第 3 章 编程基础

学习目标

本章重点学习控制程序流程，以及循环和分支结构的编程方法。

学习要求

（1）了解变量、值、类型、结构体、随机函数的基本概念。

（2）掌握 while 和 for 循环、if 语句的程序编写。

在编写复杂程序时，需要用到变量并进行程序流程控制，控制程序流程包括循环和分支。

3.1 变　量

常量是在某一变化过程中，始终保持不变的量，例如一天有 24 小时、圆周率约为 3.14 等。变量是在某一变化过程中，可以取不同数值的量。例如，用公式 $S=\pi r^2$ 求各种半径的圆面积，这里 π 是常量，S 和 r 是变量。

在 Mu 编辑器中输入

```
from microbit import *
```

单击 REPL 按钮，在 REPL 窗口中输入语句

```
score=0
```

表示这是一个名字为 score，值为 0 的变量。在此之后，Python 只要看到 score，就会用值 0 来替换 score。

继续输入

```
print (score)
```

结果如图 3.1 所示。

Python 是顺序执行命令的，在使用变量 score 之前必须先给它赋值，否则会报错。

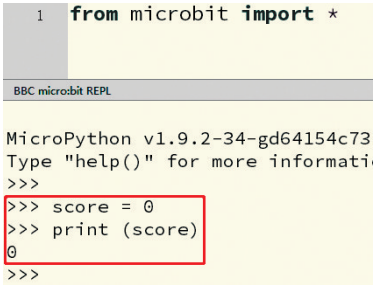


图 3.1　变量 score

如果想改变变量 score 的值，只需要给它赋一个新值，如 score＝1，当再次执行 print (score)后，结果如图 3.2 所示。

```
BBC micro:bit REPL

MicroPython v1.9.2-34-
Type "help()" for more
>>>
>>> score = 0
>>> print (score)
0
>>> score = 1
>>> print (score)
1
>>>
```

图 3.2　赋新值

小贴士

变量的概念基本上与初中代数的方程变量是一致的，只是在计算机程序中，变量不仅可以是数字，还可以是任意数据类型。

变量名必须是大小写英文、数字和下画线的组合，但不能用数字开头，不能使用 Python 关键字(如 if、for 等)。

Python 的命名习惯是使用小写字母，用下画线将单词分开，如 high_score＝100。变量的值可以是数字，也可以文字。甚至可以把同一个变量轮换赋值成数字和文字，如图 3.3 所示。当然，变量的当前值只能是一种类型。

```
>>> high_score = 100
>>> print (high_score)
100
>>> high_score = "math"
>>> print (high_score)
math
>>>
```

图 3.3　变量的值为数字或文字

3.2　值和类型

当人们看到数字 8 时，不会关心它究竟是文字，还是数字。在 Python 中，每个数据都有特定的类型，这样 Python 才知道该如何处理。通过 type()函数就可以看到 Python 数据的类型，如图 3.4 所示。

```
>>> type (8)
<class 'int'>
>>> type ("8")
<class 'str'>
```

图 3.4　数据类型

图 3.4 中，第 1 行的 8 是 int(整数 integer 的简写)数据类型，第 3 行中的 8 是 str(字符 string 的简写)数据类型，Python 认为整数 8 和字符 8 是不同的，它们的运算结果如图 3.5 所示。

图 3.5 中，第 1 行将两个数字 8 加在一起，而第 3 行却是将两个字符"8"合并在一起。由此可见，区分值的类型非常重要，如果出错，将会得到非常有意思的结果。图 3.6 显示更多的类型。

```
>>> 8+8
16
>>> "8"+"8"
'88'
```

图 3.5　不同数据类型的运算

```
>>> type (8.0)
<class 'float'>
>>> type (8>9)
<class 'bool'>
```

图 3.6　浮点数和布尔类型

图 3.6 中，第 1 行输出的是 float(一个浮点数表示一个实数，小数点位置不固定)，第 3 行输出的是 bool(布尔类型，只有两个值：True 和 False)。

1. 数值

数据的具体类型决定了 Python 可以执行哪些操作，数值(包括 int 和 float 类型)可以有两种操作类型：比较和数值操作。

(1) 比较。比较需要两个操作数，返回值为 bool 型，如表 3.1 所示。

表 3.1　数值类型的比较操作

操　作	含　义	示　例
<	小于	9<8(False)
>	大于	9>8(True)
==	等于	9==9(True)
<=	小于或等于	9<=9(True)
>=	大于或等于	9>=10(False)
!=	不等于	9!=10(True)

