%"
    text3 = font.render(texto_barra_3, 1, branco)
    tela.blit(text3, (20, 6*altura_tela/10 - 30))
```

Note que a estrutura de código do uso de Disco é parecido com os códigos do uso de memória e do uso da CPU, porém com as informações específicas do Disco utilizando a função "psutil.disk\_usage('.')".

A barra de uso do disco ficou da seguinte forma:

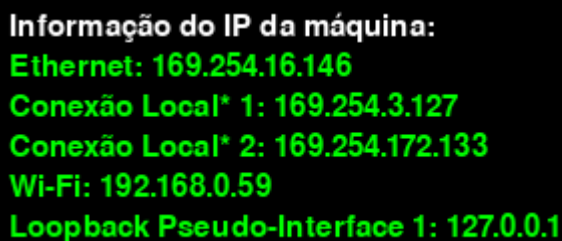


## Informações do IP da máquina:

Para exibir as informações do IP da máquina, foi utilizado uma estrutura parecida com as das informações anteriores, porém não teve a utilização da barra. Além disso, utilizamos a função “psutil.net\_if\_addrs()” para exibir as informações do IP do computador.

```
def IP_maquina():
    dic_interfaces = psutil.net_if_addrs()
    texto_ip = 'Informação do IP da máquina:'
    textip = font.render(texto_ip, 1, branco)
    tela.blit(textip, (20, 7*altura_tela/10 + 20))
    texto_ET = 'Ethernet: ' + str(dic_interfaces['Ethernet'][1].address)
    textet = font.render(texto_ET, 1, verde)
    tela.blit(textet, (20, 7*altura_tela/10 + 40))
    texto_CL1 = 'Conexão Local* 1: ' + str(dic_interfaces['Conexão Local* 1'][1].address)
    textcl1 = font.render(texto_CL1, 1, verde)
    tela.blit(textcl1, (20, 7*altura_tela/10 + 60))
    texto_CL2 = 'Conexão Local* 2: ' + str(dic_interfaces['Conexão Local* 2'][1].address)
    textcl2 = font.render(texto_CL2, 1, verde)
    tela.blit(textcl2, (20, 7*altura_tela/10 + 80))
    texto_WF = 'Wi-Fi: ' + str(dic_interfaces['Wi-Fi'][1].address)
    textwf = font.render(texto_WF, 1, verde)
    tela.blit(textwf, (20, 7*altura_tela/10 + 100))
    texto_PI = 'Loopback Pseudo-Interface 1: ' + str(dic_interfaces['Loopback Pseudo-Interface 1'][0].address)
    textpi = font.render(texto_PI, 1, verde)
    tela.blit(textpi, (20, 7*altura_tela/10 + 120))
```

A exibição das informações do IP da máquina ficou da seguinte forma:



Informação do IP da máquina:  
Ethernet: 169.254.16.146  
Conexão Local\* 1: 169.254.3.127  
Conexão Local\* 2: 169.254.172.133  
Wi-Fi: 192.168.0.59  
Loopback Pseudo-Interface 1: 127.0.0.1

## Considerações finais:

No início tive algumas dificuldades para entender a estrutura de código que foi retirado do roteiro de aprendizagem da etapa 2, porém consegui o que fazia cada linha do código. Com o entendimento, adicionei alguns outros itens como a mudança de cor na exibição do IP pra cor verde. Além disso, também adicionei a porcentagem escrita do uso da memória, da CPU e do Disco facilitando o entendimento após a exibição da aplicação.

## **Referências:**

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253423>

<https://lms.infnet.edu.br/moodle/mod/page/view.php?id=253431>