Python 编程基础学习笔记

Cheng Jun

2018 年 9 月 27 日

目录

1 前言

1.1 笔记作者

实证研究小青年，日常研究和关注经济、金融与会计等领域的问题，主要采用计量经济学和其他数据分析手法撰写学术论文和研究报告。研究之余，泛读文史哲，关注自由与开源动态。在日常研究和工作中，喜欢选用自由和开源工具，喜欢使用最新的软件工具。

邮箱：cheng081@qq.com

Github：https://github.com/chengjun90

欢迎交流。

1.2 笔记目的

本学习笔记主要为~~特定领域的数据科学~~（实证研究）服务，力求**简明够用即可**，用于构建知识体系，快速浏览全局。

更多的细节都是干中搜:

- 直接谷歌、百度
- 查找 Python 官方手册

1.3 Python 简介

Python 是非常简洁易懂的通用编程语言。Python 适应于多个应用领域，在数据科学、机器学习和深度学习等领域得到广泛的应用。

Python 代码简洁、易读，上手快。缺点之一是性能不如 C/C++ 等语言。

1.4 笔记的系统环境

```
In [1]: import sys
        import platform
        f'操作系统：{platform.platform()};Python 版本：{sys.version[0:5]}'

Out[1]: '操作系统：Windows-10-10.0.17134-SP0;Python 版本：3.7.0'

In [2]: from IPython.core.interactiveshell import InteractiveShell
        InteractiveShell.ast_node_interactivity = "all"
```

2 变量与数据类型

2.1 变量

Python 是动态类型，赋值时无需声明类型。变量可以看作名称（id）与值的关联。

2.1.1 技巧

解包

```
In [3]: a,b='hi'
        a
        b
        a,b=["哈哈","呵呵"]
        a
        b

Out[3]: 'h'

Out[3]: 'i'

Out[3]: '哈哈'

Out[3]: '呵呵'
```

多个变量赋值

```
In [4]: a,b,c="呵呵","喝喝","哈哈"
        a
        b
        c

Out[4]: '呵呵'

Out[4]: '喝喝'

Out[4]: '哈哈'
```

交换值

```
In [5]: a=1
        b=2
        a,b=b,a
        a
        b

Out[5]: 2

Out[5]: 1
```

2.2 基本数据类型

2.2.1 数值

整型 int 和浮点型 float。
高精度使用 decimal 模块。
Python 内置对复数的计算，对复数处理的数学函数在模块 cmath 中。

```
In [6]: type(1)
        type(1.1)
        type(1+2j)

Out[6]: int

Out[6]: float

Out[6]: complex

In [7]: (1+2j)*(2+3j)

Out[7]: (-4+7j)
```

2.2.2 字符

Python 中字符串不可修改，只能生成新的字符串。
字符串可以使用单引号、双引号来标记。字符串是多行的时候使用三引号。
字符串格式化，直接使用 f 方法好了。

```
In [8]: # printf style
        '我喜欢%s，学习了%d年'%('python', 3)
```

```
Out[8]: '我喜欢 python, 学习了 3 年'

In [9]: # format 方法
        '我喜欢{}, 学习了{}年'.format('python',3)
        '我喜欢{a}, 学习了{b}年'.format(b=3,a='python')

Out[9]: '我喜欢 python, 学习了 3 年'

Out[9]: '我喜欢 python, 学习了 3 年'

In [10]: # f 方法
         a="Python"
         b=3
         f'我喜欢{a}, 学习了{b}年'

Out[10]: '我喜欢 Python, 学习了 3 年'
```

2.2.3 逻辑值

逻辑值仅包括 True/False 两个, 用来配合 if/while 等语句做条件判断, 其它数据类型可以转换为逻辑值。

标准的布尔运算符为 and , or, not。

2.3 基本数据类型转换

方法	说明
int(x[,base])	将 x 转换为一个整数
float(x)	将 x 转换到一个浮点数
complex(real[,imag])	创建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效 Python 表达式, 并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
chr(x)	将一个整数转换为一个字符
unichr(x)	将一个整数转换为 Unicode 字符

