Automatizando tudo no seu computador com PyAutoGUI - Apostila Asimov Academy

Asimov Academy

ASIMOV

Conteúdo

01. Para que serve o PyAutoGUI?

Bem-vindos à apostila do curso “Automatizando tudo no seu computador com PyAutoGUI” da Asimov Academy! Nesta apostila, nosso foco é aprender a automatizar tarefas no nosso computador, deixando o mouse e o teclado executarem ações sem precisarmos utilizá-los.

Cada capítulo aborda um novo conceito ensinado ao longo das aulas do curso. Dessa forma, iremos construindo nosso conhecimento de forma incremental. Ao longo da apostila, são abordados diversos conceitos amplamente utilizados em automações de tarefas utilizando Python.

Mas afinal, qual o objetivo do PyAutoGUI?

O PyAutoGUI é uma biblioteca do Python que fornece funcionalidades para automatizar a interação com a GUI (Graphic User Interface, *traduzindo para o portugês: Interface Gráfica do Usuário*) de um computador. Ele permite que você escreva scripts para controlar o mouse e o teclado, tirar capturas de tela, realizar cliques e movimentos do mouse, entre outras operações.

Cuidados

É importante notar que a automação da interface do usuário através do PyAutoGUI pode ter limitações, especialmente em aplicações mais complexas que usam tecnologias mais avançadas. Nem sempre o PyAutoGUI vai ser capaz de performar da forma ideal, funcionando em qualquer ambiente. Existem diversos fatores que variam de caso para caso, impactando diretamente no desempenho da biblioteca no computador, como o sistema operacional, a versão de um programa, o tipo de navegador (para programas que acessam a internet), aplicativos instalados etc.

O que realmente importa ao utilizar o PyAutoGUI é a lógica pensada para realizar cada ação no escopo em que ele está sendo aplicado. **É um desafio encontrar uma forma de interação com os programas que garanta o desempenho correto da biblioteca**. Porém, aqui nesse curso, iremos abordar todas a principais operações e como elas funcionam, para que, a partir do conhecimento de todas essas ferramentas, estajamos prontos para implementar tais operações para construir um script que automatize um processo completo.

Então, vamos falar sobre a lógica do PyAutoGUI e porque é importante entendê-la para melhorarmos a qualidade dos programas que utilizam essa ferramenta poderosa.

02.Lógica do PyAutoGUI

A lógica do PyAutoGUI é baseada em simular ações do usuário, como cliques, movimentos do mouse e pressionamento de teclas, para realizar tarefas repetitivas de forma automatizada. Vamos abordar alguns pontos chave:

1. **Dinâmica de Resolução**
2. **Controle do Teclado**
3. **Controle do Mouse**
4. **Captura de Tela**
5. **Espera e Sincronização**
6. **Funções Interativas com o programa**
7. **Tratamento de Exceções**

O funcionamento do PyAutoGUI é baseado na manipulação das APIs de automação de interface gráfica fornecidas pelos sistemas operacionais, como o Windows API, X11 no Linux, e Quartz no macOS. Essas APIs permitem que o PyAutoGUI simule eventos do usuário, como cliques e movimentos do mouse, para interagir com elementos da interface gráfica.

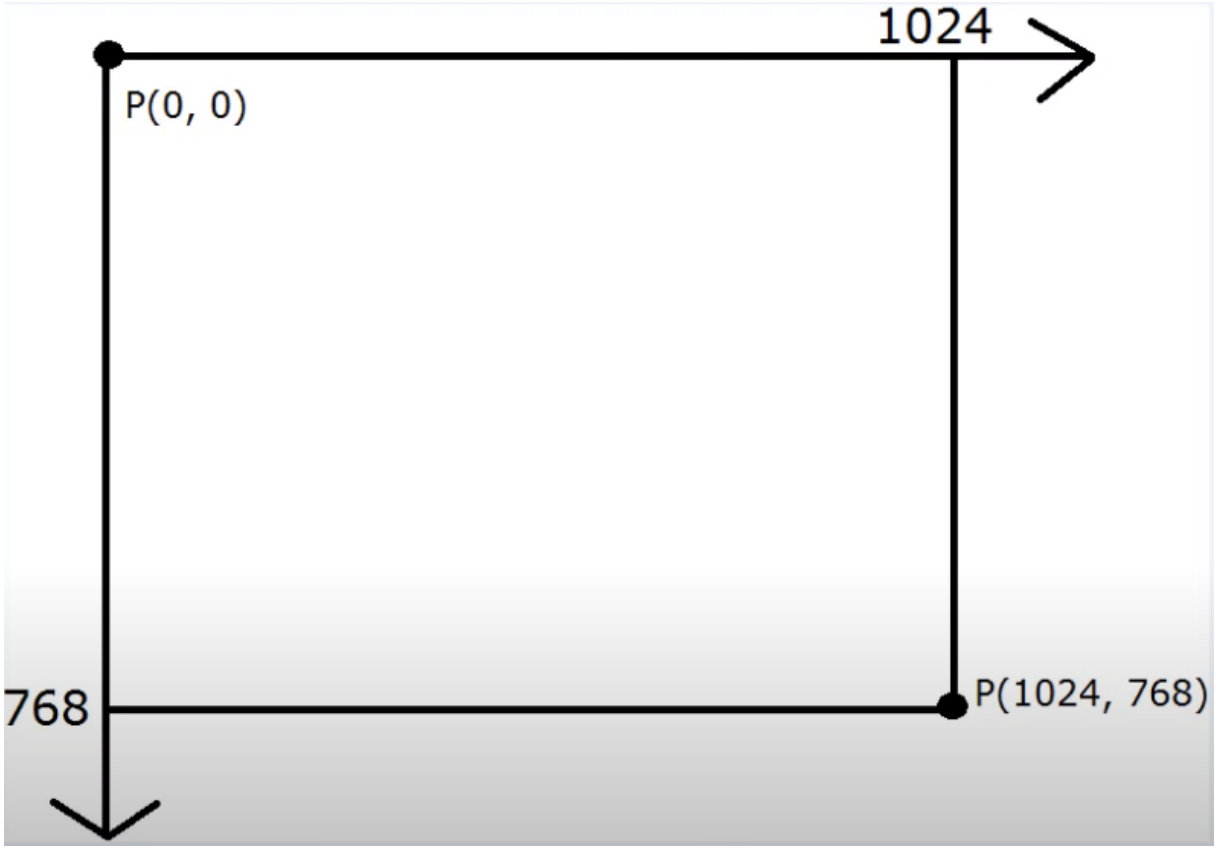
No entanto, é importante observar que o PyAutoGUI é dependente de coordenadas na tela e não tem conhecimento direto dos elementos da interface do usuário. Isso significa que as mudanças na resolução da tela ou na posição dos elementos podem afetar a robustez do código, sendo necessário ajustar as coordenadas conforme necessário.

03. Plano cartesiano na tela

O plano cartesiano da tela é representado de forma que o ponto (0,0), conhecido também como ponto de origem, geralmente está no canto superior esquerdo da tela, com os valores positivos aumentando para a direita (eixo x) e para baixo (eixo y). Essa abordagem pode ser um pouco diferente da abordagem convencional quando estamos tratando do eixo y, em que o eixo y cresce para cima.

