[**Integração com APIs**](#_317rn8o38077) **2**

[**Resumo de Métodos — Básicos Python**](#_uyy90vj5du4h) **2**

[Arquivos Externos - .txt](#_k59kbtp1mw0r) 2

[Strings](#_bfwu8fmgdyew) 2

[**Resumo de Métodos — Listas**](#_8rkdo8qk7ma) **4**

[Criar e Adicionar Elementos:](#_jc03uksex68z) 4

[Remover e Excluir Elementos:](#_2kgg5zltand0) 4

[Obter Informações:](#_bxklr6s3ahz5) 4

[**Resumo de Métodos — Funções**](#_umgc3xpe373a) **5**

[**Resumo de Métodos — Dicionários**](#_3h3zds1aql1q) **5**

[**Resumo de Métodos - Pandas**](#_jwsx8tfmdcbb) **5**

[Criação e Abertura](#_juckhd7pzi0w) 5

[Visualização](#_ulmm7zy83wnc) 6

[Edição](#_4il7ojgfavil) 6

[Bibliotecas Adicionais](#_mpi6k4xuf8mc) 8

[**Resumo de Métodos - PyAutoGui**](#_y39hyiitvgwr) **9**

[**Resumo de Métodos - Pathlib**](#_demzxmxgcke1) **9**

[Biblioteca Adicional](#_cvi81fcbfwn9) 10

[**Resumo de Métodos - YaGmail**](#_rvi6j05zyxry) **10**

[**Resumo de Métodos - Extração de E-mail**](#_fuxwc47ehdp5) **10**

[Localizar Anexo Específico](#_1aj698qxgdxd) 11

[**Resumo de Métodos - Selenium**](#_fwuyzf9bi5o3) **11**

[Configurações no Google Colab](#_xlcglf7t1w1r) 11

# Integração com APIs

# Resumo de Métodos — Básicos Python

**\:** permite continuar o código na linha de baixo para não estender em horizonta

ctrl + /: transforma toda a seleção em comentário

## Arquivos Externos - .txt

**with open (‘DiretoriodoArquivo.txt’, ‘r’) as arquivo2:** Abre e fecha o arquivo dentro da iteração

**arquivo = open(‘DiretoriodoArquivo.txt’, ‘r’):** para abrir o arquivo

r: apenas ler

w: escrever (subescrevendo, substituindo)

a: de append, para adicionar informações ao arquivo

**arquivo.close()**: fecha e salva as alterações no arquivo aberto com Open

**arquivo.read()**: lê o arquivo de forma bruta, transformando tudo numa string

**arquivo.readlines()**: transforma cada linha do arquivo numa lista

## Strings

**texto.replace("Original", "Substituído")**: substitui o valor na frase apenas para a exibição em questão, mas não na variável original

**texto = texto.replace("Original", "Substituição")**: substitui o valor definitivamente ou atribui outra variável com o texto atualizado

**texto.upper()**: transforma a string inteira em maiúscula

**texto.lower()**: transforma a string inteira em minuscula

**texto.capitalize()**: toda a string fica minúscula, exceto a primeira letra

**texto.title()**: sempre que houver um espaço, o primeiro caractere fica maísculo

**texto.strip()**: remove todos os espaços inúteis no início e no fim da string

**texto.lstrip ou rstrip()**: remove somente os espaços no final ou no começo da string

**texto.split()**: divide a string em cada espaço, criando uma lista com todas as palavras

**"-".join(texto)**: unifica uma lista com strings colocando "-" entre cada valor da lista

**len(texto)**: calcula o número de índices em uma lista

**texto.count(x)**: calcula o número de vezes que o valor aparece na lista, pode ser especificado o começo e fim do intervalo