可以在 Python 解释器中输入任何操作符进行验证，如图 3.7 所示。

(2) 数值操作。数值操作返回一个数值类型，如表 3.2 所示。

表 3.2　数值操作

操　作	含　义	示　例
+	加	2+2==>4
-	减	3-2==>1
*	乘	2*3==>6
/	除	10/5==>2
%	求余	5%2==>1
**	乘方	4**2==>16
int()	转换为 int 型	int(3.2)==>3
float()	转换为 float 型	float(3)==>3.0

在程序中使用数值运算，通常都将其返回值赋值给某个变量，如图 3.7 所示。

```
>>> 9>=8
True
>>> 9!=10
True
>>> number_1 = 3
>>> print (number_1)
3
>>> number_2 = number_1**2
>>> print (number_2)
9
```

图 3.7　操作符验证与数值操作

2. 字符串

字符串类型可以保存任何文字，包括单个数据和一组字母。创建字符串只需要将数据用“”或者“" "”括起来即可。在 Python 中，两者都可以。首选后者，因为它可以处理含有“”的字符串。

小贴士

在 Python 3 中，字符串的编码格式为 Unicode，因此 Python 的字符串支持多种语言。

当源代码中包含中文时，就必须指定保存为 UTF-8 编码，通常在文件开头写上以下两行：

```
#!/usr/bin/env python3
#-*-coding: utf-8-*-
```

其中，第一行注释是为了告诉 Linux 或 Mac OS X 系统，这是一个 Python 可执行程序。第二行注释是为了告诉 Python 解释器，按照 UTF-8 编码读取源代码，否则在源代码中写的中文输出可能会有乱码。

此外，还要确保文本编辑器正在使用 UTF-8 编码。

与数值类型相似，Python 提供了一些字符串操作方法，如表 3.3 所示。图 3.8 是在 Python 解释器中的部分结果。

表 3.3　字符串操作

操　作	含　义	示例
string[x]	获取第 $x+1$ 个字符	"abcde"[1]==>"b"
string[x:y]	获取所有从 $x+1$ 到 y 的字符	"abcde"[1:3]==>"bc"
string[:y]	获取从字符串开始到第 y 个的字符	"abcde"[:3]==>"abc"
string[x:]	获取从第 $x+1$ 个开始到字符串结束的字符	"abcde"[3:]==>"de"
len(string)	返回字符串长度	len("abcde")==>5
string+string	合并两个字符串	"abc"+"def"==>"abcdef"

注：在 Python 中，计数从 0 开始，所以对应人类语言变为 $x+1$。

```
>>> "abcde"[1:3]
'bc'
>>> "abcde"[:3]
'abc'
>>> "abcde"[3:]
'de'
```

图 3.8 字符串操作验证

3. 布尔值

布尔类型非常简单，只有 True 和 False 两种取值。

小贴士

在 Python 中，这两个值的首字母要大写，并且不需要任何引号。

这两个值通常不存在变量中。

它们通常用于条件语句（如 if）的判断条件中，其主要操作符是与（and）、或（or）和非（not）。

- 非：简单地取值转换。
- 与：需要两个操作数，如果两个数都为真，则返回真，否则，返回假。
- 或：也需要两个操作数，如果两个数中任何一个为真，则返回真。

True 和 False 的操作结果如图 3.9 所示。

```
>>> not True
False
>>> not False
True
>>> True and False
False
>>> True and True
True
>>> False and False
False
>>> True or False
True
>>> True or True
True
>>> False or False
False
```

图 3.9 非与或操作验证

4. 数据类型转换

使用函数 int()、float()和 str()可以转换数据类型。它们分别将其他数据类型转换为整数、浮点数和字符串。但是它们相互之间不能随意转换，如果将浮点数转为整数，Python 将舍去所有小数部分。

当字符串中只有一个字符时，才能转换成数字，而其他类型几乎都可以转换成字符串。

3.3 结 构 体

除了简单数据类型，Python 还允许将数据用不同方式组合起来创建结构体。

1. 列表和元组

最简单的结构体是 sequences(线性结构)，它将信息一个接一个地存储起来，sequences 分为两类：list(列表)和 tuple(元组)。

list(列表)是一种有序的集合，可以随时添加和删除其中的元素。tuple(元组)与 list 非常类似，但是 tuple 一旦初始化就不能修改。

用“[]”将数字括起来可构成列表，用“()”将数字括起来可构成元组。在结构体名后面跟“[]”，并在其中填入下标就可以访问单个元素。

注意：下标从 0 开始，因此 list_1[0]和 tuple_1[0]可以访问线性结构中的第一个元素。

大多数情况下，列表和元组是相似的，如图 3.10 所示。

```
>>> list_1 = [1,2,3,4]
>>> tuple_1 = (1,2,3,4)
>>> list_1[1]
2
>>> tuple_1[1]
2
```

