



山东科技大学——测绘与空间信息学院

Python程序设计

地理信息科学系 刘洪强

J6-557 电话：86081170

2021年3月5日星期五

课程安排:

36个学时，其中授课24个学时，实验12个学时

章节内容

第1章 认识Python

第2章 Python编程基础

第3章 函数、类、包和模块

第4章 文件操作

第5章 地图文档管理与数据访问

第6章 空间数据定位与查询

第7章 空间数据分析

第8章 地图制图与输出

第3章 函数、类、 包和模块

主要内容

函数
类
包
模块

第3章 函数、类、包和模块

3.1 函数

函数是组织好的，可**重复使用的**，用来实现单一，或相关联功能的**代码段**。

函数能提高应用的模块性，和代码的重复利用率。Python提供了许多内建函数，比如`print()`,`int()`。但你也可以自己创建函数，这被叫做用户自定义函数。

第3章 函数、类、包和模块

3.1 函数

- 将可能需要反复执行的代码封装为函数，并在需要该功能的地方进行调用，不仅可以实现**代码复用**，更重要的是可以**保证代码的一致性**，只需要修改该函数代码则所有调用均受到影响。
- 设计函数时，应注意**提高模块的内聚性**，同时**降低模块之间的隐式耦合**。
- 在实际项目开发中，往往会把一些通用的函数封装到一个模块中，并把这个通用模块文件放到顶层文件夹中，这样更方便管理。

第3章 函数、类、包和模块

3.1 函数

- 在编写函数时，应尽量减少副作用，尽量**不要修改参数本身**，不要修改除返回值以外的其他内容。
- 不要在一个函数中执行太多的功能，尽量只让一个函数完成一个高度相关且大小合适的任务，一个函数的代码尽量能在**一个屏幕**内完整显示。
- 尽量减少不同函数之间的隐式耦合，减少全局变量的使用，使得函数之间仅通过**调用**和**参数传递**来显式体现其相互关系

第3章 函数、类、包和模块

3.1 函数

❖函数定义语法:

```
def 函数名([参数列表]):  
    ''' 注释 '''  
    函数体
```

❖注意事项:

- ✓函数形参不需要声明类型，也不需要指定函数返回值类型
- ✓即使该函数不需要接收任何参数，也必须保留一对空的圆括号
- ✓括号后面的冒号必不可少
- ✓函数体相对于def关键字必须保持一定的空格缩进

第3章 函数、类、包和模块

3.1 函数

(1) 定义函数

简单的规则：

- (1) 函数代码块以**def**关键词开头，后接函数**标识符名称和圆括号()**。
- (2) 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于**定义参数**。
- (3) 函数的第一行语句可以选择性地使用文档字符串——用于存放函数说明。
- (4) 函数内容以**冒号起始**，并且**缩进**。
- (5) **return** [表达式] 结束函数，选择性地返回一个值给调用方。不带表达式的**return**相当于返回 **None**。

