Python金融计算（第一讲）：Python基础

谢文杰

华东理工大学金融专业硕士（MF）· 量化金融模块专业课

2022年春季

目录

目录

本节要点

本节包括以下知识点：

- 算法及表达式
- 常量与变量
- 语句及函数
- 模块
- 程序及字符串

算法与表达式

算法

算法犹如菜谱，告诉你如何完成特定的任务。从本质上说，编写计算机程序就是使用计算机能够理解的语言（如Python）描述一种算法。

表达式

表达式为程序的一部分，结果为一个值。例如，2+2就是一个表达式，结果为4。

简单表达式是使用运算符（如+或%）和函数（如pow）将字面值（如2或"Hello"）组合起来得到的。通过组合简单的表达式，可创建复杂的表达式，如(2+2)*(3-1)。表达式还可能包含变量。

常量与变量

常量

Python的**常量**主要是指在程序运行的过程中不可变的量。**其特点为：** 一旦绑定，不能更改。

如数字7和字符串"abc"在运行时一直都是数字7跟字符串"abc"，不会更改成其他的量。

变量

Python中在程序运行时可以随着程序的运行更改的量称之为**变量**。**其特点为：** 即使赋值，也可以更改。

比如我们可以定义一个变量i，并将数字5赋给变量i（即$i = 5$），然后再将数字7再赋给变量i（即$i = 7$），那么这个时候i的值就变成了7，i的值是可以改变的。

语句及函数

语句

语句是让计算机执行特定操作的指示。这种操作可能是修改变量（通过赋值）、将信息打印到屏幕上（如print("Hello, world!")）、导入模块或执行众多其他任务。

函数

Python函数类似于数学函数，它们可能接受参数，并返回结果。Python提供了很多函数，你也可以自己编写函数，这些函数可用来完成很多神奇的任务。

在学习编写自定义函数时，你将发现函数实际上可以在返回前做很多事情。

模块

模块

模块是扩展，可通过导入它们来扩展Python的功能。例如，模块math包含多个很有用的函数。

要导入模块，可使用特殊命令import。例如函数floor包含在模块math中。

```
>>> import math
>>> math.floor(32.9)
```

如果确定不会从不同模块导入多个同名函数，你可能不想每次调用函数时都指定模块名。可使用import的如下变种：

```
>>> from math import sqrt
>>> sqrt(9)
```

程序及字符串

交互式解释器是Python的亮点之一，它让你能够实时地测试解决方案以及尝试使用Python。然而，等你退出交互式解释器时，你在其中编写的所有代码都将丢失。你的终极目标是编写自己和他人都能运行的程序。

保存并执行程序

首先，你需要一个文本编辑器——最好是专门用于编程的。创建一个简单的程序：

```
print("Hello, world!")
```

选择菜单File→Save保存程序（其实就是一个纯文本文件），并指定合理的存储位置及文件名，如hello.py（扩展名.py很重要）。最后，可以运行这个程序，方法是选择菜单Run→Run Module。

程序及字符串

字符串

字符串其实就是一段文本，其中的字符是用Unicode码点表示的。对于字符串，需要学习的知识有很多。

字符串用途众多，但主要用途是表示一段文本，如感叹句“Hello, world!”。我们可以使用引号（'或"）来创建字符串。

字符串可以使用操作符+，但其功能和数学中的不一样，他会对前后字符进行拼接操作。如：

```
>>> "Hello, " + "world!"
'Hello, world!'
```

目录

本节要点

本节包括以下知识点：

- 序列概述
- 成员资格
- 列表：Python的主力
- 元组：不可修改的序列

序列概述

在Python中，最基本的数据结构为**序列**（sequence）。序列中的每个元素都有编号，即其位置或索引，**其中第一个元素的索引为0，第二个元素的索引为1**，依此类推。Python内置了多种序列，本章重点讨论其中最常用的两种：列表和元组。另一种重要的序列是字符串，将在下一节更详细地讨论。

通用的序列操作

有几种操作适用于所有序列，包括**索引、切片、相加、相乘和成员资格检查**。另外，Python 还提供了一些内置函数，可用于确定序列的长度以及找出序列中最大和最小的元素。