图 3.10 list 与 tuple 相似处

但是，在更新元素时就会发现列表和元组之间的差别：可以更新列表中的单个元素，但不能更新元组中的单个元素，如图 3.11 所示。

```
>>> list_1[1] = 88
>>> list_1
[1, 88, 3, 4]
>>> tuple_1[1] = 88
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>>
```

图 3.11 list 与 tuple 更新单个元素时的差别

如果想更新元组中的单个元素，可以在一次性覆盖元组中的所有元素时，告诉 Python 将变量 tuple_1 赋一个新值以取代旧值，如图 3.12 所示。

字符串的操作符可以用于列表和元组，具体方法参考表 3.3，部分操作如图 3.13 所示。

```
>>> tuple_1 = (1,88,3,4)
>>> tuple_1
(1, 88, 3, 4)
```

图 3.12 tuple 更新元素

```
>>> len(list_1)
4
>>> tuple_1[:3]
(1, 88, 3)
```

图 3.13 list 与 tuple 的字符串操作

2. 列表操作方法

列表的操作方法如表 3.4 所示。

表 3.4 列表操作方法

操　作	含　义	示　例
list.append(item)	添加元素到列表尾部	list_1.append(0)
list.extend(list_2)	合并 list_2 到列表尾部	list_1.extend([0,-1])
list.insert(x,item)	插入元素到第 $x+1$ 个位置	list_1.insert(1,88)
list.sort()	排序列表	list_1.sort()
list.index(item)	返回列表中第一次出现该元素的位置	list_1.index(0)
list.count(item)	计算列表中该元素出现的次数	list_1.count(0)
list.remove(item)	删除列表中第一次出现的该元素	list_1.remove(0)
list.pop(x)	返回并删除第 $x+1$ 个元素	list_1.pop(1)

注：在 Python 中，计数从 0 开始，所以对应人类语言多为 $x+1$。

前 7 个操作的结果如图 3.14 所示，它们中有些返回一个值，有些改变了 list_1 的值，有些改变了元素的顺序。

```
>>> list_1 = [1,2,3,4]
>>> list_1.append(0)
>>> list_1
[1, 2, 3, 4, 0]
>>> list_1.extend([0,-1])
>>> list_1
[1, 2, 3, 4, 0, 0, -1]
>>> list_1.insert(1,88)
>>> list_1
[1, 88, 2, 3, 4, 0, 0, -1]
```

```
>>> list_1.sort()
>>> list_1
[-1, 0, 0, 1, 2, 3, 4, 88]
>>> list_1.index(0)
1
>>> list_1.count(0)
2
>>> list_1.remove(0)
>>> list_1
[-1, 0, 1, 2, 3, 4, 88]
```

图 3.14 列表操作结果

pop(x)比较特殊，首先，它返回列表中第 $x+1$ 个位置的元素值，随后从列表中删除该元素，如图 3.15 所示。

```
>>> list_1
[-1, 0, 1, 2, 3, 4, 88]
>>> out = list_1.pop(1)
>>> out
0
>>> list_1
[-1, 1, 2, 3, 4, 88]
```

图 3.15 pop(x)的操作结果

3. 元组操作方法

元组除了不能被修改外，其他与列表非常类似。所有对列表的操作方法，只要不改变元素的值，都可以用于元组；如果改变了元素的值，则出现错误信息，如图 3.16 所示。

```
>>> tuple_1 = (1,2,3,4)
>>> tuple_1.index(1)
0
>>> tuple_1.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'tuple' object has no attribute 'sort'
```

图 3.16 对元组的操作

4. 字典

列表和元组是元素的集合，每个元素都对应了其中的一个下标。在列表["a","b",

"c","d"]中,a 的下标是 0,b 的下标是 1,以此类推。

如果要创建一个数据结构,把学号与名字关联起来,就要用到字典(dictionary,简称 dict)。Python 内置的字典使用键-值(key-value)存储,具有极快的查找速度。

小贴士

dict 的实现原理和查字典是一样的。

假设字典包含了 1 万个汉字,要查某个字时,一种办法是把字典从第一页往后翻,直到找到想要的字为止,这种方法就是在 list 中查找元素的方法,list 越大,查找越慢。第二种方法是先在字典的索引表里(比如部首表)查这个字对应的页码,然后直接翻到该页,找到这个字。无论找哪个字,这种查找速度都非常快,不会随着字典大小的增加而变慢。dict 就是用第二种实现方式。

在 Python 中,可以使用通过"{}"来定义的字典,如图 3.17 所示。

```
>>> name = {"8108311" : "Mary", "8108312" : "John"}
```

