Simulate Keypresses In Python

This demonstrates how to press keys with Python. Using pynput we are able to simulate key presses into any window. This will show you how to press and release a key, type special keys and type a sentence.

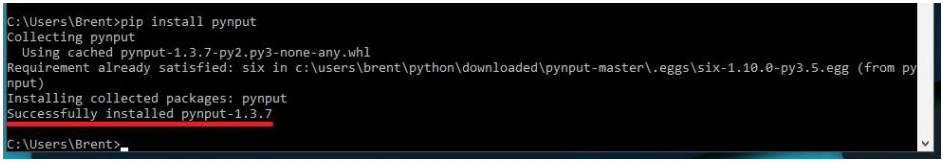
PIP

If you haven't used or setup pip before, go to my tutorial at how-to-setup-pythons-pip to setup pip.

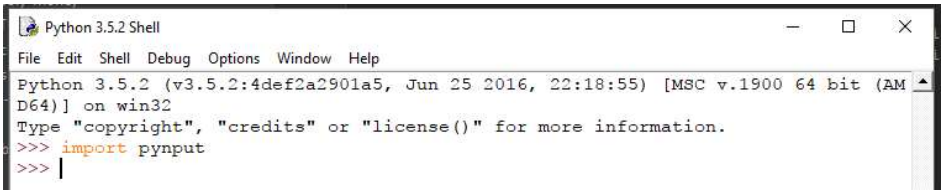
Installing Pynput

We will be using the pynput module to listen to mouse events. To install this module execute `pip install pynput` in cmd. Watch the output to make sure no errors have occurred; it will tell you when the module has

been successfully installed.



To double-check that it was installed successfully, open up IDLE and execute the command `import pynput`; no errors should occur.



Simulating Keys

Create a new script and save it somewhere so you can easily run the script. Import `Key` and `Controller` from `pynput.keyboard`.

```
from pynput.keyboard import Key, Controller
```

Make a variable called keyboard and set it to an instance of `Controller`. Now using the keyboard variable we can press and release keys.

```
keyboard = Controller()
```

Pressing and Releasing Keys

Using `keyboard.press` we can press keys and with `keyboard.release` we can release a key. This allows us to type a key by pressing and releasing. You can only supply this method with one key at a time. Here is an example of how to type the letter 'a'.

```
keyboard.press('a')
keyboard.release('a')
```

Pressing and Releasing Special Keys

For special keys that can't be put into a string like shift or control, you will need to refer to the page here to look at the Key class for supported keys. Using these in the press or release methods will press/release the key matching it. For example, if I wanted to press the Windows key, I would look at that page for the key. 'cmd' has the description "*A generic command button. On PC platforms, this corresponds to the Super key or Windows key, and on Mac it corresponds to the Command key*" which is what I am looking for. Now for the code.

```
keyboard.press(Key.cmd)
keyboard.release(Key.cmd)
```

This method also allows us to press a key while holding another key, for example, ctrl+c to copy. To do this we will need to press ctrl, press and release c and then release ctrl.

```
keyboard.press(Key.ctrl)
keyboard.press('c')
keyboard.release('c')
keyboard.release(Key.ctrl)
```

Here are a few other common special keys:

- Key.alt_l: Left ALT
- Key.backspace: Backspace
- Key.ctrl_l: Left Ctrl
- Key.delete: Delete
- Key.enter: Enter
- Key.esc: Escape
- Key.f1: F1
- Key.f5: F5

- Key.media_play_pause: Play/Pause
- Key.page_down: Page Down
- Key.up: Up Arrow Key
- The rest can be found in the pynput docs for the Key class.

Typing Multiple Keys

A cool feature supplied by the class is the type method. This method allows us to type more than one key at a time but it has to be a string of characters. So if we wanted to type "Nitratine" we would execute:

```
keyboard.type('Nitratine')
```

This method does also support spaces but when it comes to enters, use a new line character (\n) and a tab character (\t) for tabs.

```
keyboard.type('This is one line.\nAnd this is the next
line.\n\tThis line has been tabbed in.')
```

Putting A Random Delay Between Each Keypress

To put a random delay between each keypress, you can use `time.sleep` with a random number passed to it. Here is a small example function I made:

```
import time
import random
from pynput.keyboard import Controller

keyboard = Controller()  # Create the controller

def type_string_with_delay(string):
    for character in string:  # Loop over each character in the
string
        keyboard.type(character)  # Type the character
        delay = random.uniform(0, 2)  # Generate a random number
between 0 and 10
        time.sleep(delay)  # Sleep for the amount of seconds
generated

type_string_with_delay("This is my string typed with a delay")
```

Common Issues and Questions

How can I use the keyboard and mouse controllers at the same time?

