PyAutoGUI Documentation

Release 1.0.0

Al Sweigart

Aug 07, 2017

Contents

Cross-platform GUI automation for human beings.

PyAutoGUI is a Python module for programmatically controlling the mouse and keyboard.

PyAutoGUI can be installed from the `pip` tool or downloaded from PyPI: https://pypi.python.org/pypi/PyAutoGUI

The source is available on: https://github.com/asweigart/pyautogui

Contents:

CHAPTER 1

Introduction

Purpose

The purpose of PyAutoGUI is to provide a cross-platform Python module for GUI automation *for human beings*. The API is designed to be as simple as possible with sensible defaults.

For example, here is the complete code to move the mouse to the middle of the screen on Windows, OS X, and Linux:

```
>>> import pyautogui
>>> screenWidth, screenHeight = pyautogui.size()
>>> pyautogui.moveTo(screenWidth / 2, screenHeight / 2)
```

And that is all.

PyAutoGUI can simulate moving the mouse, clicking the mouse, dragging with the mouse, pressing keys, pressing and holding keys, and pressing keyboard hotkey combinations.

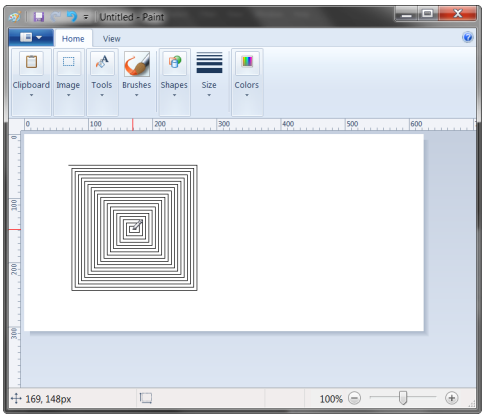
Examples

```
>>> import pyautogui
>>> screenWidth, screenHeight = pyautogui.size()
>>> currentMouseX, currentMouseY = pyautogui.position()
>>> pyautogui.moveTo(100, 150)
>>> pyautogui.click()
>>> pyautogui.moveRel(None, 10)  # move mouse 10 pixels down
>>> pyautogui.doubleClick()
>>> pyautogui.moveTo(500, 500, duration=2, tween=pyautogui.easeInOutQuad)  # use␣
↪tweening/easing function to move mouse over 2 seconds.
>>> pyautogui.typewrite('Hello world!', interval=0.25)  # type with quarter-second␣
↪pause in between each key
>>> pyautogui.press('esc')
>>> pyautogui.keyDown('shift')
>>> pyautogui.press(['left', 'left', 'left', 'left', 'left', 'left'])
```

```
>>> pyautogui.keyUp('shift')
>>> pyautogui.hotkey('ctrl', 'c')
```

This example drags the mouse in a square spiral shape in MS Paint (or any graphics drawing program):

```
>>> distance = 200
>>> while distance > 0:
        pyautogui.dragRel(distance, 0, duration=0.5)   # move right
        distance -= 5
        pyautogui.dragRel(0, distance, duration=0.5)   # move down
        pyautogui.dragRel(-distance, 0, duration=0.5)  # move left
        distance -= 5
        pyautogui.dragRel(0, -distance, duration=0.5)  # move up
```



Dependencies

On Windows, PyAutoGUI has no dependencies (other than Pillow and some other modules, which are installed by pip along with PyAutoGUI). It does **not** need the `pywin32` module installed since it uses Python's own `ctypes` module.

On OS X, PyAutoGUI requires PyObjC installed for the AppKit and Quartz modules. The module names on PyPI to install are `pyobjc-core` and `pyobjc` (in that order).

On Linux, PyAutoGUI requires `python-xlib` (for Python 2) or `python3-Xlib` (for Python 3) module installed.

Fail-Safes

Like the enchanted brooms from the Sorcerer's Apprentice programmed to keep filling (and then overfilling) the bath with water, your program could get out of control (even though it is following your instructions) and need to be stopped. This can be difficult to do if the mouse is moving around on its own, preventing you from clicking on the program's window to close it down.

