INSTITUTO INFNET

ESCOLA SUPERIOR DE TECNOLOGIA

GRADUAÇÃO EM ANÁLISE E DESENVOLVIMENTO

PYTHON PARA DADOS

TESTE DE PERFORMANCE – AT

ALUNO: GABRIEL GOMES DE SOUZA

PROFESSOR(A): DÁCIO MOREIRA DE SOUZA

E-MAIL: gabriel.gsouza@al.infnet.edu.br

Sumário

1. Conteúdo

Mini Projeto 01

PS4

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Define o caminho para salvar o arquivo CSV
PATH = 'Python para
Dados\\AT\\Mini-Projeto1\\PlayStation4\\'

def requisicao_url(url):
    response = requests.get(url)
    if response.status_code == 200:
        return BeautifulSoup(response.text, 'html.parser')
    else:
        print("Erro ao fazer a requisição:",
response.status_code)
        return None

def extrair_cabecalhos(tabela):
    return [th.text.strip() for th in
tabela.find_all('tr')[0].find_all('th')][:-1]

def extrair_linhas(tabela):
    linhas = []
    for linha in tabela.find_all('tr')[1:]:
        linha_valor = []
        for td in linha.find_all(['td', 'th'])[:-1]:
            text = td.text.strip()
            if not text:
                text = 'Não definido'
            linha_valor.append(text)
        linhas.append(linha_valor)
    return linhas
```

```
def criar_dataframe(linhas, cabecalhos):
    df = pd.DataFrame(linhas, columns=cabecalhos)
    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
    return df

def salvar_csv(df, caminho, nome_arquivo):
    df.to_csv(f'{caminho}{nome_arquivo}', index=False)

url =
"https://pt.wikipedia.org/wiki/Lista_de_jogos_para_PlayStati
on_4"
soup = requisicao_url(url)
if soup:
    tabela = soup.find('table', {'class': 'wikitable
sortable'})
    cabecalhos = extrair_cabecalhos(tabela)
    linhas = extrair_linhas(tabela)
    df = criar_dataframe(linhas, cabecalhos)
    salvar_csv(df, PATH, 'ps4_jogos.csv')
```

PS5

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Define o caminho para salvar o arquivo CSV
PATH = 'Python para
Dados\\AT\\Mini-Projeto1\\PlayStation5\\'

url =
"https://pt.wikipedia.org/wiki/Lista_de_jogos_para_PlayStati
on_5"
```

```
response = requests.get(url)

if response.status_code == 200:
    soup = BeautifulSoup(response.text, 'html.parser')
    tabela = soup.find('table', {'id': 'softwarelist'})
    cabecalhos = []
    for th in tabela.find('tr').find_all('th'):
        if not th.has_attr('colspan'):
            cabecalhos.append(th.text.strip())
    cabecalhos = cabecalhos[:-2]

    # Extrai as regiões
    regioes = []
    for th in tabela.find_all('tr')[1].find_all('th'):
        regioes.append(th.text.strip())

    for region in regioes:
        cabecalhos.append(region)

    # Extrai as linhas da tabela
    linhas = []
    for linha in tabela.find_all('tr')[2:]:
        linha_valor = []
        for td in linha.find_all(['th', 'td']):
            linha_valor.append(td.text.strip() or 'Não
definido')
        linhas.append(linha_valor[:-2])

    # Converte os dados em um DataFrame do pandas
    df = pd.DataFrame(linhas, columns=cabecalhos)

    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
    df.replace("—", "Não definido", inplace=True)
```

```
    df.to_csv(f'{PATH}ps5_jogos.csv', index=False)

else:
    print("Erro ao fazer a requisição:",
response.status_code)
```

Xbox360

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Define o caminho para salvar o arquivo CSV
PATH = 'Python para Dados\\AT\\Mini-Projeto1\\Xbox360\\'

url =
"https://pt.wikipedia.org/wiki/Lista_de_jogos_para_Xbox_360"
resposta = requests.get(url)

if resposta.status_code == 200:
    soup = BeautifulSoup(resposta.text, 'html.parser')
    tabela = soup.find('table', {'id': 'softwarelist'})

    # Extrai os cabeçalhos e as linhas da tabela
    colunas = [th.text.strip() for th in
tabela.find('tr').find_all('th')[:-3] if not
th.has_attr('colspan')]
    regioes = [th.text.strip() for th in
tabela.find_all('tr')[1].find_all('th')]

    for linha in regioes:
        colunas.append(linha)

    linhas = []

    for linha in tabela.find_all('tr')[1:]:
```

```
        valores_linha = [td.text.strip() or 'Não definido'
for td in linha.find_all('td')[:-3]]
        linhas.append(valores_linha)