方法	说明
ord(x)	将一个字符转换为它的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串

2.4 字节和 Unicode

在 Python 3 及以上版本中，字符串类型是 Unicode 编码。

可以用 encode 将这个 Unicode 字符串编码为 UTF-8：

```
In [11]: var="呵呵"
         var.encode('utf-8')
         type(var.encode('utf-8'))
         var.encode('utf-8').decode('utf-8')

Out[11]: b'\xe5\x91\xb5\xe5\x91\xb5'

Out[11]: bytes

Out[11]: '呵呵'
```

3 数据结构

数据结构是用来存储一系列相关数据的集合。

数据结构也可以称之为容器。Python 提供了许多高效的容器类型，其中可以存储对象的集合。

Python 中有四种内置的数据结构：列表（List）、元组（Tuple）、字典（Dictionary）和集合（Set）。

- 列表（list）：列表则可以删除、添加、替换、重排序列中的元素。可变类型。列表是异质的，这意味着列表中的数据不必要同一个类。
- 元组（Tuple）：元组是不可变类型。
- 字典（Dictionary）：字典是通过键值 key 来索引元素 value，而不是象列表是通过连续的整数来索引。字典是可变类型，可以添加、删除、替换元素。
- 集合：集合是不重复元素的无序组合。用 set() 创建空集，可用 set() 从其它序列转换生成集合。

```
In [12]: mylist=['a','b','b',123]
         mylist
```

```
Out[12]: ['a', 'b', 'b', 123]
```

```
In [13]: mytuple=('a','b','b',123)
         mytuple
```

```
Out[13]: ('a', 'b', 'b', 123)
```

```
In [14]: mydict={"name":"zhangsan","age":18}
         mydict
```

```
Out[14]: {'name': 'zhangsan', 'age': 18}
```

```
In [15]: myset = {1,2,1,3}
         myset
```

```
Out[15]: {1, 2, 3}
```

Python 3.7 引入了 dataclass 。

dataclass 一个简单的数据容器，支持通过的对象属性进行访问。

```
In [16]: from dataclasses import dataclass
         from typing import Any

         @dataclass
         class Student:
             name: str
             age: int=18
             score: Any=""

             def level(self,hour):
                 if self.score/hour>1:
                     return "优秀"
                 else:
                     return "其他"

         lisi = Student(name="李四",score=92.5)
         lisi.level(20)
```

```
Out[16]: '优秀'
```

4 判断与循环

4.1 判断

if-elif-else 语句

```
In [17]: import random
         a=random.gauss(0,1)

         if a>=1.65:
             print("真大")
         elif a>0:
             print("还行")
         else:
             print("结束")
```

```
结束
```

三元操作符是 if-else 语句也就是条件操作符的一个快捷方式:
[表达式为真的返回值] if [表达式] else [表达式为假的返回值]

```
In [18]: a=random.gauss(0,1)

         "还行" if a>=0 else "结束"
```

```
Out[18]: '还行'
```

4.2 循环

Python 提供了 for 循环和 while 循环。
迭代循环 for:

```
for <变量> in <可迭代对象>:
    <语句块>
    break # 跳出循环
    continue # 略过余下循环语句
else: # 迭代完毕，则执行
    <语句块>
```

while:

```
while 条件:
    语句块
    break # 跳出循环
    continue # 跳出余下的语句块
    语句块
else: # 条件不满足则跳出，执行以下语句块
    语句块
```

控制循环的语句:

循环控制语句	描述
break	在语句块执行过程中终止循环，并且跳出整个循环
continue	在语句块执行过程中终止当前循环，跳出该次循环，执行下一次循环
pass	pass 是空语句，是为了保持程序结构的完整性

```
In [19]: s="我的数据分析工具箱"
         for  i in "的箱":
             if i in s:
                 print(i)
```

```
的
箱
```

```
In [20]: list1 = ["这是", "一个", "测试"]

         for index, item in enumerate(list1):
             print(index, item)

         for index, item in enumerate(list1, 100):
             print(index, item)
```

```
0 这是
1 一个
```

```
2 测试
100 这是
101 一个
102 测试
```

```
In [21]: a=[1,2,3]
         b=["a","b","c"]
         c=zip(a,b)
         c
         list(c)

         for i,j in zip(a,b):
             print(i,j)
```

```
Out[21]: <zip at 0x1903cffcc88>
```

```
Out[21]: [(1, 'a'), (2, 'b'), (3, 'c')]
```

```
1 a
2 b
3 c
```

```
In [22]: a=4

         while a:
             print(a)
             if a<3:
                 break
             a=a-1
         else:
             print("结束")
```

```
4
3
2
```