Esses valores de X e Y representam a quantidade de pixels existente na tela.

Exemplo do plano cartesiano de uma tela com dimensões de 1024x768 pixels:



Porém, como vocês já devem imaginar, as dimensões das telas variam de modelo a modelo, podendo existir diversos formatos, como 1920x1080, 2560x1440, 1080x1920, 1024x600 etc. Essas diferenças podem impactar a automação com PyAutoGUI, especialmente se os scripts foram originalmente desenvolvidos em uma tela com uma resolução específica e, posteriormente, são executados em uma tela com uma resolução diferente. E é por isso que mais pra frente iremos abordar estratégias para evitar esses problemas de dimensões.

O plano cartesiano nos permite movimentar o mouse de forma relativa e de forma absoluta, identificar componentes em coordenadas específicas e determinar limites. Essas são propriedades fundamentais

para conseguirmos desenvolver códigos que executem ações com precisão e de forma correta.

04. Inserção de texto, teclas de atalho e navegação

As interações com o teclado no PyAutoGUI são realizadas através de funções específicas que simulam a pressão e liberação de teclas, teclas de atalho e a entrada de texto. Vamos explorar essas funcionalidades com mais detalhes.

Inserção de Texto

Para inserir texto, como se o usuário estivesse digitando no teclado, existem as funções `pyautogui.typewrite()` e `pyautogui.write()`

Ambas possuem exatamente o mesmo objetivo e significado, são opções idênticas. Mas para focar nossos próximos exemplos, vamos utilizar somente a função `pyautogui.typewrite()`.

Essa função possui um parâmetro `interval` que é opcional, referente ao intervalo de tempo, em segundos, entre a digitação de cada caractere do texto inserido. Por exemplo:

```
pyautogui.typewrite("Ola, mundo!", interval=0.10)
```

Nesse caso, será escrito na tela a frase “Olá, mundo!”, sendo que cada caractere terá um intervalo de 0.10 segundos entre a sua digitação e a digitação do próximo caractere. Obviamente, quanto maior for o atributo passado para o parâmetro `interval`, maior será o tempo para finalizar a escrita do texto. Caso não seja passado nenhum valor para o `interval`, por default, o valor será igual a 0 segundos, ou seja, o texto será escrito instantâneamente, sem o efeito visual da digitação.

Pressionar teclas

Essa é uma funcionalidade bastante intuitiva no PyAutoGUI, basicamente você só precisa informar dentro da função `pyautogui.press()` o nome da tecla que desejar pressionar, conforme o exemplo a seguir:

```
pyautogui.press('enter')
```

Neste exemplo, a tecla ‘enter’ será executada na altura de código em que estiver inserida. Então, um uso muito comum dessa função é logo após uma inserção de texto, de forma que o texto escrito na linha anterior do código seja confirmado a partir da tecla ‘enter’, como quando enviamos mensagem em alguma plataforma de chat. Por exemplo:

```
pyautogui.typewrite("Bom-dia, amigos!", interval=0.10)
pyautogui.press('enter')
```

Nesse caso, se o texto fosse inserido em uma conversa do WhatsApp, por exemplo, o texto “Bom-dia, amigos!” seria enviado após a tecla ‘enter’ ser executada pela função `pyautogui.press`

Além da função `pyautogui.press`, existe as funções `pyautogui.keyDown` e `pyautogui.keyUp` que também representam a ação de pressionar teclas, mas essas duas últimas possuem uma funcionalidade específica para manter pressionada uma tecla e liberar o pressionamento dessa tecla, respectivamente.

Essas funções são úteis para casos em que você quiser pressionar outras teclas do teclado enquanto alguma outra tecla estiver sendo pressionada. Um caso prático em que isso é aplicado, é quando queremos aumentar ou diminuir o nível de zoom de uma página, em que apertando "ctrl" + "+" aumenta o zoom e "ctrl" + "-" diminui o zoom.

Por exemplo, um caso em que sempre que uma página é acessada pela primeira vez, as letras e os componentes de um site são muito pequenos. Nesse caso, poderia ser aplicado um aumento de zoom 2x, 3x ou mais.

```
pyautogui.keyDown('ctrl')
pyautogui.press('+')
pyautogui.press('+')
pyautogui.press('+')
pyautogui.keyUp('ctrl')
```

No caso acima, é aplicado um zoom de 3x.

Outro caso prático seria pra inserir um texto com letras maiúsculas, para isso bastaria manter a tecla 'shift' pressionada e inserir um texto enquanto ela é mantida pressionado, e depois liberá-la da execução (*não esqueça de sempre liberar uma tecla que se manteve pressionada*), conforme demonstrado abaixo:

```
pyautogui.keyDown('shift')
pyautogui.typewrite("Ola, mundo!", interval=0.10)
pyautogui.keyUp('shift')
```

Teclas de atalho

No PyAutoGUI, podemos configurar teclas de atalhos para serem apertadas de forma simultânea, para exercer algum tipo de ação específica, como:

- Alt + Tab
- Ctrl + C
- Ctrl + V

Para configurar uma tecla de atalho, basta utilizar a função `pyautogui.hotkey()` e passar como atributo, dentro dos parênteses, a sequência de teclas que devem ser pressionadas ao mesmo tempo ' Por exemplo:

```
pyautogui.hotkey('alt', 'tab')
```

Nesse caso, essa linha de código irá trocar a aba que está aparecendo na sua tela, alternando com alguma outra aba que esteja aberta.

Outro exemplo:

```
pyautogui.hotkey('alt', 'f4')
```

Essa linha de código irá fechar a aba ou programa que está aparecendo na sua tela, finalizando o processo.

Navegação

O PyAutoGUI fornece funções para auxiliar na execução do programa, tornando a navegação mais lenta quando preciso.

Essa é uma funcionalidade muito importante para garantir que uma ação seja executada somente após um determinado tempo desde a última ação. Um exemplo prático seria pensar quando entramos em um site novo, em que existe um tempo de carregamento das informações (e, dependendo do site, esse tempo pode ser maior do que em outros sites), impactando diretamente no tempo que vai levar até aparecer na tela alguns componentes desse site.

Outro caso prático, ao abrir a caixa de e-mail. Após acessar o site do seu provedor de e-mail, às vezes o site atualiza para mostrar os e-mails mais recentes, e essa atualização leva um tempo. Caso o objetivo de um programa seja pegar os 5 últimos e-mails recebidos, por exemplo, seria necessário aguardar essa atualização padrão que o site realiza.

Quando se está navegando por pastas ou páginas, esse tipo de espera é comum, e por isso precisamos lidar com esse caso. E, para isso, utilizaremos a função `pyautogui.sleep()`, que pausa o programa por um determinado tempo, em segundos, passado como atributo para dentro da função. Desta forma, sempre que essa função for inserida no código, a execução do código será pausada na linha em que a função for inserida.

Por exemplo:

```
pyautogui.sleep(5)
```

Nesse caso, o programa ficaria parado por 5 segundos, e após esse tempo retomaria a execução do código a partir da próxima linha, após a chamada da função.

