目錄

MuPython

MuPython 是基于Mu给初学者的 Python 编辑器，可以为「指间ESP」开发板编写 MicroPython，也可以使用标准的Python3编写代码。它旨在使学习体验更加愉快，它使学生能够在早期体验成功，这在你学习任何新知识的时候都很重要。

Mu 是 Nicholas Tollervey 的心血结晶。Nicholas 是一位受过古典音乐训练的音乐家，在担任音乐老师期间，他在职业生涯早期就开始对 Python 和开发感兴趣。他还写了 Python in Education，这是一本可以从 O'Reilly 下载的免费书。

Nicholas 曾经寻找过一个更简单的 Python 编程界面。他想要一些没有其他编辑器（甚至是 Python 附带的 IDLE3 编辑器 ）复杂性的东西，所以他与 Raspberry Pi 基金会（赞助他的工作）的教育总监 Carrie Ann Philbin 合作开发了Mu。

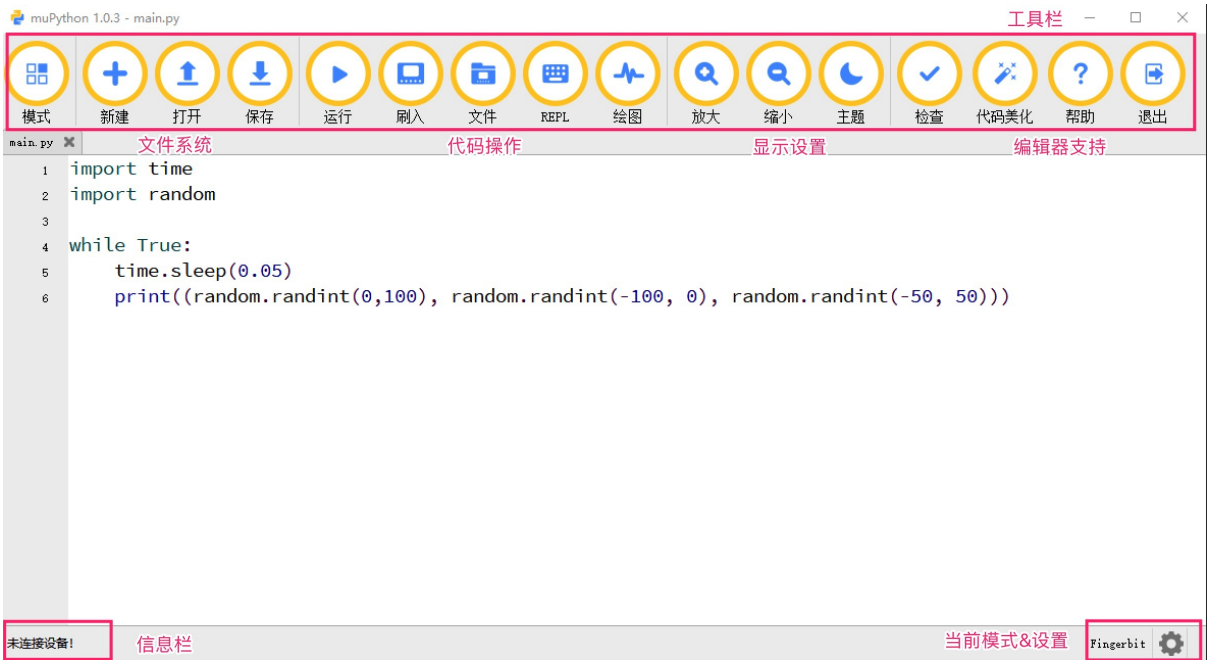
Mu是一个用 Python 编写的开源程序（在 GNU GPLv3 许可证下）。它最初是为 Micro:bit 迷你计算机开发的，但是其他老师的反馈和请求促使他将 Mu 重写为通用的 Python 编辑器。

软件下载

下载地址：https://www.microduino.cn/download/mupython/

系统要求：Win7 +

软件界面



1、工具栏

将鼠标悬停在按钮上（而不是实际点击它），您会看到一个小工具提示弹出按钮，其中包含有关按钮功能的更多信息。试试吧！大多数按钮保持不变，但有些按钮会根据您当时的操作而改变。

模式：

单击它可以更改MuPython的当前模式。每种模式都有不同的功能，具体取决于您想要实现的目标。

文件系统：

- **新建**：创建一个新的空白文件。
- **打开**：打开电脑上的Python文件。
- **保存**：将Python文件保存到电脑上。

代码操作：

具体取决于您当前使用的模式，每种模式都有不同的功能，为您提供了与您编写的代码进行交互的有趣方式。如图所示：

- **运行**：将python代码直接运行可以与「指间ESP」开发板交互。
- **刷入**：将python文件保存到「指间ESP」开发板中。
- **文件**：显示「指间ESP」开发板中的文件列表，可进行打开，删除操作。
- **REPL**：是一种用Python与计算机交谈的交互方式。

- **绘图：**监听有效的数据源（Python程序，REPL中的某些内容或来自连接设备的串行数据）进行绘图。

显示设置：

- **放大：**放大Python代码显示。
- **缩小：**缩小Python代码显示。
- **主题：**三种主题：白天、夜晚、高对比度。

编辑器支持：

- **检查：**分析代码并提出改进方法。
- **代码美化：**代码格式化操作。
- **帮助：**在浏览器中打开帮助文档。
- **退出：**关闭muPython编辑器。

2、信息栏

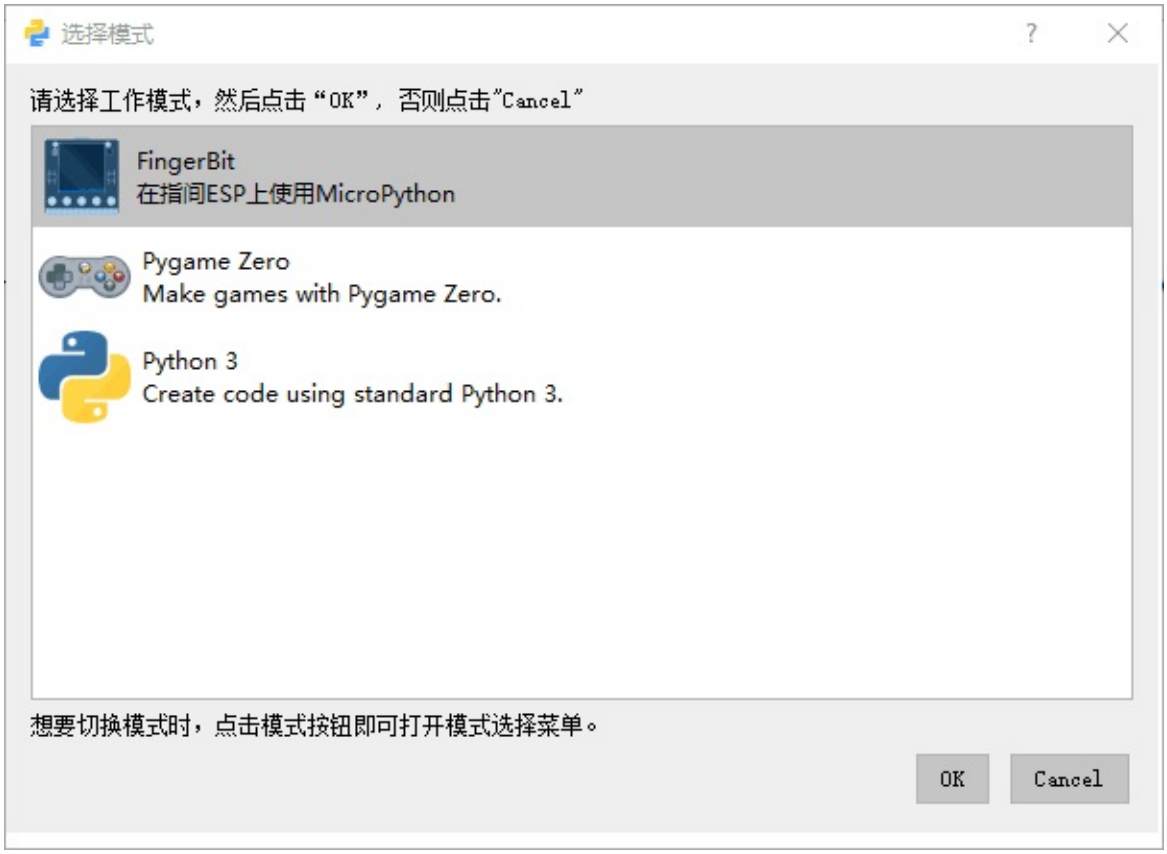
显示硬件连接状态及操作提示

3、显示当前模式及设置按钮

- **当前日志：**显示操作日志，反馈bug时，可以复制日志给开发哥哥。
- **烧录固件：**烧录「指间ESP」固件，烧录会删除开发板内的文件，烧录前需备份开发板的文件。
- **关于：**查看软件版本号等信息。