4.3 运算符：

Python 语言支持以下类型运算符：

- 算术运算符
- 比较运算符
- 赋值运算符
- 逻辑运算符：not，and，or
- 成员操作符：in，not in
- 身份运算符：is

5 高级特性

5.1 切片

取一个 list 或 tuple 的部分元素是非常常见的操作。比如，一个 list 如下：

```
In [23]: L = ['a', 'b', 'c', 'd', 'e']
         L[2:4]


         strs="取一个 list 或 tuple 的部分元素是非常常见的操作。"
         strs[-6::]
```

```
Out[23]: ['c', 'd']
```

```
Out[23]: '常见的操作。'
```

5.2 迭代器

5.3 生成器

可以节约内存。

```
In [24]: a=[0,1,2,3,4,5]
         a2=(2*i for i in a)
         a2
         for i in a2:
             print(i)
```

```
Out[24]: <generator object <genexpr> at 0x000001903CFEC840>

0
2
4
6
8
10
```

5.4 推导式

可以生成 list、dict 和 set。

- 代码简洁
- 可读性强

[表达式 for 变量 in 可迭代对象 if 逻辑条件]
{键值表达式: 元素表达式 for 变量 in 可迭代对象 if 逻辑条件}
{元素表达式 for 变量 in 可迭代对象 if 逻辑条件}

```
In [25]: a=[0,1,2,3,4,5]
         [i+100 for i in a if i%2==0]

Out[25]: [100, 102, 104]

In [26]: {k:v for k,v in zip('推导式',range(3))}

Out[26]: {'推': 0, '导': 1, '式': 2}
```

6 函数式编程

```
In [27]: def mysum(num1,num2):
             num=num1+num2
             return num
         mysum(1,2)

Out[27]: 3
```

通过上面的学习，可以知道通过 return 语句返回值。

不带参数值的 return 语句返回 None。

函数可以返回多个值。该函数其实只返回了一个对象，也就是一个元组，最后该元组会被拆包到各个结果变量中。

```
In [28]: def f():
             a = 5
             b = 6
             c = 7
             return a, b, c

         a, b, c = f()
         a
         b
         c

Out[28]: 5

Out[28]: 6

Out[28]: 7
```

默认参数

```
In [29]: def mysum(num1=0,num2=0):
             num=num1+num2
             return num
         mysum(num2=100)

Out[29]: 100
```

匿名函数

```
In [30]: list(map(lambda x: x * x, [1, 2, 3, 4]))

Out[30]: [1, 4, 9, 16]
```