图 3.17 定义字典

字典中的元素称为键值对,其中第一部分是键(key),第二部分是值(value)。只需要给定一个新键及其对应的值,就可以在字典中添加新元素,如图 3.18 所示。

```
>>> name["8108310"] = "Hein"
>>> name
{'8108312': 'John', '8108310': 'Hein', '8108311': 'Mary'}
```

图 3.18 添加新元素

小贴士

dict 的特点如下:

- 查找和插入的速度极快,不会随着 key 的增加而变慢。
- 需要占用大量的内存,内存浪费多。

list 的特点如下:

- 查找和插入的时间随着元素的增加而增加。
- 占用空间小,浪费内存很少,所以 dict 是用空间来换取时间的一种方法。

5. 集合

与列表和元组使用下标、字典使用键不同,Python 的集合(set)允许将一堆数据放在一起而不用指定下标或序号。

集合 set 和字典 dict 的唯一区别仅在于没有存储对应的 value,Python 用于集合的操作方法如表 3.5 所示。

表 3.5 集合操作方法

操　作	含　义
item in set_1	测试给定的值是否在集合中
set_1 & set_2	返回两个集合共有的元素
set_1 | set_2	合并两个集合中的元素
set_1 - set_2	set_1 中存在 set_2 中不存在的元素
set_1 ^ set_2	set_1 或 set_2 中存在的元素，不包括两个集合共有的元素

对两个集合 herbs 和 spices 的上述操作结果如图 3.19 所示。

```
>>> herbs = {'thyme','dill','corriander'}
>>> spices = {'cumin','chilli','corriander'}
>>> "thyme" in herbs
True
>>> herbs & spices
{'corriander'}
>>> herbs | spices
{'chilli', 'cumin', 'corriander', 'dill', 'thyme'}
>>> herbs - spices
{'thyme', 'dill'}
>>> herbs ^ spices
{'chilli', 'cumin', 'dill', 'thyme'}
```

图 3.19 集合操作结果

3.4 控制程序流程

控制程序流程包括循环和分支。

如果需要让程序等待某事件发生，就需要围绕一段代码定义循环，这段代码定义了如何对某些预期事件(例如按下按钮)作出反应。

3.4.1 while 循环

while 循环是一种最简单的循环，关键字 while 用来判断条件是否为真。如果是，它会运行一个代码块，这个代码块称为循环体；如果不是，则中断循环(忽略这个循环体)，程序的其余部分继续执行。

如果条件始终为真，它将一直循环下去，直到条件为假，如图 3.20 所示是一个简单的例子。

```
>>> while True:
...     print("Hein is handsome")
...
Hein is handsome
Hein is handsome
Hein is handsome
```

图 3.20 while 循环

小贴士

条件后面要加上“:”,接下来的一行要缩进,所有缩进部分都属于循环体。

要在 REPL 中运行这段代码,必须在输入 print 语句之后按回车(Enter)键,然后按退格(Backspace)键删掉自动产生的 tab(缩进),最后再按回车键。

图 3.20 中的矩形框部分就是用退格键删掉缩进的部分。

最后按回车键是告诉 Python 循环体结束并执行这段代码。

这段代码会陷入死循环,不断地执行 print()语句。按 Ctrl+C 键可以将其终止。

为了不陷入死循环,通常需要一个或多个变量在循环内部改变判断条件,使程序最终能够跳出循环。

【例 3.1】 while 循环。

编写程序,代码如下:

```
from microbit import *

while running_time()<10000:
    display.show(Image.ASLEEP)

display.show(Image.SURPRISED)
```

