Python 的内容比较多，可能要花个几篇来讲完。

上一篇文章讲了 Python 的一些运行环境和数据基础，本篇文章将来讲解高级的，函数参数这些。

控制流

在 Python 中有三种控制流语句，if、for 和 while。

1. 条件

if age >= 18:

注意不要少写了冒号 ：。

elif 是 else if 的缩写，完全可以有多个 elif，所以 if 语句的完整形式就是：

if <条件判断 1>: <执行 1> elif <条件判断 2>: <执行 2> elif <条件判断 3>: <执行 3> else: <执行 4>

if 语句执行有个特点，它是从上往下判断，如果在某个判断上是 True，把该判断对应的语句执行后，就忽略掉剩下的 elif 和 else。

if 判断条件还可以简写，比如写：

if x: print('True')

只要 x 是非零数值、非空字符串、非空 list 等，就判断为 True，否则为 False。

2. 循环

Python 的循环有两种。

第一种是 for...in 循环，依次把 list 或 tuple 中的每个元素迭代出来

所以 for x in ...循环就是把每个元素代入变量 x，然后执行缩进块的语句。

Python 提供一个 range()函数，可以生成一个整数序列，再通过 list()函数可以转换为 list。比如 range(5)生成的序列是从 0 开始小于 5 的整数：

```
In [8]: list(range(5))

Out[8]: [0, 1, 2, 3, 4]
```

计算：

```
In [9]: sum = 0
        for x in range(101):
            sum = sum + x
        print(sum)

        5050
```

第二种循环是 while 循环：

```
sum = 0
n = 99
while n > 0:
    sum = sum + n
    n = n - 2
print(sum)
```

```
2500
```

在循环中，break 语句可以提前退出循环。

```python
n = 1
while n <= 100:
    if n > 10: # 当n = 11时，条件满足，执行break语句
        break # break语句会结束当前循环
    print(n)
    n = n + 1
print('END')
```

```
1
2
3
4
5
6
7
8
9
10
END
```

在循环过程中，也可以通过 continue 语句，跳过当前的这次循环，直接开始下一次循环。continue 的作用是提前结束本轮循环，并直接开始下一轮循环。

```python
n = 0
while n < 10:
    n = n + 1
    print(n)
```

```
1
2
3
4
5
6
7
8
9
10
```

```python
n = 0
while n < 10:
    n = n + 1
    if n % 2 == 0: # 如果n是偶数，执行continue语句
        continue # continue语句会直接继续下一轮循环，后续的print()语句不会执行
    print(n)
```

```
1
3
5
7
9
```

要特别注意，不要滥用 break 和 continue 语句。break 和 continue 会造成代码执行逻辑分叉过多，容易出错。大多数循环并不需要用到 break 和 continue 语句，上面的两个例子，都可以通过改写循环条件或者修改循环逻辑，去掉 break 和 continue 语句。

有些时候，如果代码写得有问题，会让程序陷入“死循环”，也就是永远循环下去。这时可以用 Ctrl+C 退出程序，或者强制结束 Python 进程。

函数

1. 函数是什么

函数（Functions）是指可重复使用的程序片段。它们允许你为某个代码块赋予名字，允许你通过这一特殊的名字在你的程序任何地方来运行代码块，并可重复任何次数。这就是所谓的调用（Calling）函数。

在 Python 中，函数可以通过关键字 def 来定义。这一关键字后跟一个函数的标识符名称，再跟一对圆括号，其中可以包括一些变量的名称，再以冒号结尾，结束这一行。随后而来的语句块是函数的一部分。

在定义函数时给定的名称称作“形参”（Parameters），在调用函数时你所提供给函数的值称作“实参”（Arguments）。

2. 调用函数

要调用一个函数，需要知道函数的名称和参数。函数的参数只是输入到函数之中，以便我们可以传递不同的值给它，并获得相应的结果。

Python 内置的常用函数包括数据类型转换函数，比如 int()函数可以把其他数据类型转换为整数。用 input()读取用户的输入：

```
In [1]: birth = input('birth:')
        if birth < 2000:
            print('00前')
        else:
            print('00后')

        birth:1982

        ---------------------------------------------------------------------------
        TypeError                                 Traceback (most recent call last)
        <ipython-input-1-15df9f612505> in <module>()
              1 birth = input('birth:')
        ----> 2 if birth < 2000:
              3     print('00前')
              4 else:
              5     print('00后')

        TypeError: '<' not supported between instances of 'str' and 'int'
```

