目录

简介

M60是一个60%机械键盘，支持USB和蓝牙5.0，能在10个蓝牙设备间任意切换，可热插拔换轴，带RGB背光灯。与其它键盘相比，M60有一个独特的特性——可以跑Python代码，因此，M60几乎拥有最强的自定义宏和扩展功能。

M60是延迟最低的USB键盘之一，采用中断方式检测按键，快速响应按键。并且精心地优化过功耗，键盘用1200mAh电池可以使用1个月。

特性

- USB蓝牙双模
- 轴支持热插拔
- 支持10个蓝牙设备间快速切换
- 经典60%布局，61键
- RGB背光灯
- 兼容Windows、macOS、Linux、Raspberry Pi、Android、iOS
- 支持固件更新
- USB Type-C

手上还没有M60键盘？可以在以下链接获取

- Makerdiary Store
- 淘宝

规格参数

硬件参数

	M60 键盘参数
主模块	nRF52840, Arm Cortex-M4F, 64MHz, M.2 KEY-E
RAM	256KB RAM
Flash	1MB FLASH + 8MB QSPI Flash
无线	Bluetooth Low Energy 5.0, NFC
USB	Type-C
布局	60% (61键)
热插拔	是
轴规格	兼容Cherry MX轴
背光	64个RGB背光灯
电池接口	JST 1.25mm 3-Pin，又称 ZH 1.25mm 3-Pin
天线	2.4GHz外接天线, NFC外接天线
尺寸	285 mm x 94.6 mm

功耗

- 关机 15uA左右
- 睡眠 50~60uA
- 连上蓝牙 160uA左右

多设备连接

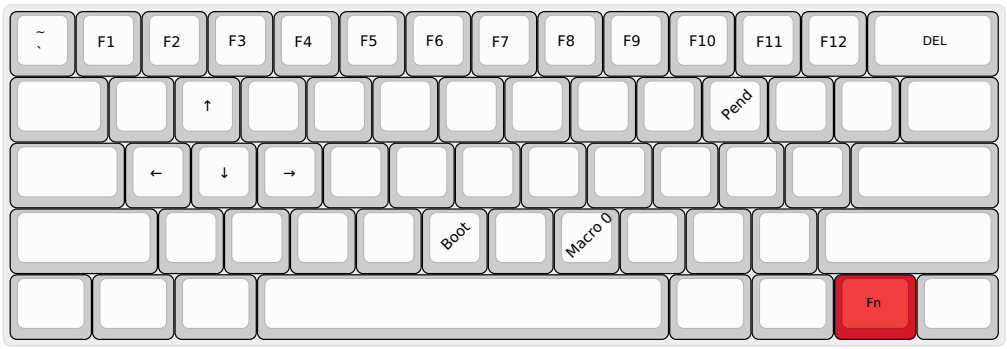
支持在10个蓝牙设备间任意切换

M60键盘上手

M60通过USB连上电脑即可使用，默认的键值为：

多层级keymap

由于M60只有61个键，缺少`F1`~`F12`等键，这里引入了多层键值，通常称之为keymap，上图为第0层，而`F1`至`F12`放在第1层（见下图），按住`Fn`则激活第1层。有多个被激活的层时，数值大的层优先被使用。



其中

- `Fn` + `M` 触发0号宏，会输入一串字符
- `Fn` + `P` 在电池供电时，将键盘挂起休眠，之后按任意键可重启键盘；USB供电时，会被忽略
- `Fn` + `B` 让键盘进入Bootloader，只在升级固件时使用

除了`Fn`这个专门用作切换层级的特殊键，普通键可以通过区分短按和长按来实现额外的功能，这类键称之为Tap-key。

`B` 键配置Bluetooth和USB

这里将`B`键用作了Tap-key。短按`B`键，正常输出`b`；按住`B`不放，则激活一个新的层，用来设置蓝牙（Bluetooth）和USB：

- `B` + `Esc` 切换蓝牙开关状态。蓝牙未开，则开启；蓝牙已开，则关闭
- `B` + `0` ~ `9` 切换蓝牙ID（从0到9），用于在多设备间切换
- `B` + `U` 切换USB开关状态

`D` 键导航

`D`也被用作Tap-key。短按，输出`d`；长按`D`不放，则激活一个新的层，用来快速导航：

- D + H → ←
- D + J → ↓
- D + K → ↑
- D + L → →
- D + U → PgUp
- D + N → PgDn

; 键用作 Ctrl

还有;被用作Tap-key，短按输出；，不过，长按不是用来切换层级，而是用作Ctrl。

- ; + c = Ctrl + c
- ; + v = Ctrl + v
- ; + x = Ctrl + x
- ; + a = Ctrl + a

以上将D和;用作Tap-key，目的是让我们在打字的时候手指尽量停留在home row（A S D F、H J K L ;）上，这不仅可以提升效率，也有助于手指、手腕关节的健康。 就具体应用场景而言，在VS Code里面双手打字时，用左右手配合按; + C比左手按Ctrl + C要方便一些。值得一提的是，VS Code中未选中文本时，Ctrl + C是复制光标所在的行，之后Ctrl + V，则把复制的行粘贴到光标下新的一行。再配合D + H、J、K、L移动光标，可以很便捷。

而在浏览器中，可用; + Tab替代Ctrl + Tab切换标签页，; + T打开新标签页，; + W关闭标签页。

结对快捷键

另外，键盘还支持同时（或10ms以内）按下两个键触发特殊功能，这里称之为结对快捷键（pair-keys）。默认的pair-keyes包括J K、U I两对，你可以文本编辑器中试试同时按下J和K，看看有什么输出。

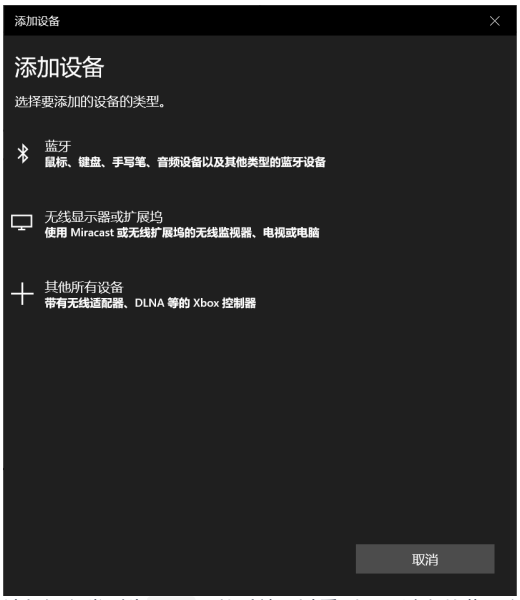
连接蓝牙

我们通过组合键B + 1开启蓝牙广播来连接第一个设备，蓝牙处于广播状态时，1键的背光呈蓝色呼吸状：

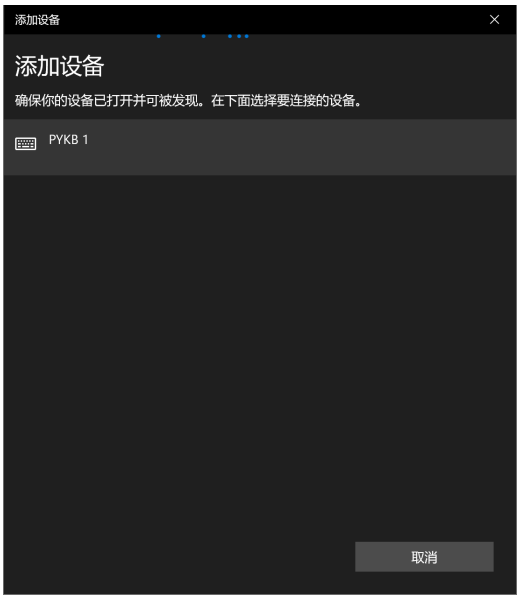
当需要要连接第二个设备时，使用 `B` + `2` 开启蓝牙广播，然后进行连接。键盘支持10个设备，可以通过 `B` + `0` ~ `9` 进行切换。