    # Converte os dados em um DataFrame do pandas
    df = pd.DataFrame(linhas, columns=colunas)

    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
    df.replace("—", "Não definido", inplace=True)

    # Exporta o DataFrame para um arquivo CSV
    df.to_csv(f'{PATH}xbox360_jogos.csv', index=False)

else:
    print("Erro ao fazer a requisição:",
resposta.status_code)
```

XboxSeries

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

# Define o caminho para salvar o arquivo CSV
PATH = 'Python para Dados\\AT\\Mini-Projeto1\\XboxSeries\\'

url =
"https://pt.wikipedia.org/wiki/Lista_de_jogos_para_Xbox_Seri
es_X_e_Series_S"
resposta = requests.get(url)

if resposta.status_code == 200:
    soup = BeautifulSoup(resposta.text, 'html.parser')
    tabela = soup.find('table', {'id': 'softwarelist'})
```

```
    # Extrai os cabeçalhos e as linhas da tabela
    cabecalhos = [th.text.strip() for th in
tabela.find('tr').find_all('th') if not
th.has_attr('colspan')][:-2]
    regioes = [th.text.strip() for th in
tabela.find_all('tr')[1].find_all('th')]

    for linha in regioes:
        cabecalhos.append(linha)

    linhas = []

    for linha in tabela.find_all('tr')[1:]:
        valores_linha = [td.text.strip() or 'Não definido'
for td in linha.find_all(['th', 'td'])[:-2]]
        linhas.append(valores_linha)

    # Converte os dados em um DataFrame do pandas
    df = pd.DataFrame(linhas, columns=cabecalhos)

    df.dropna(inplace=True)
    df.drop_duplicates(inplace=True)
    df.replace("—", "Não definido", inplace=True)

    df.to_csv(f'{PATH}xbox_series_jogos.csv', index=False)

else:
    print("Erro ao fazer a requisição:",
resposta.status_code)
```