第3章 函数、类、包和模块

3.1 函数

(1) 定义函数

`def` 函数名（参数列表）：

函数体

或者更直观的表示为：

`def <name>(arg1, arg2,... argN):`

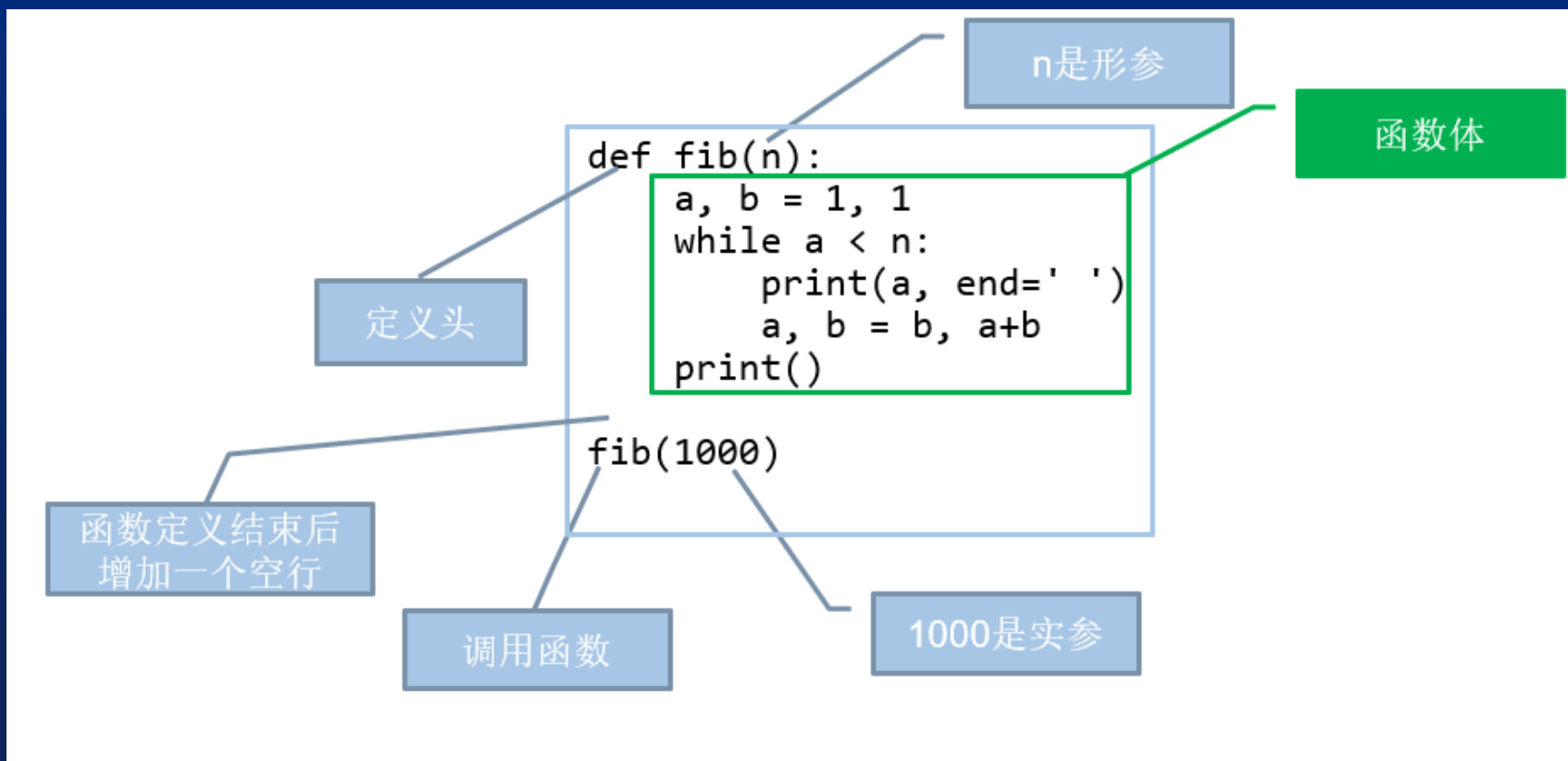
`<statements>`

函数的名字也必须以**字母开头**，可以包括下划线“”，同定义变量一样，不能把Python的关键字定义成函数的名字。函数内的语句数量是任意的，每个语句至少有一个空格的缩进，以表示此语句属于这个函数的。函数体必须保持一致的缩进，因为在函数中，**缩进结束的地方，表示函数结束。**

第3章 函数、类、包和模块

3.1 函数

- 生成斐波那契数列中小于n的所有数值的函数定义：



第3章 函数、类、包和模块

3.1 函数

- 在定义函数时，开头部分的注释并不是必需的，但如果为函数的定义加上注释的话，可以为用户提供友好的提示。

```
>>> def fib(n):  
    '''accept an integer n.  
       return the numbers less than n in Fibonacci sequence.'''  
    a, b = 1, 1  
    while a < n:  
        print(a, end=' ')  
        a, b = b, a+b  
    print()
```

```
>>> fib(  
    (n)  
    accept an integer n.  
    return the numbers less than n in Fibonacci sequence.
```