Windows连接蓝牙

打开 `设置` / `设备`，点击 `添加蓝牙或其他设备`



选择设备类型为 `蓝牙`，然后就可以看到M60键盘的蓝牙名称 `PYKB 1` 出现在搜索到的设备列表，点击 `PYKB 1` 即可连接键盘

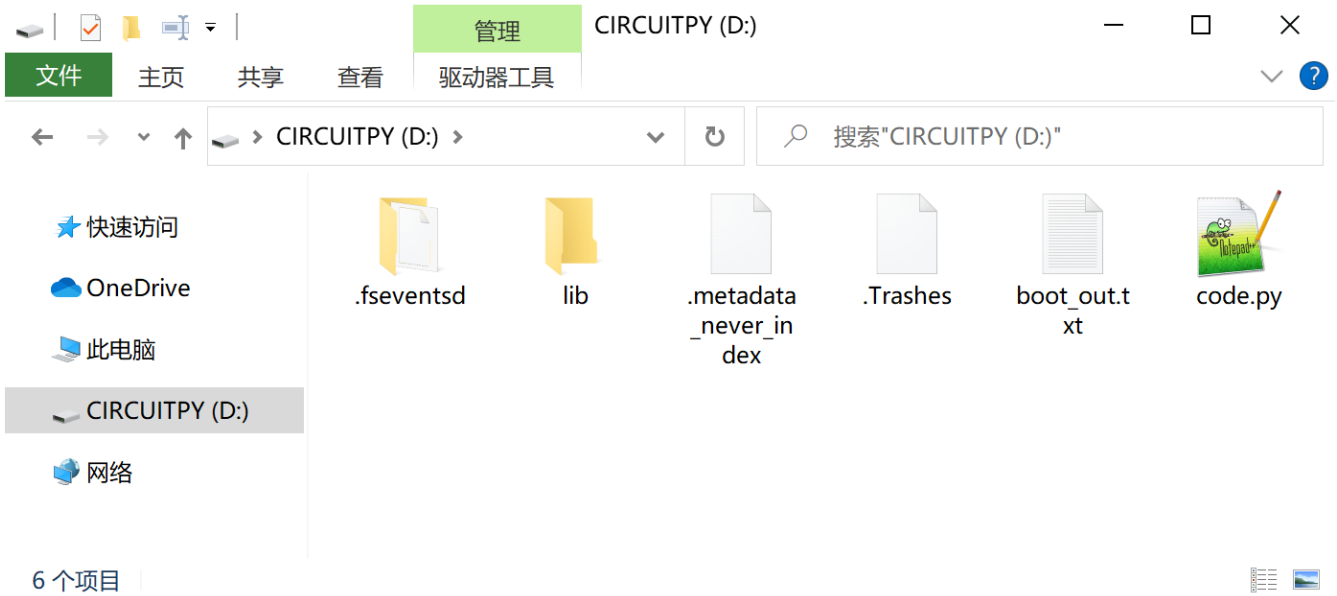


更进一步

我们希望M60默认的配置能展现给你一种提升效率的思路，你可以根据自己的想法和打字习惯调整键盘，找到最适合自己的配置。为此，我们设计了一种灵活的键盘配置方案——让键盘可以跑Python，通过修改简单的Python代码配置和扩展键盘。但不需要你一定懂Python，通过参考配置键盘文档就可以现实很多强大的个性化功能，开启你的探索~

配置键盘

M60不仅是一个键盘（HID设备），也是一个U盘，U盘中可以保存Python文件，而文件中的Python代码可以被M60执行。其中，M60启动时会默认运行U盘中的 `code.py` 。要配置或扩展键盘功能，我们用任意编辑器修改 `code.py` ，保存之后，即刻生效。



键盘通过USB连上电脑，会有一个名为 `CIRCUITPY` 的U盘出现，U盘中的 `code.py` 默认内容为：

```
from PYKB import *


keyboard = Keyboard()

___ = TRANSPARENT
BOOT = BOOTLOADER
L1 = LAYER_TAP(1)
L2D = LAYER_TAP(2, D)
L3B = LAYER_TAP(3, B)
LSFT4 = LAYER_MODS(4, MODS(LSHIFT))
RSFT4 = LAYER_MODS(4, MODS(RSHIFT))

# Semicolon & Ctrl
SCC = MODS_TAP(MODS(RCTRL), ';')

keyboard.keymap = (
    # layer 0
    (
        ESC,   1,   2,   3,   4,   5,   6,   7,   8,   9,   0, '-', '=', BACKSPACE,
        TAB,   Q,   W,   E,   R,   T,   Y,   U,   I,   O,   P, '[', ']', '|',
        CAPS,  A,   S, L2D,   F,   G,   H,   J,   K,   L, SCC, '"',   ENTER,
        LSFT4, Z,   X,   C,   V, L3B,   N,   M, ',', '.', '/',       RSFT4,
        LCTRL, LGUI, LALT,         SPACE,          RALT, MENU,  L1, RCTRL
    ),

    # layer 1
    (
        '`',  F1,  F2,  F3,  F4,  F5,  F6,  F7,  F8,  F9, F10, F11, F12, DEL,
        ___, ___,  UP, ___, ___, ___, ___, ___, ___, ___, ___,___,___,___,
        ___,LEFT,DOWN,RIGHT,___, ___, ___, ___, ___, ___, ___, ___,     ___,
        ___, ___, ___, ___, ___,BOOT, ___,MACRO(0), ___, ___, ___,      ___,
        ___, ___, ___,              ___,             ___, ___, ___,  ___
    ),

    # layer 2
    (
```

```
        '`',  F1,  F2,  F3,  F4,  F5,  F6,  F7,  F8,  F9, F10, F11, F12, DEL,
        ___, ___, ___, ___, ___, ___, ___,PGUP, ___, ___, ___,AUDIO_VOL_DOWN,AUDIO_VOL_UP,AUDIO_MUTE,
        ___, ___, ___, ___, ___, ___,LEFT,DOWN, UP,RIGHT, ___, ___,      ___,
        ___, ___, ___, ___, ___, ___,PGDN, ___, ___, ___, ___,           ___,
        ___, ___, ___,               ___,              ___, ___, ___,  ___
    ),

    # layer 3
    (
        BT_TOGGLE,BT1,BT2, BT3,BT4,BT5,BT6,BT7, BT8, BT9, BT0, ___, ___, ___,
        ___, ___, ___, ___, ___, ___,___,USB_TOGGLE,___,___,___,___,___, ___,
        ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,      ___,
        ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,           ___,
        ___, ___, ___,               ___,              ___, ___, ___,  ___
    ),

    # layer 4
    (
        '`', ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,
        ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,
        ___, ___, ___,   D, ___, ___, ___, ___, ___, ___, ';', ___,      ___,
        ___, ___, ___, ___, ___,   B, ___, ___, ___, ___, ___,           ___,
        ___, ___, ___,               ___,              ___, ___, ___,  ___
    ),
)


def macro_handler(dev, n, is_down):
    if is_down:
        dev.send_text('You pressed macro #{}\n'.format(n))
    else:
        dev.send_text('You released macro #{}\n'.format(n))

def pairs_handler(dev, n):
    dev.send_text('You just triggered pair keys #{}\n'.format(n))


keyboard.macro_handler = macro_handler
keyboard.pairs_handler = pairs_handler

# Pairs: J & K, U & I
keyboard.pairs = [{35, 36}, {20, 19}]

keyboard.verbose = False

keyboard.run()
```

