





目录 Contents

- 第一部分  Python的常量、变量和数据类型
- 第二部分  Python的运算符和表达式
- 第三部分  计算器的基本原理
- 第四部分  计算器的代码实现



第一部分

Python的常量、变量和数据类型

■ 常量

- 一个字面意义上的常量的例子是如同5、1.23、9.25e-3这样的数，或者如同‘This is a string’、‘It’s a string!’这样的字符串。
- 它们被称作字面意义上的常量，因为它们具备“字面”的意义——按照它们的字面意义使用它们的值。例如数2总是代表它自己，而不会是别的什么东西——它是一个常量，不能改变它的值。因此，所有这些都被称常量。

```
x = 38 # 38是常量
```

```
str = 'hello world' # 'hello world'是常量
```

■ 变量

- 仅仅使用字面意义上的常量很快就会不能满足我们的需求——我们需要既可以储存信息又可以对它们进行操作（改变它的内容）。这是为什么要引入 *变量*。
- 变量的值可以*变化*，即可以使用变量存储任何东西。变量只是计算机中存储信息的一部分内存。与字面意义上的常量不同，需要一些能够访问这些变量的方法，因此要给变量命名。

```
x = 38 # x是变量  
y = x - 4 # x,y都是变量  
str = 'hello world' # str是变量
```

标识符

- 变量是标识符的例子。标识符是用来标识 *某样东西* 的名字。在命名标识符的时候，要遵循这些规则：
 - 标识符的第一个字符必须是字母表中的字母（大写或小写）或者一个下划线（‘_’）。
 - 标识符名称的其他部分可以由字母（大写或小写）、下划线（‘_’）或数字（0-9）组成。
 - 标识符名称是对大小写敏感的。例如，`myname`和`myName`不是一个标识符。注意前者中的小写`n`和后者中的大写`N`。
 - 有效标识符名称的例子有：`i`、`__my_name`、`name_23`和`a1b2_c3`。
 - 无效标识符名称的例子有：`2things`、`this is spaced out`和`my-name`。

■ 关键字

- 关键字是Python里事先定义的，有特别意义的标识符，有时又叫保留字，是特别意义的变量。

and	del	from	not	while
as	elif	global	or	with
assert	else	if	pass	yield
break	except	import	print	
class	exec	in	raise	
continue	finally	is	return	
def	for	lambda	try	

■ 数据类型

- Python的数据类型主要包括数值类型、字符串类型、布尔类型、列表类型、字典类型和元组类型，本章我们主要将前三种类型。

1

数值类型

2

字符串类型

3

布尔类型

■ 数值类型

- Python的常规数值类型包括整数、浮点数、长整型、有虚部的复数等。

① 整数类型示例

```
a = 15  
b = 15 * 4 + 20 - 10  
c = 15/3  
print (a,b,c)
```



运行结果: 15 70 5.0

② 浮点数类型示例

```
a = 15.8  
b = 15.8 * 20  
print (a,b)
```



运行结果: 15.8 316.0

数值类型

- Python的常规数值类型包括整数、浮点数、长整型、有虚部的复数等。

③ 长整型类型示例

```
a = 3457475  
b = 45555000  
c = a + b  
print (a,b,c)
```



```
运行结果:  
3457475  
45555000  
49012475
```

④ 复数类型示例

```
a=1.5+0.5j  
b=a.real  
c=a.imag  
print (a,b,c)
```



```
运行结果:  
(1.5+0.5j) 1.5 0.5
```

■ 字符串类型

- 字符串是一个字符串序列，序列中的元素包含了一个从左到右的顺序序列。其中的元素根据它们的相对位置进行存储和读取的。

① 单字符使用方法示例

```
a = 'm'  
b = 'a'  
c = 'n'  
d = a + b + c  
print (a,b,c,d)
```



运行结果: m a n man

字符串类型

- 字符串是一个字符串序列，序列中的元素包含了一个从左到右的顺序序列。其中的元素根据它们的相对位置进行存储和读取的。

② 字符串使用方法与单字符相同。(注: ' ' 或 " " 对字符内容没有影响)

```
a = "butter"  
b = 'fly'  
c = a + b  
print (a,b,c)
```