因为 input()返回的数据类型是 str，str 不能直接和整数比较，必须先把 str 转换成整数。Python 提供了 int()函数来完成这件事情：

```
In [5]: s = input('birth:')
        birth = int(s)
        if birth < 2000:
            print('00前')
        else:
            print('00后')

        birth:1980
        00前
```

函数名其实就是指向一个函数对象的引用，完全可以把函数名赋给一个变量，相当于给这个函数起了一个“别名”：

```
a = abs # 变量a指向abs函数
a(-1) # 所以也可以通过a调用abs函数
```

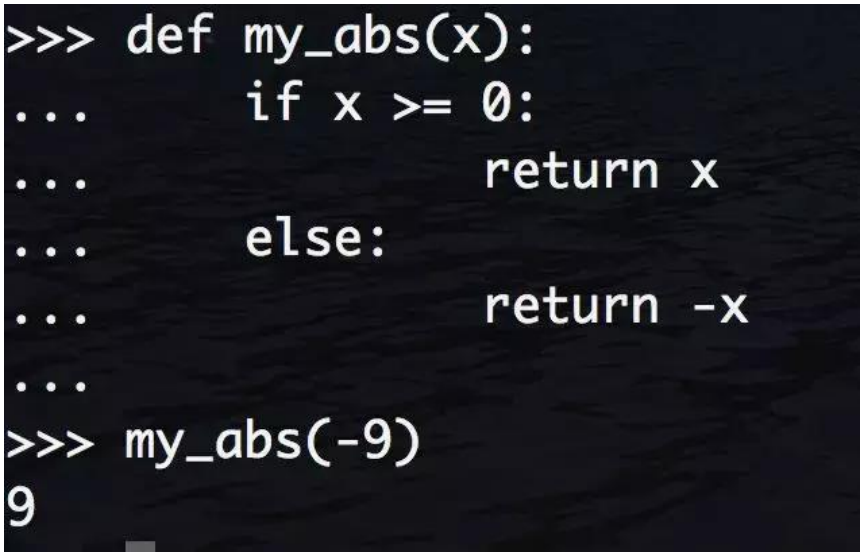
```
1
```

如果函数调用出错，一定要学会看错误信息。

3.定义函数

在 Python 中，定义一个函数要使用 def 语句，依次写出函数名、括号、括号中的参数和冒号:，然后，在缩进块中编写函数体，函数的返回值用 return 语句返回。

在 Python 交互环境中定义函数时，注意 Python 会出现...的提示。函数定义结束后需要按两次回车重新回到>>>提示符下：



如果你已经把 my_abs()的函数定义保存为 abstest.py 文件了，那么，可以在该文件的当前目录下启动 Python 解释器，用 from abstest import my_abs 来导入 my_abs()函数，注意 abstest 是文件名（不含.py 扩展名）。

定义一个什么事也不做的空函数，可以用 pass 语句：

```
def nop(): pass
```

pass 语句什么都不做，实际上它可以用作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个 pass，让代码能运行起来。

pass 还可以用在其他语句里，比如：

```
if age >= 18: pass
```

缺少了 pass，代码运行就会有语法错误。

数据类型检查可以用内置函数 isinstance()实现。

Python 的函数返回多值其实就是返回一个 tuple；Python 函数返回的是单一值时，返回值仍然是一个 tuple。但是，在语法上，返回一个 tuple 可以省略括号，而多个变量可以同时接收一个 tuple，按位置赋给对应的值。函数可以同时返回多个值，但其实就是一个 tuple。

函数执行完毕也没有 return 语句时，自动 return None。

4.函数的参数

Python 的函数定义非常简单，但灵活度却非常大。除了正常定义的必选参数外，还可以使用默认参数、可变参数和关键字参数，使得函数定义出来的接口，不但能处理复杂的参数，还可以简化调用者的代码。