其中，`keymap`对应键盘各层的键值（action code），`macro_handler`是宏的处理函数，`pairs_handler`为pair-keys的处理函数。

`code.py`更改保存后，会被立即运行。当修改的**`code.py`**时，需要注意一下是否有拼写错误和语法错误，若有错误会让键盘罢工，但这并不会损坏硬件，不用过分担心，只需将**`code.py`**恢复至以上的默认内容就可以让键盘恢复。

如果你懂一点Python，尽管大胆尝试；如果还没接触过Python，抓住这个机会，学一下Python，也是极好的。当然，没学Python，也可以配置键盘，依葫芦画瓢即可，这里列一些配置实例作为参考。

1. 交换`Caps`和`LCtrl`的位置，将`keymap`的`#layer 0`中`CAPS`和`LCTRL`替换即可：

    ```
        # layer 0
        (
            ESC,   1,   2,   3,   4,   5,   6,   7,   8,   9,   0, '-', '=', BACKSPACE,
            TAB,   Q,   W,   E,   R,   T,   Y,   U,   I,   O,   P, '[', ']', '|',
            LCTRL,  A,   S, L2D,   F,   G,   H,   J,   K,   L, SCC, '"',    ENTER,
            LSFT4, Z,   X,   C,   V, L3B,   N,   M, ',', '.', '/',        RSFT4,
            CAPS, LGUI, LALT,         SPACE,           RALT, MENU,  L1, RCTRL
        ),
    ```

2. 用`Caps`替换`D`作为Tap-key（因为`D`作为Tap-key可能跟游戏中用将`D`用作方向键冲突）同样是修改`keymap`的`#layer 0`：

```
# layer 0
(
   ESC,   1,   2,   3,   4,   5,   6,   7,   8,   9,   0, '-', '=', BACKSPACE,
   TAB,   Q,   W,   E,   R,   T,   Y,   U,   I,   O,   P, '[', ']', '|',
   LAYER_TAP(2, CAPS),  A,   S,  D,   F,   G,   H,   J,   K,   L, SCC, '"',    ENTER,
   LSFT4, Z,   X,   C,   V, L3B,   N,   M, ',', '.', '/',        RSFT4,
   LCTRL, LGUI, LALT,         SPACE,          RALT, MENU,  L1, RCTRL
),
```

3. 添加一个新的宏，比如，用`Fn`和`Enter`触发1号宏，由于`Fn`被用来激活第1层（layer 1），则在`keymap`的`#layer 1`中添加宏：

```
# layer 1
(
   '`',  F1,  F2,  F3,  F4,  F5,  F6,  F7,  F8,  F9, F10, F11, F12, DEL,
   ___, ___,  UP, ___, ___, ___, ___, ___, ___, ___,SUSPEND,___,___,___,
   ___,LEFT,DOWN,RIGHT,___, ___, ___, ___, ___, ___, ___, ___,    MACRO(1),
   ___, ___, ___, ___, ___,BOOT, ___,MACRO(0), ___, ___, ___,      ___,
   ___, ___, ___,              ___,             ___, ___, ___,  ___
),
```

   另外，定义宏的功能可以参考键盘宏文档

4. 将右下角的`RShift`、`Menu`、`Fn`、`RCtrl`，短按时作为方向键，长按时作为`RShift`、`RGUI`、`Fn`、`RCtrl`，这里修改`#layer 0`：

```
# layer 0
(
   ESC,   1,   2,   3,   4,   5,   6,   7,   8,   9,   0, '-', '=', BACKSPACE,
   TAB,   Q,   W,   E,   R,   T,   Y,   U,   I,   O,   P, '[', ']', '|',
   CAPS,  A,   S, L2D,   F,   G,   H,   J,   K,   L, SCC, '"',    ENTER,
   LSFT4, Z,   X,   C,   V, L3B,   N,   M, ',', '.', '/',   MODS_TAP(MODS(RSHIFT), UP),
   LCTRL, LGUI, LALT,         SPACE,     RALT, MODS_TAP(MODS(RGUI), LEFT), LAYER_TAP(1, DOWN), MODS_TAP(MODS(RCTRL), RIGHT)
),
```

键盘宏

在M60键盘默认配置中，`Fn` + `M`被用作0号宏，对应keymap中第1层的`MACRO(0)`。宏被触发后，会运行`code.py`中`macro_handler`：

```
def macro_handler(dev, n, is_down):
    if is_down:
        dev.send_text('You pressed macro #{}\n'.format(n))
    else:
        dev.send_text('You released macro #{}\n'.format(n))
```

其中，`macro_handler`带3个参数：

- `dev`用来提供键盘的部分接口，比如，`dev.send_text`用来发送一段字符串，`dev.send`用来发送keycode
- `n`为宏的编号，0号宏则`n`为0
- `is_down`表示宏状态，被按下时为`True`，被释放时为`False`

通过修改`macro_handler`可以实现丰富有趣的功能。

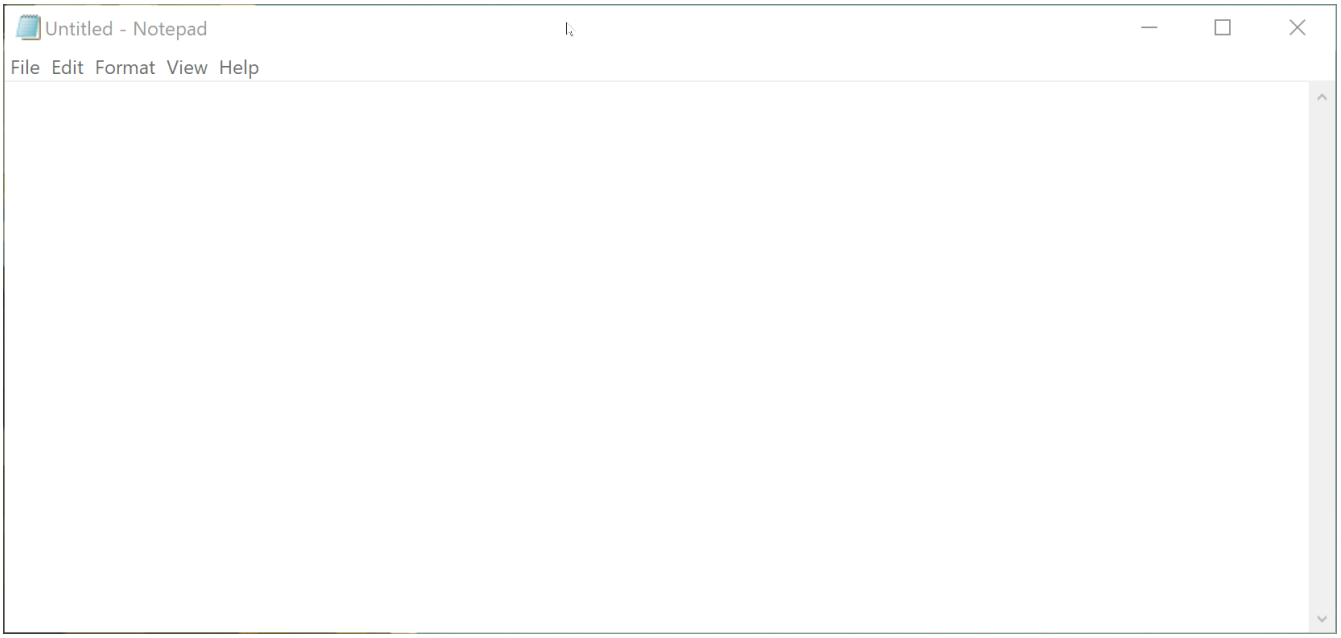
一键打开网页

在Windows上，我们可以设置宏一键用浏览器打开网页，只需将上面的`macro_handler`函数改为：

```
def macro_handler(dev, n, is_down):
    if is_down and n == 0:
        dev.send(GUI, R)
        time.sleep(0.1)
        dev.send_text('https://makerdiary.com\n')
```