使用模式

模式可以轻松自定义muPthon的行为，同时简化muPython：而不是试图一次性提供所有可能的功能，模式将相关的特性和功能组织在一起。由于您需要使用不同类型的功能，只需单击muPython中左上角的“模式”按钮即可更改模式。



- **IdeaBit**：「指间ESP」开发板，基于ESP32的开发板，可以运行MicroPython。
- **PyGame Zero**： 编写使用Python的图形游戏。
- **Python 3**：Python 3模式提供了入门所需的工具，包括一个简单易用的可视化调试器，可帮助您观察代码在运行时的运行情况。

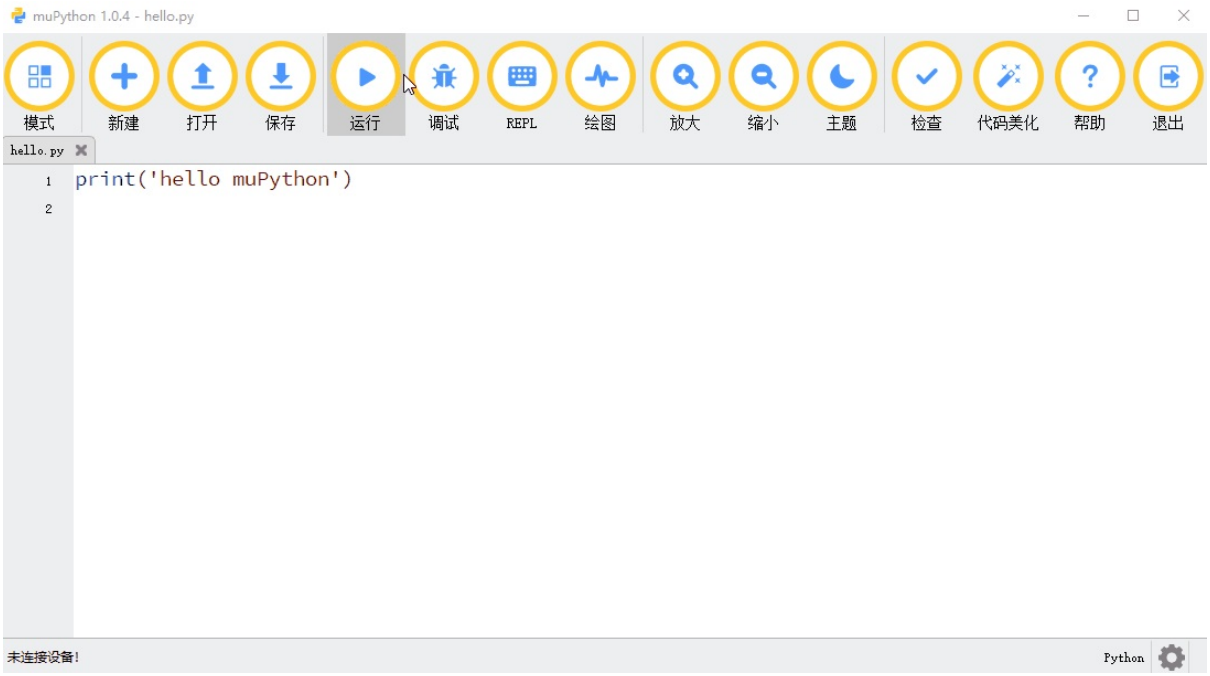
Python3

python 3是世界上最流行的专业编程语言之一。它也很容易学。

python 3模式提供的功能包含在以下4个按钮中：

1、运行

单击“运行”按钮运行当前脚本，任何文本输入或输出都将软件底部窗口中显示。当代码运行时，“运行”按钮变成“停止”按钮。单击“停止”可以退出。



2、调试

单击“调试”将启动可视化调试器。调试器开始运行您的代码（就像“运行”按钮一样），但它以一种特殊的方式运行，允许您暂停代码，查看程序中的状态，并逐步执行代码，这样您就可以按照python解释程序的方式进行操作。如果代码中有错误，这将非常有用。


muPython 1.0.4 - debug.py
模式
新建
打开
保存
运行
调试
REPL
绘图
放大
缩小
主题
检查
代码美化
帮助
退出
debug.py
def add(x, y):
return x+y
values = [
(1, 2),
(2, 3),
(3, 4),
]
for x, y in values:
print(add(x, y))
未连接设备!
Python