4.1 位置参数：

```
def power(x, n):
    s = 1
    while n>0:
        n = n-1
        s = s * x
    return s
```

```
power(2, 2)
```

```
4
```

power(x, n)函数有两个参数：x 和 n，这两个参数都是位置参数，调用函数时，传入的两个值按照位置顺序依次赋给参数 x 和 n。

4.2 默认参数：

对于一些函数来说，你可能为希望使一些参数可选并使用默认的值，以避免用户不想为他们提供值的情况。默认参数值可以有效帮助解决这一情况。你可以通过在函数定义时附加一个赋值运算符=来为参数指定默认参数值。要注意到，默认参数值应该是常数。更确切地说，默认参数值应该是不可变的。

```python
def power(x, n=2):
    s = 1
    while n > 0:
        n = n - 1
        s = s * x
    return s
```

```python
power(5)
```

```
25
```

n = 2 是默认参数

定义默认参数要牢记一点：默认参数必须指向不变对象。且只有那些位于参数列表末尾的参数才能被赋予默认参数值，意即在函数的参数列表中拥有默认参数值的参数不能位于没有默认参数值的参数之前。

4.3 可变参数：

有时你可能想定义的函数里面能够有任意数量的变量，也就是参数数量是可变的，这可以通过使用星号来实现。即传入的参数个数是可变的。

要定义出这个函数，我们必须确定输入的参数。由于参数个数不确定，我们首先想到可以把a，b，c……作为一个list或tuple传进来，这样，函数可以定义如下：

```
def calc(numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

但是调用的时候，需要先组装出一个list或tuple：

```
>>> calc([1, 2, 3])
14
>>> calc((1, 3, 5, 7))
84
```

如果利用可变参数，调用函数的方式可以简化成这样：

```
>>> calc(1, 2, 3)
14
>>> calc(1, 3, 5, 7)
84
```

所以，我们把函数的参数改为可变参数：

```
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

定义可变参数和定义一个list或tuple参数相比，仅仅在参数前面加了一个 * 号。在函数内部，参数 numbers 接收到的是一个tuple，因此，函数代码完全不变。但是，调用该函数时，可以传入任意个参数，包括0个参数：

如果已经有一个list或者tuple，要调用一个可变参数怎么办？可以这样做：

```
>>> nums = [1, 2, 3]
>>> calc(nums[0], nums[1], nums[2])
14
```

这种写法当然是可行的，问题是太繁琐，所以Python允许你在list或tuple前面加一个 * 号，把list或tuple的元素变成可变参数传进去：

```
>>> nums = [1, 2, 3]
>>> calc(*nums)
14
```

*nums 表示把 nums 这个list的所有元素作为可变参数传进去。这种写法相当有用，而且很常见。

我们声明一个诸如 *param 的星号参数时，从此处开始直到结束的所有位置参数（Positional Arguments）都将被收集并汇集成一个称为 param 的元组（Tuple）。

类似地，当我们声明一个诸如 **param 的双星号参数时，从此处开始直至结束的所有关键字参数都将被收集并汇集成一个名为 param 的字典（Dictionary）。

4.4 关键字参数：

如果你有一些具有许多参数的函数，而你又希望只对其中的一些进行指定，那么你可以通过命名它们来给这些参数赋值——这就是关键字参数（Keyword Arguments）——我们使用命名（关键字）而非位置来指定函数中的参数。

关键字参数允许你传入 0 个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个 dict。

举个例子，扩展函数的功能。试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，利用关键字参数来定义这个函数就能满足注册的需求。

```
def person(name, age, **kw):
    print('name:', name, 'age:', age, 'other:', kw)
```

```
person('Sue', 22, job = 'Data Scientist')
```

```
name: Sue age: 22 other: {'job': 'Data Scientist'}
```

和可变参数类似，也可以先组装出一个 dict，然后，把该 dict 转换为关键字参数传进去：

```
>>> extra = {'city': 'Beijing', 'job': 'Engineer'}
>>> person('Jack', 24, **extra)
name: Jack age: 24 other: {'city': 'Beijing', 'job': 'Engineer'}
```