其中，宏被触发之后会发送`Win`+`r`到电脑，延时0.1秒等待电脑响应，然后输入`https://makerdiary.com`+回车，电脑就把用默认的浏览器打开makerdiary网站。

一键写代码



要实现一键写代码，可先将提前写好的代码文件保存在M60的虚拟U盘中，然后在`macro_handler`中读取代码文件，最后将代码一个字母一个字母的敲出来。虽然是假装自动写代码，不过用来录屏还是极好的。

```
def macro_handler(dev, n, is_down):
    if is_down and n == 0:
        with open('code.py', 'r') as f:
```

```
            for line in f:
                dev.send_text(line)
```

一键循环发送 `666`

如果需要刷屏，一直循环地发送 `666` ，可以把 `macro_handler` 改成这样

```
def macro_handler(dev, n, is_down):
    if n == 0 and not is_down:
        t = time.time()
        dt = 1                # seconds
        while dev.scan() < 2:
            if time.time() > t:
                dev.send_text('666\n')
                t += dt
```

这个宏触发之后会一直输出 `666` +回车，直到有新的按键被按下为止。

当然，一直循环输出同样的内容有点单调，我们还可以添加一点随机性，比如，随机输出 `awesome` 、 `wonderful` 、 `superb` 、 `bravo` 等来刷弹幕：

```
import random

def macro_handler(dev, n, is_down):
    if n == 0 and not is_down:
        t = time.time()
        dt = 1
        dic = ['awesome', 'wonderful', 'superb', 'bravo', 'great', 'excellent']
        while dev.scan() < 2:
            if time.time() > t:
                i = random.randint(0, 3)
                dev.send_text(dic[i] + '\n')
                t += dt
        print('done')
```

结对快捷键

这里将同时（或10ms以内）按下两个键用于触发自定义功能，这样的两个键称之为结对快捷键（pair-keys）。 要设置pair-keys，先选两个按键，最好是相邻的，根据以下信息找到按键对应的编号，然后配置 `code.py` 中 `keyboard.pairs` 和 `keyboard.pairs_handler`。

```
# ESC(0)   1(1)  2(2)  3(3)  4(4)  5(5)  6(6)  7(7)  8(8)  9(9)  0(10) -(11) =(12) BACKSPACE(13)
# TAB(27)  Q(26) W(25) E(24) R(23) T(22) Y(21) U(20) I(19) O(18) P(17) [(16) ](15)  \(14)
# CAPS(28) A(29) S(30) D(31) F(32) G(33) H(34) J(35) K(36) L(37) ;(38) "(39)     ENTER(40)
#LSHIFT(52) Z(51) X(50) C(49) V(48) B(47) N(46) M(45) ,(44) .(43) /(42)         RSHIFT(41)
# LCTRL(53) LGUI(54) LALT(55)           SPACE(56)        RALT(57) MENU(58) Fn(59) RCTRL(60)
```

我们通过几个案例来熟悉pair-keys的使用：

1. 利用右下角的6个键组成4个pair-keys，用作方向键，其中:
   - `/` `RSHIFT` 作为 `↑`
   - `MENU` `Fn` 作为 `↓`
   - `RALT` `MENU` 作为 `←`
   - `Fn` `RCTRL` 作为 `→`

   ```
   # Pairs: / & RSHIFT, MENU & Fn, RALT & MENU, Fn & RCTRL
   keyboard.pairs = [{42, 41}, {58, 59}, {57, 58}, {59, 60}]

   def pairs_handler(dev, n):
       if n == 0:
           dev.send(UP)
       elif n == 1:
           dev.send(DOWN)
       elif n == 2:
           dev.send(LEFT)
       elif n == 3:
           dev.send(RIGHT)

   keyboard.pairs_handler = pairs_handler
   ```

2. 在Windows 10中，用 `D` `F` 和 `J` `K` 切换桌面
   - `D` `F` 输出 `CTRL` + `GUI` + `LEFT`
   - `J` `K` 输出 `CTRL` + `GUI` + `RIGHT`

   ```
   # Pairs: D & F, J & K
   keyboard.pairs = [{31, 32}, {35, 36}]

   def pairs_handler(dev, n):
       if n == 0:
           dev.send(CTRL, GUI, LEFT)
       elif n == 1:
           dev.send(CTRL, GUI, RIGHT)

   keyboard.pairs_handler = pairs_handler
   ```

键盘延迟

键盘按键的延迟，即按下按键到电脑响应按键之间的时间差，其影响因素包括：通信协议限制（USB和蓝牙）、矩阵矩阵扫描方式（周期扫描或者中断检测扫描）、防抖方式、键盘微处理器处理速度、电脑处理速度，甚至键程……

其中，通信协议限制是关键因素，现在广泛使用的通信协议是USB和蓝牙，其它方式大多也是通过USB转换，同样受到USB限制。

测按键的延迟比较难，简化一点，我们测一测键盘在极限条件下的响应速度，根据极限响应速度大致可以了解键盘的延迟。其中，这里设定的极限条件为:

> 键盘以最快的速度扫描按键矩阵，假装每一次都扫描到了一个键按下或释放（实际手速没用这么快），每次都上报按键事件

先说结论：

使用跑Windows 10的Surface Book，M60在USB连接下，可以稳定地每 1.1-1.2ms 处理一个按键事件（按下或释放）；而在低功耗蓝牙连接下，大多数时候可在 1-2ms 间处理一个按键事件，但偶尔会有一个非常大的延迟。

USB对延迟的影响

键盘是USB中的标准 HID (Human Input Device) 设备，HID设备采用USB协议的中断传输方式 (Interrupt Transfer)，虽然名字中有中断二字，但实际上是电脑以大致的周期轮询设备，其中，最小的轮询间隔 (即中断间隔，Interrupt Interval) 可设置为1ms，即最高频率为1000Hz。因此，很多游戏键盘以1000Hz矩阵扫描频率，高了也没多少用。

为了测响应速度，这里设计了一个小实验：

> 键盘上，模拟字母a到z依次按下、释放、发送给电脑，在键盘端测量每次按键扫描、发送的处理时间，同时也在电脑端测量a到z按下、释放的时间间隔。

这里键盘端用自定义宏来实现，将键盘U盘中 `code.py` 中 `macro_handler` 更改为:

```
def macro_handler(dev, n, is_down):
    if is_down:
        return

    a = get_action_code(A)
    data = []
    t = time.monotonic_ns()
    for i in range(26):
        t1 = time.monotonic_ns()
        dev.scan()
        dev.kbd.press(a + i)
        t2 = time.monotonic_ns()
        dev.scan()
        dev.kbd.release(a + i)
        t3 = time.monotonic_ns()
        data.append((t2 - t1) // 100000 / 10.)
        data.append((t3 - t2) // 100000 / 10.)

    average = (time.monotonic_ns() - t) // (26 * 2 * 100000) / 10.
    print('average: {}, max: {}, min: {}, data: {}'.format(average, max(data), min(data), data))
```