**texto.find(x)**: indica o índice que determinado valor começa dentro da lista/string (-1 significa inexistente)

**x in texto**: retorna o valor verdadeiro ou falso se existe o elemento procurado

**end=""**: não faz nada no fim da linha, grudando o próximo print com este

**Cores:** Letra, Fundo

**Black:** 30, 40 | Red: 31, 41 | Green: 32, 42 | Yellow: 33, 43 | Blue: 34, 44 | Magenta: 35, 45 |

Cyan: 36, 46 | Grey: 37, 47 | White: 97, 107 |

# 

# Resumo de Métodos — Listas

## Criar e Adicionar Elementos:

**a = list(range(x,y,z)**: cria uma lista de x até y contabilizando múltiplos de z

**lista.append(x)**: adiciona o elemento depois do último índice da lista

**lista.insert(slot, x)**: insere elemento x no índice determinado

**lista2 = lista1.copy**: copia uma lista na outra, mas totalmente independente (uma não altera a outra)

**set(lista)**: cria um set retirando que é similar a lista e tupla, mas sem valores repetidos

## Remover e Excluir Elementos:

**lista.remove(0):** procura e remove o valor no primeiro índice possível da lista

**lista.pop()**: procura e remove determinado valor no índice escolhido OU o último índice caso o valor não seja informado.

**lista.clear()**: limpa a lista e devolve uma vazia, é mais forte/melhor do que fazer manualmente, apaga o cache

## Obter Informações:

**len(lista):** calcula o número de índices em uma lista

**max(lista)**: informa o maior valor da lista

**lista.count(x,y)**: calcula o número de vezes que o valor aparece no intervalo ou na lista completa (padrão)

**lista.index(0)**: informa o primeiro índice que contém o valor procurado

**lista.find(x)**: indica o índice que determinado valor começa dentro da lista (-1 significa inexistente) percorrendo até o fim

**x in lista**: retorna o valor verdadeiro ou falso se existe o valor procurado na lista

**lista.slice(0:0:0)**: cria uma sublista dentro da lista somente para determinada exibição

**lista.reverse(0)**: põe a ordem oposta dos valores da lista

**lista.sort(key=None, reverse=False)**: organiza a lista em ordem crescente, modificando-a

key=None: usa uma function para cada elemento no iterable a ser ordenado

reverse=True/False: padrão de forma crescente, para reverter usar True

**sorted(lista)**: function que retorna um valor a ser atribuído numa variável

**comparação de listas (>, =>, ==, etc.)** = compara o primeiro índice que os elementos forem diferentes

# Resumo de Métodos — Funções

map(função, lista): aplica uma função em todos os elementos de uma lista

usar list para transformar em lista, pois assim como zip retornará um objeto

LAMBDA FUNCTION (sem nome e para criar função em uma linha)

variável = lambda argumentos: retorno

Funções que terminam com Lambda são criadores de funções.

# Resumo de Métodos — Dicionários

**dicionario["chave"] = 0**: adiciona uma nova chave no dicionario

**dict.fromkeys(chave, valor)**: cria um dicionario com os valores determinados, podendo padrozinar o valor também é possível criar através de tuplas com dois elementos

**zip(lista1, lista2)**: junta duas listas em uma tupla podendo criar dicionário

**dicionario.items()**: cria uma lista de tuplas com os dois elementos, chave e valor

**dicionario.keys()**: vai dar os nomes das chaves (mudar o dicionário muda essas listas, o list impede)

**dicionario.values()**: vai dar os valores das chaves (mudar o dicionário muda essas listas, o list impede)

# Resumo de Métodos - Pandas

pandas as pd

## Criação e Abertura

**tabela = pandas.read\_excel(r"C:\Users\Download\Arquivo.xls", sheets = 0, sep=None):** carrega uma planilha

r: opcional, para ler como texto bruto (as contra-barras não virarão comandos)

sheets: seleciona na pasta de trabalho qual planilha será carregada

sep: mostra o parâmetro de separação de colunas, o padrão é vírgula (Excel usa “;”)

encoding ='latin1', 'ISO-8859-1', 'utf-8', 'cp1252': caso bugue a base de dados por conta de caracteres