4.5 命名关键字参数：

如果要限制关键字参数的名字，就可以用命名关键字参数，例如，只接收 city 和 job 作为关键字参数。这种方式定义函数并调用：

```
def person(name, age, *, city, job):
    print(name, age, city, job)
```

```
person('Sue', 22, city='Shanghai', job='Data Scientist')
```

```
Sue 22 Shanghai Data Scientist
```

和关键字参数**kw 不同，命名关键字参数需要一个特殊分隔符*，*后面的参数被视为命名关键字参数。

命名关键字参数必须传入参数名，这和位置参数不同。如果没有传入参数名，调用将报错。

使用命名关键字参数时，要特别注意，如果没有可变参数，就必须加一个*作为特殊分隔符。如果缺少*，Python 解释器将无法识别位置参数和命名关键字参数，即缺少 *，city 和 job 被视为位置参数。

如果函数定义中已经有了一个可变参数，后面跟着的命名关键字参数就不再需要一个特殊分隔符 * 了：

```
def person(name, age, *args, city, job):
    print(name, age, args, city, job)
```

4.6 参数组合：

在 Python 中定义函数，可以用必选参数、默认参数、可变参数、关键字参数和命名关键字参数，这 5 种参数都可以组合使用。

但是参数定义的顺序必须是：必选参数、默认参数、可变参数、命名关键字参数和关键字参数。虽然可以组合多达 5 种参数，但不要同时使用太多的组合，否则函数接口的可理解性很差。

通过一个 tuple 和 dict，你也可以调用函数：

```
def f1(a, b, c=0, *args, **kw):
    print('a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw)
```

```
args = (1, 2, 3, 4)
kw = {'d': 99, 'x': '#'}
```

```
f1(*args, **kw)
```

```
a = 1 b = 2 c = 3 args = (4,) kw = {'d': 99, 'x': '#'}
```

对于任意函数，都可以通过类似 func(*args, **kw)的形式调用它，无论它的参数是如何定义的。

5. 递归函数

如果一个函数在内部调用自身本身，这个函数就是递归函数。理论上，所有的递归函数都可以写成循环的方式，但循环的逻辑不如递归清晰。

使用递归函数需要注意防止栈溢出。在计算机中，函数调用是通过栈（stack）这种数据结构实现的，每当进入一个函数调用，栈就会加一层栈帧，每当函数返回，栈就会减一层栈帧。由于栈的大小不是无限的，所以，递归调用的次数过多，会导致栈溢出。

通过下面的代码可以查看你的电脑最大算到多少：

```
def fact(n):
    if n==1:
        return 1
    return n * fact(n-1)
```

```
fact(2971)
```

```
69251026566049636281834760348995573534614515322048681418675506546743706723821478866315332876O
822106214754903923512586151845331083772732377804319650186840690923737967338032523146142008327
898361066315839620308863295578235549847079741187511878079561979452703415208369768322252500687
061416986104644786830930151201489268857171158283660474545095283144986424492557948920879738742
427631247079167578853396953892468956263233135650698158437664628824339892003518844396985033415
732017011730323282403508866742218659436488794914792609148463587899656709592647667908132076102
```

解决递归调用栈溢出的方法是通过尾递归优化，事实上尾递归和循环的效果是一样的，所以，把循环看成是一种特殊的尾递归函数也是可以的。

尾递归是指，在函数返回的时候，调用自身本身，并且，return 语句不能包含表达式。这样，编译器或者解释器就可以把尾递归做优化，使递归本身无论调用多少次，都只占用一个栈帧，不会出现栈溢出的情况。

要改成尾递归方式，需要多一点代码，主要是要把每一步的乘积传入到递归函数中。Python 标准的解释器没有针对尾递归做优化，任何递归函数都存在栈溢出的问题。

```
def fact(n):
    if n==1:
        return 1
    return n * fact(n-1)
```

```
fact(2971)
```

```
69251026566049636281834760348995573534614515322048681418675506546743706723821478866315332876 0
82210621475490392351258615184533108377273237780431965018684069092373796733803252314614200832 7
89836106631583962030886329557823554984707974118751187807956197945270341520836976832225250068 7
06141698610464478683093015120148926885717115828366047454509528314498642449255794892087973874 2
42763124707916757885339695389246895626323313565069815843766462882433989200351884439698503341 5
73201701173032328240350886674221865943648879491479260914846358789965670959264766790813207610 2
```