其中，用 `Fn` + `M` 触发这个宏。

电脑端使用了keyboard库，代码如下：

```
import time
import keyboard


data = lambda: None
data.events = None

# {"event_type": "down", "scan_code": 29, "name": "ctrl", "time": 0, "is_keypad": false}
def print_pressed_keys(e):

    if e.name == 'a' and e.event_type == 'down':
        data.start = time.monotonic_ns()
        data.events = [e]
```

```
    elif data.events:
        data.events.append(e)
        if e.name == 'z' and e.event_type == 'up':
            data.end = time.monotonic_ns()
            t = []
            for i in range(1, len(data.events)):
                dt = data.events[i].time - data.events[i - 1].time
                dt = int(dt * 100000) / 100.
                t.append(dt)

            average = (data.end - data.start) // (len(data.events) * 100000) / 10.
            print('average: {}, max: {}, min: {}, data: {}'.format(average, max(t), min(t), t))
            data.events = None

keyboard.hook(print_pressed_keys)
keyboard.wait()
```

这里附上次4次测量数据，需要注意的是，键盘端测量是扫描+发送的处理时间，电脑端是两个按键事件的时间差。

键盘端测量数据

输出4串字母a到z，获得的扫描+发送的处理时间

```
average: 1.1, max: 1.1, min: 0.9, data: [0.9, 0.9, 1.0, 0.9, 1.0, 0.9, 0.9, 0.9, 1.0, 0.9, 1.0, 0.9, 1.0, 0.9, 0.9, 0.9, 1.0, 0.9, 1.0, 0.9, 1.0, 0.9, 0.9, 0.9,
average: 1.1, max: 1.1, min: 0.9, data: [1.0, 1.0, 1.0, 0.9, 1.0, 0.9, 1.1, 0.9, 1.0, 0.9, 1.0, 0.9, 0.9, 1.0, 1.0, 0.9, 1.0, 0.9, 0.9, 1.0, 1.0, 0.9, 1.0, 0.9,
average: 1.1, max: 1.2, min: 0.9, data: [1.0, 1.0, 1.0, 1.1, 1.0, 0.9, 1.0, 0.9, 1.0, 0.9, 1.0, 1.0, 1.1, 0.9, 0.9, 1.0, 1.0, 0.9, 1.0, 0.9, 1.0, 1.0, 1.1, 0.9,
average: 1.1, max: 1.1, min: 0.9, data: [1.0, 1.0, 1.0, 1.0, 0.9, 1.0, 1.0, 1.0, 1.1, 0.9, 0.9, 1.0, 1.0, 0.9, 1.0, 0.9, 1.0, 1.0, 1.1, 0.9, 1.0, 0.9, 1.0, 0.9,
```

电脑端测量数据

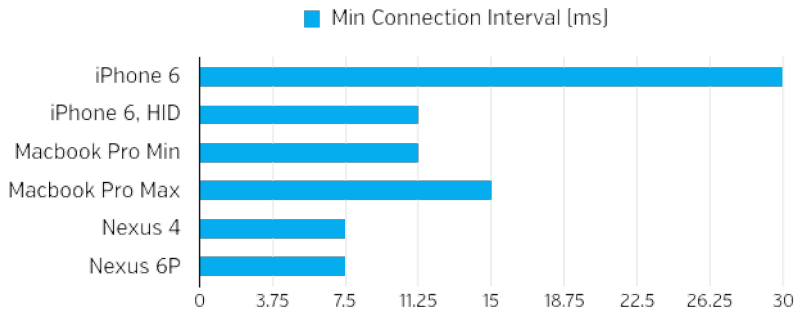
输入4串字母a到z，获得的按键事件时间间隔

```
average: 1.2, max: 4.02, min: 0.0, data: [3.82, 2.68, 2.0, 4.02, 0.99, 0.99, 1.99, 2.94, 2.99, 0.99, 1.0, 0.99, 1.99, 0.0, 2.0, 0.98, 0.99, 0.99, 0.99, 0
average: 1.2, max: 2.76, min: 0.0, data: [1.99, 1.99, 1.99, 1.95, 2.0, 0.99, 1.99, 2.76, 1.83, 1.0, 0.99, 0.0, 0.99, 0.99, 1.0, 0.99, 0.99, 0.99, 1.99, 0
average: 1.2, max: 3.14, min: 0.0, data: [0.95, 1.98, 3.14, 1.95, 2.02, 2.89, 2.6, 2.11, 1.0, 1.0, 0.99, 0.0, 0.99, 0.99, 0.99, 0.99, 0.0, 0.99, 0.0, 1.67
average: 1.1, max: 4.51, min: 0.0, data: [1.0, 0.99, 2.99, 1.99, 1.92, 1.35, 4.41, 4.51, 0.0, 1.99, 0.99, 0.99, 3.0, 1.98, 0.99, 1.0, 0.99, 1.36, 1.59, 1
```

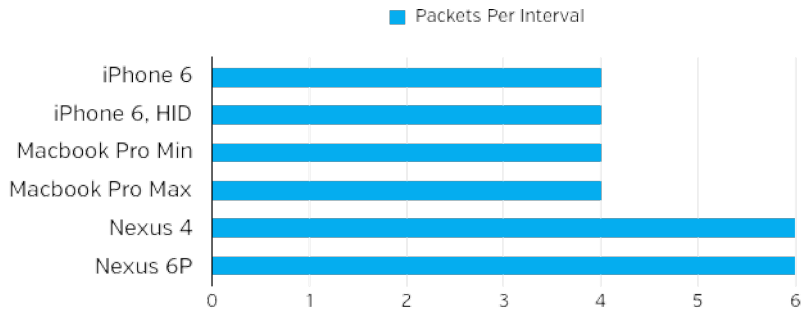
从数据数据中可以看出，键盘端的数据要稳定一些，电脑端波动大一些，也许是因为Windows系统的非实时性导致获得事件时间不太准确。

BLE对延迟的影响

BLE在对键盘的支持上借用了USB HID协议，设计了一个叫 HID over GATT 的蓝牙 Profile。在BLE协议中，对延迟影响最大是两个设备连接之后的连接间隔（Connection Interval）。在不同的系统上，最小的连接间隔有所不同，其中，Android上最小为7.5ms；苹果系统上最小可以到11.25ms；Windows上，没用找说明，Surface Book上实测最小为11.25ms。



这里的连接间隔和前面USB HID的中断间隔是不同的，因为蓝牙的一个连接间隔里面可以发几包数据，而最大是几包，这又是因系统而异：Android上可以发6包数据；苹果系统上是4，Windows上结合下面的数据可能是6。



那么，键盘在Android上最快是每秒发 1000 * 6 / 7.5 = 800 次数据，平均间隔 1.25 ms；苹果系统上，最快是 1000 * 4 / 11.25 ~= 355 次/s，平均间隔 2.8125 ms；Windows上，可能是 1000 * 6 / 11.25 ~= 533 次/s，平均间隔 1.875 ms。