3、REPL

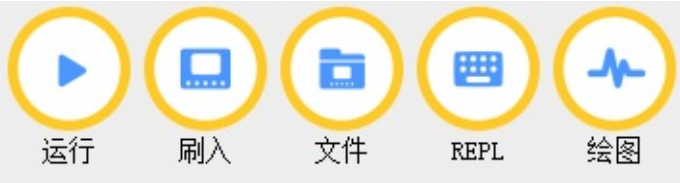
4、绘图

指间ESP

「指间ESP」基于ESP32芯片的一种小型开发板，可以用MicroPython编程。

当「指间ESP」通过USB连接电脑后，muPython会自动连接，并更新连接状态（在界面左下角有提示信息）。

IdeaBit模式提供的功能包含在以下5个按钮中：

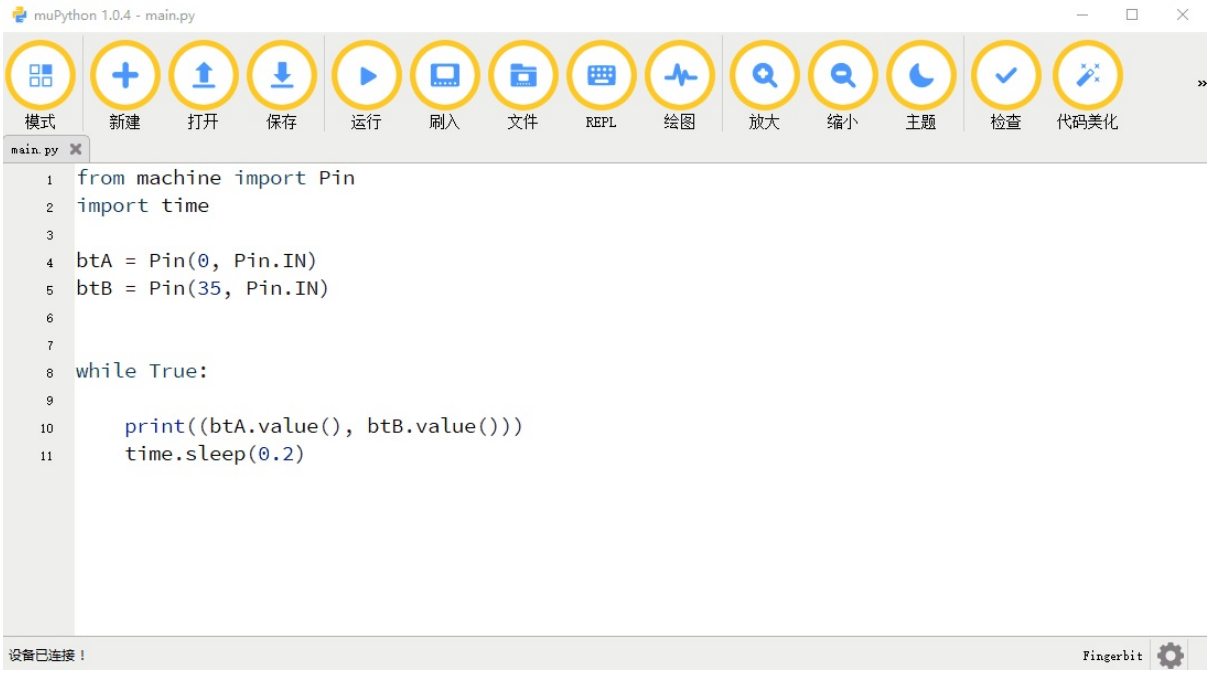


注意：该模式下所有操作都需要「指间ESP」正常连接状态下进行

1、运行

MicroPython程序未刷入「指间ESP」开发板，可以单击“运行”按钮，运行MicroPython程序。

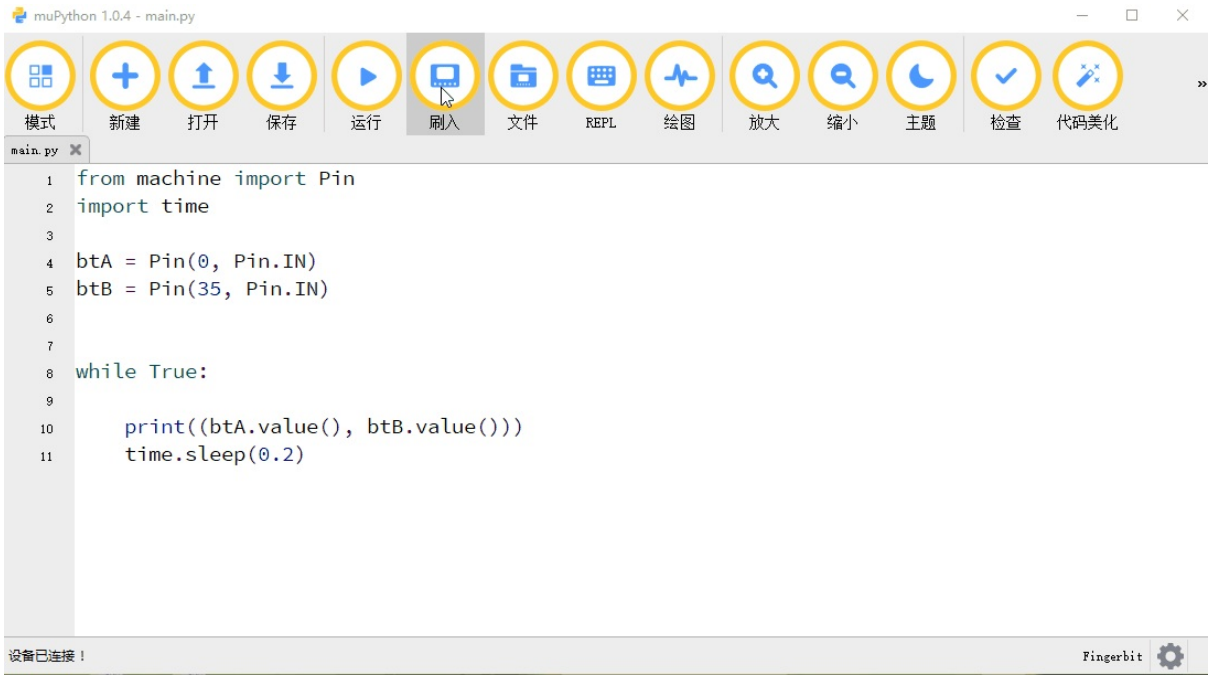
单击“REPL”按钮，用“CTRL+C”快捷键，可以停止当前运行程序。



2、刷入

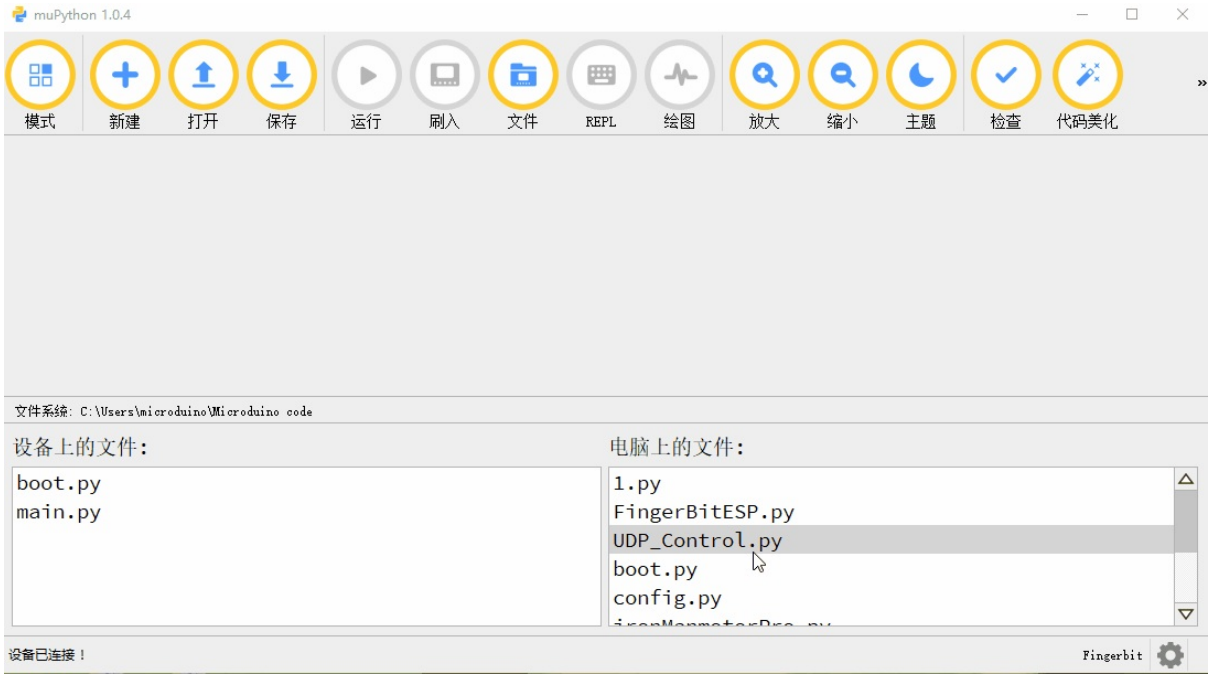
单击“刷入”，可以把当前的MicroPython程序复制到「指间ESP」开发板中。

单击“REPL”按钮，用“CTRL+D”快捷键，可以刷新设备，程序会自动运行。用“CTRL+C”快捷键，可以停止当前运行程序。



3、文件

单击“文件”将打开文本编辑器和页脚之间的两个窗口。左边的窗口列出了「指间ESP」上的所有文件，右边的窗口列出了您计算机上Microduino code目录中的所有文件。在每个文件之间拖动文件以复制它们。要删除「指间ESP」上的文件，请右键单击该文件并选择“删除”。



4、REPL

5、绘图

PyGame Zero

Pygamezero是一个初学者友好的包装器，它围绕着使用Python编写视频游戏的强大PyGame库。用Pygamezero只需几行python代码就可以非常容易地编写一个有趣的游戏。

Pygamezero模式提供的功能包含在以下5个按钮中：

1、Play

单击"Paly"按钮后会变成“Stop”按钮。当游戏开始时，任何文本输入或输出都将软件底部窗口中显示。

最重要的是，pygamezero创建的新游戏窗口将出现在桌面上。这使得你很容易看到，测试和检查你的游戏是如何工作。

2、图片、字体、音效、音乐

“图像”、“字体”、“音效”和“音乐”按钮的工作方式完全相同。

单击这四个按钮中的每一个都会打开一个目录，您应该将这些游戏资源放入其中（只需使用操作系统的文件管理器将其拖放到其中即可）。


muPython 1.0.4 - mygame0.py
模式
新建
打开
保存
Play
图片
字体
音效
音乐
放大
缩小
主题
检查
代码美化
mygame0.py
1 import pygame
2 import os
3 import random
4
5
6 WIDTH = 400
7 HEIGHT = 400
8 GRID_WIDTH = WIDTH // 20
9 FPS = 30
10
11 # define colors
12 WHITE = (255, 255, 255)
13 BLACK = (0, 0, 0)
14 RED = (255, 0, 0)
15 GREEN = (0, 255, 0)
16 BLUE = (0, 0, 255)
17
18 pygame.init()

REPL

REPL是一种用Python与计算机交谈的交互方式。为了使这项工作，计算机做了四件事：

1、Read the user input (你的Python命令).

2、Evaluate your code (弄清楚你的意思).

3、Print any results (你可以看到计算机的响应).

4、Loop back to step 1 (继续第一步).

术语“REPL”是Read，Evaluate，Print和Loop的首字母缩写，因为这正是计算机所做的.....