第3章 函数、类、包和模块

3.1 函数

(1) 定义函数

几点注意：

- 1) 如果没有`return`语句，函数执行完毕后也会返回结果，只是结果为`None`。
- 2) `return None`可以简写为`return`。
- 3) 在Python中定义函数时，需要保持函数体中同一层级代码的缩进一致。

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

定义带参的函数，参数称为**形参**；调用时一般也需要传入参数，称为**实参**。

定义和调用函数时可以使用以下参数类型：

- 1) **必须参数**。
- 2) **关键字参数**。
- 3) **默认参数**。
- 4) **可变参数**。
- 5) **组合参数**。

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

1) 必须参数

必须参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

```
def paramone(str):  
    print('the param is:',str)  
    print('我是一个传入参数，我的值是：',str)  
paramone('hello,world')
```

定义了必须传入一个参数的函数paramone(str)，其传入参数为str，结果即是将'hello,world'这个值传给str。

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

2) 关键字参数

函数调用使用关键字参数来确定传入的参数值。

使用关键字参数Python 解释器能够用参数名匹配参数值。

```
def personinfo(age,name):
```

```
    print('年龄: ',age)
```

```
    print('名称: ',name)
```

```
    return
```

```
print('-----按参数顺序传入参数-----')
```

```
personinfo(21,'小萌')
```

```
print('-----不按参数顺序传入参数，指定参数名-----')
```

```
personinfo(name='小萌',age=21)
```

```
print('-----按参数顺序传入参数，并指定参数名-----')
```

```
personinfo(age=21,name='小萌')
```

-----按参数顺序传入参数-----

年龄: 21

名称: 小萌

-----不按参数顺序传入参数，指定参数名-----

年龄: 21

名称: 小萌

-----按参数顺序传入参数，并指定参数名-----

年龄: 21

名称: 小萌

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

2) 关键字参数

通过关键参数，实参顺序可以和形参顺序不一致，但不影响传递结果，避免了用户需要牢记位置参数顺序的麻烦。

```
>>> def demo(a, b, c=5):  
        print(a, b, c)  
  
>>> demo(3, 7)  
3 7 5  
>>> demo(a=7, b=3, c=6)  
7 3 6  
>>> demo(c=8, a=9, b=0)  
9 0 8
```

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

3) 默认参数。

调用函数时，如果没有传递参数，则会使用默认参数。所谓使用默认参数，就是我们在定义函数时，给参数一个**默认值**，当没有给调用该函数时的该参数赋值时，调用的函数就使用这个默认的值。

```
def defaultparam(name,age=23):
```

```
    print('hi, 我叫: ',name)
```

```
    print('我今年: ',age)
```

```
    return
```

```
defaultparam('小萌')
```

```
hi, 我叫: 小萌  
我今年: 23
```

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

3) 默认参数。

调用带有默认值参数的函数时，**可以不对默认值参数进行赋值，也可以为其赋值**，具有很大的灵活性。

```
>>> def say(message, times=1):  
    print(message*times)  
  
>>> say('hello')  
hello  
>>> say('hello', 3)  
hello hello hello  
>>> say('hi', 7)  
hi hi hi hi hi hi hi
```

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

3) 默认参数。

注意：默认值参数必须出现在函数参数列表的最右端，**任何一个默认值参数右边不能有非默认值参数。**

```
>>> def func(a=3, b, c=5):    # 失败，带默认值的参数后面有不带默认值的参数
    print(a, b, c)
```

SyntaxError: non-default argument follows default argument

```
>>> def func(a=3, b):          # 失败，带默认值的参数后面有不带默认值的参数
    print(a, b)
```

SyntaxError: non-default argument follows default argument

```
>>> def func(a, b, c=5):      # 成功
    print(a, b, c)
```