Mini Projeto 02

```
import json
import re
import pandas as pd
```

```
import datetime

# Caminho dos arquivos
csv_file = 'Python para
Dados\\AT\\Mini-Projeto2\\DadosAnalisados\\dadosAT.csv'
json_file = 'Python para
Dados\\AT\\Mini-Projeto2\\DadosAnalisados\\dadosATAntigo.jso
n'
excel_file = 'Python para
Dados\\AT\\Mini-Projeto2\\DadosAnalisados\\dadosATAntigo.xls
x'

def ler_arquivos(csv_path, json_path, excel_path):
    try:
        df_csv = pd.read_csv(csv_path)

        with open(json_path) as f:
            json_data = json.load(f)
        df_json = pd.DataFrame(json_data)

        df_excel = pd.read_excel(excel_path)
        return df_csv, df_json, df_excel
    except Exception as e:
        print(f"Erro ao ler arquivos: {e}")
        return None, None, None

def formatar_data(data):
    try:
        # Verifica se a data está no formato dd/mm/yyyy ou
dd-mm-yyyy
        if '/' in data:
            data = data.replace('/', '-')

        # Verifica se a data está no formato dd-mm-yyyy
        match = re.match(r'(\d{2})-(\d{2})-(\d{4})', data)
```

```
        if match:
            return
f"{match.group(3)}-{match.group(2)}-{match.group(1)}"
        else:
            return data
    except Exception as e:
        print(f"Erro ao formatar data: {e}")
        return data

def validar_email(email):
    # Expressão regular para validar o formato do email
    regex_email =
r'[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+'
    return re.match(regex_email, email)

def limpar_dados(df):
    try:
        df.drop_duplicates(inplace=True)
        df.fillna('', inplace=True)

        if 'data_nascimento' in df.columns:
            # Aplica formatação da coluna de data de
nascimento se necessário
            df['data_nascimento'] =
df['data_nascimento'].apply(lambda x: formatar_data(str(x)))

            df = df.fillna('N/A') # Preenche valores nulos
com 'N/A'

            df['email_valido'] = df['email'].apply(lambda x:
validar_email(str(x))) # Validar formato do email

            df = df[df['email_valido'].notna()] # Remover
registros com emails inválidos
```

```
            # Remover duplicatas baseadas em Data de
Nascimento e Email, mantendo o primeiro com mais campos
preenchidos
            df['num_campos_preenchidos'] = df.apply(lambda
x: x.count(), axis='columns')
            df =
df.sort_values(by=['num_campos_preenchidos'],
ascending=[False])
            df =
df.drop_duplicates(subset=['data_nascimento', 'email'],
keep='first')

            # Remover colunas auxiliares
            df = df.drop(columns=['num_campos_preenchidos',
'email_valido'])

            # Ordenar por ID
            df = df.sort_values(by='id')

            return df
        else:
            print("Coluna 'data_nascimento' não encontrada
no DataFrame:")
            print(df.head())  # Mostra as primeiras linhas
do DataFrame para inspeção
            return None
    except Exception as e:
        print(f"Erro ao limpar dados: {e}")
        return None

def consolidar_dados(csv_df, json_df, excel_df):
    try:
        df_consolidado = pd.concat([csv_df, json_df,
excel_df], ignore_index=True)
```

```
        # Remover duplicatas após concatenar DFs
        df_consolidado.drop('id', axis='columns',
inplace=True)
        df_consolidado.drop_duplicates(inplace=True)

        # Substituir células vazias por "Não definido"
        df_consolidado.replace('', 'Não definido',
inplace=True)

        return df_consolidado
    except Exception as e:
        print(f"Erro ao consolidar DataFrames: {e}")
        return None

def exportar_excel(df, output_path):
    try:
        df.to_excel(output_path, index=False)
        print(f"Dados exportados para {output_path} com
sucesso!")
    except Exception as e:
        print(f"Erro ao exportar dados para Excel: {e}")

def main():
    df_csv, df_json, df_excel = ler_arquivos(csv_file,
json_file, excel_file)

    if df_csv is None or df_json is None or df_excel is
None:
        print("Erro ao ler os arquivos. Verifique os
caminhos e tente novamente.")
        return

    df_csv = limpar_dados(df_csv)
    df_json = limpar_dados(df_json)
    df_excel = limpar_dados(df_excel)
```

```
    if df_csv is None or df_json is None or df_excel is
None:
        print("Erro ao limpar os dados. Verifique os dados e
tente novamente.")
        return

    # Consolidar todos os DataFrames em um único DataFrame
    df_consolidado = consolidar_dados(df_csv, df_json,
df_excel)

    if df_consolidado is None:
        print("Erro ao consolidar os dados. Verifique os
dados e tente novamente.")
        return

    # Analisar o número de duplicatas após a consolidação
    num_duplicatas =
df_consolidado.duplicated(subset=['data_nascimento',
'email']).sum()
    if num_duplicatas > 0:
        print(f"Foram encontradas {num_duplicatas}
duplicatas no DataFrame consolidado.")
        df_consolidado =
df_consolidado.drop_duplicates(subset=['data_nascimento',
'email'], keep='first')
        print(f"Duplicatas removidas. Novo tamanho do
DataFrame: {len(df_consolidado)}")

    exportar_excel(df_consolidado, 'Python para
Dados\\AT\\Mini-Projeto2\\usuarios-consolidados.xlsx')
main()
```

Mini Projeto 03

```
import pandas as pd
from sqlalchemy import create_engine
import collections

EXCEL_PATH = 'Python para
Dados\\AT\\Mini-Projeto2\\usuarios-consolidados.xlsx'
DB_PATH = 'Python para
Dados\\AT\\Mini-Projeto3\\banco_de_dados.sqlite'

def ler_arquivo_excel(caminho):
    try:
        df = pd.read_excel(caminho)
        return df
    except Exception as e:
        print(f"Erro ao ler o arquivo Excel: {e}")
        return None

def jogos_separados(df):
    jogos = set()
    for row in df['jogos_preferidos']:
        jogos.update(row.split('|'))
    return jogos # lista com todos os jogos separados

def jogos_populares(df):
    jogos = set()
    for row in df['jogos_preferidos']:
        jogos.update(row.split('|'))

    contador = collections.Counter()
    for row in df['jogos_preferidos']:
        contador.update(row.split('|'))

    jogos_populares = contador.most_common(5) # Retorna os 5
jogos mais comuns
    return jogos_populares
```

```
def jogos_especificos(df):
    jogos = set()
    for row in df['jogos_preferidos']:
        jogos.update(row.split('|'))

    contador = collections.Counter()
    for row in df['jogos_preferidos']:
        contador.update(row.split('|'))

    jogos_especificos = [jogo for jogo, contagem in
contador.items() if contagem == 1]

    return jogos_especificos # lista de jogos que foram
citados uma única vez

def exportar_sqlite(db_path, jogos_separados,
jogos_populares, jogos_especifico):
    try:
        engine = create_engine(f'sqlite:///{db_path}')
        conn = engine.connect()

        pd.DataFrame(list(jogos_separados),
columns=['Jogo']).to_sql('jogos_separados', engine,
if_exists='replace', index=False)

        pd.DataFrame(list(jogos_populares), columns=['Jogo',
'Quantidade']).to_sql('jogos_populares', engine,
if_exists='replace', index=False)

        pd.DataFrame(list(jogos_especifico),
columns=['Jogo']).to_sql('jogos_especifico', engine,
if_exists='replace', index=False)

        print(f"Dados exportados com sucesso no dados!")
```

```
    except Exception as e:
        print(f"Erro ao exportar os dados para o banco de
dados SQLite: {e}")
    conn.close()

df = ler_arquivo_excel(EXCEL_PATH)

separados = jogos_separados(df)
populares = jogos_populares(df)
especifico = jogos_especificos(df)

exportar_sqlite(DB_PATH, separados, populares, especifico)
```