计算机通过向您显示三个V形符号 `>>>` 或编号提示符 `In [1]:` 来告诉您它正在等待指令。

你在需要“解决问题”时使用REPL，由于您从REPL获得的即时反馈，因此可以轻松地即兴发挥，深入研究计算机正在做什么。

虽然所有REPL都以相同的方式工作，但REPL的特性和功能将根据您目前使用的模式而有所不同。但是，有两个命令适用于所有版本的REPL，这些命令非常有用： `dir` 和 `help` 。

该 `dir` 令告诉你它是什么。如果您自己使用它，它会告诉您Python目前所知道的内容：

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__']
```

虽然这个名称列表目前可能看起来很神秘，但如果你创建一个新变量，你会在Python知道的事物列表中看到它：

```
>>> name = "Nicholas"
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__',
'__package__', '__spec__', 'name']
```

但是 `dir` 可以做得更多！如果你使用Python知道的东西的名称，那么 `dir` 将返回该东西的所有属性。例如， `name` 变量实际上是一个字符串对象，字符串对象具有各种有用的属性。如果我想知道它们是什么，我会将 `name` 对象传递给 `dir` 这样：

```
>>> dir(name)
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__',
'__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__getnewargs__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__iter__', '__le__', '__len__', '__lt__', '__mod__', '__mul__', '__ne__',
'__new__', '__reduce__', '__reduce_ex__', '__repr__', '__rmod__', '__rmul__',
'__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'capitalize',
'casefold', 'center', 'count', 'encode', 'endswith', 'expandtabs', 'find',
```

```
 'format', 'format_map', 'index', 'isalnum', 'isalpha', 'isdecimal', 'isdigit',
 'isidentifier', 'islower', 'isnumeric', 'isprintable', 'isspace', 'istitle',
 'isupper', 'join', 'ljust', 'lower', 'lstrip', 'maketrans', 'partition',
 'replace', 'rfind', 'rindex', 'rjust', 'rpartition', 'rsplit', 'rstrip',
 'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title', 'translate',
 'upper', 'zfill']
```

这有很多属性！鉴于其中一些的名称，它们可能非常有用。例如，假设我想获得该 `name` 对象的大写版本。我注意到 `upper` 返回的列表中有一个属性， `dir` 所以我需要一些方法来检查它的作用。这是 `help` 它自己的地方。如果你传递了Python知道的东西， `help` 它会回复一些有用的文档：

```
>>> help(name.upper)
Help on built-in function upper:

upper(...) method of builtins.str instance
    S.upper() -> str

    Return a copy of S converted to uppercase.
```

第一行告诉您这 `upper` 是内置字符串类的方法。第二行告诉您调用的 `upper` 结果会产生一个新的字符串（ `-> str` ）。最后一行包含一个简单的英文描述 `upper` 。所以，我应该期望它返回值的大写版本 `name` ：

```
>>> name.upper()
`NICHOLAS`
```

只需使用 `dir` 和 `help` 命令，您就可以查询和查询整个Python。

举个例子

让我们假装你需要做一些算术，那么为什么不把REPL用作计算器呢？

```
>>> 1 + 1
2
>>> 10 - 1
9
>>> (5 * 5 + 5 ) / 4
7.5
```

更高级的数学最初可能不会出现。但是，如果我们导入 `math` 模块并使用 `dir` ， `help` 我们很快就会找到我们可能需要的东西：

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos',
```

```
'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log', 'log10', 'log1p',
'log2', 'modf', 'nan', 'pi', 'pow', 'radians', 'sin', 'sinh', 'sqrt', 'tan',
'tanh', 'tau', 'trunc']
>>> help(math.sqrt)
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.

>>> math.sqrt(12345)
111.1080555135405
```

就像在“普通”Python中一样，您可以导入模块，为对象赋值，使用函数和创建循环：

```
>>> import random
>>> names = ['Lisa', 'Kushal', 'Mary', 'Kate', 'Carlos', 'Zander', ]
>>> while True:
...   print(random.choice(names))
...   wait = input()
...
Kate
Kate
Zander
Mary
Zander
Kushal
```

要打破无限循环，你需要按 `CTRL-C` 。这将导致显示如下消息：

```
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
KeyboardInterrupt
```

绘图

绘图，可以轻松地显示代码可能创建的数字数据。

输出采用折线图的形式，时间沿x（水平）轴运行，数据值沿y（垂直）轴绘制。

要在支持它的那些模式下激活绘图，只需单击“绘图”按钮即可打开或关闭它。

当绘图处于活动状态时，它开始监听有效的数据源。这可能是您的Python程序，REPL中的某些内容或来自连接设备的串行数据。

注意：

发出元组的通常模式是循环执行。

您必须在循环中的某处包含一个短暂停顿，以使绘图正常工作。

否则，您的代码可能会使绘图充满数据，它将无法跟上并抱怨。

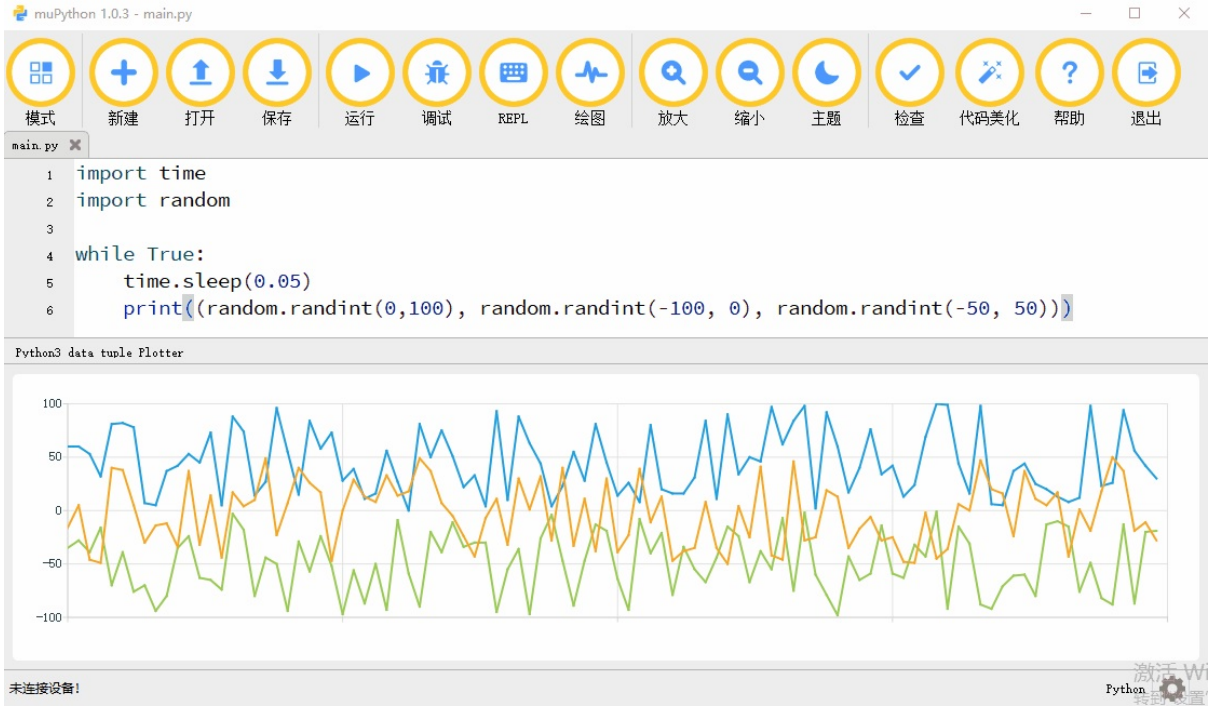
暂停可能非常小，但仍然必须在那里。实现这一目标的最佳方法是使用Python `time.sleep()` （以秒为单位）或 `microbit.sleep()` （以毫秒为单位）来实现此目的。

从一个为绘图创建三个随机数据源的示例中可以清楚地了解上述所有内容：

示例：

```
import time
import random

while True:
    # Just keep emitting three random numbers in a Python tuple.
    time.sleep(0.05)
    print((random.randint(0, 100), random.randint(-100, 0), random.randint(-50, 50)
))
```


muPython 1.0.3 - main.py
模式
新建
打开
保存
运行
调试
REPL
绘图
放大
缩小
主题
检查
代码美化
帮助
退出
main.py
1 import time
2 import random
3
4 while True:
5 time.sleep(0.05)
6 print((random.randint(0,100), random.randint(-100, 0), random.randint(-50, 50)))
Python3 data tuple Plotter
100
50
0
-50
-100
未连接设备!
Python

MicroPython

Damien George是一名计算机工程师，他每天都要使用Python语言工作，同时也在做一些机器人项目。有一天，他突然冒出了一个想法：能否用Python语言来控制单片机，进而实现对机器人的操纵呢？ 要知道，Python是一款比较容易上手的脚本语言，而且有强大的社区支持，一些非计算机专业领域的人都选它作为入门语言。遗憾的是，它不能实现非常底层的操控，所以在硬件领域并不起眼。

Damien为了突破这种限制，他花了六个月的时间来打造MicroPython。它基于ANSI C标准，语法跟Python 3基本一致，拥有自家的解析器、编译器、虚拟机和类库。 微控制器通常以C编程，可以直接访问和控制寄存器来使用外围设备，为目标微控制器进行交叉编译和构建固件代码，并使用合适的编程器进行烧录，而MicroPython集成了所有这些步骤。借助MicroPython，用户完全可以通过Python脚本语言实现硬件底层的访问和控制，比如说控制LED灯泡、LCD显示器、读取电压、控制电机、访问SD卡等。

与桌面版本的Python不同，MicroPython是微控制器的精简版本，因此它并非支持所有Python的库和功能。当你学过Python后再学习MicroPython是很容易的事，但不是所有的Python语法都适用于MicroPython，这点要尤为注意。