Naturalemente, quanto maior o valor passado como atributo da função, maior será o tempo de espera.

De forma semelhante ao `pyautogui.sleep()`, existe outra função que possui o mesmo objetivo, mas que é aplicada de modo geral, sendo inserida apenas uma vez e no início do código. Essa função é

a `pyautogui.PAUSE`. Basicamente, o que ela faz é configurar um tempo, em segundos, de espera para a execução de qualquer função do PyAutoGUI. Por exemplo:

```
pyautogui.PAUSE = 2.5
```

Nesse caso, todas as funções do PyAutoGUI teriam um intervalo de 2.5 segundos entre suas execuções.

05. Coordenadas do mouse na tela

Para dar início ao tópico sobre interação com mouse utilizando PyAutoGUI, é fundamental o compreendimento do tópico [03. Plano cartesiano na tela], pois é por meio das coordenadas da tela que o cursor do mouse se movimentará.

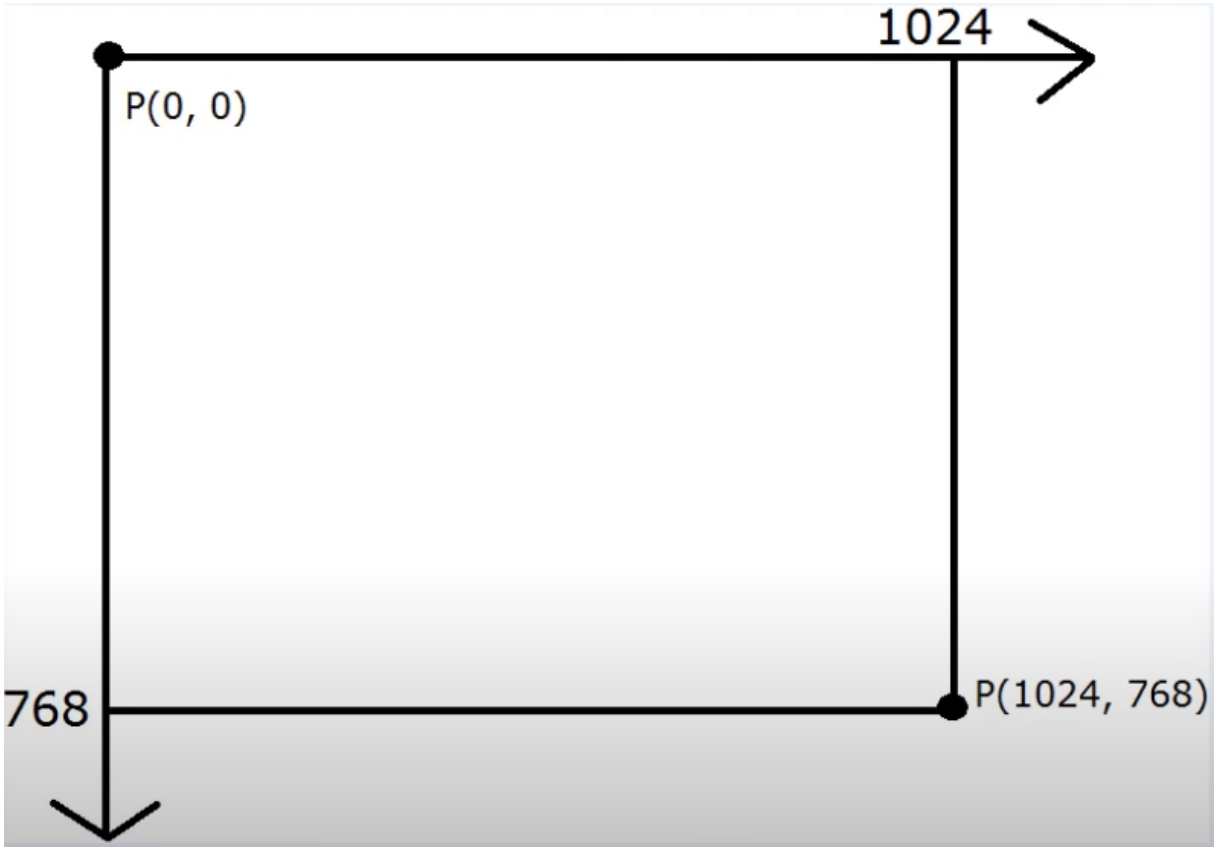
Sendo assim, é necessário identificarmos precisamente os valores das coordenadas na tela em que o programa que utilize a navegação do mouse com PyAutoGUI está sendo executado, pois a partir dos valores das coordenadas, é possível informar qual região específica da tela que o mouse deve atingir. Lembrando que os valores do plano cartesiano de cada tela muda de acordo com o modelo, tamanho e marca, então cada tela terá suas próprias coordenadas, o que torna ainda mais importante identificarmos as coordenadas da tela onde o código será executado.

Portanto, abaixo está um pequeno script que, ao ser executado, irá imprimir no terminal as coordenadas X e Y em que o mouse está localizado. Desta forma, você poderá movimentar seu mouse e acompanhar os valores das coordenadas em tempo real mudando conforme o mouse for movimentado.

Script de coordenadas

```
import pyautogui
print('Aperte Ctrl-C para sair.')
try:
    while True:
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        print(positionStr, end='')
        print('\b' * len(positionStr), end='', flush=True)
except KeyboardInterrupt:
    print('\n')
```

