基本语法文档(七)

Author: Limzh

Section1. 序言

考虑以下代码的功能

```
import numpy as np
def main():
    array = np.zeros(100)
    print(array)
main()
```

惹人关注的是第一行

```
import numpy as np
```

这行代码实际做的事情是从库文件中导入模块，并在之后的文档中像调用对象方法一样使用其中的函数。

Section2.import库文件的重要意义

试想我们在开发一个巨大无比的项目，有数以千万计的代码需要我们去写。我们能把这些代码都写到同一个文件中嘛？如果这样做，会导致大量的问题，例如：

1. 作用域和函数命名冲突
2. 运行时间慢
3. 堆栈崩溃
4. 难以优化

一种有效的替代方式是，将整个项目分成若干个子文件，在每一个子文件中实现一部分项目的功能，并使用某种规则让子文件之间相互“沟通”。就像机器的组件一样，先实现每一块组件的细节，再考虑将组件与组件之间相互连接。这种连接方式，就是`import`的语句发挥的作用。那么被引入的文件我们称之为导入文件或者库文件，库文件实现的函数称之为库函数。这个名称是一种形象的比喻，就像我们要写一篇论文，最优先的想法肯定是从文库中导入一些参考资料。

这里，我们得到了import库文件的第一个重要意义： 为自己写的大工程文件代码结构进行合理的划分

import语句提供的第二个便利是，我们可以通过这种方式自由地将别人已经实现的函数或者代码导入到自己的文件中来。试想，如果我们要实现一个项目，这个项目需要有大量数学函数，与概率分布。难道我们要将这些底层的函数都自己实现一边嘛？我稍微想想就不由得打起了退堂鼓。但假如有一个官方的数学函数库，能为我们提供所有的底层实现，使我们可以专心于主体逻辑框架，岂不美哉？

因此，我们得到了第二个重要意义： import语句调用库文件能够将别人已经实现好的函数放到自己的文件中，起到优化代码，优化底层的作用。

Section3. import 语句的基本语法和一些高级特性

“库文件和库函数”这种思想所带来的便利实际上就是“模块化”思想带来的便利。那么问题就来到了具体语法上。假设我们现在有一个文件叫做`mathlibrary_limzh.py`用以提供一些重要数学函数，有一个主文件`calculator.py`来实现一个复杂的计算器。那么，我们可以通过以下语法将`mathlib.py`导入主文件。

```
import mathlibrary_limzh
```

这实在是太简单了！在主文件的某一行使用`import xxx`语句，那么该语句之后的部分就都会收到语句的影响。一般地，我们把import语句放到文件开头。

成功导入库文件之后，如何使用这个自定义的数学库文件的库函数呢？请考虑下面一段代码的效用。

```
import mathlibrary_limzh
x, y = 10, 20
z = mathlibrary_limzh.sum1(x, y) # 假如该库文件有sum这个函数
print(z) # 30
```

第二行，我们像调用对象的方法一样调用了该库文件中的库函数。实际上这就是使用库函数的方法了，对象调用方法之于库文件调用库函数。两者有极大的相似之处。

import语句的基本语法到这里就介绍完了，下面开始介绍一些高级特性。

3.1 import语句高级特性： 更换库文件名

以上面的代码段为例子，有一个让人相当困扰的难题：每次调用库函数的时候，都要打一遍该库文件的名字，但是`mathlibrary_limzh`这个名字特别特别繁琐！我们能不能换一下名字呢？答案是可以。考虑下面代码，

```
import mathlibrary_limzh as mathlib
x, y = 10, 20
z = mathlib.sum1(x, y)
print(z)
```

3.2 import语句高级特性：导入库文件的某一个函数

有的时候，我们只需要用到库文件的某一个函数或者某一个类（忘记提及，在库文件中实现的所有东西，函数、类甚至是变量都可以被导入）而不希望导入并使用其他的函数，那么我们该怎么办呢？考虑以下代码的效用：

```
from mathlibrary_limzh import sum1
x, y = 10, 20
z = sum1(x, y)
```

3.3 import语句高级特性：导入库文件中的所有内容

有时，我们甚至希望把整个库文件的内容全盘导入，调用函数或者类的时候直接调用函数名即可。这个时候使用以下语法，

```
from mathlibrary_limzh import *
x, y = 10, 20
z = sum1(x, y)
v = minus(y, x) # mathlib里假如还有minus这个函数
```

Section4. 面向对象、作用域和库文件

介绍完如何导入、使用库函数，接下来让我们抓住python语言特性的内核：面向对象。
也尝试用作用域的概念去理解在导入库文件的时候究竟发生了什么。

要理解导入库文件时候发生了什么，首先我们要问一个问题：为什么使用库函数的语法和使用对象中的函数的语法如此相像？

一个简洁的回答是，因为每一个文件、模块都可以被视作是一个对象，但本质上是作用域的影响。

正如每一个函数有着自己的作用域一样，每一个文件也有着自己的作用域。当一个文件导入另外一个文件时，实际上做的事情时把另外一个文件的作用域（连同着该作用域里面定义的内容）一起导入当前文件的作用域。打一个比方，我买了两个糖果袋，现在我要把这两袋糖果汇合成一袋糖果，那我必然要将一袋糖果放到另一袋糖果中去。这样的导入有两种结果，第一种是保留着该作用域的边界，即：连同袋子一起放入主袋糖果里。这样附袋内部的糖果就和主袋内部的其他糖果隔离开，每次我要吃附袋糖果都要先打开附袋包装。

这也就是下段代码的使用方式

```
import mathlibrary_limzh as mathlib
mathlib.sum1()
```

另一种导入结果是，我直接将所有糖果倾倒到主袋中，把附袋扔掉。

这也就是

```
from mathlibrary_lib import *
sum1()
```

当然，我也可以只从附袋中取一颗入主袋

```
from mathibrary_lib import sum1
```

这就是一种作用域视角上的理解。而面向对象，仅仅是逻辑层面上的认识：每一个模块，也是一个对象，可以调用函数，可以调用变量。

练习

1. 请打开`day7/calculator.py`,导入`mathlib.py`库（也已放到`day7`），并使用库函数实现一个复杂版的计算器。
2. 请阅读相关资料，理解接下来我们要学习的`numpy`，`matlibplot`和`pandas`等库其实从语法层面上仅仅是库而已。