1、print 输出

使用print()函数可以将数据打印到终端。

print()函数可以直接在终端执行，也可以写在Python文件中，通过运行文件来执行。

注意：

当我们在交互环境下编写代码时，>>>是MicroPython解释器的提示符，不是代码的一部分。前面没有>>>或...的“Hello World！”为程序运行的结果。

print()函数中可以有多个字符串，用逗号“,”隔开。

示例：

```
>>> print("hello", "world!")
hello world!
```

print 默认输出换行，如果在末尾加上end=" "则不换行。

示例：

```
print("hello", end=" ")
print("world")
```

运行结果：

```
hello world
```

2、注释

代码中的注释有助于我们理解代码，在程序运行时，会忽略注释。

单行注释

MicroPython中的单行注释以#开头，后面的文字直到行尾都算注释。

示例：

```
>>> print("hello world") #This is a annotation
hello world
```

多行注释

如果要进行多行的注释，可以使用多个#号， 三个单引号（'''）或三个双引号（"""）。

示例：

```
'''
This is a multi-line comment.
Prints hello world.
'''
print("hello world")
```

3、缩进

MicroPython使用缩进来区分不同的代码块，不需要使用大括号 {} 。

```
if True:
  print ("True")
else:
  print ("False")
```

缩进的空格数是可变的，但是同一代码块缩进必须一致。

示例：

```
if True:
  print ("Answer")
  print ("True")
else:
 print ("Answer")
  print ("False")    # 缩进不一致，会导致运行错误
```

以上程序由于缩进不一致，执行后会出现类似以下错误

```
*test.py:6:2: unexpected indent
  print ("False")    # 缩进不一致，会导致运行错误
 ^
syntax finish.
```

4、help()函数

调用 MicroPython的 help() 函数查看一些基本信息。

示例：

```
>>> help(max)  #查看max()函数的基本信息
object <function> is of type function
>>>
>>> help(sys) #查看sys模块的一些信息
object <module 'sys'> is of type module
  __name__ -- sys
  path -- ['', '/lib']
```

```
  argv -- []
  version -- 3.4.0
  version_info -- (3, 4, 0)
  implementation -- (name='mi 0>
  stdout -- <io.FileIO 1>
  stderr -- <io.FileIO 2>
  modules -- {'flashbdev': <module 'flashbdev'>}
  print_exception -- <function>
>>>
>>> help(sys.path)
object ['', '/lib'] is of type list
  append -- <ert -- <function>
  pop -- <function>
  remove -- <function>
  reverse -- <function>
  sort -- <function>
```

5、常量、变量

常量

如10、100这样的数值或“hello world!”这样的字符串，就是常量。常量的值不可变，MicroPython提供了一个const关键字，表示其值不可更改。

示例：

```
from micropython import const

a = const(33)
print(a)
```

变量

创建一个变量很简单，只需要起个名，给他赋予一个值，在赋值的时候不需要指定变量的数据类型，因为变量是一个引用，它通过赋值来访问不同数据类型的对象。这点与其他语言中的变量不同，要注意（关于数据类型在下一小节讲）。

示例：

```
>>> a = "abcd" #引用a指向一个字符串对象“abcd”
>>> print(a)
abcd
>>> a = 32 #引用a指向一个整型对象
>>> print(a2)
32
>>> a1 = a #a1指向引用a指向的对象
>>> print(a)
32
```

给变量命名必须遵循以下规则：

◆变量名只能包含数字、字母、下划线

◆变量名的第一个字符必须是字母或下划线

◆变量名区分大小写

MicroPython中基本的数据类型有Number（数字）、String（字符串）、List（列表）、Tuple（元组）、Dictionary（字典）等。

用type()可以查看变量和常量的数据类型。

示例：

```
>>> a1 = 23
>>> print(type(a1))
<class 'int'>
>>> a, b, c, d=20, 5.5, True, 4+3j
>>> print(type(a), type(b), type(c), type(d))
<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
>>>
>>> l, t, s = [3,2,5], (1,3,4), {'tom':4, 'jerry':6, 'seiya':25}
>>> print(type(l), type(t), type(s))
<class 'list'> <class 'tuple'> <class 'dict'>
```

1、Number（数字）

MicroPython支持 int、float、bool、complex（复数）。

当定义一个变量时，Number 对象被创建。

示例：

```
>>> var1 = 2
>>> var2 = 6
```

创建的Number对象，可以通过del语句进行删除。

示例：

```
del var
del var_a, var_b
```

注意：

1、可以同时为多个变量赋值，如a, b = 1, 2。

2、数值的除法（/）总是返回一个浮点数，如1/1，结果为1.0。

3、在混合计算时，MicroPython会把整型转换成为浮点型

int（整型）

MicroPython可以处理任意大小的整数（包括负整数），整数的表示方法和数学上的写法一样，如

```
1, 100, -8080, 0 , ……
```

MicroPython中用十六进制表示整数，如下

```
0xff00, 0xa5b3c3d2, ……
```

float（浮点型）

浮点数就是小数。按科学计数法表示时，浮点数的小数点是可变的，如1.23×10^9和12.3×10^8是相等的。可以把10用e替代，1.23×10^9就是1.23e9，或是12.3e8，0.000012可以写成1.2e-5。

示例：

```
>>> c1 = 1.34e3
>>> print(c1)
1340.0
>>> c2 = 1.343e-3
>>> print(c2)
0.001343
```

浮点数和整数在计算机内部存储方式不同，整数是精确的，而浮点数运算时会有四舍五入的误差。

bool（布尔型）

布尔值只有True、False两种值，非True则False，注意大小写。

示例：连续输出4个“hello”

```
>>> flag = True
>>> num = 4
>>> while flag:
...   print("hello")
...   num = num-1
...   if num == 0:
...     flag = False
...
...
...
hello
hello
hello
hello
```

上述示例中while为一个循环语句，当flag为True时，循环执行后面的语句，if为一个判断语句，当num=0成立时，执行下面的语句。在上面示例中要注意if num == 0中的“==”不要写成“=”。“==”表示判断是否相等，而“=”表示赋值，例如flag=False表示将False的值赋给flag变量。

注意：

前面讲过在交互环境下编写代码时，>>>是MicroPython解释器的提示符，不是代码的一部分，同样上面示例中出现的...也是一个提示符，表示待输入代码。

complex（复数）

复数由实数部分和虚数部分构成，可以用a + bj，或者complex(a, b)表示， 复数的实部 a 和虚部 b 都可以是浮点型。

示例：

```
>>> complex(1, 2)
(1+2j)
```

数字类型转换

有时候，我们需要对数据内置的类型进行转换，具体如下

int(x)： 将x转换为一个整数。

float(x)： 将x转换到一个浮点数。

complex(x)： 将x转换到一个复数，实数部分为 x，虚数部分为 0。

complex(x, y)： 将 x 和 y 转换到一个复数，实数部分为 x，虚数部分为 y。x 和 y 是数字表达式。

示例：

```
>>> a = 1.0
>>> print(int(a)) #float转换成int
1
>>>
>>> x, y = 2.3, 5
>>> c = complex(x, y)
>>> print(c)
(2.3+5j)
```

2、String（字符串）

字符串是以"或'括起来的任意文本（如'abc'，"xyz"）。

注意：

"或'本身只是一种表示方式，不是字符串的一部分。因此，字符串'abc'只有a，b，c这三个字符。

示例：

```
>>> str = 'xy,z'
>>> print(str)
xy,z
```

访问子字符串，可以使用方括号来截取字符串。

示例：

```
>>> str = "Hello World"
>>> print("str[0]:", str[0])
str1[0]: H
>>>
>>> print("str[0:5]:", str[0:5])
str1[0:5]: Hello
```

你可以截取字符串的一部分并与其他字符串拼接。

```
>>> str = "Hello"
>>> print(str[:6] + " muPython")
Hello muPython
```

3、List（列表）

列表（list）是MicroPython中最基本的数据结构。列表中的每个元素都分配一个索引，第一个索引是0，第二个是1，依次类推。

在MicroPython中定义列表需要使用方括号，列表中的数据项都包含在方括号中，数据项之间使用逗号分隔。

创建列表

(1) 列表中的数据可以是任意数据类型，甚至可以是不同类型的混合。

示例：

```
>>> li = [1, 2, 'a', "hello"]
>>> print(li)
[1, 2, 'a', 'hello']
```

(2) 列表中的数据除了基本的数据类型，还可以是其他复杂的数据结构。

示例：

```
>>> li = [ 1, 2, 'a', [1, 2, 5] ]
>>> print(li)
[1, 2, 'a', [1, 2, 5]]
```

操作列表

列表创建完成后，可以进行访问、修改、删除、插入等操作，即列表是可变的数据类型。

(1) 访问列表中的值

使用下标索引来访问列表中的值，同样也可以使用方括号的形式截取字符。

示例：

```
>>> li = ['physics', 'chemistry', 1997, 2000]
>>> print("li[0]:", li[0])  #Access the first element of the list.
li[0]: physics
>>> print("li[2]:", li[2])
li[2]: 1997
```