运行结果: butter fly butterfly

■ 布尔类型

- 二进制数的“0或1”与逻辑值“真或假”相对应，用户可以通过布尔值判断变量的真假，以此来控制程序的路径走向。

```
a = False;  
print (a)  
b = True  
print (b)
```



运行结果: False True

■ 布尔类型

- 将变量做为参数传给bool方法，返回True 或者 False。空值或者None（Python中类似其它语言Null 或者 Nil 的值）都会被认为是 False，而其它情形则被认为是 True。

```
a = bool("")  
b = bool('hello')  
c = None  
d = bool(c)  
e = bool(1)  
print (a,b,c,d,e)
```



运行结果: False True None
False True



第二部分 Python的运算符和表达式

运算符和操作数

- 编写的大多数语句（逻辑行）都包含**表达式**。一个简单的表达式例如 $2 + 3$ 。一个表达式可以分解为运算符和操作数。
- **运算符**的功能是完成某件事，它们由如“+”这样的符号或者其他特定的关键字表示。运算符需要数据来进行运算，这样的数据被称为 **操作数**。在这个例子中，2和3是操作数。
- 在Python中，表达式可以作为语句，但表达式结果不会存储。

运算符及其用法

运算符	名称	说明	例子
+	加	两个对象相加	3 + 5得到8。'a' + 'b'得到'ab'。
-	减	得到负数或是一个数减去另一个数	-5.2得到一个负数。50 - 24得到26。
*	乘	两个数相乘或是返回一个被重复若干次的字符串	2 * 3得到6。'la' * 3得到'lalala'。
**	幂	返回x的y次幂	3 ** 4得到81（即3 * 3 * 3 * 3）
/	除	x除以y	4/3得到1（整数的除法得到整数结果）。4.0/3或4/3.0得到1.3333333333333333
//	取整除	返回商的整数部分	4 // 3.0得到1.0
%	取模	返回除法的余数	8%3得到2。-25.5%2.25得到1.5
<<	左移	把一个数的比特向左移一定数目（每个数在内存中都表示为比特或二进制数字，即0和1）	2 << 2得到8。——2按比特表示为10
>>	右移	把一个数的比特向右移一定数目	11 >> 1得到5。——11按比特表示为1011，向右移动1比特后得到101，即十进制的5。

运算符及其用法

运算符	名称	说明	例子
&	按位与	数的按位与	5 & 3得到1。
	按位或	数的按位或	5 3得到7。
^	按位异或	数的按位异或	5 ^ 3得到6
~	按位翻转	x的按位翻转是-(x+1)	~5得到6。

运算符	名称	说明	例子
not	布尔“非”	如果x为True，返回False。如果x为False，它返回True。	x = True; not y返回False。
and	布尔“与”	如果x为False，x and y返回False，否则它返回y的计算值。	x = False; y = True; x and y，由于x是False，返回False。在这里，Python不会计算y，因为它知道这个表达式的值肯定是False（因为x是False）。这个现象 称为短路计算。
or	布尔“或”	如果x是True，它返回True，否则它返回y的计算值。	x = True; y = False; x or y返回True。

运算符及其用法

运算符	名称	说明	例子
<	小于	返回x是否小于y。所有比较运算符返回1表示真，返回0表示假。这分别与特殊的变量True和False等价。注意，这些变量名的大写。	5 < 3 返回0（即False）而3 < 5 返回1（即True）。比较可以被任意连接：3 < 5 < 7 返回True。
>	大于	返回x是否大于y	5 > 3 返回True。如果两个操作数都是数字，它们首先被转换为一个共同的类型。否则，它总是返回False。
<=	小于等于	返回x是否小于等于y	x = 3; y = 6; x <= y 返回True。
>=	大于等于	返回x是否大于等于y	x = 4; y = 3; x >= y 返回True。
==	等于	比较对象是否相等	x = 2; y = 2; x == y 返回True。x = 'str'; y = 'stR'; x == y 返回False。x = 'str'; y = 'str'; x == y 返回True。
!=	不等于	比较两个对象是否不相等	x = 2; y = 3; x != y 返回True。