可以套用前面的小实验，代码不用变，电脑连上蓝牙，用 `B` + `U` 把USB HID关掉，用蓝牙发送按键信息。

键盘端测量数据

```
average: 2.7, max: 31.6, min: 1.0, data: [1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.1, 31.6, 1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.2, 1.5, 1.1, 1.0, 1.1, 1
average: 2.0, max: 19.9, min: 1.0, data: [1.2, 1.1, 1.1, 1.0, 1.1, 1.1, 1.2, 1.5, 1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.2, 1.3, 11.4, 2.1, 2.1, 1.9, 1.8, 1.4, 1.9, 1
average: 3.5, max: 36.6, min: 1.0, data: [1.2, 1.1, 1.1, 1.1, 1.2, 1.2, 1.3, 1.0, 1.1, 1.0, 1.1, 27.8, 1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.2, 1.5, 1.4, 1.0, 1.1, 1
average: 2.6, max: 34.3, min: 1.0, data: [1.1, 1.1, 1.1, 1.0, 1.1, 1.2, 1.1, 1.0, 1.1, 1.0, 1.1, 28.1, 1.1, 1.0, 1.1, 1.0, 1.1, 1.0, 1.3, 1.5, 2.0, 1.0, 1.1, 1
```

电脑端测量数据

```
average: 3.9, max: 38.06, min: 0.0, data: [2.53, 8.0, 11.95, 0.99, 0.99, 0.99, 1.0, 0.99, 0.99, 5.16, 0.99, 0.66, 31.19, 1.02, 0.61, 20.85, 1.02, 0.99,
average: 1.5, max: 24.81, min: 0.0, data: [1.95, 0.99, 1.98, 0.0, 0.99, 0.0, 0.99, 0.99, 0.0, 0.99, 0.99, 0.0, 0.99, 0.99, 0.0, 1.0, 0.77, 0.44, 1.7, 1.0,
average: 3.3, max: 33.06, min: 0.0, data: [0.95, 1.99, 1.99, 0.98, 0.0, 0.99, 1.4, 1.0, 18.18, 1.02, 0.0, 0.99, 9.6, 0.0, 1.0, 33.06, 0.0, 0.99, 1.0, 0.0,
average: 2.7, max: 62.49, min: 0.0, data: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 15.63, 0.0, 0.0, 0.0, 0.0, 0.0
```

另外，把跑Windows 10的Surface Book切换为小米Mix 2，测了键盘端的延迟。

Android连接时键盘端数据

```
average: 1.2, max: 2.1, min: 0.8, data: [0.9, 0.8, 0.9, 0.9, 1.8, 0.8, 0.9, 0.8, 0.9, 0.8, 0.9, 0.8, 0.9, 0.8, 2.0, 1.9, 1.5, 0.8, 0.9, 0.8, 1.0, 0.8, 1.9, 1.4,
average: 1.2, max: 2.1, min: 0.8, data: [0.9, 0.8, 1.0, 0.9, 1.7, 0.8, 0.9, 0.8, 0.9, 0.8, 0.9, 0.8, 1.0, 0.8, 2.1, 2.0, 1.2, 0.8, 0.9, 0.8, 0.9, 1.0, 2.1, 0.8,
average: 1.2, max: 2.1, min: 0.8, data: [0.9, 0.8, 0.9, 0.8, 0.9, 0.9, 1.0, 1.9, 1.5, 0.8, 0.9, 0.8, 0.9, 0.8, 1.0, 1.0, 2.0, 1.7, 0.9, 0.8, 0.9, 0.8, 1.0, 0.8,
average: 1.2, max: 2.1, min: 0.8, data: [1.0, 0.8, 1.3, 0.8, 0.9, 0.8, 0.9, 0.8, 0.9, 0.8, 0.9, 0.9, 1.1, 1.9, 2.1, 1.5, 0.9, 0.8, 0.9, 0.8, 1.6, 1.7, 0.9, 0.8,
```

从数据中，可以看出在Windows中蓝牙连接下，大多数时候可在 1~2ms 间处理发送一个按键事件，但偶尔会有一个非常大的延迟，键盘端延迟可达36.6ms，电脑端延迟可达62.49ms。 而Android上蓝牙的延迟更小，也更稳定。 键盘端和电脑端的数据有差异，不过在同一个数量级，能反应出键盘的响应速度，从中了解到键盘的延迟。

Keycodes

占位键

名称	描述
NO	空键
TRANSPARENT	透明键，使用下一层键值

Bluetooth

名称	描述
BT0 ~ BT9	切换蓝牙设备
BT_TOGGLE	切换蓝牙开关状态

特殊键

名称	描述
BOOTLOADER	进入bootloader
HEATMAP	生成按键heatmap（todo）
SUSPEND	休眠，仅用于电池供电，休眠之后，可通过任意键唤醒
SHUTDOWN	关机，仅用于电池供电，关机之后，只能通过背面按键开机

功能键

功能键用于切换层级、组合普通键与一个或多个修饰键、定义于宏。

- MODS_KEY(mods, key) 发送一个或多个修饰键（modifier） + 一个普通键，其中， mods 需用到 MODS()

  MODS_KEY(MODS(LCTRL), C)， MODS_KEY(MODS(LCTRL, LSHIFT), C)， MODS_KEY(MODS(LCTRL, LSHIFT, LALT), C)
- LAYER_TOGGLE(n) 切换第n层状态
- MACRO(n) 定义第n号宏

TAP-Key

TAP-Key 区分长按和短按：

- LAYER_TAP(n, key)，长按激活第n层，释放后取消激活；短按则发送普通键值 key。若没有第2个参数，则短按不做处理。
- LAYER_TAP_TOGGLE(n)，长按激活第n层，释放后取消激活；短按则切换第n层状态。
- LAYER_MODS(n, mods)，长按激活第n层，并发送一个或多个修饰键；短按发送一个或多个修饰键， mods 需用 MODS() 生成

  LAYER_MODS(1, MODS(LCTRL))， LAYER_MODS(1, MODS(LCTRL, LSHIFT))
- MODS_TAP(mods, key) 长按发送一个或多个修饰键；短按发送 key

  MODS_TAP(MODS(LCTRL), ';')， MODS_TAP(MODS(LCTRL, LALT), LEFT)

媒体键

```
AUDIO_MUTE
AUDIO_VOL_UP
AUDIO_VOL_DOWN
TRANSPORT_NEXT_TRACK
TRANSPORT_PREV_TRACK
TRANSPORT_STOP
TRANSPORT_STOP_EJECT
```

```
TRANSPORT_PLAY_PAUSE
# application launch
APPLAUNCH_CC_CONFIG
APPLAUNCH_EMAIL
APPLAUNCH_CALCULATOR
APPLAUNCH_LOCAL_BROWSER
# application control
APPCONTROL_SEARCH
APPCONTROL_HOME
APPCONTROL_BACK
APPCONTROL_FORWARD
APPCONTROL_STOP
APPCONTROL_REFRESH
APPCONTROL_BOOKMARKS
# supplement for Bluegiga iWRAP HID(not supported by Windows?)
APPLAUNCH_LOCK
TRANSPORT_RECORD
TRANSPORT_FAST_FORWARD
TRANSPORT_REWIND
TRANSPORT_EJECT
APPCONTROL_MINIMIZE
# https://docs.microsoft.com/en-us/windows-hardware/drivers/hid/display-brightness-control
DISPLAY_BRIGHTNESS_UP
DISPLAY_BRIGHTNESS_DOWN
```