(2) 修改列表项

可以对列表的数据项进行修改。

示例：

```
>>> li = ['physics', 'chemistry', 1997, 2000]
>>> print("li[2]:", li[2])
li[2]: 1997
>>> li[2] = 2003 #修改列表
>>> print("li[2]:", li[2])
li[2]: 2003
>>> print(li)
['physics', 'chemistry', 2003, 2000]
```

(3) 删除列表中的元素

可以使用del或pop()函数来删除列表中指定位置的元素。

示例：

```
>>> li = ['physics', 'chemistry', 1997, 2000]
>>> print(li)
['physics', 'chemistry', 1997, 2000]
>>> del li[2]
>>> print(li)
['physics', 'chemistry', 2000]
>>> li.pop(1)
'chemistry'
>>> print(li)
['physics', 2000]
```

(4) 在某一位置插入元素

用 insert(i, x) 函数在位置 i 处插入元素 x，原来 i 位置及其后的元素依次后移，也可以使用append()方法在末尾添加元素。

示例：

```
>>> li = ['physics', 'chemistry', 1997, 2000]
>>> print(li)
['physics', 'chemistry', 1997, 2000]
>>> li.insert(2, 'hello')
```

```
>>> print(li)
['physics', 'chemistry', 'hello', 1997, 2000]
>>>
>>> li = ['physics', 'chemistry', 1997, 2000]
>>> li.append("end")
>>> print(li)
['physics', 'chemistry', 1997, 2000, 'end']
```

MicroPython列表脚本操作符

MicroPython表达式	结果	描述
len([1,2,3])	3	列表元素个数
[1,2,3]+[4,5,6]	[1,2,3,4,5,6]	多个列表组合成一个列表
['ha!']* 3	['ha!','ha!','ha!']	重复若干个元素组成列表
3 in [1,2,3]	True	判断元素是否存在于列表中

MicroPython列表截取 有一个列表：li=['hello','muPython',2019,4,10]

MicroPython表达式	结果	描述
li[2]	2019	读取列表中第三个元素
li[-1]	3	读取列表中倒数第1个元素
li[1:]	['muPython', 2018, 1, 3]	截取从第2个元素开始及其后的元素

MicroPython列表操作的函数和方法

一、列表操作包含以下函数：

1、cmp(list1, list2)：比较两个列表的元素

2、len(list)：返回列表元素个数

3、max(list)：返回列表元素最大值

4、min(list)：返回列表元素最小值

5、list(seq)：将元组转换为列表

二、列表操作包含以下方法:

1、list.append(obj)：在列表末尾添加新的对象

2、list.count(obj)：统计某个元素在列表中出现的次数

3、list.extend(seq)：在列表末尾一次性追加另一个序列中的多个值（用新列表扩展原来的列表）

4、list.index(obj)：从列表中找出某个值第一个匹配项的索引位置

5、list.insert(index, obj)：将对象插入列表

6、list.pop(obj=list[-1])：移除列表中的一个元素（默认最后一个元素），并且返回该元素的值

7、list.remove(obj)：移除列表中某个值的第一个匹配项

8、list.reverse()：反向列表中元素

9、list.sort([func])：对原列表进行排序

4、Tuple（元组）

元组（tuple）和列表十分类似，只是元组和字符串一样是不可变的，即不能修改元组。正是由于元组不可变，一般用于MicroPython中定义一组不需要改变的值。

在MicroPython中定义元组使用圆括号，元组中的项目同样使用逗号分隔。

示例：

```
>>> tu = (1, 2, 4, 'hello')
>>> print(tu)
(1, 2, 4, 'hello')
```

空的元组由一对空的圆括号组成，比如t1=()。

注意：

定义只有1个元素的元组时，必须要在这个元素后面跟一个逗号。

示例：

```
>>> a = (1)
>>> type(a)
<class 'int'>
>>> print(a)
1
>>>
>>> tu = (1, )
>>> type(tu)
<class 'tuple'>
>>> print(tu)
(1,)
```

5、Dictionary（字典）

字典（dict）这种数据结构类似通讯录，有一个名字和名字对应的信息，可以通过名字查找对应的信息，在字典中名字叫‘键’，对应的内容叫‘值’。字典就是一个键/值对（key/value）的集合。

在MicroPython中定义字典使用花括号，字典中的键/值对之间使用逗号分隔，键和值之间用冒号分隔。

示例：

```
>>> d = {'tom':4, 'jerry':6, 'seiya':25}
```

```
>>> print(d)
{'tom': 4, 'jerry': 6, 'seiya': 25}
>>> print(d['jerry'])
6
```

操作字典

(1) 把数据放入dict

可以直接对单个键赋值的方法来将数据放入字典。

```
>>> d = {'tom':4, 'jerry':6, 'seiya':25}
>>> print(d)
{'tom': 4, 'jerry': 6, 'seiya': 25}
>>> d['panda']=12
>>> print(d)
{'jerry': 6, 'panda': 12, 'seiya': 25, 'tom': 4}
```

(2) 由于一个key只对应一个value，所以，多次对一个key放入value，后面的值会把前面的值覆盖。

```
>>> d = {'tom':4, 'jerry':6, 'seiya':25}
>>> d['panda']=12
>>> print(d)
{'jerry': 6, 'panda': 12, 'seiya': 25, 'tom': 4}
>>> d['panda']=10
>>> print(d)
{'jerry': 6, 'panda': 10, 'seiya': 25, 'tom': 4}
```

(3) 删除键/值对

用pop(key)的方法删除一个键值对，对应的value也会从dict中删除。

```
>>> d = {'tom':4, 'jerry':6, 'seiya':25}
>>> d.pop('seiya')
25
>>> print(d)
{'tom': 4, 'jerry': 6}
```

1、算术运算符

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	定义一个负数，或两个对象相减	-5.2，或50-24得到26
*	乘	两数相乘或返回一个被重复若干次的字符串、列表、元组等	2*3得到6。'la'*3得到'lalala'
/	除	x除以y	4 / 3得到1.333333。6.0/2.0得到3.0
//	取整除	返回商的整数部分	4 // 3得到1
%	取模	返回除法的余数	8 % 3得到2。-25.5 % 2.25得到1.5
**	幂	返回x的y次幂	2**3得到8（即 2*2*2）

示例：

```
>>> print(3 + 5, 'a' + 'b') # x + y
8 ab
>>> print(-5.2) # - x
-5.2
>>> print(50 - 24) # x - y
26
>>> print(2 * 3, 'la' * 3)  # x * y
6 lalala
>>> print(4 / 3, 6.0 / 2.0) # x / y
1.333333 3.0
>>> print(4 // 3) # x // y
1
>>> print(8 % 3, -25.5 % 2.25) # x % 3
2 1.5
>>> print(2 ** 3) # x ** y
8
```

2、位运算符

位运算符是把数字看作二进制来进行计算的，如5的二进制位为0101。

运算符	名称	说明	例子
<<	左移	把<<左边的运算数的各二进制位全部左移若干位（由<<右边的数指定移动的位数），高位丢弃，低	2 << 2得到8（即0010左移两位，右边补0，结果为

	移	位补0	1000）
>>	右移	把>>左边的运算数的各二进制位全部右移若干位，（由>>右边的数指定移动的位数），低位丢弃，高位补0	11 >> 1得到5（即1011右移一位，左边补0，结果为0101）
&	按位与	参与运算的两个值，如果相应的两个位都为1，则对应位按位与运算的结果为1，否则为0	5 & 3得到1（即0101 & 0011，结果为0001）
|	按位或	两个数对应的二进制位有一个为1时，则对应位按位或运算的结果为1，否则为0	5 | 3得到7（即0101 | 0011，结果为0111）
^	按位异或	两个数对应的二进制位不相同时，则该对应位按位异或运算的结果为1，否则为0	5 ^ 3得到6（即0101^0011，结果为0110）
~	按位取反	每个二进制位取反，即把1变为0，把0变为1。~x 类似于 -x-1	~5得到-6（0101， 在一个有符号二进制数的补码形式。）

示例：

```
>>> print(2 << 2) # x << y
8
>>> print(11 >> 1) # x >> y
5
>>> print(5 & 3) # x & y
1
>>> print(5  | 3) # x | y
7
>>> print(5 ^ 3) # x ^ y
6
>>> print(~5) # ~x
-6
```

3、比较运算符

运算符	名称	说明	例子
<	小于	返回x是否小于y。若为真返回True，为假返回False	5<3返回False，3<5返回True
>	大于	返回x是否大于y	5>3返回True
<=	小于等于	返回x是否小于或等于y	3<=4返回True
>=	大于等于	返回x是否大于或等于y	4>=6返回False
==	等于	比较是否相等	'str'=='stR'返回False