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

4) 可变参数

如果需要一个函数能处理比当初声明时更多的参数，这些参数叫做可变参数，和前面所述两种参数不同，可变参数声明时不会命名。

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

4) 可变参数

可变长度参数主要有两种形式：在参数名前加1个星号*或2个星号**

◆ *parameter用来接收多个位置实参并将其放在元组中。

◆ **parameter用来接收多个关键参数并存放到字典中。

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

4) 可变参数

◆ *parameter的用法。

```
>>> def demo(*p):  
        print(p)
```

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

```
>>> demo(1, 2)
```

```
>>> demo(1, 2, 3, 4, 5, 6, 7)
```

(1, 2, 3)

(1, 2)

(1, 2, 3, 4, 5, 6, 7)

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

4) 可变参数

```
def personinfo(arg, *vartuple):  
    print(arg)  
    for var in vartuple:  
        print('我属于不定长参数部分:', var)  
    return  
print('-----不带可变参数-----')  
personinfo('小萌')  
print('-----带上两个可变参数-----')  
personinfo('小萌', 21, 'beijing')  
print('-----带上五个可变参数-----')  
personinfo('小萌', 21, 'beijing', 123, 'shanghai', 'happy')
```

```
-----不带可变参数-----  
小萌  
-----带上两个可变参数-----  
小萌  
我属于不定长参数部分: 21  
我属于不定长参数部分: beijing  
-----带上五个可变参数-----  
小萌  
我属于不定长参数部分: 21  
我属于不定长参数部分: beijing  
我属于不定长参数部分: 123  
我属于不定长参数部分: shanghai  
我属于不定长参数部分: happy
```

第3章 函数、类、包和模块

3.1 函数

(2) 函数的参数与调用

4) 可变参数

◆ **parameter的用法。

```
>>> def demo(**p):  
    for item in p.items():  
        print(item)  
  
>>> demo(x=1, y=2, z=3)
```

```
('x', 1)  
('y', 2)  
('z', 3)
```

第3章 函数、类、包和模块

3.1 函数

5) 组合参数

几种不同类型的参数可以混合使用，但是不建议这样做。

```
def func(a, b, c=4, *aa, **bb):  
    print(a, b, c)  
    print(aa)  
    print(bb)
```

```
>>> func(1, 2, 3, 4, 5, 6, 7, 8, 9, xx='1', yy='2', zz=3)
```

```
1 2 3  
(4, 5, 6, 7, 8, 9)  
{ 'xx': '1', 'yy': '2', 'zz': 3}  
>>> func(1, 2, 3, 4, 5, 6, 7, xx='1', yy='2', zz=3)  
1 2 3  
(4, 5, 6, 7)  
{ 'xx': '1', 'yy': '2', 'zz': 3}
```

第3章 函数、类、包和模块

3.1 函数

传递参数时，可以通过在**实参序列前加一个星号**将其**解包**，然后传递给多个单变量形参。

```
def demo(a, b, c):  
    print(a+b+c)  
  
seq = [1, 2, 3]  
demo(*seq)  
print('-----')  
tup = (1, 2, 3)  
demo(*tup)
```

6

6

第3章 函数、类、包和模块

3.1 函数

如果函数实参是字典，可以在前面加两个星号进行解包，等价于关键参数。

```
def demo(a, b, c):  
    print(a+b+c)  
  
dic = {'a':1, 'b':2, 'c':3}  
demo(**dic)  
print('-----')  
demo(a=1, b=2, c=3)  
print('-----')  
demo(*dic.values())
```

6

6

6

第3章 函数、类、包和模块

3.1 函数

(3) 递归函数

一个函数在内部调用自身，这个函数就称作递归函数。
递归的简单定义如下：

```
def recursion():
```

```
    return recursion()
```