相关序列操作，将在实践中讲解演示。

成员资格

要检查特定的值是否包含在序列中，可使用运算符in。这个运算符与前面讨论的运算符（如乘法或加法运算符）稍有不同。它检查是否满足指定的条件，并返回相应的值：满足时返回True，不满足时返回False。这样的运算符称为**布尔运算符**，而前述真值称为布尔值。

in的使用示例

```
>>> permissions = 'rw'
>>> 'w' in permissions
True
>>> 'x' in permissions
False
```

列表：Python的主力

本节主要讨论列表不同于元组和字符串的地方——列表是可变的，即可修改其内容。另外，列表有很多特有的**方法**。

函数list

鉴于不能像修改列表那样修改字符串，因此在有些情况下使用字符串来创建列表很有帮助。为此，可使用函数list。

```
>>> list('Hello')
['H', 'e', 'l', 'l', 'o']
```

请注意，可将任何序列（而不仅仅是字符串）作为list的参数。

基本的列表操作

1. 修改列表：给元素赋值；2. 删除元素；3. 给切片赋值；

列表：Python的主力

列表方法

方法是与对象（列表、数、字符串等）联系紧密的函数。通常，像下面这样调用方法：

object.method(arguments)

方法调用与函数调用很像，只是在方法名前加上了对象和句点。列表包含多个可用来查看或修改其内容的方法。

列表常用方法

有：append、clear、copy、count、extend、index、insert、pop、remove、reverse、sort等

元组：不可修改的序列

与列表一样，元组也是序列，唯一的差别在于元组是不能修改的（字符串也不能修改）。元组语法很简单，只要将一些值用逗号分隔，就能自动创建一个元组。

```
>>> 1, 2, 3
(1, 2, 3)
```

元组并不太复杂，而且除创建和访问其元素外，可对元组执行的操作不多。元组的创建及其元素的访问方式与其他序列相同。

为何要熟悉元组？

- 用作映射中的键（以及集合的成员），而列表不行
- 有些内置函数和方法返回元组

一般而言，使用列表足以满足对序列的需求。

目录

本节要点

本节包括以下知识点：

- 字符串基本操作
- 设置字符串格式
- 字符串方法

字符串基本操作

所有标准序列操作（索引、切片、乘法、成员资格检查、长度、最小值和最大值）都适用于字符串，但别忘了字符串是不可变的，因此所有的元素赋值和切片赋值都是非法的。

```
>>> website = 'http://www.python.org'
>>> website[-3:] = 'com'
Traceback (most recent call last):
website[-3:] = 'com'
TypeError: object doesn't support slice assignment
```

设置字符串格式

将值转换为字符串并设置其格式是一个重要的操作，需要考虑众多不同的需求，因此随着时间的流逝，Python提供了多种字符串格式设置方法。

字符串设置运算符——百分号

运算符的行为类似于C语言中的经典函数printf：在%左边指定一个字符串（格式字符串），并在右边指定要设置其格式的值。

```
>>> format = "Hello, %s. %s enough for ya?"
>>> values = ('world', 'Hot')
>>> format % values
'Hello, world. Hot enough for ya?'
```

设置字符串格式

%s称为**转换说明符**，指出了要将值插入什么地方。s意味着将值视为字符串进行格式设置。如果指定的值不是字符串，将使用str将其转换为字符串。其他说明符将导致其他形式的转换。例如，%.3f将值的格式设置为包含3位小数的浮点数。

字符串设置运算符——模板字符串

它使用类似于UNIX shell的语法，旨在简化基本的格式设置机制。

```
>>> from string import Template
>>> tmpl = Template("Hello, $who!what enough for ya?")
>>> tmpl.substitute(who="Mars", what="Dusty")
'Hello, Mars! Dusty enough for ya?'
```

包含等号的参数称为关键字参数，相关知识将在后续介绍。

字符串方法

字符串有很多方法，有些很有用（如split和join），有些很少用到（如istitle 和capitalize）。字符串有很多方法都是从模块string那里“继承”而来的。