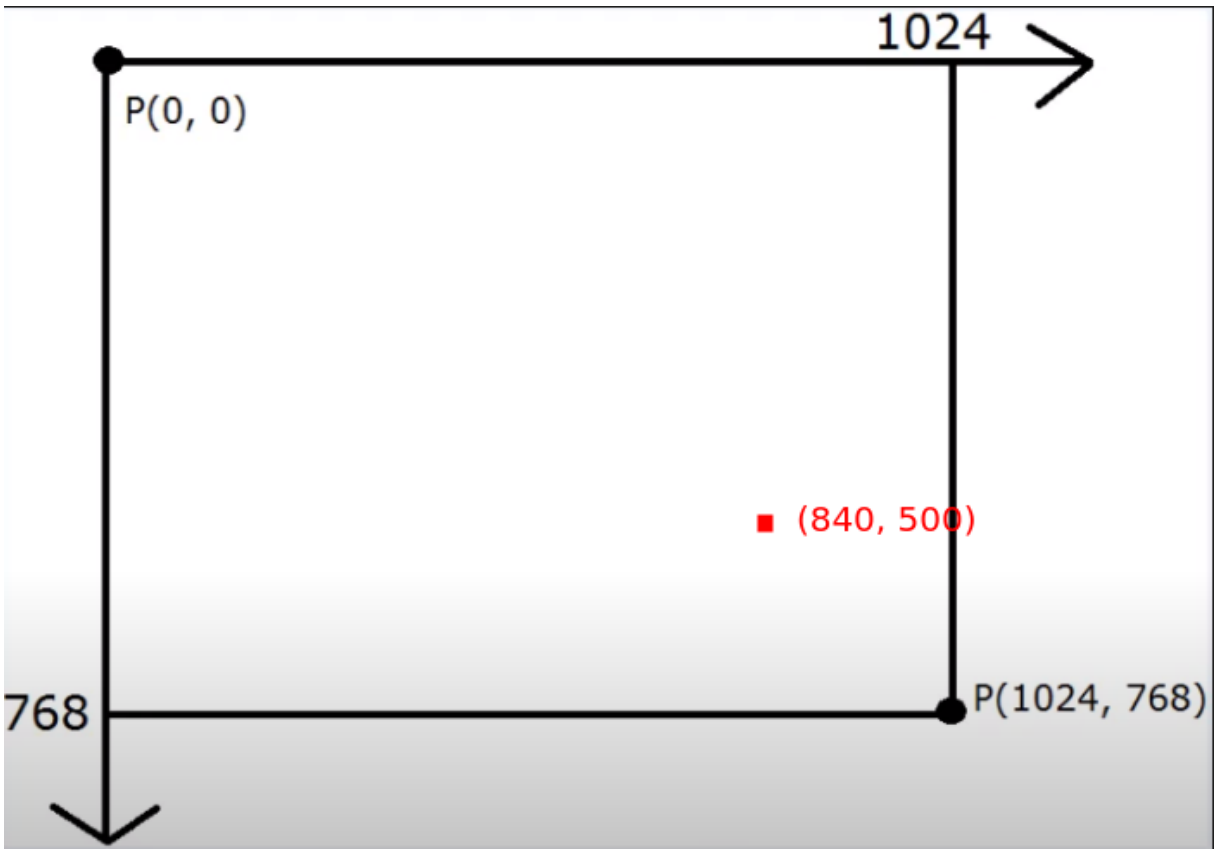
Vamos analisar novamente a imagem do plano cartesiano da tela logo abaixo:



Considerando que o script informado logo acima da imagem do plano cartesiano está sendo executado, vamos pegar como exemplo o seguinte valor de coordenadas:

X: 840 Y:500

Esse conjunto de coordenadas representa a exata localização de onde o mouse do cursor está aparecendo na tela, conforme imagem abaixo:



Desta forma, se na loocalização (840, 500) existe algum coomponente que deve ser clicado pelo mouse (veremos na sequência os tipos de cliques), então esses valores de coordenadas devem ser informados à alguma função de movimento do mouse do PyAutoGUI, que é o que veremos no próximo tópico.

06. Movimentos com o mouse

Agora que você já entendeu como funcionam as coordenadas da tela e qual sua função, vamos abordar as funções de movimentar o mouse.

Movimento absoluto

O movimento absoluto representa o deslocamento para uma coordenada exata. Desta forma, quando for informado um conjunto de coordenadas, o cursor do mouse será movido exatamente para a localização desse conjunto de coordenadas.

A função `pyautogui.moveTo()` representa este tipo de movimento. Iremos passar apenas dois atributos para essa função, a coordenada X e a coordenada Y, respectivamente.

Abaixo, um exemplo de aplicação:

```
pyautogui.moveTo(x=220, y=100)
```

O resultado do código acima será o cursor do mouse movimentado para o ponto exato na tela representado pelo conjunto de coordenadas (X=220, Y=100), conforme demonstrado na imagem abaixo:

Movimento relativo

Uma informação importante para entendermos melhor como o movimento relativo do mouse é executado: cada valor passado como atributo para a função do movimento relativo representa a quantidade de pixels que o cursor do mouse se deslocará.

O movimento relativo representa o deslocamento a partir de uma coordenada, da coordenada em que o mouse se encontra no momento. Diferente do movimento absoluto, em que o mouse erá movimentado exatamente para a localização das das coordenadas informadas nos atributos, no movimento relativo o cursor do mouse será deslocado em uma quantidade referente aos valores passado como atributo da função.

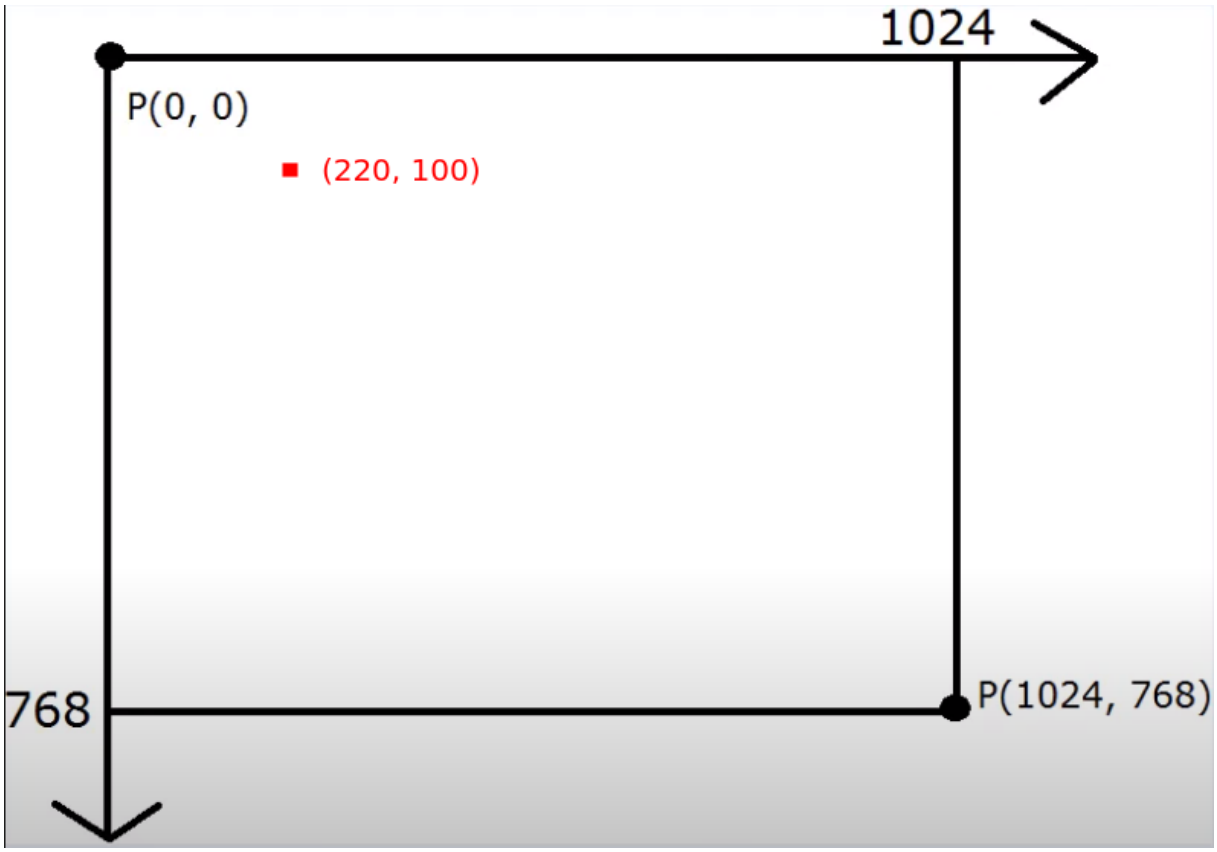
A função `pyautogui.moveRel()` representa este tipo de movimento. Iremos passar apenas dois atributos para essa função, o deslocamento em X e o deslocamento em Y, respectivamente.