递归函数应该满足如下条件：

- (1) 大的问题可以分解为相同或相似的过程完成。
- (2) 至少有一个能返回具体的实例（即有出口）。

第3章 函数、类、包和模块

3.1 函数

(3) 递归函数

计算阶乘 $n! = 1 \times 2 \times 3 \times \dots \times n$ ，用函数 $\text{fact}(n)$ 表示，可以看出：

$$\text{fact}(n) = n! = 1 \times 2 \times 3 \times \dots \times (n-1) \times n = (n-1)! \times n = \text{fact}(n-1) \times n$$

$\text{fact}(n)$ 可以表示为 $n \times \text{fact}(n-1)$ ，只有 $n=1$ 时需要特殊处理。

$\text{fact}(n)$ 用递归的方式的函数定义如下：

```
def fact(n):
```

```
    if n==1:
```

```
        return 1
```

```
    return n * fact(n - 1)
```

1	fact (4)
24	

第3章 函数、类、包和模块

3.1 函数

- `return`语句用来从一个函数中返回一个值，同时结束函数。
- 对于以下情况，Python将认为该函数以`return None`结束，返回空值：
 - 函数没有`return`语句；
 - 函数有`return`语句但是没有执行到；
 - 函数有`return`也执行到了，但是没有返回任何值。

第3章 函数、类、包和模块

3.1 函数

(4) 匿名函数

Python使用**lambda**来创建匿名函数。

lambda只是一个**表达式**，函数体比def简单很多。

lambda的主体是一个表达式，而不是一个代码块。

lambda 函数只包含一个语句：

```
lambda [arg1 [,arg2,.....argn]]:expression
```

第3章 函数、类、包和模块

3.1 函数

(4) 匿名函数

- `lambda`表达式可以用来声明匿名函数（也可以定义具名函数），也就是没有函数名字的临时使用的小函数，尤其适合需要一个函数作为另一个函数参数的场合。
- `lambda`表达式只**可以包含一个表达式**，该表达式可以任意复杂，其计算结果可以看作是函数的返回值。

第3章 函数、类、包和模块

3.1 函数

(4) 匿名函数

使用def语句求两个数之和示例：

```
def func(x, y):  
    return x + y
```

```
func(3, 4)
```

使用lambda表达式求两个数之和示例：

```
f= lambda x, y: x + y  
f(3,4)
```

第3章 函数、类、包和模块

3.1 函数

(4) 匿名函数

```
>>> f = lambda x, y, z: x+y+z      #可以给lambda表达式起名字
>>> f(1,2,3)                      #像函数一样调用
6
>>> g = lambda x, y=2, z=3: x+y+z  #参数默认值
>>> g(1)
6
>>> g(2, z=4, y=5)                #关键参数
11
```

第3章 函数、类、包和模块

3.2 类

(1) 类定义

```
class <类名>:
```

```
    <方法定义>
```

方法定义同函数定义，如下：

```
    def <方法名>(<self,其他参数>):
```

```
    ...
```

方法是依附于类的函数,普通函数则是独立的.

方法的**第一个参数是专用的**,习惯用名字**self**.

只能**通过向对象发消息来调用方法**.

第3章 函数、类、包和模块

- 面向对象程序设计（Object Oriented Programming, OOP）主要针对大型软件设计而提出，使得软件设计更加灵活，能够很好地支持**代码复用和设计复用**，并且使得代码具有更好的可读性和可扩展性。
- 面向对象程序设计的一条基本原则是**计算机程序由多个能够起到子程序作用的单元或对象组合而成**，大大地降低了软件开发的难度。
- 面向对象程序设计的一个关键性观念是将数据以及对数据的操作封装在一起，组成一个相互依存、不可分割的整体，即**对象**。对于相同类型的对象进行分类、抽象后，得出共同的特征而形成了**类**，面向对象程序设计的关键就是如何**合理地定义和组织这些类以及类之间的关系**。

