# 模块

金融科技协会 2020 年 11 月 26 日

## 目录

# 1. 模块

## 1.1 模块的定义

- 简单地说，模块就是一个保存了 Python 代码的文件。模块能定义函数，类和变量。模块里也能包含可执行的代码。
- 模块让你能够有逻辑地组织你的 Python 代码段。

## 1.2 模块类型

- 自定义模块：我们只需要写一个 python 文件即可，也就是说写一个.py 为后缀的文件，
- 内置标准模块：Python 自带的标准库
- 开源模块（第三方）：这些库需要先进行安装

## 1.3 模块的使用

- import module1,module2
- from 模块名 import 函数名
- from 模块名 import 函数名 as 函数别名
- import 模块名 as 函数别名

In [7]:
```
import numpy as np  # import 语句导入整个模块内的所有成员（包括变量、函数、类等）
import random
import matplotlib.pyplot as plt
from pandas import *
from numpy import zeros
```

图 1：模块导入方法示例

图 1 展示了几种不同的模块导入方法，其中的核心就是 import 关键字，不建议使用 from 模块名 import * 这种方法。

In [2]:
```
# hello.py 文件
class test:                    # 定义一个类，进行测试
    def __init__(self):
        self.string = "hello aft!"
    def print_class(self):
        print(self.string)

def print_func():              #定义一个函数进行测试
    print("hello 2020!")

if __name__ == '__main__':
#代码只有在文件作为脚本直接执行才会被执行，而 import 到其他脚本中是不会被执行的。
    s = test()
    s.print_class()
    print_func()
```

```
hello aft!
hello 2020!
```

```
In [3]: # example.py 文件
        # import importlib
        # hello=importlib.reload(hello)

        import hello     #导入hello.py 模块

        s = hello.test()        #测试hello.py中定义的类
        s.print_class()

        hello.print_func()     #测试hello.py中定义的函数

        hello aft!
        hello 2020!
```

图 2：个人编写模块并导入使用测试示例

在图 2 中，编写了一个 hello.py 文件，其中定义了一个类和函数，我们把这个文件放在当前目录下，然后在 example.py 文件中简单调用了这个模块。

```
In [42]: import seaborn as sns

         x = np.random.normal(size=10000)  #生成10000个标准正态分布的数
         sns.distplot(x)

Out[42]: <matplotlib.axes._subplots.AxesSubplot at 0x288a9ce5b00>
```

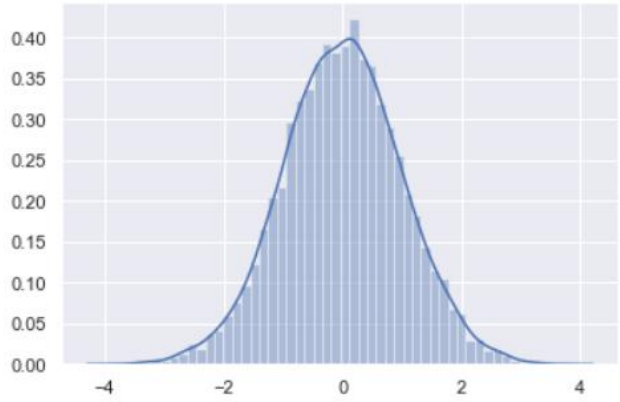


图 3：导入 seaborn 模块并进行调用示例

在图 3 中，调用 numpy 模块生成 1000 个数据，并使用 seaborn 库画出这些数据的直方图并拟合，由此可见模块功能非常强大。

## 1.4 一些关于模块的问题

• 模块导入多次为何仅仅导入一次没有区别？

模块并不是用来执行操作的，而是用来**定义变量、函数、类等**。因为定义只需要做一次，所以导入模块多次和导入一次的效果相同。

• 出于性能考虑，每个模块在每个解释器会话中只导入一遍。如果你修改了你的模块，需要导入修改后的模块：

（1）需要重启解释器；

（2）可以用 importlib.reload() 重新加载，例如:

```
import importlib
importlib.reload(模块名)
```

- 如何让模块可用？

（1） 将模块放到正确的位置

（2） 告诉解释器到哪里去查找

当你导入一个模块，Python 解释器对模块位置的搜索顺序是：

（1）当前目录：所以将自己写的模块直接放在当前目录，解释器就可以找到；

（2）如果不在当前目录，Python 则搜索在环境变量 PYTHONPATH 下的每个目录： 修改环境变量。

```
In [4]: #### 第一种方法:
        import sys
        sys.path
        # 所以我们只需要将我们写的模块放到这里表示的任何一个位置中,
        # 解释器就可以找到, 从而使自己写的模块可用
        sys.path.append('D:\学习\coding')

In [5]: sys.path

Out[5]: ['C:\\Users\\HP\\python\\AFT',
         'C:\\ProgramData\\Anaconda3\\python37.zip',
         'C:\\ProgramData\\Anaconda3\\DLLs',
         'C:\\ProgramData\\Anaconda3\\lib',
         'C:\\ProgramData\\Anaconda3',
         '',
         'C:\\Users\\HP\\AppData\\Roaming\\Python\\Python37\\site-packages',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\win32\\lib',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\Pythonwin',
         'C:\\ProgramData\\Anaconda3\\lib\\site-packages\\IPython\\extensions',
         'C:\\Users\\HP\\.ipython',
         'D:\\学习\\coding']
```