When you import the classes, Controller will be set to the last one imported. To show what the issue was, ask yourself, what controller did you use to set the mouse and what one to set the keyboard? You would have used the same, but they need to be from the different classes. So then you should use:

```
from pynput.keyboard import Key, Controller as KeyboardController
from pynput.mouse import Button, Controller as MouseController
```

Now when you want to use the controller for the mouse use MouseController and KeyboardController for the keyboard.

```
keyboard = KeyboardController()
mouse = MouseController()
```

ModuleNotFoundError/ImportError : No module named 'pynput'

Did you install pynput? This error will not occur if you installed it properly. If you have multiple versions of Python, make sure you are installing pynput on the same version as what you are running the script with.

I got a SyntaxError

Syntax errors are caused by you and there is nothing I can offer to fix it apart from telling you to read the error. They always say where the error is in the output using a ^. Generally, people that get this issue have incorrect indentation, brackets in the wrong place or something spelt wrong. You can read about SyntaxError on Python's docs here.

The Key Presses Work in Notepad But Not My Game

pynput uses a Win32API function called `SendInput`. The SendInput function will insert input events into the same queue as a hardware device but the events are marked with a `LLMHF_INJECTED` flag that can be detected by hooks and then filtered. To avoid this flag you probably have to write a custom driver *(ref: stackoverflow/Anders)*.

It would be ideal for most games to look for these events if they want to reduce 'bot' activity as it stops packages like these being used.

In github.com/Gautam-J/Self-Driving-Car I had seen that the file directkeys.py contained the following:

```
import ctypes
import time

SendInput = ctypes.windll.user32.SendInput

W = 0x11
A = 0x1E
S = 0x1F
D = 0x20
UP = 0xC8
LEFT = 0xCB
RIGHT = 0xCD
DOWN = 0xD0
ENTER = 0x1C
ESC = 0x01
TWO = 0x03

# C struct redefinitions
PUL = ctypes.POINTER(ctypes.c_ulong)


class KeyBdInput(ctypes.Structure):
    _fields_ = [("wVk", ctypes.c_ushort),
                ("wScan", ctypes.c_ushort),
                ("dwFlags", ctypes.c_ulong),
                ("time", ctypes.c_ulong),
                ("dwExtraInfo", PUL)]


class HardwareInput(ctypes.Structure):
    _fields_ = [("uMsg", ctypes.c_ulong),
                ("wParamL", ctypes.c_short),
                ("wParamH", ctypes.c_ushort)]


class MouseInput(ctypes.Structure):
    _fields_ = [("dx", ctypes.c_long),
                ("dy", ctypes.c_long),
                ("mouseData", ctypes.c_ulong),
                ("dwFlags", ctypes.c_ulong),
                ("time", ctypes.c_ulong),
                ("dwExtraInfo", PUL)]


class Input_I(ctypes.Union):
    _fields_ = [("ki", KeyBdInput),
                ("mi", MouseInput),
                ("hi", HardwareInput)]
```

```
class Input(ctypes.Structure):
    _fields_ = [("type", ctypes.c_ulong),
                ("ii", Input_I)]


# Actuals Functions
def PressKey(hexKeyCode):
    extra = ctypes.c_ulong(0)
    ii_ = Input_I()
    ii_.ki = KeyBdInput(0, hexKeyCode, 0x0008, 0,
ctypes.pointer(extra))
    x = Input(ctypes.c_ulong(1), ii_)
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x),
ctypes.sizeof(x))


def ReleaseKey(hexKeyCode):
    extra = ctypes.c_ulong(0)
    ii_ = Input_I()
    ii_.ki = KeyBdInput(0, hexKeyCode, 0x0008 | 0x0002, 0,
                        ctypes.pointer(extra))
    x = Input(ctypes.c_ulong(1), ii_)
    ctypes.windll.user32.SendInput(1, ctypes.pointer(x),
ctypes.sizeof(x))


# directx scan codes
# http://www.gamespp.com/directx/directInputKeyboardScanCodes.html

if __name__ == '__main__':
    while (True):
        PressKey(0x11)
        time.sleep(1)
        ReleaseKey(0x11)
        time.sleep(1)
```

> The license for this piece of code can be found here.

This file demonstrates how we can press keys using DirectX key codes. The link in the file no longer exists but it can still be found on the wayback machine. This page provides other codes for keys that can be simulated.

Unfortunately it will only work on Windows (due to the usage of `ctypes.windll`) but some may find that it solves their issues with the `LLMHF_INJECTED`flag.