小贴士

函数 running_time()的作用是在设备启动后返回时间(单位:毫秒)。

语句 while running_time()<10000 检查运行时间是否小于 1×10^4 ms(10s)。

如果是,则属于事件执行范围,设备将会显示 Image.ASLEEP(睡眠)。

若运行时间大于或等于 1×10^4 ms,即 running_time()小于 1×10^4 ms,while 条件不成立。

在这种情况下,循环终止,程序将会继续执行 while 循环之后的代码块,显示 Image.SURPRISED。

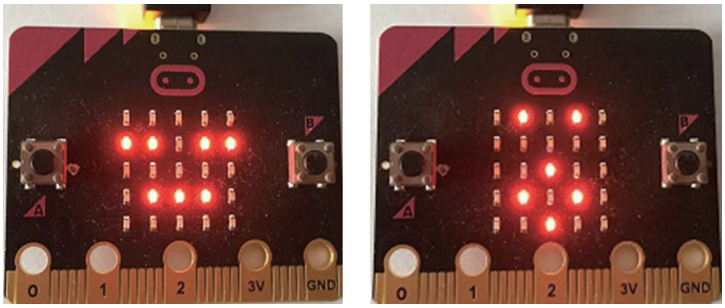
看起来像是设备休眠 10s 后,露出“惊喜”的表情。

单击“刷入”按钮,10s 前与 10s 后在 micro:bit 上显示结果如图 3.21 所示。

【例 3.2】 动画-闪烁的星星。

夏天的晚上,人们经常看到天上的星星闪闪发光,非常好看。联欢晚会的舞台上,灯光配合舞者的动作闪烁的效果赏心悦目。如何通过 micro:bit 实现闪烁的效果呢?

由于在图案库里面没有星星的图案,所以必须自己绘制,星星的图像绘制出来后,如何实现闪烁呢?



(a) 10s前执行循环中语句　　(b) 10s秒后执行循环外语句

图 3.21　while 循环中的条件判断

把闪烁的现象通过慢动作来分析后会发现，星星闪烁实际上就是星星亮一段时间灭一段时间；同时，它不是闪烁一次就结束了，所以就需要使用循环让它一直执行下去。

编写程序，代码如下：

```
from microbit import *

while True:
    star=Image("00900:"
               "09990:"
               "99999:"
               "09990:"
               "00900")
    display.show(star)
    sleep(2000)
    display.clear()
    sleep(2000)
```

小贴士

永久循环语句 while True 的使用：

如果要运行某代码块，while 会检查该事件是否为 True，由于 True 事件永久为 True，这样，一个永久循环就实现了。

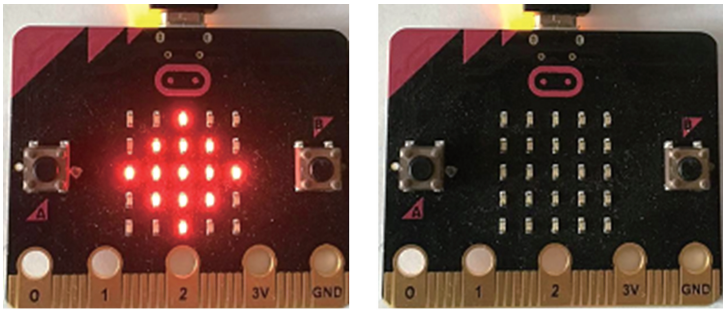
display 的 clear 方法将 LED 显示屏中每个像素的亮度置 0（将灯熄灭）。

单击“刷入”按钮，“闪烁的星星”效果如图 3.22 所示。

复习思考题

（1）如果不用 clear 方法，该如何实现相应的功能呢？

（2）如果不把灯熄灭，而是显示相似的图案，效果又怎么样呢？



(a) 明亮的星星　　(b) 暗淡的星星

图 3.22　永久循环

3.4.2　for 循环

for 循环可以用来遍历数据，它在每次循环中实现对每个数据的处理，如图 3.23 所示。

```
>>> for i in range(1,6):
...     print (i, "times sever is", i*7)
...
1 times sever is 7
2 times sever is 14
3 times sever is 21
4 times sever is 28
5 times sever is 35
```

图 3.23　for 循环代码与执行结果

循环中，rang(x，y)遍历 x～$y-1$ 的每个数据。

rang(x，y，z)的第三个参数 z，可以设定两个连续数字之间的间隔。例如把 range(1,6)改成 range(1,6,2)，它将只计算 1～5 的所有奇数；把 range(1,6)改成 range(2,6,2)，它将只计算 1～5 的所有偶数，如图 3.24 所示。

```
>>> for i in range(1,6,2):
...     print (i, "times sever is", i*7)
...
1 times sever is 7
3 times sever is 21
5 times sever is 35
>>> for i in range(2,6,2):
...     print (i, "times sever is", i*7)
...
2 times sever is 14
4 times sever is 28
```