As a safety feature, a fail-safe feature is enabled by default. When `pyautogui.FAILSAFE = True` PyAutoGUI functions will raise a `pyautogui.FailSafeException` if the mouse cursor is in the upper left corner of the screen. If you lose control and need to stop the current PyAutoGUI function, keep moving the mouse cursor up and to the left. To disable this feature, set `FAILSAFE` to `False`:

```
>>> import pyautogui
>>> pyautogui.FAILSAFE = False # disables the fail-safe
```

You can add delays after all of PyAutoGUI's functions by setting the `pyautogui.PAUSE` variable to a float or integer value of the number of seconds to pause. By default, the pause is set to 0.1 seconds. This can be helpful when interacting with other applications so that PyAutoGUI doesn't move too fast for them. For example:

```
>>> import pyautogui
>>> pyautogui.PAUSE = 2.5
>>> pyautogui.moveTo(100, 100); pyautogui.click()    # there will be a two and a half␣
↪second pause after moving and another after the click
```

All PyAutoGUI functions will block until they complete. (It is on the roadmap to add an optional non-blocking way to call these functions.)

It is advised to use FAILSAFE along with setting PAUSE.

CHAPTER 2

Installation

To install PyAutoGUI, install the `pyautogui` package from PyPI and dependencies.

On Windows, this is:

```
C:\Python34\pip.exe install pyautogui
```

(Though you may have a different version of Python installed other than 3.4)

On OS X, this is:

```
pip3 install pyobjc-core
pip3 install pyobjc
pip3 install pyautogui
```

If you are running El Capitan and have problems installing pyobjc try:

```
MACOSX_DEPLOYMENT_TARGET=10.11 pip install pyobjc
```

On Linux, this is:

```
pip3 install python3-xlib
sudo apt-get install scrot
sudo apt-get install python3-tk
sudo apt-get install python3-dev
pip3 install pyautogui
```

PyAutoGUI will try to install Pillow (for its screenshot capabilities). This happens when pip installs PyAutoGUI.

CHAPTER 3

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 position
```

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 position
```

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='left')
```

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, newline is Enter
```

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 it is found
(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), (933, 877, 70, 13)]
```

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.

CHAPTER 4

General Functions

position() - Returns tuple of integers: (x, y) for the current position of the mouse cursor.

size() - Returns tuple of integers: (width, height) for size of the main monitor. TODO - add multi monitor support.

CHAPTER 5

Mouse Control Functions

The Screen and Mouse Position

Locations on your screen are referred to by X and Y Cartesian coordinates. The X coordinate starts at 0 on the left side and increases going right. Unlike in mathematics, the Y coordinate starts at 0 on the top and increases going down.

```
0,0       X increases -->
+---------------------------+
|                           | Y increases
|                           |     |
|   1920 x 1080 screen      |     |
|                           |     V
|                           |
|                           |
+---------------------------+ 1919, 1079
```

The pixel at the top-left corner is at coordinates 0, 0. If your screen's resolution is 1920 x 1080, the pixel in the lower right corner will be 1919, 1079 (since the coordinates begin at 0, not 1).

The screen resolution size is returned by the `size()` function as a tuple of two integers. The current X and Y coordinates of the mouse cursor are returned by the `position()` function.

For example:

```
>>> pyautogui.size()
(1920, 1080)
>>> pyautogui.position()
(187, 567)
```

Here is a short Python 3 program that will constantly print out the position of the mouse cursor:

```
#! python3
import pyautogui, sys
print('Press Ctrl-C to quit.')
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')
```

Here is the Python 2 version:

```
#! python
import pyautogui, sys
print('Press Ctrl-C to quit.')
try:
    while True:
        x, y = pyautogui.position()
        positionStr = 'X: ' + str(x).rjust(4) + ' Y: ' + str(y).rjust(4)
        print positionStr,
        print '\b' * (len(positionStr) + 2),
        sys.stdout.flush()
except KeyboardInterrupt:
    print '\n'
```

To check if XY coordinates are on the screen, pass them (either as two integer arguments or a single tuple/list arguments with two integers) to the `onScreen()` function, which will return `True` if they are within the screen's boundaries and `False` if not. For example:

```
>>> pyautogui.onScreen(0, 0)
True
>>> pyautogui.onScreen(0, -1)
False
>>> pyautogui.onScreen(0, 99999999)
False
>>> pyautogui.size()
(1920, 1080)
>>> pyautogui.onScreen(1920, 1080)
False
>>> pyautogui.onScreen(1919, 1079)
True
```

Mouse Movement

The `moveTo()` function will move the mouse cursor to the X and Y integer coordinates you pass it. The `None` value can be passed for a coordinate to mean "the current mouse cursor position". For example:

```
>>> pyautogui.moveTo(100, 200)    # moves mouse to X of 100, Y of 200.
>>> pyautogui.moveTo(None, 500)   # moves mouse to X of 100, Y of 500.
>>> pyautogui.moveTo(600, None)   # moves mouse to X of 600, Y of 500.
```

Normally the mouse cursor will instantly move to the new coordinates. If you want the mouse to gradually move to the new location, pass a third argument for the duration (in seconds) the movement should take. For example:

```
>>> pyautogui.moveTo(100, 200, 2)    # moves mouse to X of 100, Y of 200 over 2 seconds
```

(If the duration is less than `pyautogui.MINIMUM_DURATION` the movement will be instant. By default, `pyautogui.MINIMUM_DURATION` is 0.1.)

If you want to move the mouse cursor over a few pixels *relative* to its current position, use the `moveRel()` function. This function has similar parameters as `moveTo()`. For example:

```
>>> pyautogui.moveTo(100, 200)    # moves mouse to X of 100, Y of 200.
>>> pyautogui.moveRel(0, 50)      # move the mouse down 50 pixels.
>>> pyautogui.moveRel(-30, 0)      # move the mouse left 30 pixels.
>>> pyautogui.moveRel(-30, None)   # move the mouse left 30 pixels.
```

Mouse Drags

PyAutoGUI's `dragTo()` and `dragRel()` functions have similar parameters as the `moveTo()` and `moveRel()` functions. In addition, they have a `button` keyword which can be set to `'left'`, `'middle'`, and `'right'` for which mouse button to hold down while dragging. For example:

```
>>> pyautogui.dragTo(100, 200, button='left')     # drag mouse to X of 100, Y of 200␣
↪while holding down left mouse button
>>> pyautogui.dragTo(300, 400, 2, button='left')  # drag mouse to X of 300, Y of 400␣
↪over 2 seconds while holding down left mouse button
>>> pyautogui.dragRel(30, 0, 2, button='right')    # drag the mouse left 30 pixels␣
↪over 2 seconds while holding down the right mouse button
```

Tween / Easing Functions

Tweening is an extra feature to make the mouse movements fancy. You can probably skip this section if you don't care about this.

A tween or easing function dictates the progress of the mouse as it moves to its destination. Normally when moving the mouse over a duration of time, the mouse moves directly towards the destination in a straight line at a constant speed. This is known as a *linear tween* or *linear easing* function.

PyAutoGUI has other tweening functions available in the `pyautogui` module. The `pyautogui.easeInQuad` function can be passed for the 4th argument to `moveTo()`, `moveRel()`, `dragTo()`, and `dragRel()` functions to have the mouse cursor start off moving slowly and then speeding up towards the destination. The total duration is still the same as the argument passed to the function. The `pyautogui.easeOutQuad` is the reverse: the mouse cursor starts moving fast but slows down as it approaches the destination. The `pyautogui.easeOutElastic` will overshoot the destination and "rubber band" back and forth until it settles at the destination.

For example:

```
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInQuad)      # start slow, end fast
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeOutQuad)     # start fast, end slow
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInOutQuad)   # start and end fast,␣
↪slow in middle
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInBounce)    # bounce at the end
>>> pyautogui.moveTo(100, 100, 2, pyautogui.easeInElastic)   # rubber band at the end
```

These tweening functions are copied from Al Sweigart's PyTweening module: https://pypi.python.org/pypi/PyTweening https://github.com/asweigart/pytweening This module does not have to be installed to use the tweening functions.

If you want to create your own tweening function, define a function that takes a single float argument between `0.0` (representing the start of the mouse travelling) and `1.0` (representing the end of the mouse travelling) and returns a float value between `0.0` and `1.0`.

Mouse Clicks

The `click()` function simulates a single, left-button mouse click at the mouse's current position. A "click" is defined as pushing the button down and then releasing it up. For example:

```
>>> pyautogui.click()  # click the mouse
```

To combine a `moveTo()` call before the click, pass integers for the `x` and `y` keyword argument:

```
>>> pyautogui.click(x=100, y=200)  # move to 100, 200, then click the left mouse␣
↪button.
```

To specify a different mouse button to click, pass `'left'`, `'middle'`, or `'right'``for the ``button` keyword argument:

```
>>> pyautogui.click(button='right')  # right-click the mouse
```

To do multiple clicks, pass an integer to the `clicks` keyword argument. Optionally, you can pass a float or integer to the `interval` keyword argument to specify the amount of pause between the clicks in seconds. For example:

```
>>> pyautogui.click(clicks=2)  # double-click the left mouse button
>>> pyautogui.click(clicks=2, interval=0.25)  # double-click the left mouse button,␣
↪but with a quarter second pause in between clicks
>>> pyautogui.click(button='right', clicks=3, interval=0.25)  ## triple-click the␣
↪right mouse button with a quarter second pause in between clicks
```

As a convenient shortcut, the `doubleClick()` function will perform a double click of the left mouse button. It also has the optional `x`, `y`, `interval`, and `button` keyword arguments. For example:

```
>>> pyautogui.doubleClick()  # perform a left-button double click
```

There is also a `tripleClick()` function with similar optional keyword arguments.

The `rightClick()` function has optional `x` and `y` keyword arguments.

The mouseDown() and mouseUp() Functions

Mouse clicks and drags are composed of both pressing the mouse button down and releasing it back up. If you want to perform these actions separately, call the `mouseDown()` and `mouseUp()` functions. They have the same `x`, `y`, and `button`. For example:

```
>>> pyautogui.mouseDown(); pyautogui.mouseUp()  # does the same thing as a left-
↪button mouse click
>>> pyautogui.mouseDown(button='right')  # press the right button down
>>> pyautogui.mouseUp(button='right', x=100, y=200)  # move the mouse to 100, 200,␣
↪then release the right button up.
```

Mouse Scrolling

The mouse scroll wheel can be simulated by calling the `scroll()` function and passing an integer number of "clicks" to scroll. The amount of scrolling in a "click" varies between platforms. Optionally, integers can be passed for the the `x` and `y` keyword arguments to move the mouse cursor before performing the scroll. For example:

```
>>> pyautogui.scroll(10)    # scroll up 10 "clicks"
>>> pyautogui.scroll(-10)   # scroll down 10 "clicks"
>>> pyautogui.scroll(10, x=100, y=100)  # move mouse cursor to 100, 200, then scroll␣
↪up 10 "clicks"
```

On OS X and Linux platforms, PyAutoGUI can also perform horizontal scrolling by calling the hscroll() function. For example:

```
>>> pyautogui.hscroll(10)    # scroll right 10 "clicks"
>>> pyautogui.hscroll(-10)    # scroll left 10 "clicks"
```

The `scroll()` function is a wrapper for `vscroll()`, which performs vertical scrolling.

CHAPTER 6

Keyboard Control Functions

The typewrite() Function

The primary keyboard function is `typewrite()`. This function will type the characters in the string is passed. To add a delay interval in between pressing each character key, pass an int or float for the `interval` keyword argument.

For example:

```
>>> pyautogui.typewrite('Hello world!')                    # prints out "Hello world!"␣
↪instantly
>>> pyautogui.typewrite('Hello world!', interval=0.25)   # prints out "Hello world!"␣
↪with a quarter second delay after each character
```

You can only press single-character keys with `typewrite()`, so you can't press the Shift or F1 keys, for example.

The press(), keyDown(), and keyUp() Functions

To press these keys, call the `press()` function and pass it a string from the `pyautogui.KEYBOARD_KEYS` such as `enter`, `esc`, `f1`. See *KEYBOARD_KEYS*.

For example:

```
>>> pyautogui.press('enter')  # press the Enter key
>>> pyautogui.press('f1')     # press the F1 key
>>> pyautogui.press('left')   # press the left arrow key
```

The `press()` function is really just a wrapper for the `keyDown()` and `keyUp()` functions, which simulate pressing a key down and then releasing it up. These functions can be called by themselves. For example, to press the left arrow key three times while holding down the Shift key, call the following:

```
>>> pyautogui.keyDown('shift')  # hold down the shift key
>>> pyautogui.press('left')     # press the left arrow key
>>> pyautogui.press('left')     # press the left arrow key
```

```
>>> pyautogui.press('left')      # press the left arrow key
>>> pyautogui.keyUp('shift')     # release the shift key
```

To press multiple keys similar to what `typewrite()` does, pass a list of strings to `press()`. For example:

```
>>> pyautogui.press(['left', 'left', 'left'])
```

The hotkey() Function

To make pressing hotkeys or keyboard shortcuts convenient, the `hotkey()` can be passed several key strings which will be pressed down in order, and then released in reverse order. This code:

```
>>> pyautogui.hotkey('ctrl', 'shift', 'esc')
```

. . .is equivalent to this code:

```
>>> pyautogui.keyDown('ctrl')
>>> pyautogui.keyDown('shift')
>>> pyautogui.keyDown('esc')
>>> pyautogui.keyUp('esc')
>>> pyautogui.keyUp('shift')
>>> pyautogui.keyUp('ctrl')
```

KEYBOARD_KEYS

The following are the valid strings to pass to the `press()`, `keyDown()`, `keyUp()`, and `hotkey()` functions:

```
['\t', '\n', '\r', ' ', '!', '"', '#', '$', '%', '&', "'", '(',
')', '*', '+', ',', '-', '.', '/', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', ':', ';', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`',
'a', 'b', 'c', 'd', 'e','f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o',
'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', '{', '|', '}', '~',
'accept', 'add', 'alt', 'altleft', 'altright', 'apps', 'backspace',
'browserback', 'browserfavorites', 'browserforward', 'browserhome',
'browserrefresh', 'browsersearch', 'browserstop', 'capslock', 'clear',
'convert', 'ctrl', 'ctrlleft', 'ctrlright', 'decimal', 'del', 'delete',
'divide', 'down', 'end', 'enter', 'esc', 'escape', 'execute', 'f1', 'f10',
'f11', 'f12', 'f13', 'f14', 'f15', 'f16', 'f17', 'f18', 'f19', 'f2', 'f20',
'f21', 'f22', 'f23', 'f24', 'f3', 'f4', 'f5', 'f6', 'f7', 'f8', 'f9',
'final', 'fn', 'hanguel', 'hangul', 'hanja', 'help', 'home', 'insert', 'junja',
'kana', 'kanji', 'launchapp1', 'launchapp2', 'launchmail',
'launchmediaselect', 'left', 'modechange', 'multiply', 'nexttrack',
'nonconvert', 'num0', 'num1', 'num2', 'num3', 'num4', 'num5', 'num6',
'num7', 'num8', 'num9', 'numlock', 'pagedown', 'pageup', 'pause', 'pgdn',
'pgup', 'playpause', 'prevtrack', 'print', 'printscreen', 'prntscrn',
'prtsc', 'prtscr', 'return', 'right', 'scrolllock', 'select', 'separator',
'shift', 'shiftleft', 'shiftright', 'sleep', 'space', 'stop', 'subtract', 'tab',
'up', 'volumedown', 'volumemute', 'volumeup', 'win', 'winleft', 'winright', 'yen',
'command', 'option', 'optionleft', 'optionright']
```

CHAPTER 7

Message Box Functions

PyAutoGUI makes use of the message box functions in PyMsgBox to provide a cross-platform, pure Python way to display JavaScript-style message boxes. There are four message box functions provided:

The alert() Function

```
>>> alert(text='', title='', button='OK')
```

Displays a simple message box with text and a single OK button. Returns the text of the button clicked on.

The confirm() Function

```
>>> confirm(text='', title='', buttons=['OK', 'Cancel'])
```

Displays a message box with OK and Cancel buttons. Number and text of buttons can be customized. Returns the text of the button clicked on.

The prompt() Function

```
>>> prompt(text='', title='' , default='')
```

Displays a message box with text input, and OK & Cancel buttons. Returns the text entered, or None if Cancel was clicked.

The password() Function