此外还有

- 作用域
- 闭包
- 装饰器

7 面向对象编程

Python 中的所有事物都是以对象形式存在，从简单的数值类型，到复杂的代码模块，都是对象。

同一个类（class）的对象具有相同的属性和方法。

对象实现了属性和方法的封装，是一种数据抽象机制。

面向对象三个特点：

- 继承
- 多态
- 封装性

平时的分析场景不需要多少面向对象编程的地方，简单使用即可。属性 + 方法。

下面举一个例子，学生类型，和博士生子类型。

```
In [31]: class Student:
             def __init__(self, id, score):
                 self.id = id
                 self.score = score
             def pressure(self):
                 if self.score>80:
                     print("还不错")
                 else:
                     print("需要加油")

         class Phd(Student):
             def __init__(self, id, score, paper):
                 Student.__init__(self, id, score)
                 # 或 super().__init__(id, score)
                 self.paper = paper

             def pressure(self):
                 if self.score>60:
                     print("还不错")
                 else:
                     print("需要加油")
```

```
        def pressure_paper(self,number):
            if self.paper=="A" and number>=1:
                return f"{self.id}发了{self.paper}刊共计{number}篇，可以参加预答辩了
            elif self.paper=="B" and number>=2:
                return f"{self.id}发了{self.paper}刊共计{number}篇，可以参加预答辩了
            else:
                return f"{self.id}发了{self.paper}刊共计{number}篇，尚未达到预答辩条
```

```
In [32]: zhang3=Student("2018001",81)
         zhang3.id
         zhang3.score
         zhang3.pressure()
```

```
Out[32]: '2018001'
```

```
Out[32]: 81
```

还不错

```
In [33]: li4=Phd("2018002",61,"B")
         li4.id
         li4.score
         li4.paper
         li4.pressure()
         li4.pressure_paper(2)
```

```
Out[33]: '2018002'
```

```
Out[33]: 61
```

```
Out[33]: 'B'
```

还不错

```
Out[33]: '2018002 发了 B 刊共计 2 篇，可以参加预答辩了'
```

8 数据读写

8.1 简单输入和输出

```
In [34]: use_name=input("请输入你的名字")
```

```
请输入你的名字 cheng jun
```

```
In [35]: use_name
```

```
Out[35]: 'cheng jun'
```

print(v1,v2,v3) 打印各个变量

```
In [36]: print("打印","各个变量")
```

```
打印 各个变量
```

8.2 基础 IO

- open
- read 或者 write
- close

由于文件读写时都有可能产生 IOError，所以还要借助 try 函数。

```
try:
    f = open('Python-Data-Science/file.txt', 'r')
    print(f.read())
finally:
    if f:
        f.close()
```

这句话就可以保证无论是否出错都能正确地关闭文件。
这样太麻烦。所以就引出 with 语句。

```
with open('Python-Data-Science/file.txt', 'r', encoding='gbk') as f:
    print(f.read())
```

写文件和读文件是一样的，唯一区别是调用 open() 函数时的参数不同：

- 传入标识符 `r` 或者 `rb` 表示写文本文件或写二进制文件
- 传入标识符 `w` 或者 `wb` 表示写文本文件或写二进制文件
- 在 open() 函数传入 encoding 参数，将字符串自动转换成指定编码
- 以’w’ 模式写入文件时，如果文件已存在，会直接覆盖已有文件
- 以’a’ 模式写入文件时，以追加（append）模式写入

8.3 csv

这个需要 csv 包，或者是 pandas

8.4 二进制文件

要读取二进制文件，比如图片、视频等等，用 `rb` 模式打开文件即可，写入二进制文件，用 `wb` 即可。

```
f = open('Python-Data-Science/image/lambda.png', 'rb')
f.read()[0:10]
```

9 文本处理

常见四把斧子。

- Python 的字符串属性函数
- Python 的 string 模块
- re
- 其他

10 异常处理

Python 在 builtins 模块内置了常见的异常，无需特别导入，直接就可使用。

```
In [37]: try:
             a = 1 / 0
         except ZeroDivisionError as e:
             print(e)
         else:
             print("没有错误")
         finally:
             print("运行结束。本语句一定会运行。")
```

```
division by zero
运行结束。本语句一定会运行。
```

通用异常：Exception

在 Python 的异常中，有一个通用异常：Exception，它可以捕获任意异常。

```
In [38]: try:
             print(1/0)
         except Exception as e:
             print("我去，又出错了。\n错误提示信息：", e, "。")
```

```
我去，又出错了。
错误提示信息： division by zero 。
```

11 调试与测试

11.1 调试

- print
- assert
- logging

- pdb
- IDE

asert 会对后面的表达式进行判断，如果表达式为 True，那就什么都不做，程序接着往下走；如果 False，那么就会弹出异常。

```
In [39]: def myfun(s):
             assert type(s)==int
             return s/2

In [40]: myfun(18)  # 正常运行

Out[40]: 9.0

In [41]: myfun(18.0)  # 触发错误
```

```
---------------------------------------------------------------------------

AssertionError                            Traceback (most recent call last)

<ipython-input-41-28fb6113b9bb> in <module>()
----> 1 myfun(18.0)  # 触发错误


<ipython-input-39-1715a7599229> in myfun(s)
      1 def myfun(s):
----> 2     assert type(s)==int
      3     return s/2


AssertionError:
```