engine='python': retira possíveis erros indeterminados

**tabela\_final.to\_csv(r"C:\Users\Download\Nome do Arquivo.csv", sep=';')**: transforma dataframe numa planilha

**index=False:** não coloca o indice no arquivo final

**m\_dict(dicionário)**: transforma dicionário em dataframe/tabela

**orient='index'**: opcional, faz as linhas virarem as colunas

**tabela = pd.Dataframe(dados):** cria uma nova tabela vazia utilizando um dicionário

**caminho = r’//C:/Pastas/NomeDoArquivo.xlsx’**: define o nome e local do arquivo a ser salvo

**with pd.ExcelWriter(caminho) as arquivo:** define o nome do arquivo a ser salvo

**tabela\_final.to\_excel(arquivo, index=False):** define o dataframe pra virar arquivo

## Visualização

**tabela[x:x].plot()**: print de gráfico iniciando e terminando nos valores determinado, em branco conta tudo

**.plot(figsize=(y.x)**: escolhe a largura e a altura do gráfico

**display(tabela)**: print de tabela

**.head() e .tail()**: mostra os cinco primeiros e cinco últimos

**tabela.columns**: uma lista com todas as colunas da tabela

**tabela.roles**: uma lista com todas as linhas da tabela

**tabela.info()**: mostra a quantidade de linhas e colunas, nomes das colunas, quantas linhas vazias tem e tipo

**tabela[:3]**: mostra toda a tabela até a linha 3

**tabela["coluna"]**: mostra todos os valores na coluna

**tabela[['coluna1', 'coluna2', 'coluna 3']]:** destaque ou selecione múltiplas colunas

**tabela['coluna'][32]**: para pegar um item específico, descrever coluna e linha

**tabela["x"].value\_counts()**: conta em quantas linhas cada valor de uma coluna se repete

**(normalize=True)**: para aparecer em percentual os valores percentual decimal

**pd.set\_option('display.max\_rows', None)**: mostra todas as linhas

**pd.set\_option('display.max\_columns', None)**: mostra todas as colunas

**pd.set\_option('display.width', 2000):** largura total de todas as colunas

**pd.set\_option('display.max\_colwidth', None)**: mostra largura total de cada coluna

**pd.set\_option('display.float\_format', '{:20,.2f}'.format):** casas decimais de float

**tabela = tabela.style.set\_properties(\*\*{'text-align': 'left'}):** alinha os dados na esquerda

**tabela = tabela.set\_table\_styles([dict(selector = 'th', props=[('text-align', 'left')])]):** alinha o cabeçalho na esquerda

## Edição

**tabela[‘nova coluna’] = dados[1]+dados[2]:** cria uma nova coluna na tabela

**tabela['x'] = pd.to\_datetime('', format='%d/%m/%Y')**: faz uma coluna transformar object em data

format precisa estar insinuando como estão as datas para que reconheça certinho

**tabela2 = tabela.groupby('x')**: TABELA DINÂMICA, agrupa os valores repetidos nas colunas

**tabela.groupby('x').function()**: *max, min, mean, sum, etc*.: dá pra fazer diversas coisas com os valores

exclui datas, nome e coisas que não consegue somar ou fazer a alteração na tabela

A coluna vira os índices (título das linhas) e vira dicionário

**tabela2 = tabela[(tabela['x'] > 300) & (tabela['z'] < 1000]**: FILTRAR determinada condições, igual, maior, etc, cite as colunas e comparações desejadas

**isna() ou isnull()**: método para apenas os valores vazios

**notna()**: método para apenas os valores preenchidos

**tabela2 = tabela.drop(["x","y","z"])**: excluir informação da tabela

axis = 0/1: 0 é linha, 1 coluna

**.drop\_duplicates([‘coluna’]):** retira as linhas duplicadas

* + - keep=’first’: mantém o primeiro (e no mínimo um valor)

**tabela2 = tabela[['coluna1','coluna2','etc.']]**: cria uma nova tabela só com as colunas citadas

**tabela2 = tabela.dropna(how all/any, axis)**: exclui as linhas ou colunas vazias

subset[‘x’]: escolhe a coluna a ser pesquisada

how: all/any, se todos os valores forem vazios; se ao menos um valor estiver vazio

