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 `move()` function. This function has similar parameters as `moveTo()`. For example:

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

Mouse Drags

PyAutoGUI's `dragTo()` and `drag()` functions have similar parameters as the `moveTo()` and `move()` 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
>>> pyautogui.dragTo(300, 400, 2, button='left')  # drag mouse to X of 300, Y of 400 over 2 seconds whi
>>> pyautogui.drag(30, 0, 2, button='right')  # drag the mouse left 30 pixels over 2 seconds while hol
```

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()`, `move()`, `dragTo()`, and `drag()` 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
>>> pyautogui.click(button='right', clicks=3, interval=0.25)  ## triple-click the right mouse button wi
```

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 righ
```

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.