11.2 性能测试

Python 提供 timeit 来测试性能。
profile 和 pstats 模块提供了针对更大代码块的性能测试。

```
In [42]: from timeit import timeit

         def myfun():
             [i for i in range(1000000) if i%3==0]
```

timeit(函数名 _ 字符串，运行环境 _ 字符串，number= 运行次数)

```
In [44]: timeit('myfun()', 'from __main__ import myfun', number=2)

Out[44]: 0.13829108199999496

In [45]: timeit('[i for i in range(1000000) if i%3==0]', number=2)

Out[45]: 0.13907842400001869

In [46]: %time myfun()

Wall time: 62.5 ms
```

此外还可以使用 repeat 方法。

```
from timeit import repeat
pass
repeat('func()', 'from __main__ import func', number=100, repeat=5)
```

11.3 单元测试

做探索性数据分析的时候很少做单元测试。
跳过。

12 模块与包

12.1 模块

12.1.1 导入

模块主要是以下导入方式。

```
import xx.xx

from xx.xx import xx

# 给导入的对象重命名
from xx.xx import xx as newname

# 将对象内的所有内容全部导入，容易发生命名冲突，慎用
from xx.xx import *
```

在导入的时候，需要提供导入对象的绝对路径，也就是“最顶层的包名. 次一级包名. 模块名. 类名. 函数名”。

对于 xx.xx，想导入到哪个级别，取决于自己需要。

标准库，直接 import 模块名，例如 `import os`。

12.1.2 主模块和非主模块

如果一个模块被直接使用，而没有被别人调用，称这个模块为主模块。如果一个模块被别人调用，称这个模块为非主模块。

__name__ 属性值是一个变量，且这个变量是系统给出的。利用这个变量可以判断一个模块是否是主模块。如果一个属性的值是 **main** , 那么就说明这个模块是主模块。

```
if __name__ == '__main__':
    main()
```

上面的代码很常见。

在同一目录建立以下代码。

```
def myfun():
    print(__name__)

if __name__=='__main__':
    myfun()
```

上述代码运行结果:

```
[Running] python "f:\testmain.py"
__main__
```

```
import testmain


testmain.myfun()
```

上述代码运行结果:

```
[Running] python "test.py"
testmain
```

testmain 是包的名字 testmain。

- if __name__ == '__main__'自己调用时 __name__ 为'__main__'，条件成立，执行 if 语句中函数
- if __name__ == '__main__'从别的文件调用时 __name__ 为’ 调用的文件名’，if 条件不成立，则不执行。

12.2 包

包是用以组织模块的另一种层次结构。
包是使用包含文件 __init__.py 的目录实现的。

12.3 pip 软件包管理

12.3.1 安装

```
pip install packageName 下载并安装最新的版本
pip install packageName==1.0.0下载并安装指定版本
pip install packageName>=1.0.0 下载并安装至少某个版本以上的版本的包
pip install url # 从指定网址资源安装
pip install path # 指定本地位置安装
pip install --find-links=url 从指定 url 下载安装
pip install --find-links=path 从指定 path 下载安装
pip install --upgrade packageName 更新一个已经安装过的过期模块
```

同时安装多个包：python -m pip install –user numpy scipy matplotlib ipython jupyter pandas sympy nose
提示安转时缺失包：python –m pip install qtconsole
在 Win10 的 ps 界面上使用 pip 需要管理者权限，或者是在命令后面加上--user。

12.3.2 卸载

```
pip uninstall <packageName>
```

12.3.3 查看模块信息

```
pip show <packageName>
```

查看 pip 管理了哪些模块

```
pip list -o  列示过期的包
pip list -u  列示可以更新到最新包的信息
```

pip 升级自己

```
python -m pip install --upgrade pip
```

查看 pip 版本:

```
PS C:\Users\cheng> pip -V
pip 18.0
```

此外，pip 安装在本地有比较大的缓存文件，可以手动检测并删除。
除了匹配管理包，还可以使用 conda。

12.4 pypi 镜像

https://mirrors.tuna.tsinghua.edu.cn/help/pypi/
pypi 镜像每 5 分钟同步一次。
临时使用

```
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple some-package
```