常用字符串方法

1. center：字符串居中；2. find：查找子字符串；3. join：合并序列元素；4. lower：返回字符串小写；5. replace：替换指定字符串；6：split：拆分字符串；7. strip：删除字符串开头及末尾空白；8. translate：单字符替换；9. is开头的一些方法：可用于判断字符串的特定性质。

目录

本节要点

本节包括以下知识点：

- 字典的用途
- 创建和使用字典
- 字典方法

字典的用途

字典的名称指出了这种数据结构的用途。普通图书适合按从头到尾的顺序阅读，如果你愿意，可快速翻到任何一页，这有点像Python中的列表。**字典**（日常生活中的字典和Python字典）旨在让你能够轻松地找到特定的单词（键），以获悉其定义（值）。

字典的一些用途

在很多情况下，使用字典都比使用列表更合适。

- 表示棋盘的状态，其中每个键都是由坐标组成的元组
- 存储文件修改时间，其中的键为文件名
- 数字电话/地址簿

创建和使用字典

字典以类似于下面的方式表示：
phonebook = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'} 字典由**键**及其相应的**值**组成，这种键-值对称为**项**（item）。在前面的示例中，键为名字，而值为电话号码。

函数dict

可使用函数dict从其他映射（如其他字典）或键-值对序列创建字典。

```
>>> items = [('name', 'Gumby'), ('age', 42)]
>>> d = dict(items)
>>> d
'age': 42, 'name': 'Gumby'
```

或者 d = dict(name='Gumby', age=42)。

创建和使用字典

字典的基本行为在很多方面都**类似**于序列：

- len(d)返回字典d包含的项（键-值对）数
- d[k]返回与键k相关联的值
- d[k] = v将值v关联到键k
- del d[k]删除键为k的项
- k in d检查字典d是否包含键为k的项

字典和列表的重要不同之处

- **键的类型**：字典中的键可以是整数，但并非必须是整数。
- **自动添加**：即便是字典中原本没有的键，也可以给它赋值，这将在字典中创建一个新项。
- **成员资格**：表达式k in d（d为字典）查找的是键而不是值，而表达式v in l（l为列表）查找的是值而不是索引。

字典方法

与其他内置类型一样，字典也有方法。字典的方法很有用，但其使用频率可能没有列表和字符串的方法那样高。

方法列举

1. clear：删除所有字典项；2. copy：复制字典；3. fromkeys：创建一个新字典，其中包含指定的键，且每个键对应的值都是None；4. get：访问字典项；5. items：返回一个包含所有字典项的列表；6. keys：返回一个字典视图，其中包含指定字典中的键。7. pop：获取与指定键相关联的值，并将该键-值对从字典中删除；8. popitem：而popitem随机地弹出一个字典项；9. setdefault；10； update：使用一个字典中的项来更新另一个字典；11. values：返回一个由字典中的值组成的字典视图。

目录

本节要点

本节包括以下知识点：

- 再谈print和import
- 赋值魔法
- 代码块
- 条件和条件语句
- 循环
- 简单推导
- pass、del和exec

再谈print和import

随着对Python的认识越来越深入，可能发现有些自以为很熟悉的方面隐藏着让人惊喜的特性。下面就来看看print和import隐藏的几个特性。

打印多个参数

print可用于打印多个表达式，条件是用逗号分隔它们：

```
>>> print('Age:', 42)
Age: 42
```

导入时重命名

从模块导入时，通常使用 import somemodule 或 from somemodule import somefunction但如果有两个模块，它们都包含函数open，可在语句末尾添加as子句并指定别名。

赋值魔法

即便是不起眼的赋值语句也蕴藏着一些使用窍门。

序列解包

序列解包即将一个序列解包，并将得到的值存储到一系列变量中。此操作可以用于给多个变量赋值、交换多个变量值等。

链式赋值

链式赋值是一种快捷方式，用于将多个变量关联到同一个值。
如：x = y = somefunction()

