

方法与文件

BBCLOUD 林凡

https://github.com/lvancer/course_python

lin029011@163.com

大纲

方法

异常

调试

文件

练习



方法

方法也可以叫函数，可以实现某种功能，比如len、max、min等。

方法有三种：

1、内置方法：len、max、min等可以直接调用的。

<https://docs.python.org/3.6/library/functions.html#abs> 所有的内置方法。

2、自定义方法：自己将实现某块功能的代码进行封装，开发成方法，方便调用。

前面我们就写过求和的功能，下面我们马上就可以定义一个自己的sum方法。

3、类方法：在面向对象编程中，在类中定义的方法。

比如字符串的startswith方法、列表的append方法等。基本的形式就是 变量.方法(参数)。

关于对象与类这块会在后面的课程学习。下面我们就开始学习自定义方法。

方法

方法定义：`def`。

```
def print_with_name(content):  
    print(content)  
    print('-- Green')
```

方法调用

```
print_with_name('Hello')
```

`def`表示开始定义一个方法。`def`后面跟上的是方法名，方法名后的括号内是参数。

方法名的起名规则与变量名一样，且不可重复，即不能定义两个相同的方法。

上面定义了一个实现了打印内容之后再多打印署名的功能。

这样我们就在需要署名时，直接调用这个方法就行了。

不但调用方便，而且以后要修改署名格式的时候，只需修改这个方法里的内容就可以了。

lin029011@163.com

方法

参数：

```
def print_with_name_v2(content, name):  
    print(content)  
    print('-- {}'.format(name))  
  
print_with_name_v2('Hello', 'Green')
```

多个参数之间使用逗号分隔。参数的数量尽量不要超过5个。

参数在方法的代码块里就相当于一个变量，只不过这个变量会随着调用者的输入变化。

上面的代码就实现了更灵活的功能，署名也变为一个参数动态配置。

尝试再编写一个署名格式也可以动态配置的方法。

方法的灵活度并不是越高越好，而是根据具体需求保持一个度。

方法

默认值：给参数定义一个默认值，这样就可以不用每次都输入这个参数。

```
def print_with_name_v3(content, name='Green'):  
    print(content)  
    print('-- {}'.format(name))  
  
print_with_name_v3('Hello')  
print_with_name_v3('Hello', 'Lily')
```

参数后加上**默认值**，并用**等号**连接。如上面的代码，这样name的默认值就是Green。

第一次调用时，我们就没有写入name，这时name就会**自动被赋值**为默认值Green。

也可以把参数写全。由于参数的顺序性，有默认值的参数只能放在**最后N个**。

```
def func1(a, b=1, c, d=True):    # 错误  
def func2(a, c, b=1, d=True):    # 正确
```

方法

参数赋值：基本的参数赋值是按顺序**逐个赋值**，另一种方式是根据名称**直接赋值**。

```
def func3(a, b, c=1, d=True):  
    pass  
func3(10, 20, d=False)
```

pass表示不做任何事，这里是因为这个方法暂时还没有实现，用pass做一个占位，否则语法错误。

在调用时直接用**参数名=值**的方式**传入参数**。

这样就可以跳过顺序方式时，必须写入c的参数情况。

所有**没有默认值**的参数都**必须填写**，不可跳过。当然也可以用这种方式**不按顺序**传入。

```
func3(b=10, a=20, d=False)
```

关于参数，还有最后一种**不定参数**，未来高阶一些的教程会介绍。

方法

返回值：**return**。每个方法都会返回一个值，默认是空值None。

```
def sum1(numbers):  
    answer = 0  
    for i in numbers:  
        answer = answer + i  
    return answer  
answer = sum1([10, 53, 45, 44, 907])
```

上面就是我们自己封装的sum方法。最后把answer通过**return**返回。

当遇到**return**时，方法会马上**停止**，并把return后的**值返回**。

如果**不存在**return，则自动返回**空值None**。

方法

多返回值：方法可以返回多个值，只需用逗号隔开。

```
def sum_avg(numbers):  
    answer = 0  
    for i in numbers:  
        answer = answer + i  
    return answer, answer/len(numbers)  
sum, avg = sum_avg([10, 53, 45, 44, 907])
```

上面的方法同时返回了总和与平均值。

这样返回的两个变量会被组合成一个元组。

我们在获取返回值时，可以用拆包来分别得到两个返回值。

异常

异常是一种错误，一般会造成程序不可继续执行。

一类异常是代码语法错误，这个一般Pycharm会帮我们提前告知。

```
a = 1
if a > 2
    print(1)
```

错误显示：

```
if a > 2
    ^
SyntaxError: invalid syntax
```

这类错误不应该存在。有这样的错误，程序根本不会运行。

另一类异常是运行时错误，就是在运行过程中的错误。下面的就是一个除以0的错误。

```
a = 10
b = 0
print(a / b)
```

错误显示：

```
print(a / b)
ZeroDivisionError: division by zero
```

这类错误可以被捕获并处理。

异常

异常捕获：`try`、`except`。

```
try:  
    a = 10  
    b = 0  
    print(a / b)  
except Exception as e:  
    print(e)
```