Abaixo, um exemplo de aplicação:

```
pyautogui.moveRel(xOffset=100, yOffset=150)
```

O resultado do código acima será o cursor do mouse deslocado em 100 pixels no sentido positivo no eixo X e 150 pixels no sentido positivo no eixo Y. Considerando que o mouse está nas coordenadas mostradas na imagem anterior do exemplo do movimento absoluto, após executar a linha de código

acima, do movimento relativo, o cursor do mouse se deslocará para as coordenadas 320, 250, conforme demonstrado na imagem abaixo:



Curiosidade: tanto a função `pyautogui.moveTo()` quanto `pyautogui.moveRel()` possuem como terceiro parâmetro (opcional) um valor para a duração do movimento. Este parâmetro é chamado de `duration`. Esta é uma forma de acompanhar a animação do movimento do mouse na tela, em vez de ele movimentar-se instantânemanete para o ponto destino.

Um exemplo em que a animação do movimento é executada ao longo de 5 segundos:

```
pyautogui.moveTo(x=1900, y=400, duration=5)
```

Movimento de clique e arraste

O movimento de clique e arraste representa a ação de clicar em algum elemento na tela e arrastá-lo para outro ponto da tela. Semelhante ao pressionamento de teclas do mouse, esse movimento é executado por duas funções: `pyautogui.mouseDown()` e `pyautogui.mouseUp()`

Desta forma, quando for informado um conjunto de coordenadas para a função `pyautogui.mouseDown()`, o cursor do mouse será movido exatamente para a localização desse conjunto

de coordenadas e, quando atingir o ponto informado, automaticamente o mouse será clicado e permanecerá clicado até que a função `pyautogui.mouseUp()` seja executada. Ou seja, uma função executa o clique e a outra libera o clique, sendo que enquanto o clique não for liberado, o cursor do mouse permancerá em estado de clique.

estado de clique = situação em que o botão esquerdo do mouse se encontra pressionado

Essas duas funções geralmente apresentam mais valor quando utilizadas em conjunto com alguma outra função de movimento, como a `pyautogui.moveTo()`, por exemplo, pois desta forma podemos informar uma coordenada onde se encontra um elemento específico na tela, clicar nesse elemento e trazê-lo até um outro ponto, utilizando a função `pyautogui.moveTo()`, e, ao atingir o ponto destino, liberamos o elemento do clique.

Abaixo, um exemplo de aplicação:

```
pyautogui.mouseDown(x=100, y=130)
pyautogui.moveTo(x=1900, y=400, duration=5)
pyautogui.mouseUp()
```

O resultado do código acima será o cursor do mouse movimentado para o ponto 100, 130. Ao chegar neste ponto, o mouse será clicado e mantido em estado de clique. Depois, com a função de movimento absoluto do mouse, o mouse clicado será arrastado para o ponto 1900, 400, ao longo de 5 segundos. Ao chegar no ponto destino, por fim, o mouse será liberado do estado de clique.

Observe que na função `pyautogui.mouseUp` não foi informado nenhum atributo, pois seu único objetivo é liberar o clique do mouse que foi executado pela função `pyautogui.mouseDown()`.

Movimento com clique pressionado

Semelhantemente ao movimento de clique e arraste, o movimento com clique pressionado também tem como objetivo movimentar o mouse sob o estado de clique. A diferença é que no movimento com clique pressionado, o mouse já inicia seu movimento em estado de clique.

Existem duas funções que executam o movimento com clique pressionado, em que uma realiza o movimento de forma absoluta `pyautogui.dragTo()`e a outra de forma relativa `pyautogui.dragRel()`.

O exemplo abaixo demonstra o movimento com clique pressionado.

```
pyautogui.dragTo(x=100, y=100, duration=5)
```

Observe que esse movimento também possui o parâmetro opcional `duration` para acompanhar a animação do movimento.

07. Cliques com o mouse

Agora que já sabemos como utilizar o plano cartesiano da tela, como identificar coordenadas e como movimentar o mouse para pontos específicos da tela, vamos falar sobre os cliques.

Clique com botão esquerdo

É a função que executa clique(s) com o botão esquerdo do mouse. Desta forma, sempre que a intenção do programa for clicar em um ícone da barra de tarefas, no menu iniciar, uma aba do navegador, um link, um botão, uma pasta, um arquivo ou qualquer outro componente que seja acessado/aberto por meio de clique(s), a função para executar essa tarefa é `pyautogui.click()`.

Nessa função, vamos utilizar 4 parâmetros principais: `x`, `y`, `clicks` e `interval`

`x` e `y` -> coordenadas do ponto na tela em que será executado o clique com botão esquerdo

`clicks` -> número de cliques que serão executados

`interval` -> intervalo de tempo (em segundos) entre cada execução de um clique

Os parâmetros `x` e `y` já foram abordados bastante até aqui, então vamos falar sobre os outros dois.

O parâmetro `clicks` é responsável por informar para a função quantos cliques devem ser executados em sequência. Desta forma, podemos realizar um clique simples, clique duplo, clique triplo etc. Ou seja, a função `pyautogui.click()` nos fornece recurso para lidarmos com qualquer situação de clique, desde clicar precisar clicar apenas uma vez para acessar algum elemento, até precisar clicar diversas vezes.

Já o parâmetro `interval` é responsável por informar quantos segundos devem existir de intervalo entre um clique e outro.

Portanto, podemos abrir arquivos que necessitam de dois cliques, como atalhos que estão na área de trabalho, podemos acessar aplicativos que estão na barra de tarefas com apenas um clique, podemos interagir com elmentos que necessitam de múltiplos cliques, como em uma seleção de uma determinada área em uma imagem.

Exemplo de clique simples:

```
pyautogui.click(x=470, y=855, clicks=1, interval=0)
```

Exemplo de clique duplo:

```
pyautogui.click(x=470, y=855, clicks=2, interval=0)
```

Exemplo de clique múltiplo:

```
pyautogui.click(x=470, y=855, clicks=5 interval=0)
```

Observação: os parâmetros `clicks` e `interval` possuem como valores padrões 1 e 0, respectivamente.

Curisosidade: o PyAutoGUI possui funções próprias para os casos de clique duplo e clique triplo. Caso preferir, em vez de utilizar o parâmetro `clicks`, em casos de duplo clique ou triplo clque, utilize `pyautogui.doubleClick()` ou `pyautogui.tripleClick()`.

Clique com botão direito

É a função que executa um clique com o botão direito do mouse. Desta forma, sempre que a intenção do programa for clicar com o botão direito em um ícone da barra de tarefas, no menu iniciar, uma aba do navegador, um link, um botão, uma pasta, um arquivo ou qualquer outro componente em que deseja-se ver suas propriedades por meio de um clique com o botão direito, a função para executar essa tarefa é `pyautogui.rightClick()`.