增强赋值

如代码x=x+1可写成x += 1。这称为增强赋值，适用于所有标准运算符，如*、/、%等。

代码块

什么是代码块

代码块是一组语句，可在满足条件时执行（if语句），可执行多次（循环），等等。代码块是通过缩进代码（即在前面加空格）来创建的。

在很多语言中，都使用一个特殊的单词或字符（如begin或{）来标识代码块的起始位置，并使用另一个特殊的单词或字符（如end或}）来标识结束位置。在Python中，使用冒号（:）指出接下来是一个代码块，并将该代码块中的每行代码都缩进相同的程度。发现缩进量与之前相同时，就表明当前代码块到此结束了。

条件和条件语句

到目前为止，在编写的程序中，语句都是逐条执行的。现在更进一步，让程序选择是否执行特定的语句块。

布尔值

计算机的逻辑判断，只有两种结果，就是True（英文意思是“真”）和False（英文意思是“假”）。这个计算真假的过程，叫做【布尔运算】。True和False，叫做【布尔值】。布尔值就可以帮助我们有条件的执行代码。

```
>>> True == 1
True
>>> False == 0
True
>>> True + False + 42
43
```

条件和条件语句

if语句

if语句，让你能够有条件地执行代码。这意味着如果条件（if和冒号之间的表达式）为前面定义的真，就执行后续代码块；如果条件为假，就不执行。

else子句

如果有需要，可使用else子句在if语句后增加一种选择（之所以叫子句是因为else不是独立的语句，而是if语句的一部分）。

```
name = input('What is your name?')
if name.endswith('Gumby'):
    print('Hello, Mr. Gumby')
else:
    print('Hello, stranger')
```

条件和条件语句

elif语句

要检查多个条件，可使用elif。elif是else if的缩写，由一个if子句和一个else子句组合而成，也就是包含条件的else子句。

```
num = int(input('Enter a number: '))
if num > 0:
    print('The number is positive')
elif num < 0:
    print('The number is negative')
else:
    print('The number is zero')
```

if语句还可以放在其他if语句块中，实现代码块的嵌套。

循环

如何重复操作多次呢？

while循环

while语句非常灵活，可用于在条件为真时反复执行代码块。

```
x = 1
while x <= 100:
    print(x)
```

for循环

若为序列（或其他可迭代对象）中每个元素执行代码块，可使用for循环。

```
numbers = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
for number in numbers:
    print(number)
```

简单推导

列表推导是一种从其他列表创建列表的方式，类似于数学中的集合推导。列表推导的工作原理非常简单，有点类似于for循环。

```
>>> [x * x for x in range(10)]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

这个列表由range(10)内每个值的平方组成。

如果只想打印那些能被3整除的平方值，该如何办呢？可使用求模运算符：如果y能被3整除，y % 3 将返回0（请注意，仅当x能被3 整除时，x*x才能被3整除）。为实现这种功能，可在列表推导中添加一条if语句。

```
>>> [x*x for x in range(10) if x % 3 == 0]
[0, 9, 36, 81]
```

pass、del和exec

什么都不做

有时候什么都不用做。这种情况不多，但一旦遇到，知道可使用pass语句大有裨益。那么为何需要一条什么都不做的语句呢？在你编写代码时，可将其用作占位符。例如，你可能编写了一条if语句并想尝试运行它，但其中缺少一个代码块，如下所示：

```
if name == 'Ralph Auldus Melish':
    print('Welcome!')
elif name == 'Enid':
    # 还未完成……
    pass
elif name == 'Bill Gates':
    print('Access Denied')
```

此代码无pass不能运行，因为在Python中代码块不能为空。

pass、del和exec

使用del删除

对于你不再使用的对象，Python通常会将其删除这被称为**垃圾收集**。另一种办法是使用del语句。这不仅会删除到对象的引用，还会删除名称本身。