长久使用
找个地方建立 pip.ini 文件，如 `C:\Users\Cheng\pip\pip.ini`，文件内容如下:

```
[global]
index-url = https://pypi.tuna.tsinghua.edu.cn/simple
```

然后把 `C:\Users\Cheng\pip\pip.ini` 加入到系统环境变量的 `Path` 中去。

13 杂七杂八

13.1 三个常用的函数

- help
- type
- dir

```
In [47]: help("dir")
```

```
Help on built-in function dir in module builtins:

dir(…)
    dir([object]) -> list of strings

    If called without an argument, return the names in the current scope.
    Else, return an alphabetized list of names comprising (some of) the attributes
    of the given object, and of attributes reachable from it.
    If the object supplies a method named __dir__, it will be used; otherwise
    the default dir() logic is used and returns:
      for a module object: the module's attributes.
      for a class object:  its attributes, and recursively the attributes
        of its bases.
      for any other object: its attributes, its class's attributes, and
        recursively the attributes of its class's base classes.
```

```
In [48]: type(108)
```

```
Out[48]: int
```

```
In [49]: a=[1,2,3]
         dir(a)[-5::]
```

```
Out[49]: ['insert', 'pop', 'remove', 'reverse', 'sort']
```

13.2 类型提示

```
In [50]: a: int =99
         b: bool = True
         f: float = 1.1
         s: str = "abc"
```

复杂的类型和注解，需要引入标准库 typing。

```
In [51]: import typing

         def myfun(lst: typing.List[int]) -> typing.List[int]:
             lst = [i*10 for i in lst]
             return lst

         myfun([1,5,8])

Out[51]: [10, 50, 80]
```

13.3 关键字

```
In [52]: import keyword
         len(keyword.kwlist)
         keyword.kwlist

Out[52]: 35

Out[52]: ['False',
          'None',
          'True',
          'and',
          'as',
          'assert',
          'async',
          'await',
          'break',
          'class',
```

```
        'continue',
        'def',
        'del',
        'elif',
        'else',
        'except',
        'finally',
        'for',
        'from',
        'global',
        'if',
        'import',
        'in',
        'is',
        'lambda',
        'nonlocal',
        'not',
        'or',
        'pass',
        'raise',
        'return',
        'try',
        'while',
        'with',
        'yield']
```

13.4 内置函数

在启动 Python 解释器的时候，内置函数就已经导入到内存当中供我们使用。

```
In [53]: dir(__builtins__)[0:3]

Out[53]: ['ArithmeticError', 'AssertionError', 'AttributeError']

In [54]: len(dir(__builtins__))

Out[54]: 154
```

13.5 小整数对象池

Python 初始化的时候会自动建立一个小整数对象池，范围是-5 到 256。

例如 123，即使我们在程序里没有创建它，其实在 Python 后台已经悄悄为我们创建了。

```
In [55]: a=123
         id(a)==id(123)

Out[55]: True

In [56]: a=1230
         id(a)==id(1230)

Out[56]: False
```

13.6 is 和 ==

- is 是比较两个引用是否指向了同一个对象（引用比较）。
- == 是比较两个对象是否相等。

14 未处理知识点

- 线程
- 进程

这些知识点，因为自己平时用的少，学得也是一般。跳过。

15 后记

在刚学习 Python 就开始记笔记，因为那个时候 Visual Basic 和 C 长久不用，具体知识点基本遗忘。Python 是在这样的背景下，因为有现实的需求，而重拾编程，开始学习的新语言。

一开始的笔记繁杂，几经删改，最后发现没有必要做一个 cookbook 式的笔记。简单，有体系，方便速览。有事就搜索。

后面有需要，并且有空的时候会对这份笔记进行修订和更新。