Os parâmetros utilizados nessa função são os mesmos utilizados na função do clique com o botão esquerdo, exceto o atributo `clicks` que não existe para essa função.

Exemplo de clique com o botão direito:

```
pyautogui.click(x=500, y=500, interval=0)
```

Nesse exemplo, o elemento localizado em 500, 500 será clicado com o botão direito do mouse.

Clique com o botão do meio

É a função que executa um clique com o botão do meio do mouse. Desta forma, sempre que a intenção do programa for clicar com o botão do meio para realizar desempenhar alguma funcionalidade específica, como fechar uma aba do navegador e abrir o modo nagevação em uma página, a função para executar essa tarefa é `pyautogui.middleClick()`.

Os parâmetros utilizados nessa função são os mesmos utilizados na função do clique com o botão direito.

Exemplo de clique com o botão do meio:

```
pyautogui.middleClick(x=140, y=20, interval=0)
```

08. Scroll com o mouse

Para finalizar o escopo de funções do mouse, vamos falar sobre o scroll em páginas que possuem mais conteúdo do que a tela consegue mostrar em um único frame.

O scroll representa o deslocamento vertical realizado em ambientes que possuem um contéudo que ultrapassa as margens verticais da tela. Desta forma, quando for informado um valor de scroll e um conjunto de coordenadas, o cursor do mouse realizará o scroll exatamente na linha vertical desse conjunto de coordenadas.

A função `pyautogui.scroll()` representa este tipo de deslocamento. Nessa função, iremos utilizar três parâmetros:

`clicks`-> quantidade de scrolls exectuados.

`x` e `y` -> coordenadas da região em que será executado o scroll.

No parâmetro `clicks`, a quantidade de scrolls é referente ao número de vezes que o botão (ou a "bolinha") do meio do mouse será girado. Além disso, o sentido de scroll para baixo deve ser informado como um número negativo, e o sentido de scroll para cima é positivo.

Abaixo, um exemplo de aplicação:

```
pyautogui.scroll(clicks=-5, x=500, y=150)
```

No caso acima, a página terá um scroll de 5 unidades para baixo.

Outro exemplo de aplicação:

```
pyautogui.scroll(10, 500, 150)
```

No caso acima, a página terá um scroll de 10 unidades para cima.

Note que o mouse terá seu cursor movido para as coordenadas passadas como atributos da função, e nesse ponto da tela é onde será executado o scroll. # 09. Como selecionar e localizar uma imagem na tela

Nesse tópico, iremos falar sobre uma das funcionalidades mais úteis do PyAutoGUI, que é como localizar uma imagem de interesse na tela. E essa funcionalidade é muito útil pois, como vimos anteriormente nos tópicos de interação com o mouse, nem sempre as coordenadas dos elementos que queremos acessar serão o suficiente para atingirmos nossos objetivos. Devido às limitações para lidar com a troca de posição de componentes na tela, já comentadas nos tópicos anteriores, a utilização de imagens para acessar componentes será fundamental para conseguirmos rodar nossos scripts com maestria.

Como copiar uma seção da tela

Dito isso, a primeira função que iremos ver é a `pyautogui.screenshot()`, que, como o próprio nome já sugere, é uma função para realizar um screen shot (captura da tela), ou um *print screen*.

Geralmente, realizamos uma captura de tela a partir da tela "Prt sc" ou "Print Screen" dos nossos teclados. Essa, de fato, é a forma mais prática, mas o PyAutoGUI nos fornece a função de screenshot para podermos realizar captura de telas de forma automática e em maiores quantidades.

Abaixo, um exemplo de utilização da função:

```
pyautogui.screenshot("captura_tela.png")
```

No caso acima, o PyAutoGUI irá realizar a captura da tela e salvá-la com o nome "captura_tela" na pasta do seu código.

Localização de uma imagem

Agora que já vimos como capturar uma imagem, vamos ver como localizamos uma imagem na tela. Essa função procura por uma imagem passada como atributo, sendo que no atributo deve ser passado o path do arquivo da imagem a ser localizada na tela.

A função para localizar uma imagem é `pyautogui.locateOnScreen()` e o que ela faz é identificar a imagem buscada na tela e, caso esse imagem seja encontrada, essa função retorna as coordenadas X e Y, largura e altura da imagem. Nessa função, as coordenadas de X e Y retornadas são referentes à borda da imagem.

Exemplo de utilização da função

```
pyautogui.locateOnScreen('imgs/imagem_patinho.png')
```

No caso acima, será buscada uma imagem chamada 'imagem_patinho' do tipo *.png* que está salva dentro de uma pasta 'imgs'

Existem também a função `pyautogui.locateCenterOnScreen()`, que a diferença é que essa função retorna as coordenadas do centro da imagem.

Então, a partir do conhecimento dessa função, agora podemos combiná-la com a função de mover o mouse para a coordenada específica de uma imagem, caso essa imagem esteja presente na tela:

```
local_imagem = pyautogui.locateCenterOnScreen('imgs/imagem_patinho.png')
pyautogui.moveTo(local_imagem)
```

No exemplo acima, o programa irá buscar a imagem na tela e mover o mouse para cima dessa imagem. Caso essa imagem fosse de um ícone interativo, por exemplo, poderíamos ainda passar a função de clicar, pois o cursor do mouse já estaria em cima do ícone.

Caso tenha interesse ou necessidade de identificar o tamanho da sua tela, basta utilizar a função `pyautogui.size()`.

Facilitando a extração de imagens

Para complementar a extração de imagens da tela, especialmente para casos em que queremos realizar apenas algumas capturas de tela, sem precisar fazer isso de forma automatizada, vamos abordar uma ferramenta que pode facilitar bastante esse procedimento, essa ferramenta é a Flameshot.

Flameshot é uma ferramenta que permite que possamos realizar captura de tela com especificidade. Após apertar a tecla de "Print Screen", o Flameshot nos fornece opções de customização da captura da tela, como incluir texto, inserir figuras, selecionar uma região específica para ser capturada, entre outras.

Para o nosso objetivo, o grande valor dessa ferramenta está na facilidade que ela fornece para selecionarmos apenas regiões específicas da tela para serem salvas de foto, como uma captura da tela. Basicamente é como se fossemos realizar a captura da tela só do que nos interessa. E isso é muito útil, especialmente quando queremos selecionar a imagem de algum componente que precisamos interagir, por meio das funções de localização de imagem, pois conseguimos salvar somente a imagem do componente, por exemplo.

Se tiver interesse em verificar como a ferramenta funciona ou utilizá-la, clique no link abaixo que é do site oficial do Flameshot.

https://flameshot.org/

10. Melhorando a identificação das imagens com OpenCV

Nem sempre uma imagem será encontrada na tela, mesmo que ela, de fato, exista na tela. Isso acontece porque o PyAutoGUI se baseia nos pixels da imagem, e às vezes a imagem pode não possuir uma nitidez muito boa, ou a tela pode conter muitos outros elementos que pode acabar atrapalhando o reconhecimento da imagem.

Para melhorar significativamente as buscas pelas imagens, aumentando a probabilidade de serem encontradas, podemos utilizar a biblioteca OpenCV.