图 3.24　range 第三个参数的作用

3.4.3　分支语句

Python 使用分支来控制程序流，使其根据不同条件，分别执行不同的代码。分支由 if 语句实现。

小贴士

类似 while 循环，if 语句的执行只需要一个布尔类型的条件，它后面还可以有附加语句，如 elif(else if 的缩写)和 else 语句。

if 语句可以不带 elif 或 else。

如果没有 else 语句，同时判断条件也不成立，Python 就会跳过 if 语句，不执行其中的任何代码。

一个 if 语句最多只执行一段代码，只要 Python 发现条件为真，就执行该段代码并结束整个 if 语句。

如果没有一个条件为真，则执行 else 后面的代码段。

如果希望 MicroPython 对按下按钮事件做出反应，应该把它放入一个永久循环并检查按钮是否 is_pressed(被按下)。

【例 3.3】 指示按钮。

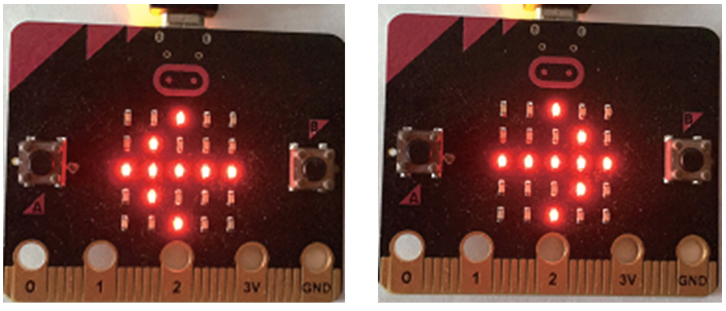
做一个方向指示按钮，实现按下左边的按钮(按钮 A)指向左边，按下右边的按钮(按钮 B)指向右边，不按的时候，没有指示。

编写程序，代码如下：

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.ARROW_W)
    elif button_b.is_pressed():
        display.show(Image.ARROW_E)
```

单击“刷入”按钮，效果如图 3.25 所示。



(a) 按下按钮A　　(b) 按下按钮B

图 3.25　指示按钮

【例 3.4】 按钮检测。

编写程序，显示按下了哪个按钮。

```
from microbit import *

while True:
    if button_a.is_pressed() and button_b.is_pressed():
        display.scroll("AB")
        break
    elif button_a.is_pressed():
        display.scroll("A")
    elif button_b.is_pressed():
        display.scroll("B")
    sleep(100)
```

小贴士

Python 中可以使用逻辑运算符与(and)、或(or)、非(not)来检查多重判断语句。

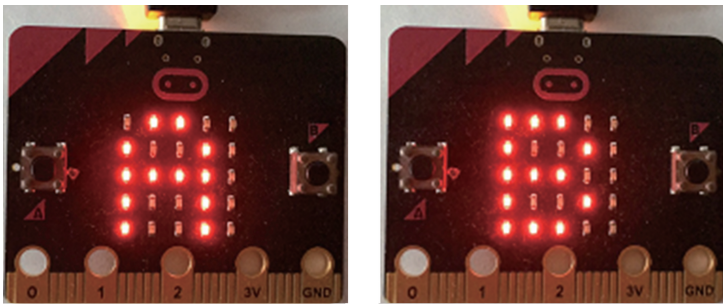
非运算就是简单地转换取值;与运算需要两个操作数,如果两个操作数都为真,则返回真,否则,返回假;或运算也需要两个操作数,如果两个操作数中任何一个为真,则返回真。

程序代码中的 and 表示与运算,即按钮 A 与按钮 B 同时按下。

break 的作用是跳出上一层的循环而执行该循环后面的语句。

通常 break 语句总是与 if 语句连在一起,即满足条件时便跳出循环。

单击“刷入”按钮,分别按下按钮 A 和按钮 B 的效果如图 3.26 所示,当同时按下按钮 A 和 B 时,滚动显示 AB。



(a) 按下按钮A　　(b) 按下按钮B

图 3.26　按钮检测

复习思考题

代码中,将 break 放置在“A”或“B”语句的下面会发生什么情况?

【例 3.5】 忧伤的宠物。

编写程序,代码如下:

```
from microbit import *

while True:
    if button_a.is_pressed():
        display.show(Image.HAPPY)
    elif button_b.is_pressed():
        break
    else:
        display.show(Image.SAD)
display.clear()
```

小贴士

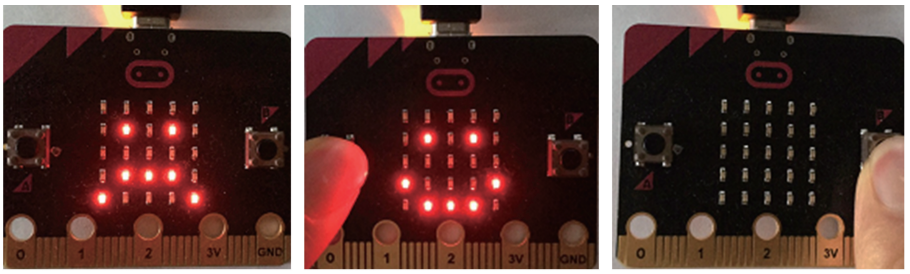
is_pressed 方法仅产生两个结果：True 或 False，按下按钮后，系统要么返回 True，要么返回 False。

这只宠物总是很悲伤，显示“悲伤”的表情，除非按下按钮 A，当按钮 A 被按下时，永久循环显示“高兴”的表情。

当按钮 B 被按下时，通过 break 语句终止该循环。

循环终止后，执行循环外的 display.clear()语句，对显示屏做 clear(清除)。

单击“刷入”按钮，效果如图 3.27 所示。



(a) 不按按钮　(b) 按下按钮A　(c) 按下按钮B

图 3.27　按钮检测

复习思考题

玩过这个游戏后，有什么办法改善这个游戏？

【例 3.6】　倒计时器。

在每年除夕的夜晚，都会有很多地方举行春节倒计时活动；又如，商家经常举办促销活动倒计时以及红绿灯倒计时等。虽然它们表现的方式不太一样，但都属于倒计时器。

想要实现倒计时，就需要一个变量，通过它不断改变数值并显示出来，实现倒计时数的改变。如果倒计时的数超过 1 位数的话，就需要移动显示，不利于观察。下面制作一个简

单的 9～0 的倒计时器。

编写程序，代码如下：