普通键

```
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

1
2
3
4
5
6
7
8
9
0

ENTER
ESCAPE
ESC
BACKSPACE
```

```
TAB
SPACE
MINUS
EQUAL
LEFTBRACE
RIGHTBRACE
BACKSLASH
HASHTILDE
SEMICOLON
APOSTROPHE
QUOTE
GRAVE
COMMA
DOT
SLASH
CAPSLOCK
CAPS

F1
F2
F3
F4
F5
F6
F7
F8
F9
F10
F11
F12

PRINTSCREEN
PRTSCN
SCROLLLOCK
PAUSE
INSERT
HOME
PAGEUP
PGUP
DELETE
DEL
END
PAGEDOWN
PGDN
RIGHT
LEFT
DOWN
UP

NUMLOCK
KPSLASH
KPASTERISK
KPMINUS
KPPLUS
KPENTER
KP1
KP2
KP3
KP4
KP5
KP6
KP7
KP8
KP9
KP0
KPDOT

# 102ND
APPLICATION
MENU
POWER
```

```
KPEQUAL

F13
F14
F15
F16
F17
F18
F19
F20
F21
F22
F23
F24

OPEN
HELP
# PROPS
SELECT
STOP
AGAIN
UNDO
CUT
COPY
PASTE
FIND
MUTE
KPCOMMA

INT1
INT2
INT3
INT4
INT5
INT6
INT7
INT8
INT9

RO
KATAKANAHIRAGANA
YEN
HENKAN
MUHENKAN
KPJPCOMMA

LANG1
LANG2
LANG3
LANG4
LANG5
LANG6
LANG7
LANG8
LANG9

HANGEUL
HANJA
KATAKANA
HIRAGANA
ZENKAKUHANKAKU

KPLEFTPAREN
KPRIGHTPAREN

LEFT_CTRL
LEFT_SHIFT
LEFT_ALT
LEFT_GUI
RIGHT_CTRL
RIGHT_SHIFT
```

```
RIGHT_ALT
RIGHT_GUI

LCTRL
LSHIFT
LALT
LGUI
RCTRL
RSHIFT
RALT
RGUI

CTRL
SHIFT
ALT
GUI
```

USB

```
USB_TOGGLE
```

升级固件

查看固件版本

要查看键盘当前使用的固件版本，可打开键盘U盘中的`boot_out.txt`文件查看版本，以下是2020年8月27日发布的固件版本信息：

```
Adafruit CircuitPython 6.0.0-alpha.1-110-g121d78ec9 on 2020-08-27; Makerdiary M60 Keyboard with nRF52840
```

若你发现 python-keyboard / firmware 中有新版本固件，可以进入键盘自带了Bootloader升级固件。

进入Bootloader

键盘有多种方式进入bootloader：

1. 键盘正常工作并处于USB连接状态时，通过组合键`Fn` + `b`进入bootloader
2. USB连接下，按住键盘背面的ON/OFF按键3秒以上进入bootloader
3. USB连接下，通过键盘的虚拟串口进入Python交互命令行模式，然后运行：

   ```
   import microcontroller as mcu
   mcu.on_next_reset(mcu.RunMode.BOOTLOADER)
   mcu.reset()
   ```

4. 未连接电池状态下，按住ON/OFF键，然后连上USB上电，进入bootloader

升级固件

键盘进入Bootloader后，会在电脑出现一个名为M60Keyboard的U盘，下载最新的键盘固件，将固件拖拽到U盘中，等待U盘消失，出现新的名为CIRCUITPY的U盘后，即完成升级。

恢复出厂设置

若是修改键盘代码导致键盘罢工，当依然可以看到 CIRCUITPY U盘时，可以将U盘中的 code.py 恢复至出厂内容：

```
from PYKB import *


keyboard = Keyboard()

___ = TRANSPARENT
BOOT = BOOTLOADER
L1 = LAYER_TAP(1)
L2D = LAYER_TAP(2, D)
L3B = LAYER_TAP(3, B)
LSFT4 = LAYER_MODS(4, MODS(LSHIFT))
RSFT4 = LAYER_MODS(4, MODS(RSHIFT))

# Semicolon & Ctrl
SCC = MODS_TAP(MODS(RCTRL), ';')

keyboard.keymap = (
    # layer 0
    (
        ESC,   1,   2,   3,   4,   5,   6,   7,   8,   9,   0, '-', '=', BACKSPACE,
        TAB,   Q,   W,   E,   R,   T,   Y,   U,   I,   O,   P, '[', ']', '|',
        CAPS,  A,   S, L2D,   F,   G,   H,   J,   K,   L, SCC, '"',    ENTER,
        LSFT4, Z,   X,   C,   V, L3B,   N,   M, ',', '.', '/',        RSFT4,
        LCTRL, LGUI, LALT,          SPACE,           RALT, MENU,  L1, RCTRL
    ),

    # layer 1
    (
        '`',  F1,  F2,  F3,  F4,  F5,  F6,  F7,  F8,  F9, F10, F11, F12, DEL,
        ___, ___,  UP, ___, ___, ___, ___, ___, ___, ___,SUSPEND,___,___,___,
        ___,LEFT,DOWN,RIGHT,___, ___, ___, ___, ___, ___, ___, ___,      ___,
        ___, ___, ___, ___, ___,BOOT, ___,MACRO(0), ___, ___, ___,       ___,
        ___, ___, ___,              ___,             ___, ___, ___,  ___
    ),

    # layer 2
    (
        '`',  F1,  F2,  F3,  F4,  F5,  F6,  F7,  F8,  F9, F10, F11, F12, DEL,
        ___, ___, ___, ___, ___, ___, ___,PGUP, ___, ___, ___,AUDIO_VOL_DOWN,AUDIO_VOL_UP,AUDIO_MUTE,
        ___, ___, ___, ___, ___, ___,LEFT,DOWN, UP,RIGHT, ___, ___,      ___,
        ___, ___, ___, ___, ___, ___,PGDN, ___, ___, ___, ___,           ___,
        ___, ___, ___,              ___,             ___, ___, ___,  ___
    ),

    # layer 3
    (
        BT_TOGGLE,BT1,BT2, BT3,BT4,BT5,BT6,BT7, BT8, BT9, BT0, ___, ___, ___,
        ___, ___, ___, ___, ___, ___,___,USB_TOGGLE,___,___,___,___,___, ___,
        ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,      ___,
        ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,           ___,
        ___, ___, ___,              ___,             ___, ___, ___,  ___
    ),

    # layer 4
    (
        '`', ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,
        ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___, ___,
        ___, ___, ___,   D, ___, ___, ___, ___, ___, ___, ';', ___,      ___,
        ___, ___, ___, ___, ___,   B, ___, ___, ___, ___, ___,           ___,
        ___, ___, ___,              ___,             ___, ___, ___,  ___
    ),
)


def macro_handler(dev, n, is_down):
```

```
    if is_down:
        dev.send_text('You pressed macro #{}\n'.format(n))
    else:
        dev.send_text('You released macro #{}\n'.format(n))

def pairs_handler(dev, n):
    dev.send_text('You just triggered pair keys #{}\n'.format(n))


keyboard.macro_handler = macro_handler
keyboard.pairs_handler = pairs_handler

# Pairs: J & K, U & I
keyboard.pairs = [{35, 36}, {20, 19}]

keyboard.verbose = False

keyboard.run()
```

若键盘仍然不正常，可以尝试一下操作

通过USB虚拟串口重置

用 putty（或其它串口工具）打开USB虚拟串口进入键盘的Python REPL（命令行模式），用另外一个键盘操作，先按 ctrl + c，然后输入：

```
import storage
storage.erase_filesystem()
```