A biblioteca OpenCV (Open Source Computer Vision) é uma poderosa utilizada para visão computacional e processamento de imagem. Desenvolvida para oferecer um conjunto abrangente de funções destinadas a resolver desafios relacionados à análise e interpretação de imagens, o OpenCV tornou-se um pilar na comunidade de aprendizado de máquina e visão computacional. Suas capacidades incluem detecção de objetos, rastreamento de movimento, reconhecimento facial, calibração de câmera e manipulação de imagens em tempo real, mas aqui no curso de PyAutoGUI iremos utilizar apenas para detecção de objetos (imagens, no nosso caso).

A forma que iremos utilizar essa biblioteca é extremamente simples, pois iremos apenas acrescentar um parâmetro na função de buscar imagem, chamado `confidence`. Esse parâmetro pode receber como atributo valores de 0 a 0.999, sendo que 0 representa o mínimo de confiança e 0.999 representa o máximo de confiança. O nível de confiança de indica o quão bem a imagem de referência precisa corresponder à imagem na tela para ser considerada uma correspondência válida. Geralmente o deixamos o valor de `confidence` no máximo pois queremos que a correspondência seja precisa e exata.

Sendo assim, o uso desse novo parâmetro na função locateOnScreen() fica conforme demonstrado abaixo:

```
pyautogui.locateOnScreen(imagem, confidence=0.999)
```

Desta forma, o nível de precisão na identificação da imagem na tela agora aumentou significativamente, pois o PyAutoGUI irá buscar somente imagens iguais, diminuindo muito as chances de ele errar e localizar outra imagem ou então não encontrar a imagem. # 11. Tratamento de exceções - Interrompendo o programa

A essa altura já sabemos que o PyAutoGUI é uma biblioteca que controla nosso mouse e teclado, podendo assim desempenhar tarefas que qualquer pessoa que estivesse comandando esses acessórios faria. Quando rodamos um script que utiliza o PyAutoGUI, damos inicio a uma série de eventos sequênciais que ocorrem um após ao outros, para que o objetivo final do script seja atingido. Mas imagine se, por algum erro, a biblioteca identifique um componente errado, ou então a página em que o script está interagindo seja carregada de uma forma diferente ou mais lenta que o normal, isso iria implicar em uma série de erros a partir do ponto em que houve esse erro.

Por exemplo, digamos que temos um script que seu objetivo é realizar o login em uma conta do Gmail. O primeiro passo a ser feito é entrar no navegador, depois procurar na barra de pesquisa por “Gmail”, depois acessar o ícone da busca correto, depois acessar a área de login, depois preencher os campos de email e senha e, por fim, realizar o login. Agora imagine nesse mesmo caso que, ao tentar acessa o ícone buscado na pesquisa por “Gmail”, o PyAutoGUI tenha acessado um componente diferente do que era pra ser acessado e, consequentemente, acabou entrando em um site diferente do Gmail. A partir desse ponto, o restante do script perdeu totalmente seu significado e propósito, pois ele não está mais navegando na página do Gmail e, consequentemente, não conseguirá realizar o login.

O ponto mais problemático de acontecer erros durante a execução do script em PyAutoGUI, é o fato de ser um script contínuo, ou seja, ele continuará rodando mesmo que algum passo não tenha sido executado conforme esperado até chegar ao final do código escrito. Mas, felizmente, o PyAutoGUI nos fornece um método para tratar esses casos de erros, um método que interrompe instantâneamente o programa, no momento em que ele estiver, a partir de um movimento com o mouse em direção ao canto superior esquerdo da tela, esse método é o `pyautogui.FailSafeException`.

Quando inserimos o comando `pyautogui.FailSafeException` em um script de PyAutoGUI, habilitamos essa funcionalidade de tratamento de exceções, de forma que, caso precisarmos pausar o script, encerrando-o no ponto em que ele estiver sendo executado, será possível fazer isso simplesmente levando o cursor do mouse para o canto superior esquerdo da tela. Esse é um movimento default desse método que nos fornece mais segurança ao executarmos códigos que utilizem PyAutoGUI.

Portanto, basta inserir a linha de código abaixo no início do seu script, que você terá uma forma de encerrar a execução do programa instantâneamente.

```
pyautogui.FailSafeException
```

12. Avisos interativos na tela e inserção de campo de texto

Por fim, vamos falar sobre as funções interativas que o PyAutoGUI disponibiliza para implementarmos em nossos scripts. Essas funções servem para interagirmos com o código, de forma que podemos passar informações para ele enquanto é executado.

Inserção dinâmica de texto

O PyAutoGUI tem uma função chamada `pyautogui.prompt()` que tem como objetivo fornecer uma caixa de texto para que, na parte do script em que for inserida, irá aparecer na tela permitindo que o usuário insira um texto. Além disso, existem dois botões, "OK" e "Cancel". Clicando no botão "OK" o script seguirá para a próxima linha de código, e clicando no botão "Cancel" o script é encerrado.

Essa função é estruturada da seguinte forma:

```
`pyautogui.prompt(text='Texto informando o que deve ser preenchido na caixa de texto',
↪  title='Título da caixa de texto')
```

[](prompt_padrao.png]

O valor preenchido no prompt pode ser salvo em uma variável para ser usado posteriormente. Assim, o que o usuário responder dentro do campo texto pode ser utilizado a partir da variável que vai armazenar a resposta. Veremos um exemplo mais prático no final deste tópico.

Inserção de avisos na tela

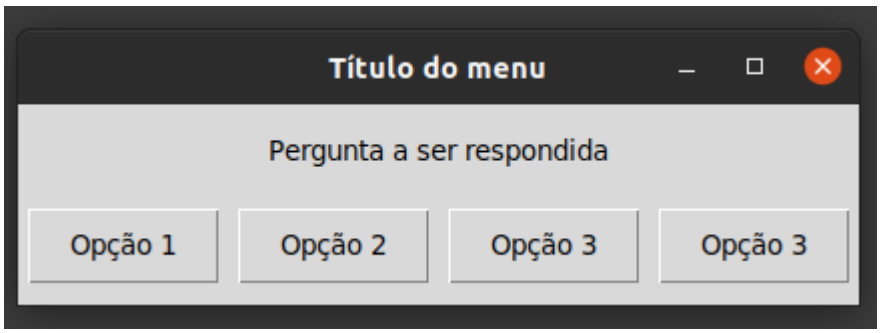
Além do prompt, existem também duas outras funções que mostram caixas interativas durante a execução de um script: `pyautogui.confirm()` e `pyautogui.alert()`

A função `pyautogui.confirm()` fornece uma espécie de menu interativo, de forma que podemos informar um tipo de pergunta enquanto o script é executado para que o usuário selecione uma opção dentre as opções disponibilizadas. Cada opção é um botão, e, da mesma forma que no prompt, a opção que o usuário escolher pode ser armazenada dentro de uma variável para que possa ser utilizada na sequência do código.

Os botões devem ser passados no formato de lista para o parâmetro `buttons`.