第3章 函数、类、包和模块

3.2 类

(1) 类定义

- Python使用**class关键字**来定义类，class关键字之后是一个空格，然后是类的名字，再然后是一个冒号，最后换行并定义类的内部实现。
- 类名的**首字母一般要大写**，当然也可以按照自己的习惯定义类名，但一般推荐参考惯例来命名，并在整个系统的设计和实现中保持风格一致，这一点对于团队合作尤其重要。

```
class Car:  
    def infor(self):  
        print(" This is a car ")
```

第3章 函数、类、包和模块

3.2 类

(1) 类定义

■定义了类之后，可以用来实例化对象，并通过“对象名.成员”的方式来访问其中的数据成员或成员方法。

```
car = Car()  
car.infor() # This is a car
```

■在Python中，可以使用内置方法isinstance()来测试一个对象是否为某个类的实例。

```
isinstance(car, Car)           # True  
isinstance(car, str)           # False
```


第3章 函数、类、包和模块

3.2 类

(2) 实例变量与局部变量

对象的数据以实例变量形式定义。

实例变量：

self.<变量名>

局部变量是在方法中定义

变量名 = <变量值>

每个类实例(对象)具有自己的实例变量副本,用来存储该对象自己的数据,实例变量的访问:

<对象>.<实例变量>

实例变量与函数局部变量不同:

同一个类的各个方法都可以访问实例变量;

类的方法中也可以定义局部变量,不能被其他方法访问。

第3章 函数、类、包和模块

3.2 类

- 类的所有实例方法都**必须至少**有一个名为**self**的参数，并且必须是方法的**第一个**形参（如果有多个形参的话），**self**参数代表将来要创建的对象本身。
- 在类的实例方法中访问实例属性时需要以**self**为前缀。
- 在外部通过**对象**调用对象方法时并**不需要**传递这个参数，如果在外部通过**类**调用对象方法则**需要**显式为**self**参数传值。

第3章 函数、类、包和模块

3.2 类

```
class Person:
    def __init__(self, n, y):
        self.name = n
        self.year = y
    def whatName(self):
        print("My name is", self.name)
    def howOld(self, y):
        age = y - self.year
        if age > 0:
            print("My age in", y, "is", age)
        else:
            print("I was born in", self.year )
```

第3章 函数、类、包和模块

3.2 类

- 在Python中，在类中定义实例方法时将第一个参数定义为“self”只是一个习惯，而实际上不必须使用“self”这个名字，尽管如此，建议编写代码时仍以self作为方法的第一个参数名字。

```
>>> class A:
    def __init__(hahaha, v):
        hahaha.value = v
    def show(hahaha):
        print(hahaha.value)
```

```
>>> a = A(3)
>>> a.show()
```

3

第3章 函数、类、包和模块

3.2 类

(3) 实例创建

类与实例:抽象与具体.

"人"是类,"张三"是人的实例

一个类可以创建任意多个实例

各实例具有相同的行为:由方法决定

但具有不同的数据:由实例变量决定

实例创建

<变量> = <类名>(<实参>)

这里<类名>相当于一个函数,称为构造器,用来构造实例.

第3章 函数、类、包和模块

3.2 类

(3) 实例创建

创建时对实例进行初始化

用构造器创建实例时,系统会自动调用`__init__`方法
通常在此方法中执行一些初始化操作

`__init__`所需的参数由构造器提供。

例如:

```
p = Person("Tom",2000)
```

```
class Person:
    def __init__(self, n, y):
        self.name = n
        self.year = y
    def whatName(self):
        print("My name is", self.name)
    def howOld(self, y):
        age = y - self.year
        if age > 0:
            print("My age in", y, "is", age)
        else:
            print("I was born in", self.year )
```

第3章 函数、类、包和模块

3.2 类

(4) 方法调用

似函数调用,但需指明实例(对象).

<实例>.<方法名>(<实参>)

<实例>就是与形参self对应的实参.

