Cheat Sheet

This is a quickstart reference to using PyAutoGUI. PyAutoGUI is cross-platform GUI automation module that works on Python 2 & 3. You can control the mouse and keyboard as well as perform basic image recognition to automate tasks on your computer.

All the keyword arguments in the examples on this page are optional.

```
>>> import pyautogui
```

PyAutoGUI works on Windows/Mac/Linux and on Python 2 & 3. Install from PyPI with `pip install pyautogui` .

General Functions

```
>>> pyautogui.position()  # current mouse x and y
(968, 56)
>>> pyautogui.size()  # current screen resolution width and height
(1920, 1080)
>>> pyautogui.onScreen(x, y)  # True if x & y are within the screen.
True
```

Fail-Safes

Set up a 2.5 second pause after each PyAutoGUI call:

```
>>> import pyautogui
>>> pyautogui.PAUSE = 2.5
```

When fail-safe mode is `True` , moving the mouse to the upper-left will raise a `pyautogui.FailSafeException` that can abort your program:

```
>>> import pyautogui
>>> pyautogui.FAILSAFE = True
```

Mouse Functions

XY coordinates have 0, 0 origin at top left corner of the screen. X increases going right, Y increases going down.

```
>>> pyautogui.moveTo(x, y, duration=num_seconds)  # move mouse to XY coordinates over num_second seconds
>>> pyautogui.moveRel(xOffset, yOffset, duration=num_seconds)  # move mouse relative to its current pos
```

If `duration` is 0 or unspecified, movement is immediate. Note: dragging on Mac can't be immediate.

```
>>> pyautogui.dragTo(x, y, duration=num_seconds)  # drag mouse to XY
>>> pyautogui.dragRel(xOffset, yOffset, duration=num_seconds)  # drag mouse relative to its current pos
```

Calling `click()` just clicks the mouse once with the left button at the mouse's current location, but the keyword arguments can change that:

```
>>> pyautogui.click(x=moveToX, y=moveToY, clicks=num_of_clicks, interval=secs_between_clicks, button='l
```

The `button` keyword argument can be `'left'` , `'middle'` , or `'right'` .

All clicks can be done with `click()` , but these functions exist for readability. Keyword args are optional:

```
>>> pyautogui.rightClick(x=moveToX, y=moveToY)
>>> pyautogui.middleClick(x=moveToX, y=moveToY)
>>> pyautogui.doubleClick(x=moveToX, y=moveToY)
>>> pyautogui.tripleClick(x=moveToX, y=moveToY)
```

Positive scrolling will scroll up, negative scrolling will scroll down:

```
>>> pyautogui.scroll(amount_to_scroll, x=moveToX, y=moveToY)
```

Individual button down and up events can be called separately:

```
>>> pyautogui.mouseDown(x=moveToX, y=moveToY, button='left')
>>> pyautogui.mouseUp(x=moveToX, y=moveToY, button='left')
```

Keyboard Functions

Key presses go to wherever the keyboard cursor is at function-calling time.

```
>>> pyautogui.typewrite('Hello world!\n', interval=secs_between_keys)  # useful for entering text, newli
```

A list of key names can be passed too:

```
>>> pyautogui.typewrite(['a', 'b', 'c', 'left', 'backspace', 'enter', 'f1'], interval=secs_between_keys
```

The full list of key names is in `pyautogui.KEYBOARD_KEYS` .

Keyboard hotkeys like Ctrl-S or Ctrl-Shift-1 can be done by passing a list of key names to `hotkey()` :

```
>>> pyautogui.hotkey('ctrl', 'c')  # ctrl-c to copy
>>> pyautogui.hotkey('ctrl', 'v')  # ctrl-v to paste
```

Individual button down and up events can be called separately:

```
>>> pyautogui.keyDown(key_name)
>>> pyautogui.keyUp(key_name)
```

Message Box Functions

If you need to pause the program until the user clicks OK on something, or want to display some information to the user, the message box functions have similar names that JavaScript has:

```
>>> pyautogui.alert('This displays some text with an OK button.')
>>> pyautogui.confirm('This displays text and has an OK and Cancel button.')
'OK'
>>> pyautogui.prompt('This lets the user type in a string and press OK.')
'This is what I typed in.'
```

The `prompt()` function will return `None` if the user clicked Cancel.

Screenshot Functions

PyAutoGUI uses Pillow/PIL for its image-related data.

On Linux, you must run `sudo apt-get install scrot` to use the screenshot features.

```
>>> pyautogui.screenshot()  # returns a Pillow/PIL Image object
<PIL.Image.Image image mode=RGB size=1920x1080 at 0x24C3EF0>
>>> pyautogui.screenshot('foo.png')  # returns a Pillow/PIL Image object, and saves it to a file
<PIL.Image.Image image mode=RGB size=1920x1080 at 0x31AA198>
```

If you have an image file of something you want to click on, you can find it on the screen with `locateOnScreen()` .

```
>>> pyautogui.locateOnScreen('looksLikeThis.png')  # returns (left, top, width, height) of first place t
(863, 417, 70, 13)
```

The `locateAllOnScreen()` function will return a generator for all the locations it is found on the screen:

```
>>> for i in pyautogui.locateAllOnScreen('looksLikeThis.png')
...
...
(863, 117, 70, 13)
(623, 137, 70, 13)
(853, 577, 70, 13)
(883, 617, 70, 13)
(973, 657, 70, 13)
(933, 877, 70, 13)
```

```
>>> list(pyautogui.locateAllOnScreen('looksLikeThis.png'))
[(863, 117, 70, 13), (623, 137, 70, 13), (853, 577, 70, 13), (883, 617, 70, 13), (973, 657, 70, 13), (9
```

The `locateCenterOnScreen()` function just returns the XY coordinates of the middle of where the image is found on the screen:

```
>>> pyautogui.locateCenterOnScreen('looksLikeThis.png')  # returns center x and y
(898, 423)
```

These functions return `None` if the image couldn't be found on the screen.

Note: The locate functions are slow and can take a full second or two.