图 4：修改 sys.path 示例

第一种方法是将自己编写的模块的路径导入到 sys.path 中，然后就可以用了。

另外一种方法：修改环境变量 PYTHONPATH。环境变量中存放的值，就是一连串的路径。系统执行用户命令时，若用户未给出绝对路径，则首先在当前目录下寻找相应的可执行文件等。若找不到，再依次在环境变量保存的这些路径中寻找相应的可执行的程序文件。所以我们可以将模块所在的目录包含在环境变量 PYTHONPATH 中，自己编写的模块就可以使用了。（http://c.biancheng.net/view/4645.html 中详细说明了在不同操作系统下修改 PYTHONPATH 环境变量的方法）

# 2. 常用模块介绍

## 2.1 time 模块

```
In [26]: print(time.time())  #以自从1970年1月1日午夜（历元）经过了多长时间来表示。

1604226654.917666
```

```
In [27]: print(time.localtime(time.time()))  #将秒数转换为表示当地时间的日期元组

time.struct_time(tm_year=2020, tm_mon=11, tm_mday=1, tm_hour=18, tm_min=30, tm_s
ec=57, tm_wday=6, tm_yday=306, tm_isdst=0)
```

```
In [15]: print(time.mktime(time.localtime(time.time()))) #将时间元组转化为秒数

1604154771.0
```

```
In [28]: #将时间元组转换为字符串
         print(time.asctime( time.localtime(time.time())))

Sun Nov  1 18:33:10 2020
```

```
In [29]: #使用 time 模块的 strftime 方法来格式化日期，：
         # 格式化成2016-03-20 11:45:39形式
         print(time.strftime("%Y-%m-%d %H:%M:%S", time.localtime()))
         # 格式化成Sat Mar 28 22:24:24 2016形式
         print(time.strftime("%a %b %d %H:%M:%S %Y", time.localtime()))
         # 将字符串转换为时间元组
         a = "Sat Mar 28 22:24:24 2016"
         print(time.strptime(a,"%a %b %d %H:%M:%S %Y"))

2020-11-01 18:34:31
Sun Nov 01 18:34:31 2020
time.struct_time(tm_year=2016, tm_mon=3, tm_mday=28, tm_hour=22, tm_min=24, tm_s
ec=24, tm_wday=5, tm_yday=88, tm_isdst=-1)
```

```
In [2]: print("Start : %s" % time.time())
        time.sleep( 5 )
        print("End : %s" % time.time())

Start : 1604492804.7039995
End : 1604492809.704309
```

图 5：time 模块常用函数示例

在 time 模块中，time()函数表示从 1970 年 1 月 1 日午夜到现在经历了多少秒，localtime()函数将秒数转化为当地时间的元组形式，mktime()将时间元组形式转化为秒数，asctime()函数将时间元组转化为字符串形式，strftime()按照我们的需要来格式化日期，其中%y 代表两位数的年份表示，%m 表示月份，%d 月内中的一天，%l12 小时制小时数等等，sleep()表示将函数阻塞多少时间。

## 2.2 random 模块

```
In [16]: random.random()    #用于生成一个0到1的随机浮点数：0 <= n < 1.0
Out[16]: 0.16123910839715017

In [17]: random.uniform(1,2)
         #用于生成一个指定范围内的随机符点数，两个参数其中一个是上限，一个是下限。
Out[17]: 1.6913146677506734

In [18]: random.randint(4,9)  #用于生成一个指定范围内的整数
Out[18]: 6

In [30]: x=[1,2,5,7,9]
         random.shuffle(x)    #用于将一个列表中的元素打乱。
         print(x)
         [2, 1, 7, 9, 5]

In [7]: print(random.sample([1,3,4,6,7,9],2))
        #sample(seq, n) 从序列seq中选择n个随机且独立的元素；\
        print(random.sample('fsdas',2))
        [4, 9]
        ['a', 'd']

In [6]: random.randrange(10, 100, 2)
        #从指定范围内，按指定基数递增的集合中 获取一个随机数。
        #如：random.randrange(10, 100, 2)，结果相当于从[10, 12, 14, 16, ... 96, 98]序列
        #中获取一个随机数
Out[6]: 42

In [37]: print(random.choice("ffdasa"))
         print(random.choice(("fas",1,3,7,"fasd")))
         print(random.choice([1,34,6435,645]))
         #从序列中获取一个随机元素。参数sequence表示一个有序类型。这里要说明 一下：
         #sequence在python不是一种特定的类型，而是泛指一系列的类型。
         #list, tuple, 字符串都属于sequence。
         f
         fasd
         645
```

图 6：random 模块常用函数示例

在 random 模块中，random()表示随机生成 0 与 1 之间的小数，uniform()生成指定范围内的小数，randint()表示生成指定范围内的整数，shuffle()表示随机打乱列表中的元素顺序，sample()表示从序列中随机选择 n 个元素，randrange()表示从生成的序列中随机获取一个数，choice()表示从序列中随机选择一个元素。