```
from microbit import *

shijian=9
while True:
    display.show(shijian)
    sleep(2000)
    shijian=shijian-1
    if shijian==0:
        display.show(Image.HAPPY)
        sleep(5000)
        shijian=9
```

小贴士

程序开始时设置变量 shijian，并给它赋初始值 9（倒计时从 9 开始）。

在循环中，每隔 2s，变量 shijian 的值减少 1（实现倒计时）。

通过 if 语句判断变量 shijian 的值是否为 0，如果为 0 则显示“笑脸”。

等待 5s 之后，把变量 shijian 重新设置为 9，重新开始倒计时，一直这样循环进行。

单击“刷入”按钮，倒计时从 9 开始，到 0 显示“笑脸”图案，不断循环显示，部分效果图如图 3.28 所示。

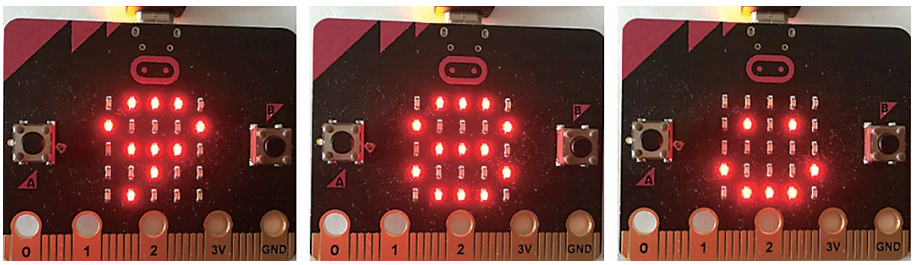


图 3.28　倒计时器的部分状态

复习思考题

（1）如何实现 0～9 的正计时？

（2）如何实现不是开机就从数字 9 开始，而是按下按键 A 之后才会从 9 开始，之后一直重复执行？

3.4.4 循环嵌套

在进行程序编写时，经常会遇到需要同时遍历多种数据的情况，这时就要用到循环嵌套。

小贴士

编写循环嵌套时要注意缩进级别：

第一个循环体的缩进为1，第二个循环体的缩进为2。只有这样，Python才能理解哪些代码属于第几个循环体以及每个循环体在何处结束。

在REPL窗口中输入如图3.29所示的程序，用来找出1～10中的所有素数。

按退格键后再按回车键，结果如图3.30所示。

```
>>> for i in range (1,10):
...     is_prime = True
...     for k in range (2,i):
...         if (i%k) == 0:
...             print (i, " is divisible by ", k)
...             is_prime = False
...     if is_prime:
...         print (i, " is prime ")
...
```

图3.29 找出1～10的所有素数的程序

```
...
1  is prime
2  is prime
3  is prime
4  is divisible by  2
5  is prime
6  is divisible by  2
6  is divisible by  3
7  is prime
8  is divisible by  2
8  is divisible by  4
9  is divisible by  3
>>>
```

图3.30 找出1～10的所有素数

小贴士

运行嵌套循环时可能会使程序变慢：

例如计算3000以内的素数（只需要将上述程序第一行中的10改成3000即可），程序运行就会用非常长的时间。

这是因为外层循环要循环上千次，每次进入内层循环也需要执行很多次。

如果正在做这个实验，会发现整个程序运行起来很慢。

如果不想等待，可以按Ctrl+C键停止运行。通过下面的方法可以改进该程序：首先使用range(1,3000,2)跳过所有的偶数，这样就直接省去一半时间；其次，在if里面增加语句break，一旦发现某个数字是非素数，就使用break跳出循环，继续执行下面一行（if is_prime:）。程序如图3.31所示。