像这样把前面的代码用`try`包起来，`try`下面就是要捕获错误的代码块。

`except`用来捕获错误，这里的`Exception`是通用异常，`as`是把异常赋值给`e`。

如果捕获到异常，就会打印出异常内容。

division by zero

这样捕获到的异常，不会造成程序退出。

但一般情况下在捕获到异常后应该立即记录错误，并退出该次运行。

异常

异常处理：异常并不是找bug，而是捕获可能的已知错误，并优雅的退出。

我们可以用多个except取获取不同的异常。

ZeroDivisionError和ValueError就是已知的异常。

当捕获到这两个异常时，做出不同的处理。

最后的Exception是保证一些未知的错误被捕获。

finally下面的代码块一定会执行，无论有没有异常。

异常的主体一般是我们无法控制的内容，

如用户输入、文件读取、接口访问、数据库连接等。

这些操作的共性是是否会成功未知，不是我们的代码可以控制的。

```
try:
    a = int(input('输入a: '))
    b = int(input('输入b: '))
    print(a / b)
except ZeroDivisionError:
    print('b不能为0')
except ValueError:
    print('输入的不是数字')
except Exception as e:
    print(e)
finally:
    print('程序结束。')
```

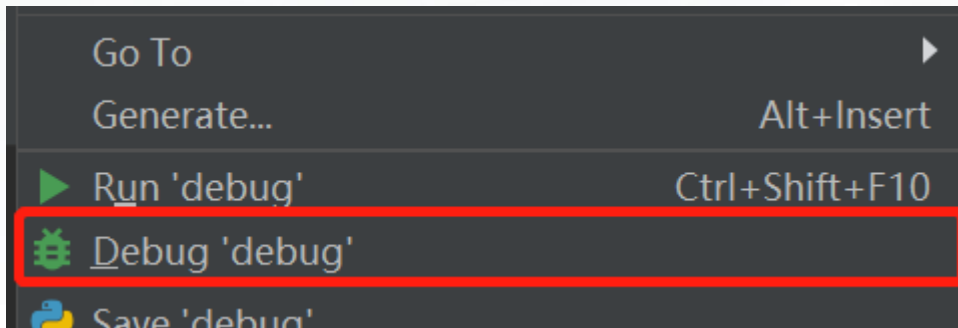

调试

调试也叫debug，就是找bug的过程了。

最原始的调试方式就是print，在程序过程中打印出各个位置的值。

这种方式是很常用的，但其实还可以用到Pycharm提供的debug功能。

debug：在启动程序的Run指令下面，就是Debug指令。



但是我们现在直接点击Debug命令，除了运行慢一点并没有什么效果。

调试

断点设置：通过设置断点，在debug下程序就会在断点处停下来。

点击行号旁边的空白处【红框位置】，就会出现一个红点，这就是断点。

```
4 a = 1
5 b = 2
6   c = a / b
7 print(c)
```