模块

学会函数，工作中让你省一半力气，但是 Python 的优势就是还能再省力。比如中位数、标准差等，依旧需要写代码，有没有现成的直接调用呢？

第一种思路是搜索，「python 中位数」和「python 标准差」的关键词在网上可以搜出一堆，直接参考即可。

第二种思路是调包，世界上好心人很多，他们已经写好了现成的诸多功能，把它们共享在了网上，这些功能统一做成了「包」。

包的概念类似于文件夹，文件夹中放着很多文件，一个.py 文件就称之为一个模块（Module）。包括 Python 内置的模块和来自第三方的模块。.py 文件中包含着具体的代码，代码太多会显得比较凌乱，为了规范和整理代码，引入了「类」。类是一种抽象的概念，是面向对象编程的核心，数据分析不用深入理解这块，只要知道类是各种函数的集合，方便复用，用起来很简单就行。

dir 函数

内置的 dir()函数能够返回由对象所定义的名称列表。

如果这一对象是一个模块，则该列表会包括函数内所定义的函数、类与变量。该函数接受参数。

如果参数是模块名称，函数将返回这一指定模块的名称列表。

如果没有提供参数，函数将返回当前模块的名称列表。

包

变量通常位于函数内部，函数与全局变量通常位于模块内部。如果你希望组织起这些模块的话，就需要包（Packages）。

Python 引入了按目录来组织模块的方法，称为包（Package）。它是一种能够方便地分层组织模块的方式。包是指一个包含模块与一个特殊的 __init__.py 文件的文件夹，这个文件是必须存在的，否则，Python 就把这个目录当成普通目录，而不是一个包。__init__.py 可以是空文件，也可以有 Python 代码，因为__init__.py 本身就是一个模块。

使用模块

在一个模块中，我们可能会定义很多函数和变量，但有的函数和变量我们希望给别人使用，有的函数和变量我们希望仅仅在模块内部使用。在 Python 中，是通过_前缀来实现的。

类似__xxx__这样的变量是特殊变量，可以被直接引用，但是有特殊用途，比如__author__，__name__就是特殊变量。我们自己的变量一般不要用这种变量名。

类似_xxx 和__xxx 这样的函数或变量就是非公开的（private），不应该被直接引用，比如_abc，__abc 等。

安装第三方模块

对应的 pip 命令是 pip3。

默认情况下，Python 解释器会搜索当前目录、所有已安装的内置模块和第三方模块，搜索路径存放在 sys 模块的 path 变量中：

```
>>> import sys
>>> sys.path
['', '/anaconda3/lib/python36.zip', '/anaconda3/lib/python3.6', '/anaconda3/lib/
python3.6/lib-dynload', '/anaconda3/lib/python3.6/site-packages', '/anaconda3/li
b/python3.6/site-packages/aeosa', '/anaconda3/lib/python3.6/site-packages/pip-10
.0.0b2-py3.6.egg', '/anaconda3/lib/python3.6/site-packages/six-1.11.0-py3.6.egg'
]
```

Python 提供了非常丰富的包和模块，合理应用这些模块将极大程度的提供数据分析能力。numpy、scipy、pandas 是数据分析最常用的三个包，matplotlib、seaborn 是常用的绘图包，scikit-learn、Gensim、NLTK 是机器学习相关的包，urllib、BeautifulSoup 是常用的爬虫包等。

后面，我会花几篇文章介绍用 numpy、Pandas 进行数据分析。

往期内容：

学习计划丨带你 10 周入门数据分析

如何炼就数据分析的思维？

数据分析惯用的 5 种思维方法

数据分析必备的 43 个 Excel 函数，史上最全！

实操：如何用 Excel 做一次数据分析

写给新人的数据库入门指南

零基础快速自学 SQL，2 天足矣！

数据分析必掌握的统计学知识