Mini Projeto 04

```
from sqlalchemy import create_engine
import requests
import pandas as pd
import time
import sqlite3

PATH = 'Python para
Dados\\AT\\Mini-Projeto3\\banco_de_dados.sqlite'
DB_PATH = 'Python para
Dados\\AT\\Mini-Projeto4\\mercado_livre.db'

def conectar_banco_de_dados(db_path, path):
    engine = create_engine(f'sqlite:///{db_path}')
    conn = sqlite3.connect(path)
    cursor = conn.cursor()
    return engine, conn, cursor

def ler_jogos_separados(cursor):
    cursor.execute('SELECT Jogo FROM jogos_separados')
```

```
    jogos_separados = [row[0] for row in cursor.fetchall()]
    return jogos_separados

def API_consulta(jogo):
    try:
        url =
f'https://api.mercadolibre.com/sites/MLB/search?category=MLB
186456&q={jogo.replace(" ", "%20")}'
        response = requests.get(url)
        response.raise_for_status() # Lança uma exceção se a
resposta da API tiver um código de erro
        return response.json()['results']
    except requests.exceptions.RequestException as e:
        print(f"Erro na requisição: {e}")

def processar_dados(dados, df):
    for dado in dados:
        nome = dado['title']
        preco = dado['price']
        permalink = dado['permalink']
        df.loc[len(df)] = [nome, preco, permalink]
        print(nome, "processado.")

def exportar_db(df, engine):
    try:
        df.to_sql('jogos_mercado_livre', engine,
if_exists='replace', index=False)
        print("Dados exportados com sucesso!")
    except Exception as e:
        print(f'Erro ao exportar para o banco de dados:
{e}')

engine, conn, cursor = conectar_banco_de_dados(DB_PATH,
PATH)
jogos_separados = ler_jogos_separados(cursor)
```

```
# Cria um DF para por os dados
df = pd.DataFrame(columns=['nome', 'preco', 'permalink'])

for jogo in jogos_separados:
    try:
        dados = API_consulta(jogo)
        processar_dados(dados, df)
        time.sleep(1.5) # evitar problema com API
    except requests.exceptions.RequestException as e:
        print(f'Erro ao consultar a API: {e}')
    except Exception as e:
        print(f'Erro ao processar a resposta: {e}')

exportar_db(df, engine)
conn.close()
```