A função é estruturada da seguinte forma:

```
pyautogui.confirm(text="Pergunta a ser respondida", title="Título do menu", buttons=['Opção
↪  1', 'Opção 2', 'Opção 3', 'Opção 3'])
```

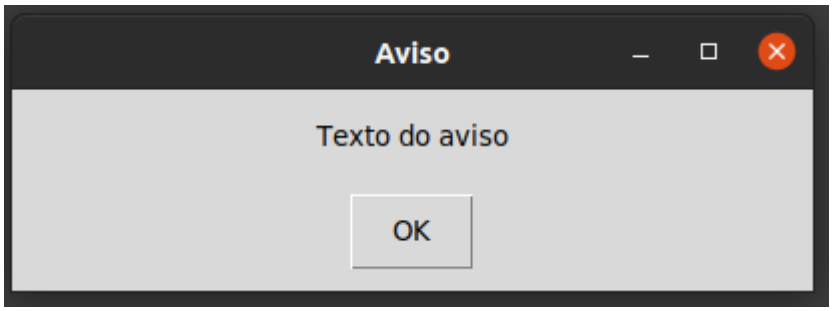


Essa função pode ser bem útil em casos em que o próximo passo do programa possua mais de uma opção que possa ser tomada e, a partir da opção que o usuário selecionar, o programa toma ação baseada nesta resposta.

Já a função `pyautogui.alert()` apenas mostra um aviso na tela, como forma de sinalizar alguma informação relevante na etapa do código em que for inserida. Junto com a mensagem de aviso, tem um botão que após ser clicado da continuidade à execução do script.

Desta forma:

```
pyautogui.alert(text='Texto do aviso', title='Aviso', button='OK')
```



Exemplo prático interativo

Por fim, vamos montar um exemplo em que as três funções interativas do PyAutoGUI são utilizadas em conjunto, conforme no código abaixo:

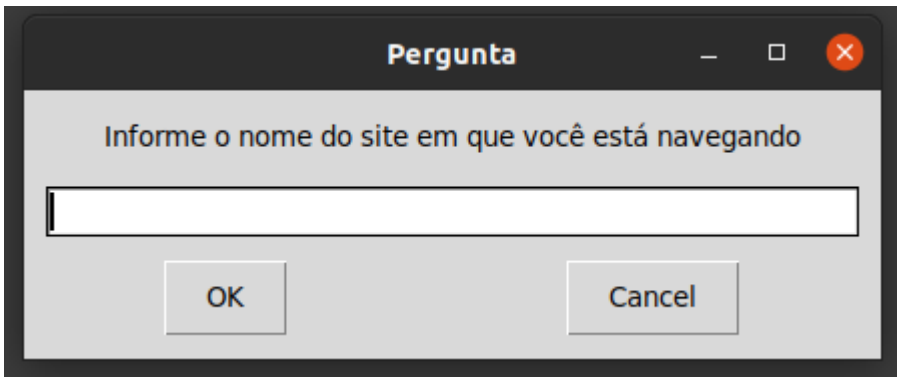
```
nome_site = pyautogui.prompt(text='Informe o nome do site em que você está navegando',
↪  title='Pergunta')

resposta_aba = pyautogui.confirm(text=f"Qual aba você quer acessar no site  {nome_site}",
↪  title="Menu interativo", buttons=['Esportes', 'Notícias', 'Horóscopo', 'Educação'])

pyautogui.alert(f"O código será redirecionado para a aba {resposta_aba}")
```

Perceba no código acima que as funções `pyautogui.prompt()` e `pyautogui.confirm()` estão inseridas dentro das variáveis "*nome_site*" e "*resposta_aba*" respectivamente. Desta forma,

as respostas que o usuário fornecer para essas funções, quando elas forem mostradas na tela, ficarão armazenadas nessas variáveis, de forma que o código consiga utilizar o valor respondido pelo usuário (que é o que é exatamente feito na função `pyautogui.confirm()` quando utiliza a variável "*nome_site*" dentro dela, e o mesmo é feito na `pyautogui.alert()` ao utilizar o valor da variável "*resposta_aba*")

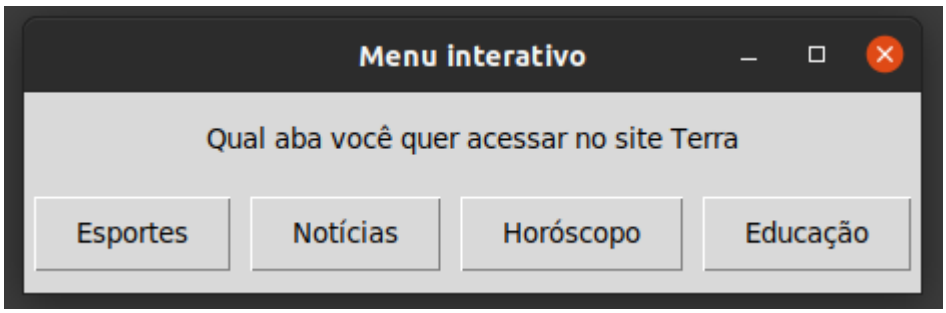
Dito isso, vamos considerar para o exemplo acima, que o script que utiliza PyAutoGUI está sendo executado no site https://www.terra.com.br/, que atualmente possui uma página inicial com o cabeçalho abaixo:

Ao iniciar a execução do script, a primeira caixa de texto que será exibida é a da imagem abaixo:

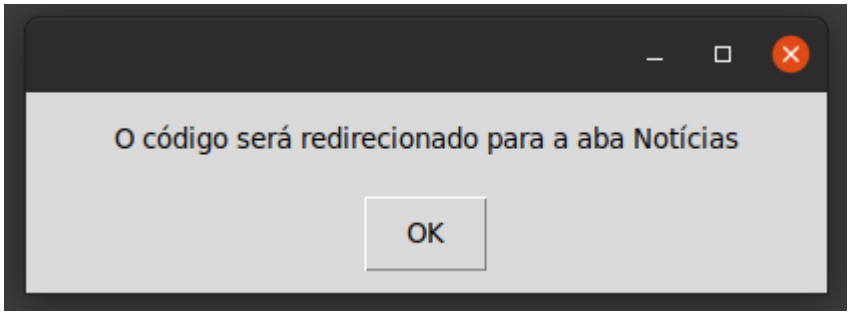


Como, no nosso exemplo, o script está sendo executado no site Terra, a resposta para o prompt acima não poderia ser outra senão "Terra".

Após responder a caixa de texto e clicar em "OK", o código segue para a próxima linha, que mostra a seguinte seleção na tela:



Considerando que o usuário selecionou a opção “Notícias”, o código automaticamente da continuidade à execução do código, que, por fim, mostra a última caixa na tela:



E após clicar em “OK”, o programa continua sendo executado normalmente a partir da próxima linha de código. Nesse caso, o mais conveniente seria inserir uma ação, utilizando as funções do PyAutoGUI, para selecionar a aba de notícias do site Terra e clicar nela, para que a página seja redirecionada para a aba de notícias. Mas essa parte nós já estamos craque em fazer, pois vimos muitos exemplos anteriormente de como fazer de diferentes formas.

Espero que tenham gostado e que este material tenha sido útil!

Um abraço e até a próxima!