例如:

```
1 p.whatName()
```

```
My name is Tom
```

```
1 p.howOld(2021)
```

```
My age in 2021 is 21
```

第3章 函数、类、包和模块

■Python提供了一个关键字“pass”，表示空语句，可以用在类和函数的定义中或者选择结构中。当暂时没有确定如何实现功能，或者为以后的软件升级预留空间，或者其他类型功能时，可以使用该关键字来“占位”。

```
class A:  
    pass
```

```
def demo():  
    pass
```

```
if 5>3:  
    pass
```


第3章 函数、类、包和模块

3.3 包

包（package）是一个有层次的文件目录结构，它定义了由n个模块或n个子包组成的python应用程序执行环境。通俗一点：包是一个包含__init__.py文件的目录，该目录下一定得有这个__init__.py文件和其它模块或子包。

通常包总是一个目录，可以使用import导入包，或者from + import来导入包中的部分模块。包目录下为首的一个文件便是__init__.py。然后是一些模块文件和子目录，假如子目录中也有__init__.py那么它就是这个包的子包了。

第3章 函数、类、包和模块

3.3 包

为了组织好模块，将多个模块分为一个包。包是python模块文件所在的目录，且该目录下必须存在 **`__init__.py`** 文件。常见的包结构如下：

package_a

└— `__init__.py`

└— `module_a1.py`

└— `module_a2.py`

package_b

└— `__init__.py`

└— `module_b1.py`

└— `module_b2.py`

第3章 函数、类、包和模块

3.3 包

如果main.py想要引用package_a中的模块module_a1，可以使用：

```
from package_a import module_a1  
import package_a.module_a1
```