运算符优先级

- 默认地，运算符优先级表决定了哪个运算符在别的运算符之前计算。
- 运算符通常由左向右结合，即具有相同优先级的运算符按照从左向右的顺序计算。例如， $2 + 3 + 4$ 被计算成 $(2 + 3) + 4$ 。一些如赋值运算符那样的运算符是由右向左结合的，即 $a = b = c$ 被处理为 $a = (b = c)$ 。
- 如果想要改变它们的计算顺序，可以使用圆括号。例如，想要在一个表达式中让加法在乘法之前计算，那么就可以写成类似 $(2 + 3) * 4$ 的样子。

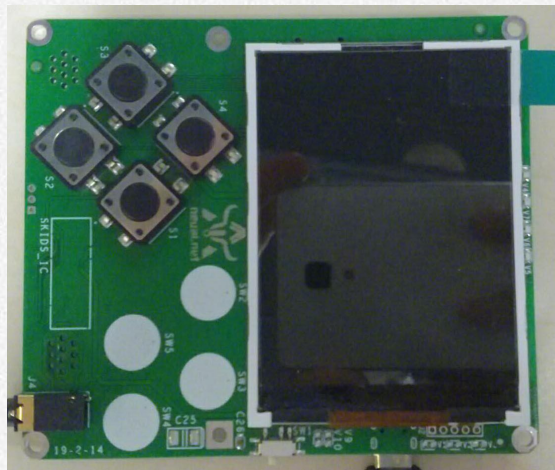
运算符	描述
lambda	Lambda表达式
or	布尔“或”
and	布尔“与”
not x	布尔“非”
in, not in	成员测试
is, is not	同一性测试
<, <=, >, >=, !=, ==	比较
	按位或
^	按位异或
&	按位与
<<, >>	移位
+, -	加法与减法
*, /, %	乘法、除法与取余
+x, -x	正负号
~x	按位翻转
**	指数



第三部分 计算器的基本原理

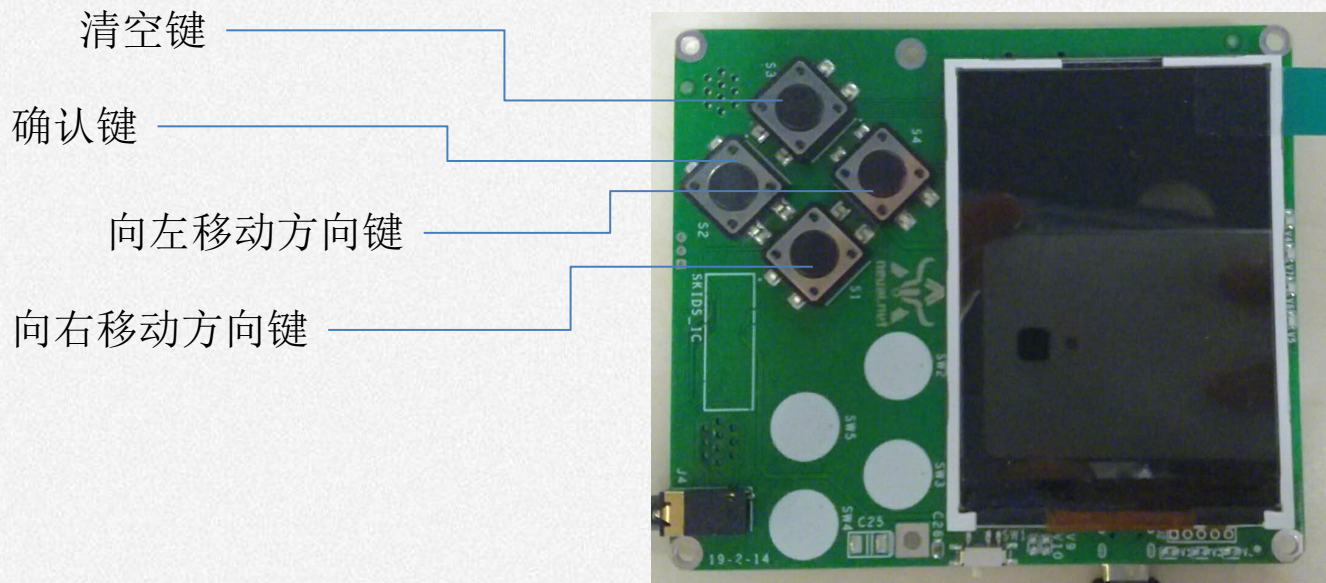
要点

- 计算器是现代进行快速、准确计算的有利工具，在超市、办公等领域都有广泛的应用。
- 我们从现存简单计算器出发，模拟其功能和特点，在skids板上通过屏幕显示出来。