```
>>> password(text='', title='', default='', mask='*')
```

Displays a message box with text input, and OK & Cancel buttons. Typed characters appear as *. Returns the text entered, or None if Cancel was clicked.

CHAPTER 8

Screenshot Functions

PyAutoGUI can take screenshots, save them to files, and locate images within the screen. This is useful if you have a small image of, say, a button that needs to be clicked and want to locate it on the screen. These features are provided by the PyScreeze module, which is installed with PyAutoGUI.

Screenshot functionality requires the Pillow module. OS X uses the `screencapture` command, which comes with the operating system. Linux uses the `scrot` command, which can be installed by running `sudo apt-get install scrot`.

Special Notes About Ubuntu

Unfortunately, Ubuntu seems to have several deficiencies with installing Pillow. PNG and JPEG support are not included with Pillow out of the box on Ubuntu. The following links have more information: https://stackoverflow.com/questions/7648200/pip-install-pil-e-tickets-1-no-jpeg-png-support http://ubuntuforums.org/showthread.php?t=1751455

The screenshot() Function

Calling `screenshot()` will return an Image object (see the Pillow or PIL module documentation for details). Passing a string of a filename will save the screenshot to a file as well as return it as an Image object.

```
>>> import pyautogui
>>> im1 = pyautogui.screenshot()
>>> im2 = pyautogui.screenshot('my_screenshot.png')
```

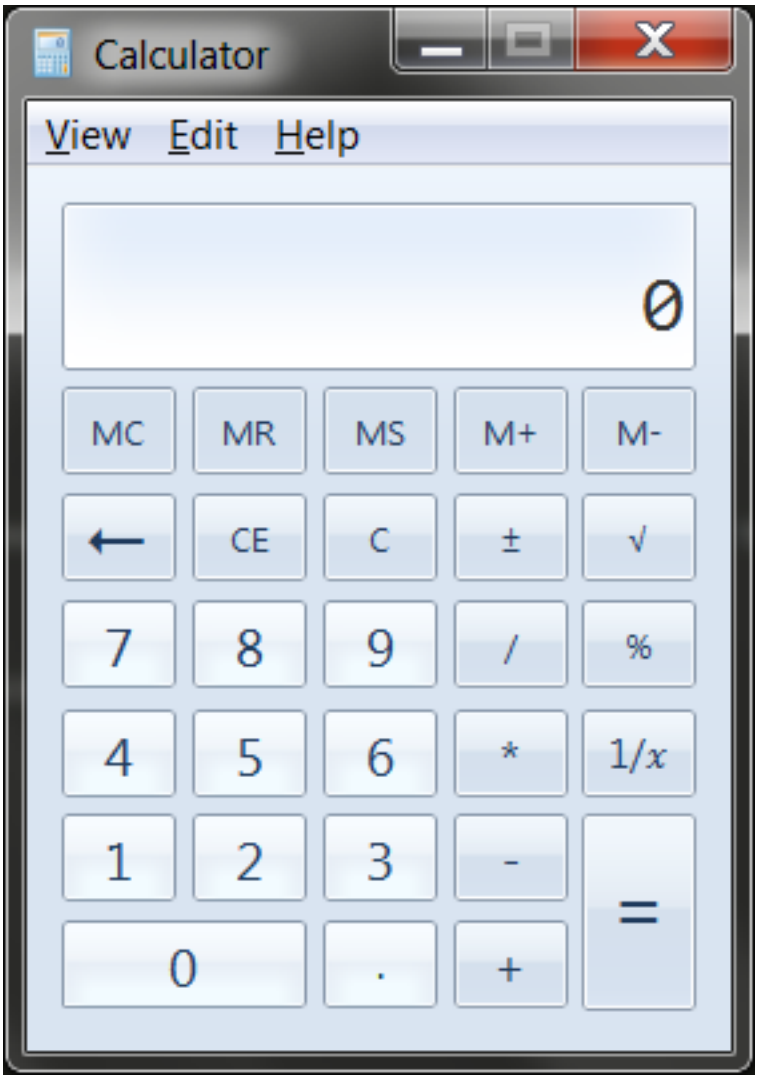
On a 1920 x 1080 screen, the `screenshot()` function takes roughly 100 milliseconds - it's not fast but it's not slow.

There is also an optional `region` keyword argument, if you do not want a screenshot of the entire screen. You can pass a four-integer tuple of the left, top, width, and height of the region to capture:

```
>>> import pyautogui
>>> im = pyautogui.screenshot(region=(0,0, 300, 400))
```

The Locate Functions

You can visually locate something on the screen if you have an image file of it. For example, say the calculator app was running on your computer and looked like this:



You can't call the `moveTo()` and `click()` functions if you don't know the exact screen coordinates of where the calculator buttons are. The calculator can appear in a slightly different place each time it is launched, causing you to re-find the coordinates each time. However, if you have an image of the button, such as the image of the 7 button:

. . . you can call the `locateOnScreen('calc7key.png')` function to get the screen coordinates. The return value is a 4-integer tuple: (left, top, width, height). This tuple can be passed to `center()` to get the X and Y

coordinates at the center of this region. If the image can't be found on the screen, `locateOnScreen()` returns `None`.

```
>>> import pyautogui
>>> button7location = pyautogui.locateOnScreen('calc7key.png')
>>> button7location
(1416, 562, 50, 41)
>>> button7x, button7y = pyautogui.center(button7location)
>>> button7x, button7y
(1441, 582)
>>> pyautogui.click(button7x, button7y)  # clicks the center of where the 7 button␣
↪was found
```

The `locateCenterOnScreen()` function is probably the one you want to use most often:

```
>>> import pyautogui
>>> x, y = pyautogui.locateCenterOnScreen('calc7key.png')
>>> pyautogui.click(x, y)
```

On a 1920 x 1080 screen, the locate function calls take about 1 or 2 seconds. This may be too slow for action video games, but works for most purposes and applications.

There are several "locate" functions. They all start looking at the top-left corner of the screen (or image) and look to the right and then down. The arguments can either be a

- `locateOnScreen(image, grayscale=False)` - Returns (left, top, width, height) coordinate of first found instance of the `image` on the screen. Returns None if not found on the screen.
- `locateCenterOnScreen(image, grayscale=False)` - Returns (x, y) coordinates of the center of the first found instance of the `image` on the screen. Returns None if not found on the screen.
- `locateAllOnScreen(image, grayscale=False)` - Returns a generator that yields (left, top, width, height) tuples for where the image is found on the screen.
- `locate(needleImage, haystackImage, grayscale=False)` - Returns (left, top, width, height) coordinate of first found instance of `needleImage` in `haystackImage`. Returns None if not found on the screen.
- `locateAll(needleImage, haystackImage, grayscale=False)` - Returns a generator that yields (left, top, width, height) tuples for where `needleImage` is found in `haystackImage`.

The "locate all" functions can be used in for loops or passed to `list()`:

```
>>> import pyautogui
>>> for pos in pyautogui.locateAllOnScreen('someButton.png')
...   print(pos)
...
(1101, 252, 50, 50)
(59, 481, 50, 50)
(1395, 640, 50, 50)
(1838, 676, 50, 50)
>>> list(pyautogui.locateAllOnScreen('someButton.png'))
[(1101, 252, 50, 50), (59, 481, 50, 50), (1395, 640, 50, 50), (1838, 676, 50, 50)]
```

These "locate" functions are fairly expensive; they can take a full second to run. The best way to speed them up is to pass a `region` argument (a 4-integer tuple of (left, top, width, height)) to only search a smaller region of the screen instead of the full screen:

```
>>> import pyautogui
>>> pyautogui.locateOnScreen('someButton.png', region=(0,0, 300, 400))
```

Grayscale Matching

Optionally, you can pass `grayscale=True` to the locate functions to give a slight speedup (about 30%-ish). This desaturates the color from the images and screenshots, speeding up the locating but potentially causing false-positive matches.