点击后：

```
4 a = 1
5 b = 2
6 ● c = a / b
7 print(c)
```

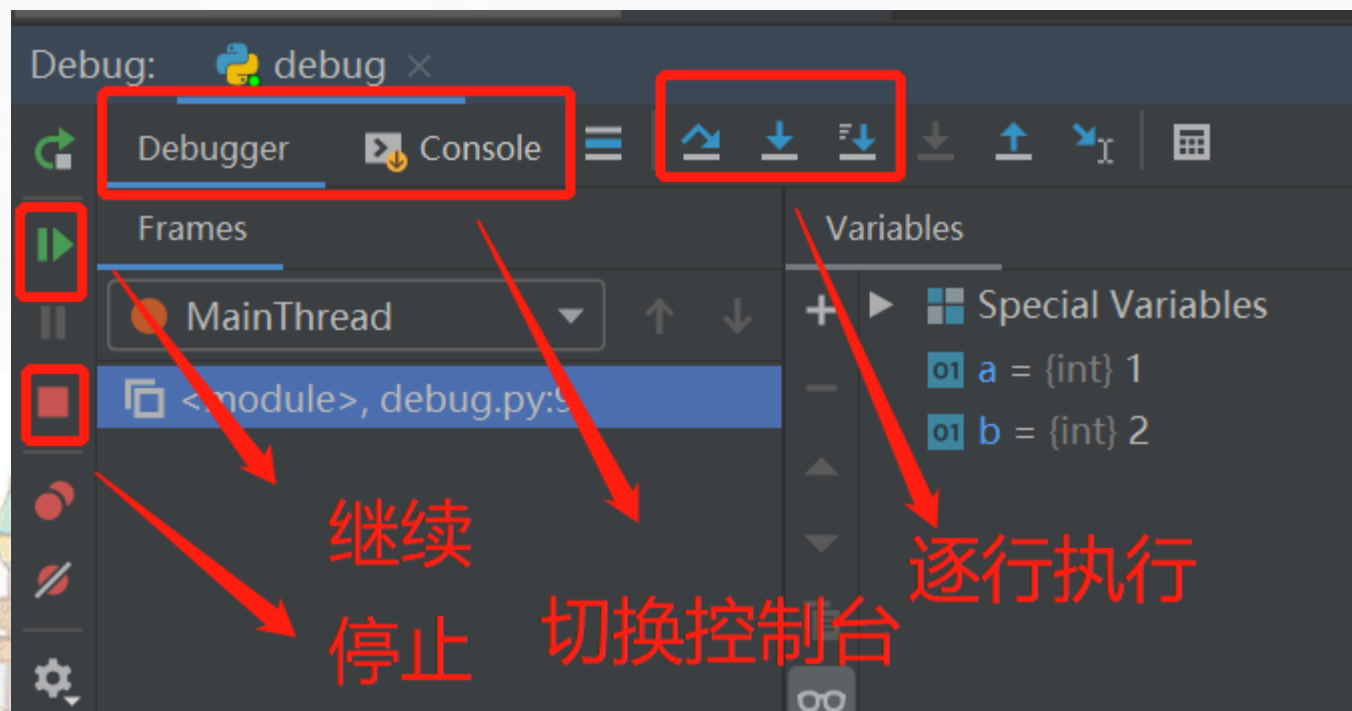
我们也可以设置多个断点，一般我们会在可能出现的地方设置。

调试

开始debug：执行Debug指令后，程序就在该命令处停下来了。

```
4 a = 1 a: 1
5 b = 2 b: 2
6 ● c = a / b
7 print(c)
```

这里我们能看到a和b变量的值
我们可以继续和停止程序。
还可以进行逐行执行，
一行一行代码执行看是否正确。



文件

文件操作是我们经常会使用到的。

读取文件：`open`。

```
f = open('a.txt', 'r') # 打开文件
content = f.read()     # 读取整个文件
print(content)
```

`open`的第一个参数是文件地址，第二参数是打开模式，`r`表示读取，也是参数的默认值。

文件地址可以是相对地址【相对于该程序文件的地址】，也可以是绝对地址。

`read`方法会读取整个文件内容。

还有一个`readlines`，会以列表方式返回每行内容。

```
f = open('a.txt')
lines = f.readlines() # 读取所有行
print(lines)
```

一次性读取整个文件，速度快，内存消耗大。

lin029011@163.com

文件

逐行读取：每次只读取一行，速度慢，但是可以应对大文件。

使用for循环直接对文件变量进行遍历。

得到的line就是每行的内容。

每行的内容都包含换行符，所以我们用strip去除。

close方法用于关闭文件，在不需要文件时要关闭。

try、except主要处理文件读取失败的情况。

```
try:
    f = open('a.txt')
    for line in f:
        line = line.strip()
        print(line)
    f.close()
except Exception as e:
    print('文件读取错误')
    exit()
```

文件

写文件：

```
f = open('b.txt', 'w')  
f.write('aa\nbb\ncc')  
f.close()
```

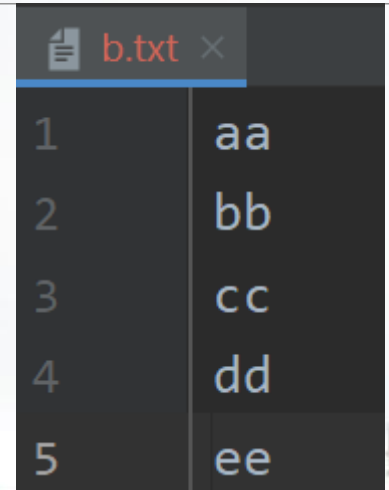
修改打开模式为w，就可以写文件。

如果文件不存在，则创建。如果文件存在，会清空该文件重新写入内容。

write方法的参数为要写入内容。

追加文件：将打开模式改为a，当文件存在时，就会直接在该文件后面进行追加内容。

```
f = open('b.txt', 'a')  
f.write('\ndd\nnee')  
f.close()
```



1	aa
2	bb
3	cc
4	dd
5	ee

文件

split是对字符串分割
去除非字母使用了正则表达式
re是正则表达式的模块

编写一个程序，计算文件里的所有不重复的单词。

```
import re    # 导入re模块
f = open('license.txt')
words = []
for line in f:
    line = line.strip()           # 去空格回车
    _words = line.lower().split(' ') # 分割
    for w in _words:
        if not w.isalpha():       # 包含非字母
            w = re.sub('[^a-zA-Z]', '', w) # 去除非字母
        if len(w.strip()) != 0:   # 保证不是空内容
            words.append(w)
words = set(words) # 去重
print(len(words))
```

练习

统计一个文件中的每个词的次数，从小到大排列，并输出到一个文件。

1、编写两个方法，一个用来统计并返回结果、一个用来将结果输出。

2、输出的文件格式：

apple 199

orange 120