设计思路

- 由于skids只有4个按键，所以我们将计算器键盘显示在屏幕，通过这4个按键来进行选择。



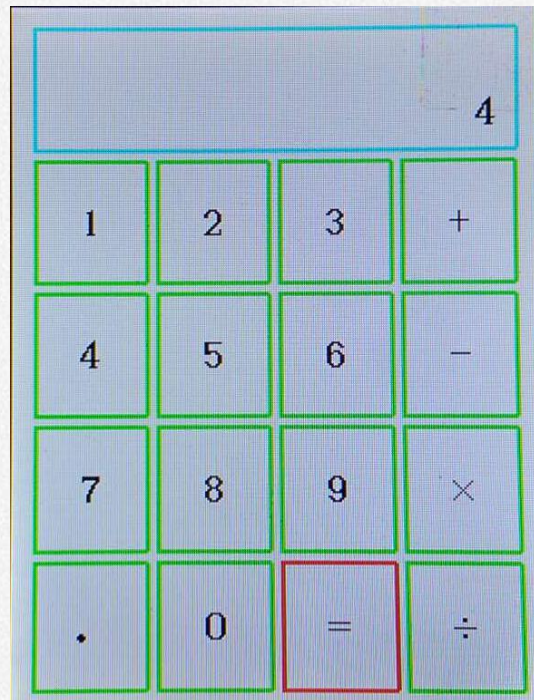
设计思路

- 屏幕布局如图所示，最上方为显示区域，显示输入和计算结果；下方为键盘区域，可通过四个按键进行选择。



基本原理

1. 利用左和下方向键（可循环移动），移动选框到要进行计算的第一个操作数，按下确认键选择，同时显示区显示该数字。
2. 用方向键去选择四个运算符，按下确认键。
3. 用方向键去选择第二个操作数，按下确认键，显示区显示该数字。
4. 用方向键去选择等号键，按下确认键，后台根据操作数和运算符进行计算，并将结果显示在屏幕上。



基本原理

- 当然也要支持浮点运算，通过选择左下角的小数点输入浮点数。
- 同时也可以通过连续选择操作数和操作符进行连续的运算。
- 按下清空键时，将清空所有已经输入的数和运算符，并清空屏幕显示区，使计算器恢复到原始状态，等待重新的输入与计算。





第四部分 计算器的代码实现

- 按键变量

```
# 按键变量
self.keys = [Pin(p, Pin.IN) for p in [35, 36, 39, 34]]
self.keymatch = ["Key1", "Key2", "Key3", "Key4"]
self.keyboard = [[1, 2, 3, 123],[4, 5, 6, 456],[7, 8, 9, 789],[10, 0, 11, 12]]
self.keydict = {1: '1', 2: '2', 3: '3', 123: '+',
                4: '4', 5: '5', 6: '6', 456: '-',
                7: '7', 8: '8', 9: '9', 789: 'x',
                10: '.', 0: '0', 11: '=', 12: '÷'}
self.startX = self.margin * 2
self.startY = self.margin * 2 + self.button_height + self.margin
self.selectXi = 0
self.selectYi = 0
```

获取按键引脚35,36,39,34
定义键盘二位列表keyboard和
它的字典keydict

定义按键1的坐标(startX,startY)
定义按键选择位置

- 布局 and 计算器变量

```
# 布局变量
self.screen_width = 240
self.screen_height = 320
self.margin = 5
self.button_width = (self.screen_width - self.margin * 7) / 4
self.button_height = (self.screen_height - self.margin * 8) / 5
```

屏幕大小为 240×320
定义按键间距为5
然后计算按键大小

```
# 计算器变量
self.l_operand = 0
self.r_operand = 0
self.operator = 123
self.result = 0
self.dotFlag = 0
self.dotLoc = 0
```