**tabela2 = tabela.reset\_index(drop=True)**: atualiza os índices após o comando dropna

**tabela3 = tabela\_1.merge(tabela\_2, on='coluna")**: vê as colunas iguais entre as planilhas para juntá-las

how=”command”: *t\_1 e t\_2 refere-se ao nome “tabela” no exemplo do método*

inner: a nova planilha utiliza apenas as linhas com valores compartilhados entre ambas

left: todas linhas da t\_1 é usada e as inexistentes na t\_2 são substituídas por NaN, linhas da t\_2 inexistentes na t\_1 são removidas

right: similar ao left, mas preservando as linhas da t\_2

outer: todas as linhas são utilizadas preenchendo com NaN as lacunas

Indicator=True: cria uma coluna “\_merge” com “both”, left\_only ou right\_only. Para excluir valores repetidos em ambas.

query(“\_merge == both”): define verdadeiro ou falso no critério

.drop(“\_merge == 1”): exclui todas as linhas com verdadeiro

**tabela2 = tabela.rename(columns={'Coluna original':'Novo nome da coluna'})**

**tabela = tabela.sort\_values('x')**: ordenar de acordo com os valores da coluna especificada

**tabela2["x"].sum()**: soma os valores da coluna

**tabela2["x"].unique()**: contabiliza apenas valores únicos

**tabela2["x"].count()**: conta a quantidade de linhas

**tabela2["x"].mean()**: dá a média da coluna

**tabela2["x"].min/max()**: maior ou menor valor da coluna

**tabela['x'].idmin/idmax()**: o índice/linha do maior ou menor valor da coluna

**tabela2["x"] = pd.to\_numeric("x", error = "coerce")**: transforma a coluna em int ou float e todas as string/texto em vazio

**.map("{:.x%}".format)**: para formatar o percentual real (x = casas decimais)

**tabela.loc['linha', 'coluna']**: para selecionar ou editar dados especificos de uma célula

**tabela.loc[[0, 3, 5]]**: dá os dados das linhas dos índices especificados na lista

**tabela.loc[0:4]**: mostra os dados das linhas até o índice selecionado

**tabela.loc['x'] == 200, 'y'] = 24**: seleciona e filtra com comparações para atribuir valor

**tabela.loc[tabela['coluna'] == 'x', 'y'] = xyz**: modifica apenas as linhas da coluna y dentro dos parâmetros x

**tabela.iloc[0, 3]**: para pegar dados específicos de uma célula, sem nome, SEMPRE NÚMERO

**tabela2 = tabela.set\_index('x')**: transforma uma coluna no índice = nome das linhas

**for i, coluna\_base in enumerate(tabela[‘Coluna Percorrida’]):**

**dado\_extraido = tabela.loc[i, Coluna]:** percorre toda as as linhas da coluna (indicada por i salvando na variável)

## Bibliotecas Adicionais

**datetime:** utilizado para formatar e trabalahr com datas

[Biblioteca explicada!](https://www.alura.com.br/artigos/lidando-com-datas-e-horarios-no-python?gclid=CjwKCAjwur-SBhB6EiwA5sKtjk_D9HjFFUEt8QlCnT4RucSV9yMWwuSthH2-y06iBoTTywY3M5YcuhoC95cQAvD_BwE)

**Plotly Express** as px: cria gráficos, sempre em duas linhas de código, uma para criar e outra para exibir

**grafico = px.histogram(tabela, x="x", colors="x")**:

**x**: define quais informações estarão em barras na horizontal

**colors**: define qual informação fará as separações das barras

**color\_discrete\_sequence=["blue, yellow"])**: define as cores utilizadas

**grafico.show():** mostra o gráfico na tela

**requests:** conseguir puxar o conteúdo de um link quando necessita requisitar a um servidor

**conteudo\_link=requests.get('URL').content**: baixa o conteudo da URL

**io**: transforma texto em um dataframe em poucas linhas de código

**arquivo= io.stringIO('conteúdo\_link.decode('latin1'):** “conteúdo\_link” criado no requests

**json:** dicionário que a maioria das APIs são importadas

**arquivo = arquivo.json()**: faz o Python ler o arquivo

**tqdm**: barra de progresso de um código, serve para avaliar desempenho

from tqdm import tqdm

**pbar = tqdm(total=len(planilha['Coluna']), position=0, leave=True):** Comando para que não seja mostrado em apenas uma linha