```
>>> import pyautogui
>>> button7location = pyautogui.locateOnScreen('calc7key.png', grayscale=True)
>>> button7location
(1416, 562, 50, 41)
```

Pixel Matching

To obtain the RGB color of a pixel in a screenshot, use the Image object's `getpixel()` method:

```
>>> import pyautogui
>>> im = pyautogui.screenshot()
>>> im.getpixel((100, 200))
(130, 135, 144)
```

Or as a single function, call the `pixel()` PyAutoGUI function, which is a wrapper for the previous calls:

```
>>> import pyautogui
>>> pyautogui.pixel(100, 200)
(130, 135, 144)
```

If you just need to verify that a single pixel matches a given pixel, call the `pixelMatchesColor()` function, passing it the X coordinate, Y coordinate, and RGB tuple of the color it represents:

```
>>> import pyautogui
>>> pyautogui.pixelMatchesColor(100, 200, (130, 135, 144))
True
>>> pyautogui.pixelMatchesColor(100, 200, (0, 0, 0))
False
```

The optional `tolerance` keyword argument specifies how much each of the red, green, and blue values can vary while still matching:

```
>>> import pyautogui
>>> pyautogui.pixelMatchesColor(100, 200, (130, 135, 144))
True
>>> pyautogui.pixelMatchesColor(100, 200, (140, 125, 134))
False
>>> pyautogui.pixelMatchesColor(100, 200, (140, 125, 134), tolerance=10)
True
```

CHAPTER 9

Testing

The unit tests for PyAutoGUI are currently not comprehensive. The tests (in basicTests.py) cover the following:

- onScreen()
- size()
- position()
- moveTo()
- moveRel()
- typewrite()
- PAUSE

Platforms Tested

- Python 3.4, 3.3, 3.2, 3.1, 2.7, 2.6, 2.5
- Windows
- OS X
- Raspberry Pi

(If you have run the unit tests successfully on other platforms, please tell al@inventwithpython.com.)

PyAutoGUI is not compatible with Python 2.4 or before.

The keyboard functions do not work on Ubuntu when run in VirtualBox on Windows.

CHAPTER 10

Roadmap

PyAutoGUI is planned as a replacement for other Python GUI automation scripts, such as PyUserInput, PyKeyboard, PyMouse, pykey, etc. Eventually it would be great to offer the same type of features that [Sikuli](http://www.sikuli.org) offers.

For now, the primary aim for PyAutoGUI is cross-platform mouse and keyboard control and a simple API.

Future features planned (specific versions not planned yet):

- "Wave" function, which is used just to see where the mouse is by shaking the mouse cursor a bit. A small helper function.
- locateNear() function, which is like the other locate-related screen reading functions except it finds the first instance near an xy point on the screen.
- Find a list of all windows and their captions.
- Click coordinates relative to a window, instead of the entire screen.
- Make it easier to work on systems with multiple monitors.
- GetKeyState() type of function
- Ability to set global hotkey on all platforms so that there can be an easy "kill switch" for GUI automation programs.
- Optional nonblocking pyautogui calls.
- "strict" mode for keyboard - passing an invalid keyboard key causes an exception instead of silently skipping it.
- rename keyboardMapping to KEYBOARD_MAPPING

Window handling features:

- pyautogui.getWindows() # returns a dict of window titles mapped to window IDs
- pyautogui.getWindow(str_title_or_int_id) # returns a "Win" object
- win.move(x, y)
- win.resize(width, height)

- win.maximize()
- win.minimize()
- win.restore()
- win.close()
- win.position() # returns (x, y) of top-left corner
- win.moveRel(x=0, y=0) # moves relative to the x, y of top-left corner of the window
- win.clickRel(x=0, y=0, clicks=1, interval=0.0, button='left') # click relative to the x, y of top-left corner of the window
- Additions to screenshot functionality so that it can capture specific windows instead of full screen.

CHAPTER 11

Cookbook and Examples

TODO - have example usage

CHAPTER 12

Frequently Asked Questions

Send questions to al@inventwithpython.com

Q: Can PyAutoGUI figure out where windows are or which windows are visible? Can it focus, maximize, minimize windows? Can it read the window titles?

A: Unfortunately not, but these are the next features planned for PyAutoGUI. This functionality is being implemented in a Python package named PyGetWindow, which will be included in PyAutoGUI when complete.

Q: Can PyAutoGUI work on Android, iOS, or tablet/smartphone apps.

A: Unfortunately no. PyAutoGUI only runs on Windows, Mac, and Linux.

This documentation is still a work in progress.

CHAPTER 13

Indices and tables

- genindex
- modindex
- search