定义左右操作数、运算符、运算结果
定义小数点有无标志、小数点位置
并对他们进行初始化

■ 界面布局

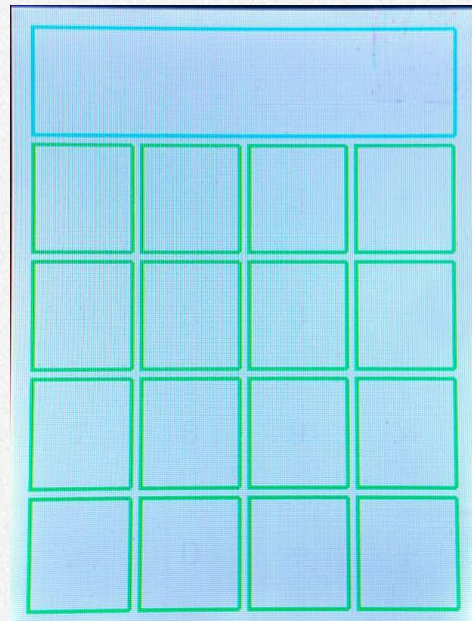
- 定义画矩形的函数，使用`screen.drawLine()`函数通过画直线的方法实现。

```
# 画矩形
def drawRect(self, x1, y1, x2, y2, lineWidth, lineColor):
    x = int(x1)
    y = int(y1)
    w = int(x2 - x1)
    h = int(y2 - y1)
    screen.drawLine(x, y, x + w, y, lineWidth, lineColor)
    screen.drawLine(x + w, y, x + w, y + h, lineWidth, lineColor)
    screen.drawLine(x + w, y + h, x, y + h, lineWidth, lineColor)
    screen.drawLine(x, y + h, x, y, lineWidth, lineColor)
```


■ 界面布局

- 调用之前画矩形的函数画出界面框架。

```
# 画界面
def drawInterface(self):
    # 显示框
    x1 = self.margin * 2
    y1 = self.margin * 2
    x2 = self.screen_width - self.margin * 2
    y2 = self.margin * 2 + self.button_height
    self.drawRect(x1, y1, x2, y2, 2, 0x00ffff)
    # 16个按键
    for i in range(4):
        y = self.startY + i * (self.button_height + self.margin)
        for j in range(4):
            x = self.startX + j * (self.button_width + self.margin)
            self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0x00ff00)
```



■ 界面布局

- 定义显示按键文字的函数，使用`text.draw()`函数，在画好的界面键盘上写上相应的数字和运算符。

```
# 显示按键文字
def showKeyboard(self):
    for i in range(4):
        for j in range(4):
            num = self.keyboard[j][i]
            x = i * (self.button_width + self.margin) + 28
            y = (j + 1) * (self.button_height + self.margin) + 30
            text.draw(self.keydict[num], int(x), int(y), 0x000000, 0xffffffff)
```



■ 界面布局

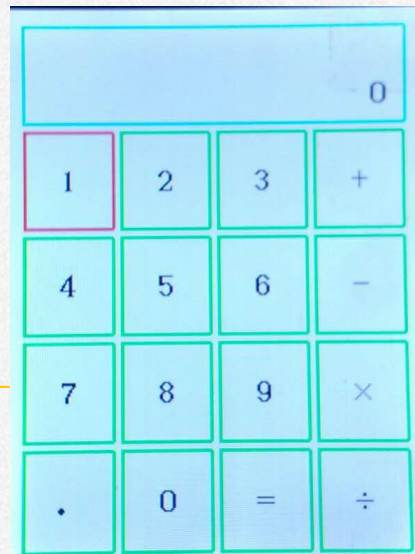
- 按键选择初始化，选中按键1，屏幕显示区显示0。

```
# 按键选择初始化
def selectInit(self):
    # 变量初始化
    self.selectXi = 0
    self.selectYi = 0

    self.l_operand = 0
    self.r_operand = 0
    self.operator = 123
    self.result = 0
    self.dotFlag = 0
    self.dotLoc = 0
```

```
# 显示初始化
x = self.margin * 3
y = self.button_height - self.margin * 3
text.draw('      0', int(x), int(y), 0x000000, 0xffffffff)
```

```
# 选择初始化
x = self.startX
y = self.startY
self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0xff0000)
```