Projeto Integrador

```
from sqlalchemy import create_engine, Column, Integer, String, ForeignKey, Float

from sqlalchemy.orm import declarative_base, sessionmaker

import pandas as pd


PATH = 'Python para Dados\\AT\\ProjetoIntegrador\\'


engine = create_engine(f'sqlite:///{PATH}/banco_jogos.db')


# Criar base de dados
```

```
    Base = declarative_base()


    class Consoles(Base):
        __tablename__ = 'consoles'
        id = Column(Integer, primary_key=True)
        console = Column(String)


    class Usuarios(Base):
        __tablename__ = 'usuarios'
        id = Column(Integer, primary_key=True)
        nome = Column(String)
        data_nascimento = Column(String)
        email = Column(String)
        cidade = Column(String)


    class UsuarioConsole(Base):
        __tablename__ = 'usuario_console'
        id = Column(Integer, primary_key=True)
        usuario = Column(Integer, ForeignKey('usuarios.id'))
                            idConsole      =      Column(Integer,
ForeignKey('consoles.id'))


    class JogosMercadoLivre(Base):
```

```
        __tablename__ = 'jogos_mercado_livre'
        id = Column(Integer, primary_key=True)
        nome = Column(String)
        preco = Column(Float)
        permalink = Column(String)

    class JogosPreferidos(Base):
        __tablename__ = 'jogos_preferidos'
        id = Column(Integer, primary_key=True)
        nome = Column(String)

    # Criar tabelas:
    Base.metadata.create_all(engine)
    Session = sessionmaker(bind=engine)
    session = Session()

    # Função para importar dados de CSV e inserir no banco de
dados
    def importar_dados(csv_path, dados):
        data = pd.read_csv(csv_path)
        try:
                                data.to_sql(dados,   con=engine,
if_exists='append', index=False)
                print(f'Dados importados para a tabela {dados}
```

```
com sucesso.')

        except Exception as e:

                print(f'Erro ao importar dados para a tabela
{dados}: {e}')


    # Função para exportar dados do banco de dados para CSV

    def exportar_dados(table, export_path):

        try:

            df = pd.read_sql_table(table, con=engine)

            df.to_csv(export_path, index=False)

              print(f'Dados exportados da tabela {table} para
{export_path} com sucesso.')

        except Exception as e:

            print(f'Erro ao exportar dados da tabela {table}:
{e}')


    PATH = 'Python para Dados\\AT\\ProjetoIntegrador\\'

    PATH_1              =              'Python              para
Dados\\AT\\Mini-Projeto1\\Arquivos_Exportados\\'

    PATH_2              =              'Python              para
Dados\\AT\\Mini-Projeto2\\usuarios-consolidados.xlsx'


    # Converter Excel para CSV e importar dados

    usuarios_excel = pd.read_excel(PATH_2)

    csv_path                                                =
```

```
f'{PATH}Arquivos_Exportados\\usuarios_consolidados.csv'
    usuarios_excel.to_csv(csv_path, index=False)
    importar_dados(csv_path, 'usuarios')

    PATH_3 = 'Python para Dados\\AT\\Mini-Projeto3\\banco_de_dados.sqlite' # consultar e pegar valores da coluna "Jogo" da tabela "jogos_separados"

    # Importar dados do outro banco SQLite
    engine_2 = create_engine(f'sqlite:///{PATH_3}')
    jogos_df = pd.read_sql_table('jogos_separados', con=engine_2, columns=['Jogo'])
    jogos_df.columns = ['nome']
    jogos_df.to_sql('jogos_preferidos', con=engine, if_exists='append', index=False)
    print('Dados importados da tabela jogos_separados para jogos_preferidos com sucesso.')

    PATH_4 = 'Python para Dados\\AT\\Mini-Projeto4\\mercado_livre.db' # consultar e pegar a coluna "nome" e "preço" da tabela "jogos_mercado_livre"

    # Importar dados do outro banco
    engine_3 = create_engine(f'sqlite:///{PATH_4}')
    jogos_mercado_livre_df =
```

```
pd.read_sql_table('jogos_mercado_livre',        con=engine_3,
columns=['nome', 'preco'])

    jogos_mercado_livre_df.to_sql('jogos_mercado_livre',
con=engine, if_exists='replace', index=False)

    print('Dados  importados  da  tabela  jogos_mercado_livre
para jogos_mercado_livre com sucesso.')



    # Exportar dados para arquivos CSV

    export_paths_and_tables = [


(f'{PATH}Arquivos_Exportados\\exported_jogos_mercado_livre.c
sv', 'jogos_mercado_livre'),


(f'{PATH}Arquivos_Exportados\\exported_jogos_preferidos.csv'
, 'jogos_preferidos'),

    ]



    for export_path, table_name in export_paths_and_tables:

        exportar_dados(table_name, export_path)



    # Recomendação dos 5 jogos mais baratos

    def recomendar_jogos_mais_baratos():

        try:

                                        jogos_mais_baratos    =
session.query(JogosMercadoLivre).order_by(JogosMercadoLivre.
preco).limit(5).all()
```

```
            print("Recomendação dos 5 jogos mais baratos são:")
            for jogo in jogos_mais_baratos:
                print(f"Nome: {jogo.nome}, Preço: R${jogo.preco:.2f}, Link: {jogo.permalink}")
        except Exception as e:
            print(f"Erro ao buscar os jogos mais baratos: {e}")


    # Chamando a função de recomendação dos 5 jogos mais baratos
    recomendar_jogos_mais_baratos()
```