```
>>> for i in range (1,3000,2):
        is_prime = True
        for k in range (2,i):
            if (i%k) == 0:
                print (i, "  is divisible by ", k)
                is_prime = False
                break
        if is_prime:
            print (i, " is prime ")
```

图 3.31　程序优化

3.5　随 机 函 数

随机性意味着无法准确预测。比如，在玩骰子的时候，人们不知道会看到 6 个面上的哪个点数；再如，在抽奖的时候，主持人从抽奖箱里随便抽取一个数字，拿到同样数字的观众就中奖了。这样的数字就是随机数。

如果想让事情偶然发生或者稍微打乱顺序，就需要使用随机函数产生随机行为。MicroPython 的 random 库（模块）可以引入随机性，只要使用代码 import random 即可。下面通过一个简单的例子进行说明。

【例 3.7】　随机数字。

编写程序，代码如下：

```
from microbit import *
import random

display.show(str(random.randint(1, 6)))
```

小贴士

MicroPython 有几个很有用的生成随机数字的方法，包括 random.randint、random.randrange、random.random 等：

random.randint(a，b)：返回两个参数 a、b 之间的自然数 N（$a \leqslant N \leqslant b$）。

random.randrange(N)：返回 0～N 的自然数（不包含 N）。

random.random 生成一个带小数点的数字，返回 0.0～1.0 的浮点数。

随机数种子用于生成伪随机数的初始数值，通过 random.seed 实现。

复习思考题

如何产生大于 1 的随机浮点数？

【例 3.8】　随机骰子。

骰子是中国古代民间娱乐用来投掷的博具，早在战国时期就有。在游戏时经常会用到

骰子，如麻将、牌九。另外，有很多桌游都以骰子的点数来决定任务角色行走的步数。最常见的骰子有 6 个面，它是一个正立方体，每面分别有 1～6 个点（或数字），其相对两面的数字之和为 7。

想要实现随机骰子，首先需要一个能够产生 1～6 的随机数，然后每一个数对应骰子的一面，把对应的图案通过 LED 点阵显示出来；同时，每次产生随机数之前，需要通过按钮被按下来触发实现。

编写程序，代码如下：

```
from microbit import *
import random

number1=Image("00000:"
              "00000:"
              "00900:"
              "00000:"
              "00000")

number2=Image("00000:"
              "00900:"
              "00000:"
              "00900:"
              "00000")

number3=Image("00000:"
              "09000:"
              "00900:"
              "00090:"
              "00000")

number4=Image("00000:"
              "09090:"
              "00000:"
              "09090:"
              "00000")

number5=Image("00000:"
              "09090:"
              "00900:"
              "09090:"
              "00000")

number6=Image("09090:"
              "00000:"
              "09090:"
              "00000:"
              "09090")
```

```
while True:
    if button_a.was_pressed():
        temp=random.randint(1, 6)
        if temp==1:
            display.show(number1)
        elif temp==2:
            display.show(number2)
        elif temp==3:
            display.show(number3)
        elif temp==4:
            display.show(number4)
        elif temp==5:
            display.show(number5)
        elif temp==6:
            display.show(number6)
```

小贴士

通过 Image 自定义 1～6 点的骰子。

设置变量 temp，用于放置产生的随机数。

根据变量 temp 的值，显示对应的骰子。

单击“刷入”按钮，按下按钮 A 产生随机骰子，部分效果图如图 3.32 所示。

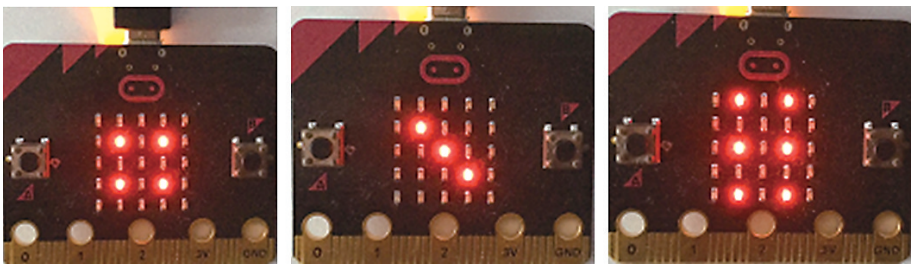


图 3.32　随机掷骰子

复习思考题

（1）如何实现随机显示√和×？

（2）用点阵显示石头剪刀布，然后两位同学进行游戏，看谁获胜的次数多？

【例 3.9】　燃烧的火焰。

编写程序，代码如下：

```
from microbit import *
import random

i=Image("00000:"*5)
```