!=	不等于	比较是否不相等	'str'!='stR'返回True

示例：

```
>>> print(5<3) # x < y
False
>>> print( 5 > 3) # x > y
True
>>> print( 3 <= 4) # x <= y
True
>>> print(4 >= 6) #x >= y
False
>>> print('str' == 'stR') # x== y
False
>>> print('str' != 'stR') # x != y
True
```

4、逻辑运算符

运算符	名称	说明	例子
not	布尔“非”	not x；如果x为True，返回False，否则返回True	x=True；not x返回False
and	布尔“与”	x and y；x，y都为True返回True，否则返回False	x=False；y=True；x and y；由于x是False，返回False
or	布尔“或”	x or y；x或y至少一个为True，返回True，否则返回False	x=True；y=False；x or y返回True

示例：

```
>>> print(5 < 3) # not x
False
>>> x = (5 < 3)
>>> print(not x)
True
>>> x = False #  x and y
>>> y = True
>>> print(x and y)
False
>>> x = True # x or y
>>> y = False
>>> print(x or y)
True
```

5、赋值运算符

算术运算符和简单的赋值运算符“=”结合可构成复杂的赋值运算符。

运算符	描述	例子
=	简单的赋值运算符	c = 10 将10赋值给c
+=	加法赋值运算符	c += a 等效于 c = c + a
-=	减法赋值运算符	c -= a 等效于 c = c - a
*=	乘法赋值运算符	c *= a 等效于 c = c * a
/=	除法赋值运算符	c /= a 等效于 c = c / a
%=	取模赋值运算符	c %= a 等效于 c = c % a
**=	幂赋值运算符	c**= a 等效于 c = c ** a
//=	取整除赋值运算符	c //= a 等效于 c = c//a

示例：

```
>>> c = 10    # =
>>> print(c)
10
>>> c += 2    # +=
>>> print(c)
12
>>> c -= 2   # -=
>>> print(c)
10
>>> c *= 2   # *=
>>> print(c)
20
>>> c /= 2   # /=
>>> print(c)
10.0
>>> c %= 3   # %=
>>> print(c)
1
>>> c = 10
>>> c **= 2   # **=
>>> print(c)
100
>>> c //= 10   # //=
>>> print(c)
10
```

注意：

赋值运算符是一个整体，中间不能有空格，否则出错。

示例：

```
>>> c = 10
>>> print(c)
10
>>> c - = 2
Traceback (most recent call last):
  File "<stdin>", line 1
SyntaxError: invalid syntax
```

1、if语句

if语句用来判断当某个条件成立（非0或为True）时，执行下一个语句。常与else一起使用，表示除if判断条件之外的其他情况。

示例：

```
>>> num = 130
>>> if num%2 == 0:
...   print(num, "is a even number")
... else:
...   print(num, "is a odd number")
...
130 is a even number
```

注意：

可以有多个elif，else是可选的。elif是“else if”的缩写，对于避免过多的缩进非常有用，else与它最近的前一个if或elif匹配。

示例：

```
>>> x = 32
>>> if x < 0:
...   print("Negative changed to zero")
... elif x == 0:
...   print("Zero")
... elif x == 1:
...   print("Single")
... else:
...   print("More")
...
More
```

注意：

由于MicroPython严格的缩进格式，为避免出错，最好用空格键进行缩进。

示例：

```
>>> x = 32
>>> if x > 0:
...   print("x > 0")
... print("hello")
... else:
...   print("x <= 0")
...
Traceback (most recent call last):
```

```
  File "<stdin>", line 3
SyntaxError: invalid syntax
```

2、while语句

while语句用于循环执行程序，即在某条件下，循环执行某段程序。

示例：

```
>>> i = 5
>>> while i > 0:
...   print(i)
...   i = i-1
...
5
4
3
2
1
```

3、for语句

for语句用于循环执行程序，并按序列中的项目（一个列表或一个字符串）顺序迭代。

示例：

```
>>> words = ['www', 'microduino', 'cn']
>>> for w in words:
...   print(w, len(w))
...
www 3
microduino 10
cn 2
```

如果需要在for循环内修改迭代的顺序或条件，可以在for循环中增加条件判断。

示例：

```
>>> words = ['www', 'microduino', 'cn']
>>> for w in words:
...   if len(w) < 7:
...     print(w)
...
...
...
www
cn
```

range()函数

如果你需要遍历一系列的数字，可以使用内置函数range()。

示例：

```
>>> for i in range(4):
...     print(i)
...
...
...
0
1
2
3
```

4、break语句

break语句用于退出for或while循环。

示例：

```
>>> for x in range(2, 10):
...   if x == 5:
...     break
...   print(x)
...
2
3
4
```

5、continue语句

continue语句用于退出for或while语句的当前循环，进入下一次循环。

示例：

```
>>> for x in range(2, 10):
...   if x == 5:
...     continue
...   print(x)
...
2
3
4
6
7
8
9
```

6、pass语句

pass语句表示空语句，不做任何事情，一般用作占位语句，用来保持程序结构的完整性。

示例：

```
>>> for letter in 'hello':
...   if letter == 'l':
...     pass
...     print("This is pass")
...   print("Current letter:", letter)
...
Current letter: h
Current letter: e
This is pass
Current letter: l
This is pass
Current letter: l
Current letter: o
```

1、函数定义

除了MicroPython内建的函数，用户也可以使用def语句自定义的函数。定义格式如下：

```
def <函数名> (<参数1, 参数2, ……>) :
    <函数体>
      ...
      ...
      ...
```

示例：

```
>>> def my_print():
...   print("hello world!")
...
>>>
```

2、函数调用

函数定义完成后，使用函数名来调用函数，从而使用其功能。

示例：

```
>>> def my_abs(x):  #求x的绝对值
>>>   if x >= 0:
>>>     return x
>>>   else:
>>>     return -x
...
...
...
>>> a = my_abs(-5)
>>> print(a)
5
```

3、函数参数

函数可以接收输入的值，并利用这些值做一些事。

多个参数传递需要用逗号隔开。

示例：

```
>>> def print_max(a, b):
...   if a > b:
...     return a
```

```
...   else:
...     return b
...
...
...
>>> a = print_max(4, 6)
>>> print(a)
6
```

4、默认值参数

在定义函数的过程中如果指定了参数，那么调用函数时一定要给所有的参数传递值，否则会出错。

但是在一些情况下，可能又希望它的参数是可选的，即不要求调用函数时一定要给所有参数都传递值，那么可以通过默认值参数来完成。

示例：

```
>>> def say_message(message, times=2):
...   print(message*times)
...
>>> say_message("hello", 3)
hellohellohello
>>> say_message("look")
looklook
```

5、关键参数

在调用函数时，还可以使用另外一种方式给函数传递值，即使用参数的名字（关键字）而不是位置，这被称为关键参数。

使用关键参数的优势：不需要担心参数顺序，使函数调用更简单。

示例：

```
>>> def print_abc(a, b, c):
...   print(a, b, c)
...
>>> print_abc(1,2,4)
1 2 4
>>> print_abc(c=4, b=2, a=1) #使用关键参数
1 2 4
```

6、return语句

return用来退出一个函数，也可以使用return从函数返回一个值，并且这个值可以赋给其他变量。

如果return语句没有返回值，等价于return None，表示无返回值。如果函数中没有明确指定return语句，都在结尾暗含有return None语句。

示例：

```
>>> def print_max(x, y):
>>>   if x > y:
>>>     return x
>>>   elif x < y:
>>>     return y
>>>   else:
...     return
...
...
...
>>> print_max(5, 3)
5
>>> print_max(5, 5)
>>> a = print_max(5, 5)
>>> print(a)
None
```

7、lambda表达式

lambda 表达式可以返回一个函数，使用lambda表达式可明显减少函数数量。定义格式如下：

```
lambda <参数1, 参数2, ……> : <表达式>
```

示例：

```
>>> def make_incrementor(n):
...   return lambda x : x+n
...
>>> f = make_incrementor(32)
>>> print(f(3))
35
>>> sum = lambda x,y : x+y
>>> print(sum(32, 3))
35
```

1、import语句

MicroPython中要引入模块，使用import语句，格式如下：

```
import <模块名>
```

注意：

如果是直接引入模块，在使用模块中函数或属性（常量、变量）时一定要指出函数或属性的所属模块，格式为：<模块名>.<函数或属性>，否则会出错。

示例：

```
>>> import random
>>>
>>> num = random.randint(1, 100) #使用random模块中的randint()函数
>>> print(num)
7
```

2、from…import语句

如果只想引入模块中的某个函数或属性，使用from…import语句，格式如下：

```
from <模块名> import <函数名或变量名>
```

示例：

```
>>> from random import randint
>>>
>>> num = randint(1, 100)
>>> print(num)
95
```