**openpyxl**: importa planilhas do excel preservando fórmulas e gráficos, mas precisa de FOR na maioria das vezes

from openpyxl import workbook, load\_workbook

var\_planilha = load\_workbook(r'NomeDoArquivo.xlsx')

plan\_ativa = var\_planilha

for celula in plan\_ativa('Coluna'):

if celula.value == 'x':

linha = celula.row

plan\_ativa[f'D{linha}'] = xyz

var\_planilha.save('NomeDoArquivo.xlsx')

# Resumo de Métodos - PyAutoGui

**pyperclip**: para copiar links ou endereços com caracteres especiais

**pyperclip.copy("texto")**: texto a ser copiado sem precisar ser selecionado

**pyautogui.press("x")**: pressiona uma tecla do teclado

**pyautogui.hotkey("x","y","z")**: pressiona um conjunto de teclas

**pyautogui.write("abc)**: digita um texto com o teclado

**pyautogui.position()**: indica as coordenadas atuais do mouse (diferente pra cada monitor)

**pyautogui.click(x=000, y=000, clicks=2)**: informa as coordenadas e o número de cliques

**pyautogui.pause = 0**: insere uma pausa entre cada comando do código

# Resumo de Métodos - Pathlib

from pathlib import Path

**Path.cwd()**: mostra o local aonde está rodando o código [current working directory]

**arquivo = Path(‘C:/Users…etc’)**: abre a pasta, sempre com barra normal e omitir o diretório até o local atual o fará ser o começo da busca

Colocar barra no final fará a pasta ser aberta

**arquivo = arquivo.iterdir()**: mostra todos os itens no diretório/local como objeto (pode percorrer com FOR)

**arquivo = arquivo.name**: transforma o arquivo numa string, possibilita o IF

**if arquivo / Path(‘NomeDoArquivo.csv’).exists()**: checa e faz ações caso no local haja o procurado

**Path(‘Pasta1/Pasta2’).mkdir()**: cria pastas dentro de pastas

## Biblioteca Adicional

**shutil:**

**arquivo\_copiado = Path(‘diretorio/NomeDoArquivo.csv’)**

**arquivo\_colar = Path(‘diretorio/NomeDaCopia.csv’)**

**shutil.copy2 (Path da cópia, Path do destino):** precisa estar em formato Path como nas variáveis arquivo\_copiado e arquivo\_colar

**shutil.move(Path do arquivo recortado, Path do local colado)**

# Resumo de Métodos - YaGmail

import yagmail

**yagmail.register(‘e-mail’,’senha’):** registra uma conta na biblioteca keyring para não expor a senha no script

**yagmail.SMTP(user=’e-mail’):** faz login na conta

**password=’senha’:** se pular o register, a senha precisará ficar no código

**yagmail.send(to=’destinatário’, subject=’assunto’, contents=’conteúdo’)**

**attachments=**’Nome do Arquivo.xlsx’: envia anexo junto, opcional

**cc = ‘e-mail’:** envia para alguém em cópia

**bc = ‘e-mail’**: envia para alguém em cópia oculta

**to/cc/bc=[‘e-mail 1’, e-mail2]:** envia para múltiplos e-mails

**contents:** pode colocar variável e dentro dela por textos com três aspas

**yagmail.SMTP('mygmailusername').send('to@someone.com', 'subject', 'This is the body'):** método para enviar e-mail numa linha só, é possível informar o password

# Resumo de Métodos - Extração de E-mail

[Repositório](https://github.com/ikvk/imap_tools#search-criteria) (com critérios após o fetch (AND, OR, etc) e métodos

[Lista de Imap](https://www.systoolsgroup.com/imap/)

**username = "seu\_email"**: salva o login numa variavel

**password = "senha"**: salva a senha numa variavel

**meu\_email = MailBox('Imap\_link').login(username, password)**: fornece acesso ao e-mail

**lista\_emails = meu\_email.fetch(AND(criterio1 = ‘informação’, criterio2 = ‘informação’)):** checar os critérios substituindo o AND e os criterio1 e 2 no Repositório, devolve objeto.