```
>>> x = 1
>>> del x
>>> x
Traceback (most recent call last):    File "<pyshell#255>", line 1, in ?
  x
NameError: name 'x' is not defined
```

pass、del和exec

有时候，你可能想动态地编写Python代码，并将其作为语句进行执行或作为表达式进行计算。

exec

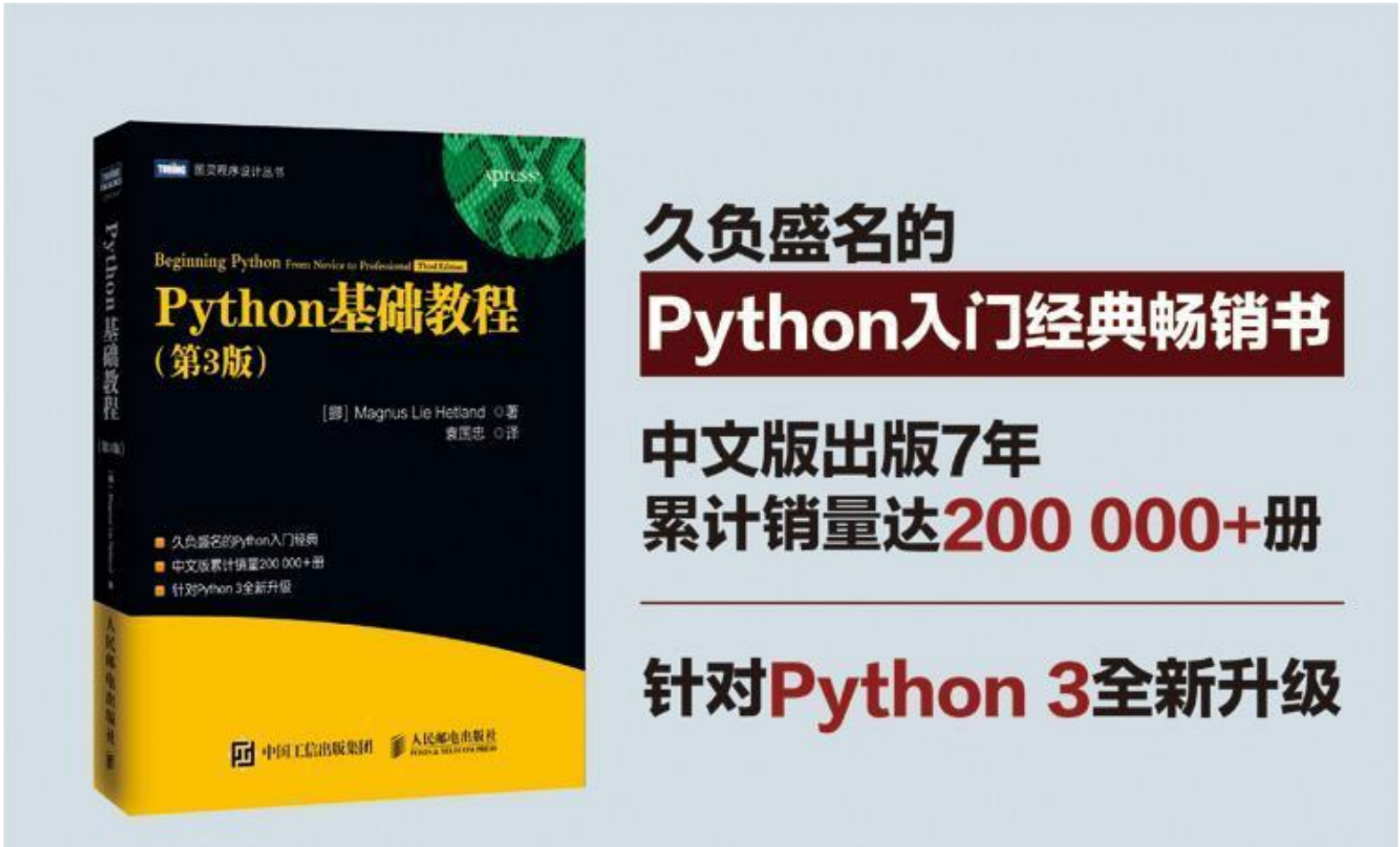
函数exec将字符串作为代码执行。