第3章 函数、类、包和模块

3.3 包

Python是开源的程序，可以方便地引入第三方的包，安装包的方法如下：

包的安装建议首先使用conda包管理工具：
conda install 包名

```
管理员: 命令提示符
Microsoft Windows [版本 10.0.18363.1379]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Windows\system32>conda install numpy
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: D:\Softwares\Anaconda3

added / updated specs:
- numpy

The following packages will be downloaded:



| package                   | build          | size   | url                                                                                                                                           |
|---------------------------|----------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| ca-certificates-2020.12.5 | h5b45459_0     | 173 KB | <a href="https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge">https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge</a> |
| certifi-2020.12.5         | py37h03978a9_1 | 143 KB | <a href="https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge">https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge</a> |
| conda-4.9.2               | py37h03978a9_0 | 3.0 MB | <a href="https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge">https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge</a> |
| numpy-1.20.1              | py37hd20adf4_0 | 5.2 MB | <a href="https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge">https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge</a> |
| openssl-1.1.1j            | h8ffe710_0     | 5.8 MB | <a href="https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge">https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/conda-forge</a> |

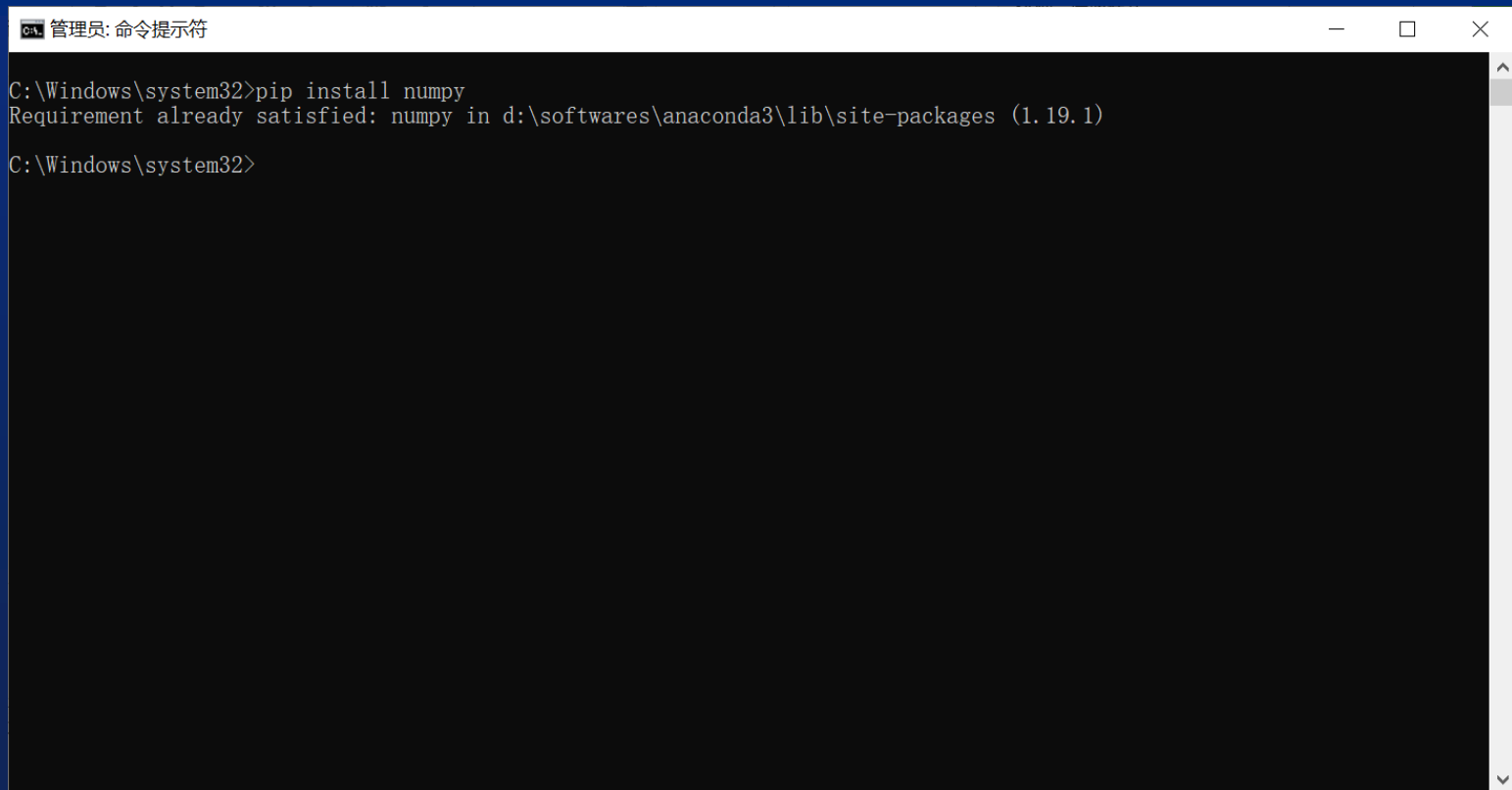

```

第3章 函数、类、包和模块

3.3 包

Python是开源的程序，可以方便地引入第三方的包，安装包的方法如下：

如果conda
未收录，再
使用pip命
令安装：
pip install
包名



```
管理员: 命令提示符
C:\Windows\system32>pip install numpy
Requirement already satisfied: numpy in d:\softwares\anaconda3\lib\site-packages (1.19.1)
C:\Windows\system32>
```

第3章 函数、类、包和模块

3.4 模块

模块(Module)是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

应用模块可条理地组织 Python 代码段，把相关的代码分配到一个模块里能让你的代码更好用，更易懂。

在模块中能定义函数，类和变量，模块里也能包含可执行的代码。

使用 import 语句来引入模块，语法如下：

```
import module1 , module2,... moduleN
```

也可以从模块中导入一个指定的部分到当前命名空间中，语法如下：

```
from modname import name1, name2, ... nameN
```

```
from modname import *
```

第3章 函数、类、包和模块

3.4 模块

- 调用模块语句:

```
import module1 , module2,... Module
```

```
import numpy
```

- 简化模块名:

```
import module as alias
```

```
import matplotlib.pyplot as plt
```

```
import pandas as pd
```

- 调用部分模块语句:

```
from modname import name1, name2, ... nameN
```

The End

人生苦短
python 當歌