通过固件重置

若以上方式都不行，可通过以下2个步骤恢复出厂设置固件：

1. 下载 factory_reset.uf2，按照升级固件文档给键盘烧写 `factoary_reset.uf2` 固件

   `factory_reset.uf2` 固件会在每次启动时将键盘U盘中的内容恢复至出厂状态。

2. 在 python-keyboard / firmware 中下载 M60 的最新固件，同样按照升级固件方式烧写这个最新固件

树莓派蓝牙连接键盘

如果在树莓派（Raspberry Pi）上用蓝牙连接M60键盘，用桌面蓝牙图形化工具连接可能会失败，这时候需要在命令行中连接，可运行以下指令：

```
bluetoothctl
pairable on
scan on
scan off
agent on
pair XX:XX:XX:XX:XX
trust XX:XX:XX:XX:XX
connect XX:XX:XX:XX:XX
info XX:XX:XX:XX:XX
exit
```

以下是一次配置的log:

```
pi@raspberrypi:~ $ sudo bluetoothctl
Agent registered
[bluetooth]# pairable on
Changing pairable on succeeded
[bluetooth]# scan on
Discovery started
[CHG] Controller DC:66:32:3B:9F:B7 Discovering: yes
[NEW] Device C5:00:01:02:03:36 PYKB 1                                   .
[bluetooth]# scan off
Discovery stopped
[CHG] Controller DC:66:32:3B:9F:B7 Discovering: no
[CHG] Device C5:00:01:02:03:36 TxPower is nil
[CHG] Device C5:00:01:02:03:36 RSSI is nil
[bluetooth]# agent on
Agent is already registered
[bluetooth]# pair C5:00:01:02:03:36
Attempting to pair with C5:00:01:02:03:36
[CHG] Device C5:00:01:02:03:36 Connected: yes
[NEW] Primary Service
        /org/bluez/hci0/dev_C5_00_01_02_03_36/service000a
        00001801-0000-1000-8000-00805f9b34fb
        Generic Attribute Profile
[NEW] Characteristic
        /org/bluez/hci0/dev_C5_00_01_02_03_36/service000a/char000b
        00002a05-0000-1000-8000-00805f9b34fb
        Service Changed
[NEW] Descriptor
        /org/bluez/hci0/dev_C5_00_01_02_03_36/service000a/char000b/desc000d
        00002902-0000-1000-8000-00805f9b34fb
        Client Characteristic Configuration
[CHG] Device C5:00:01:02:03:36 UUIDs: 00001800-0000-1000-8000-00805f9b34fb
[CHG] Device C5:00:01:02:03:36 UUIDs: 00001801-0000-1000-8000-00805f9b34fb
[CHG] Device C5:00:01:02:03:36 UUIDs: 00001812-0000-1000-8000-00805f9b34fb
[CHG] Device C5:00:01:02:03:36 ServicesResolved: yes
[CHG] Device C5:00:01:02:03:36 Paired: yes
Pairing successful
[CHG] Device C5:00:01:02:03:36 Icon is nil
[CHG] Device C5:00:01:02:03:36 Appearance is nil
[PYKB 1]# trust C5:00:01:02:03:36
[CHG] Device C5:00:01:02:03:36 Trusted: yes
Changing C5:00:01:02:03:36 trust succeeded
[PYKB 1]# connect C5:00:01:02:03:36
Attempting to connect to C5:00:01:02:03:36
Connection successful
[PYKB 1]# info C5:00:01:02:03:36
Device C5:00:01:02:03:36 (random)
        Name: PYKB 1
        Alias: PYKB 1
        Paired: yes
        Trusted: yes
        Blocked: no
```

```
        Connected: yes
        LegacyPairing: no
        UUID: Generic Access Profile    (00001800-0000-1000-8000-00805f9b34fb)
        UUID: Generic Attribute Profile (00001801-0000-1000-8000-00805f9b34fb)
        UUID: Human Interface Device    (00001812-0000-1000-8000-00805f9b34fb)
[PYKB 1]# quit
pi@raspberrypi:~ $
```

FAQ

1. 怎么使用最新代码

   下载仓库的代码，把仓库中的整个 `keyboard` 目录拷贝到键盘U盘中的 `lib` 目录下(完成后有 `lib/keyboard/model/m60.py` 的文件结构)，并且将仓库中的 `code.py` 覆盖U盘根目录下的 `code.py`

2. 键盘没有RGB背光灯效

   可以参考第1条，使用最新代码。更新玩最新代码后，可以用 b + tab 切换灯效，b + caps 开关背光

3. 更改代码后，键盘没有反应

   a. 先确定更改的代码有没有语法错误； b. 可以用 putty 打开键盘模拟出来的串口，查看信息，调试代码； c. 恢复出产设置

进阶开发

入坑指南

通过修改 `code.py` 就可以实现大量功能，如果 `code.py` 能满足你的配置需求，可以就此止步。

入坑

入坑有风险，可能会导致你的键盘变成砖

入坑指南对你有以下要求：

- 会Python语言
- 熟练掌握键盘的恢复出厂设置功能
- 不怕折腾

指南

想进一步扩展键盘功能，可以修改python-keyboard仓库中的 `keyboard` 库。键盘默认使用固件 `PYKB` 库，而 `PYKB` 库是 `keyboard` 库一个只读版本，这里要切换到 `keyboard` 库：

1. 将python-keyboard仓库中整个 `keyboard` 文件夹复制到键盘U盘的 `lib` 目录, 完成后有 `lib/keyboard/model/m60.py` 的文件结构
2. 将 `code.py` 中 `from PYKB import *` 改成为 `from keyboard import *`

完成以上2个步骤，就可以通过修改U盘中 `lib/keyboard` 中的文件，任意定制键盘。

若出现难以恢复的错误，请尝试恢复出厂设置。

调试

M60键盘，除了自带HID设备、U盘储存设备，还有一个USB虚拟串口。可以用 putty、terminal-s等串口工具打开这个虚拟串口（波特率等参数可任意设置）。串口中可以查看键盘的打印信息，或者用 `Ctrl` + `c` 进入CircuitPython的REPL，在命令行中运行Python代码。

避免在REPL里面用M60按 `Ctrl` + `c` 中断M60自己的运行，需要另一个键盘来操作

硬件信息

主模块

M.2 KEY-E型M.2模块，板载nRF52840，为Arm Cortex-M4F, 64MHz, 256KB RAM, 1MB FLASH, 8MB QSPI Flash

电池接口

JST 1.25mm 3-Pin，又称为 ZH 1.25mm 3-Pin

尺寸

285 mm x 94.6 mm

矩阵键盘IO信息

	1	2	3	4	5	6	7	8
ROW	P0_05	P0_06	P0_07	P0_08	P1_09	P1_08	P0_12	P0_11
COL	P0_19	P0_20	P0_21	P0_22	P0_23	P0_24	P0_25	P0_26

开机

- USB通电时，键盘直接启动
- 电池供电时，按住键盘背面的ON/OFF键启动键盘
- IO口 `P0_28` 输出为 0，则电源保持使能

电源键

键盘背面的ON/OFF键连接的IO是`P0_27`。

LEDs on M.2

LEDs	Pin
Red LED	P0_30
Green LED	P0_29
Blue LED	P0_31

RGB矩阵灯

RGB矩阵灯采用IS32FL3733驱动：

name	Pin	note
Power	P1_04	1: on, 0: off
I2C SDA	P1_05	
I2C SCL	P1_06	
Interrupt	P1_07	

电池

name	Pin	note
充电检测	P0_03	0: charging
电池电压	P0_02	AIN0