■ 计算器功能实现

- 实现四则运算、浮点运算、连续计算。

```
# 计算器算法
def sendData(self, num):
    # 数字0-9
    if num < 10:
        if self.operator == 11:
            self.r_operand = 0
            self.operator = 123
        if self.dotFlag == 0:
            self.r_operand = self.r_operand * 10 + num
        else:
            self.dotLoc = self.dotLoc + self.dotFlag
            self.r_operand = self.r_operand + num / (10 ** self.dotLoc)
        self.result = self.r_operand
    # 小数点.
    elif num == 10:
        if self.dotFlag == 0:
            self.dotFlag = 1
```

```
# 等号=
elif num == 11:
    self.dotFlag = 0
    self.dotLoc = 0
    self.r_operand = self.calculate(self.l_operand, self.operator, self.r_operand)
    self.l_operand = 0
    self.operator = num
    self.result = self.r_operand
# 运算符+ - * /
elif num > 11:
    self.dotFlag = 0
    self.dotLoc = 0
    self.l_operand = self.calculate(self.l_operand, self.operator, self.r_operand)
    self.r_operand = 0
    self.operator = num
    self.result = self.l_operand
else:
    print('input error')
```


■ 按键响应

- 右移按键响应，先取消前一个选择（用原色重新画一下边框），再选择右边一个按键（用另一种颜色画一下边框）。

```
# 按键事件处理
def keyboardEvent(self, key):
    # 右移选择键
    if self.keymatch[key] == "Key1":
        # 取消前一个选择
        num = self.keyboard[self.selectYi][self.selectXi]
        x = self.selectXi * (self.button_width + self.margin) + self.startX
        y = self.selectYi * (self.button_height + self.margin) + self.startY
        self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0x00ff00)
        # 选择右边一个
        self.selectXi = (self.selectXi + 1) % 4
        num = self.keyboard[self.selectYi][self.selectXi]
        x = self.selectXi * (self.button_width + self.margin) + self.startX
        self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0xff0000)
```

■ 按键响应

- 下移按键响应，先取消前一个选择（用原色重新画一下边框），再选择下边一个按键（用另一种颜色画一下边框）。

```
# 纵向移动键
elif self.keymatch[key] == "Key2":
    # 取消前一个选择
    num = self.keyboard[self.selectYi][self.selectXi]
    x = self.selectXi * (self.button_width + self.margin) + self.startX
    y = self.selectYi * (self.button_height + self.margin) + self.startY
    self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0x00ff00)
    # 选择右边一个
    self.selectYi = (self.selectYi + 1) % 4
    num = self.keyboard[self.selectYi][self.selectXi]
    y = self.selectYi * (self.button_height + self.margin) + self.startY
    self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0xff0000)
```

■ 按键响应

- 确认按键响应，先通过当前坐标位置获取所选按键传给计算器核心算法，并将结果显示在屏幕上（先清楚之前的内容，再写新内容）。

```
# 确认键
elif self.keymatch[key] == "Key3":
    num = self.keyboard[self.selectYi][self.selectXi]
    self.sendData(num)
    # 清空显示区
    x = self.margin * 3
    y = self.button_height - self.margin * 3
    text.draw('      ', int(x), int(y), 0x000000, 0xffffffff)
    # 显示结果
    results = str(self.result)
    length = len(results)
    if length >= 13:
        length = 13
    x = self.screen_width - self.margin * 3 - 16 * length
    y = self.button_height - self.margin * 3
    text.draw(results[0:13], int(x), int(y), 0x000000, 0xffffffff)
```


■ 按键响应

- 清空按键响应，先取消当前的按键选择，并调用按键初始化函数进行复位，重新开始计算。

```
# 清空键
else:
    # 取消前一个选择
    num = self.keyboard[self.selectYi][self.selectXi]
    x = self.selectXi * (self.button_width + self.margin) + self.startX
    y = self.selectYi * (self.button_height + self.margin) + self.startY
    self.drawRect(x, y, x + self.button_width, y + self.button_height, 2, 0x00ff00)
    # 按键选择初始化
    self.selectInit()
```

- 通过循环不断检测4个按键引脚电平，按键按下时电平变低，然后将序号传给按键事件处理函数进行响应的处理。

```
# 开始运行
def start(self):
    while True:
        i = 0
        j = -1
        for k in self.keys:
            if (k.value() == 0):
                if i != j:
                    j = i
                    self.keyboardEvent(i)
                i = i + 1
            if (i > 3):
                i = 0
        time.sleep_ms(130) # 按键去抖
```



THANK YOU

牛艾科技