```
>>> exec("print('Hello, world!')")
Hello, world!
```

eval

eval是一个类似于exec的内置函数。exec执行一系列Python语句，而eval计算用字符串表示的Python表达式的值，并返回结果（exec什么都不返回，因为它本身是条语句）。

Python参考教材


Python基础教程
(第3版)
[挪] Magnus Lie Hetland 著
久负盛名的
Python入门经典畅销书
中文版出版7年
累计销量达200 000+册
针对Python 3全新升级

Python安装

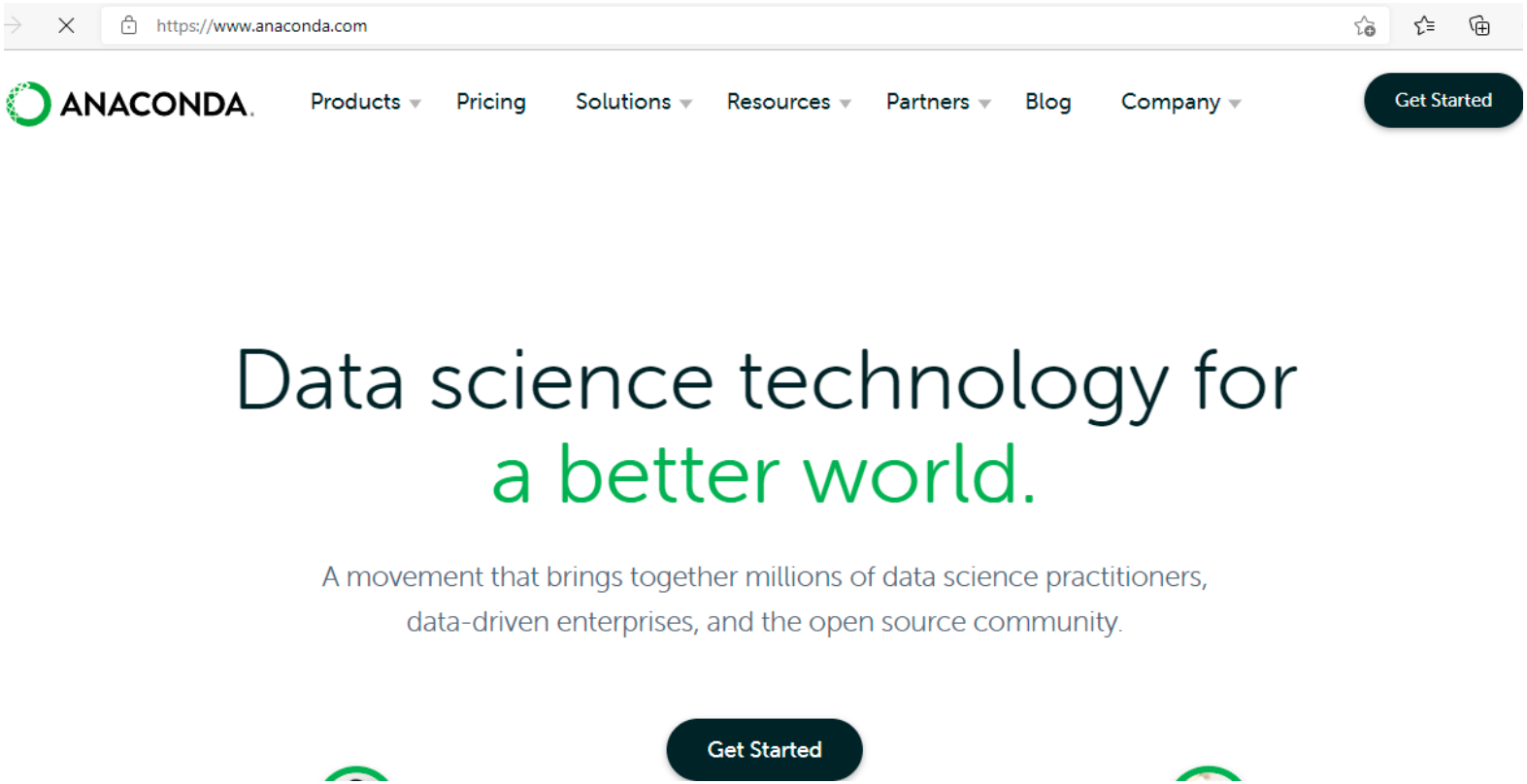


图: https://www.anaconda.com/

Python安装

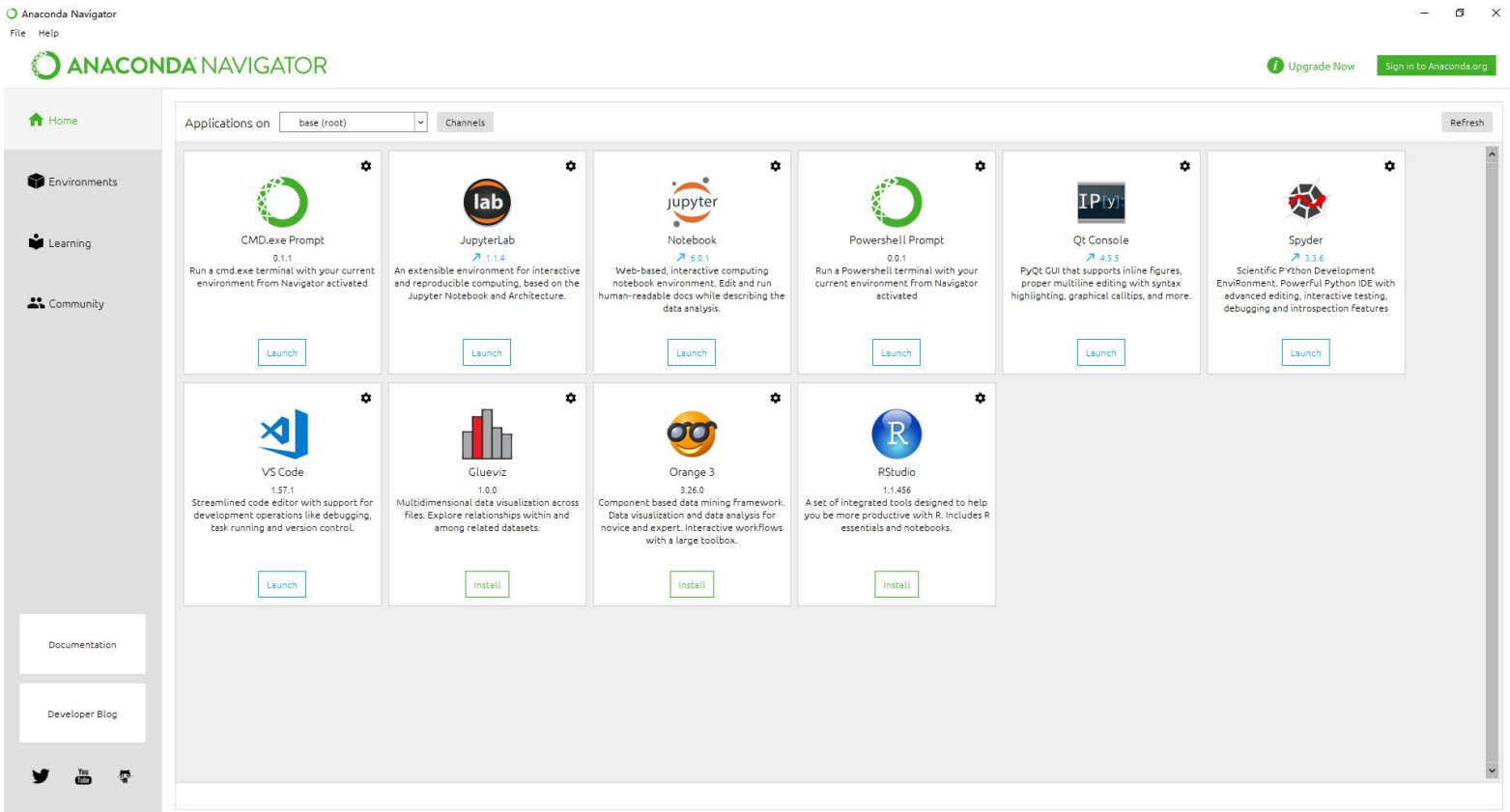


图: https://www.anaconda.com/

Python参考文档

图：https://docs.python.org/zh-cn/3/