## Localizar Anexo Específico

**lista\_emails = meu\_email.fetch(AND(from\_="remetente")):** busca todos os e-mails de um remetente, o critério pode ser alternado conforme Repositório, AND ou OR pode ser substituído

**for email in lista\_emails:** passa por todos os e-mails que atendem ao critério

**if len(email.attachments) > 0:** checa se há pelo menos um anexo em cada e-mail dentro do critério

**for anexo in email.attachments:** SE anexos for maior que 0, passa por todos eles

**if "TituloAnexo" in anexo.filename:** critério verificado ao passar pelos os anexos

**salvar\_anexo = anexo.payload:** SE critério atendido, salva anexo na variável

**with open("Nome\_do\_Arquivo.xlsx", 'wb') as arquivo\_excel**: cria o arquivo

**arquivo\_excel.write(anexo.payload):** escreve no arquivo os dados armazenados

# Resumo de Métodos - Selenium

Baixar o “ChromeDriver” (depende do navegador) e colocar na pasta Anaconda3 na pasta do Usuário, depois dar um Pip Install para utilização no computador.

**from selenium import webdriver**

Para não mostrar o navegador abrindo, utilizar no add\_argument o “headless”

**options\_navegador = webdriver.ChromeOptions()**: compila as configurações desejadas

**options\_navegador.add\_argument(‘x’)**: adiciona uma configuração ao compilado

**navegador = webdriver.Chrome()**: cria a instância do navegador ou se conecta com ele

webdriver.Edge(executable\_path=r'C:\Pasta\Arquivo.exe'): para o “ChromeDriver”

**navegador.get(‘www.link.com’’)**: abre o site desejado no navegador

**navegador.switch\_to.frame(navegador.find\_element\_by\_tag\_name(‘iframe’)):** acessa uma página dentro de outra página

**navegador.swith\_to.default.content():** volta para a tela principal quando entra no site

## Interação com a Página

**navegador.execute\_script(window.scroll(x, y):** mover uma barra em horizontal ou vertical

**interacao.get\_attribute(‘x’):** pega o valor de determinado atributo do elemento escolhido

Pode ser utilizado junto de uma lista de elementos da mesma classe e pegar determinada informação dela.

Método 1 de selecionar e interagir com um campo:

**interacao = navegador.find\_elements\_by\_name(‘x’)**: armazena campos ou botões numa lista

Também pode ser utilizado no singular com “element” ou sem atribuir variável

[Lista de Localização de Elementos](https://selenium-python.readthedocs.io/locating-elements.html)

xpath: É modificável por alterações, mas certeiro.

**interacao.send\_keys(‘xxx’)**: preenche o formulário com ‘xxx’

Método 2 de selecionar e interagir com um campo:

**navegador.find\_elements\_by\_name(‘x’)[0].send\_keys(z):** procura pelo item ‘x’ na página do navegador e põe todos numa lista, seleciona o índice a ser preenchido com ‘z’.

**.submit():** envia o formulário previamente preenchido

**.click()**: clica num botão identificado

## Aguardando Conteúdo

Método 1 de aguardar conteúdo aparecer na tela:

**from selenium.webdriver.common.by import By**

**from selenium.webdriver.support.ui import WebDriverWait**

**from selenium.webdriver.support import expected\_conditions as EC**

**interacao = WebDriverWait (navegador, 30).until(EC.presence\_of\_element\_located((By.CLASS\_NAME, ‘x’))):**

Aguarda a condição do Until ser atingida ou o tempo de ‘30’ segundos.

[Lista de opções do Until](https://selenium-python.readthedocs.io/locating-elements.html)

Método 2:

from time import sleep

**while len(navegador.find\_elements\_by\_class\_name(‘x’)) == 0:**

time.sleep (1)

**navegador.find\_element\_by\_class\_name(‘x’).click()**

## Configurações no Google Colab

!pip install selenium

!apt-get update

!cp /usr/lib/chromium-browser/chromedriver /usr/bin

!apt install chromium-chromedriver

import sys

sys.path.insert(0,'/usr/lib/chromium-browser/chromedriver')

from selenium import webdriver

chrome\_options = webdriver.ChromeOptions()

chrome\_options.add\_argument('--headless')

chrome\_options.add\_argument('--no-sandbox')

chrome\_options.add\_argument('--disable-dev-shm-usage')

# Resumo de Métodos - Python em .exe

Transforma arquivos .py em .exe em que estejam totalmente funcionais.