在使用from...import语句从模块中引入函数时，为避免冲突和便于理解，可以使用as语句给引入的函数换个名字，如下：

```
from <模块名> import <函数名或变量名> as <自定义名>
```

示例：

```
>>> from random import randint as ra
>>>
>>> num = ra(1, 100)
>>> print(num)
30
```

3、自定义的模块

每个Python文件，只要它保存在MicroPython的文件系统中，就是一个模块。

引入自定义的模块，需要模块文件位于MicroPython环境变量路径下或与当前运行程序在同一路径下。

注意：

不能引入workSpace目录中的文件，编写好后，必须要下载到开发板上才可以引入它。

4、dir()函数

dir()函数是micropython内置的函数，用来列出模块中的函数、类和属性。

如果给dir()函数提供一个模块名称，它返回该模块中的名称列表，如果不提供，则返回当前模块的名称列表。

示例：

```
>>> import random
>>>
>>> dir(random)
['__name__', 'getrandbits', 'seed', 'randrange', 'randint', 'choice', 'random', 'un
iform']
>>> print( dir() )
['myfile', 'print_x', 'a', 'gc', 'print_abc', 'num', 'sys', 'bdev', 'randint', '__n
ame__', 'os', 'x', 'random', 'print_max', 'ra', 'uos']
```

dir()函数没有列出内置函数和变量的名字，这些函数和变量定义在builtins模块中，通过它你可以看到具体的内容。

示例：

```
>>> import builtins
>>> dir(builtins)
['__name__', '__build_class__', '__import__', '__repl_print__', 'bool', 'bytes', 'b
ytearray', 'complex', 'dict', 'enumerate', 'filter', 'float', 'frozenset', 'int', '
list', 'map', 'memoryview', 'object', 'property', 'range', 'reversed', 'set', 'slic
e', 'str', 'super', 'tuple', 'type', 'zip', 'classmethod', 'staticmethod', 'Ellipsi
s', 'NotImplemented', 'abs', 'all', 'any', 'bin', 'callable', 'compile', 'chr', 'de
lattr', 'dir', 'divmod', 'eval', 'exec', 'execfile', 'getattr', 'setattr', 'globals'
, 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'isinstance', 'issubclass', 'ite
r', 'len', 'locals', 'max', 'min', 'next', 'oct', 'ord', 'pow', 'print', 'repr', 'r
ound', 'sorted', 'sum', 'BaseException', 'ArithmeticError', 'AssertionError', 'Attr
ibuteError', 'EOFError', 'Exception', 'GeneratorExit', 'ImportError', 'IndentationE
rror', 'IndexError', 'KeyboardInterrupt', 'KeyError', 'LookupError', 'MemoryError',
'NameError', 'NotImplementedError', 'OSError', 'OverflowError', 'RuntimeError', 'St
opAsyncIteration', 'StopIteration', 'SyntaxError', 'SystemExit', 'TypeError', 'Unic
odeError', 'ValueError', 'ZeroDivisionError', 'input', 'open']
```

1、类

类定义格式如下：

```
class <类名>:
    类体
```

类中除了定义函数，还可以包含其他语句。一个类中的函数定义通常有一个特殊的参数列表形式，以在类外调用这个函数。

2、类对象

类支持两种操作：属性引用和实例化。

属性引用

调用类的属性：obj.name，name是类中定义的变量或函数的名字。

示例：

```
class MyClass:
  i=12345

  def f(self):
    print("hello world")
```

上面示例中MyClass.i和MyClass.f是有效的属性引用，分别引用一个整数和一个函数。

类实例化

示例：

```
x = MyClass()
```

上面的示例，创建该类的新实例并将对象分配给本地变量x。

一个类可以定义一个名为init()的特殊方法。

示例：

```
def __init__(self,  data):
  self.data=23
```

当一个类定义了一个init()方法时，类会自动调用init()新创建的类实例。可以通过x=MyClass()的方式获得。在这种情况下，赋予类实例化的参数被传递给init()。

示例：

```
>>> class ComplexClass:
...   def __inis__(self, r, i):
...     self.r = r
...     self.i = i
...     print(self.r + self.i)
...
...
...
>>> x = ComplexClass(3, 6)
9
>>> x.r, x.i
(3, 6)
```

实例对象

示例：

```
class Complex:
  def __init__(self, r, i):
    self.r = r
    self.i = i
    print(self.r + self.i)

x = Complex(3, 6)
x.counter = 1
while x.counter < 10:
    print("hello",x.counter)
    x.counter = x.counter * 2
print(x.counter)
del x.counter
```

运行结果：

```
9
hello 1
hello 2
hello 4
hello 8
16
```

1、文件读取

想要读取文件中的数据，首先要打开文件（文件已存在）。

```
open(filename, mode)
    类体
```

```
mode：模式
    r ：以只读方式打开
    rw：以读写方式打开
```

示例：

```
f = open ( 'test.py', 'r')
```

注意：

只能打开下载到开发板中的文件。

示例：

```
print(f.read())
```

读取完成后，记得使用close()关闭文件，释放资源。

示例：

```
f.close()
```

综合示例：

```
>>> f = open('test.py', 'r')
>>> print(f.read())
import time
from machine import Pin
led=Pin(2,Pin.OUT)

while True:
  led.value(1)
  time.sleep(0.5)
  led.value(0)
  time.sleep(0.5)
>>> f.close()
```

2、文件写入

向文件中写入数据同样要先打开文件（文件已存在）。

```
f = open(filename, mode)
```

然后向文件中写入数据。

```
fd.write(str)
```

示例：

```
fd.write('hello muPython')
```

最后不要忘了关闭文件以释放资源。

```
f.close()
```

综合示例：

```
>>> f = open('test.py', 'w')
>>> f.write('#muPython')
8
>>> f.close()
>>> f = open('test.py', 'r')
>>> print(f.read())
#muPython
>>> f.close()
```

注意：

向一个已经存在内容的文件写入数据时，会覆盖原来的内容。

1、异常

即使语句或表达式在语法上是正确的，但是执行它时却出现错误，我们把这种在执行过程中检测到的错误称为异常。

示例：

```
c = 10/0
print(c)
```

运行结果：

```
syntax finish.
>>>
>>>
Ready to download this file,please wait!
download ok
exec(open('test.py').read(),globals())
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<string>", line 1, in <module>
ZeroDivisionError: division by zero
```

上面的例子语法没有错误，但是引发ZeroDivisionError异常而产生中断，使后面的程序不能正确执行。

2、处理异常

try……except

在MicroPython中用try……except语句来处理异常，将可能引发异常的语句放到try中执行，当异常发生时，跳过try中剩余的语句，直接跳转至except中的语句来处理异常。

示例：

```
>>> try:
...   a = 10/0
...   print(a)
... except:
...   print("error")
...
error
```

except语句也可以专门处理指定的异常，即在except语句后跟异常名称，如果不指定异常名称则表示处理所有异常。

示例：

```
def divide(x, y):
  try:
    i = x/y
  except ZeroDivisionError:
    print("division by zero!")
  else:
    print("result is", i)

divide(5, 0)
divide(5, 2)
```

运行结果：

```
division by zero!
2.5
```

在上面的示例中，try……except语句有一个可选的else子句。如果try子句不引发异常，则必须执行该代码。

try……finally

无论是否发生异常都会执行finally中的语句块，它可以和try……except……else一起使用。

示例：

```
def divide(x, y):
  try:
    i = x/y
  except ZeroDivisionError:
    print("division by zero!")
  else:
    print("result is", i)
  finally:
    print("executing finally clause")

divide(5,0)
print() #打印一个空行
divide(5, 2)
```

运行结果：

```
division by zero!
executing finally clause

result is 2.5
executing finally clause
```

V1.0.4(2019/4/17)

- 新增Tkinter库，解决一些电脑上不能绘制图形的问题
- 修改部分配置文件的生成路径，防止因应用权限不够而造成显示异常
- 修复一些电脑上不能正常烧录固件的bug
- 修改烧录时间统计方法，增加烧录时间预估的准确性
- 新增音频支持库，确保pygame中音效即背景音乐播放的稳定性

V1.0.3(2019/4/4)

- 代码刷入加入语法自动检查
- 新增代码自动美化功能
- 新增在micropython在线运行功能
- Ideabit模式加入Plotter绘图功能
- 新增Micropython固件烧录功能
- 修复通信bug

V1.0.2 (2019/3/22)

- 刷入py文件后可以直接执行
- 修复python3运行模式下，界面重启问题
- 修复microPython模式下，双击文件会推出的问题

第三方库

Tkinter

turtle

opencv-python

urllib3

pillow

pickleshare

numpy

matplotlib

开源软件鸣谢

Mu