第3章

观看视频

Python程序设计基本概念

本章主要介绍 Python 程序设计的基本概念，内容涵盖 Python 的文件类型、Python 源程序结构、关键词、内置函数、名字空间，以及环境变量的配置与使用等。

本章的学习目标：

- 掌握 Python 源程序的结构、编码、标识符、注释(含 docstring)的使用；
- 掌握 Python 关键词、常见的内置函数、Python 名字空间概念；
- 掌握 PYTHONPATH 环境变量的配置与使用、Python 模块与包的使用。

3.1 Python 程序概念

计算机虽然很强大，而且比人类更快地完成任务，但计算机并不智能，需要人类编写指令并告诉它该做什么，这些指令就是程序。

3.1.1 程序的定义

在维基百科上，计算机程序[①](Computer Program)的定义是：指一组指示电子计算机动作的指令，一般使用程序设计语言编写，然后被编译成计算机能够读懂的格式，最后由计算机执行，如 C++、Java、Python 等，并通过编译器或解释器翻译为机器可以执行的代码。这些指令使用 Python 语言来编写，就称为 Python 源程序。程序通常包括变量、数据类型、控制流语句、循环结构、函数等元素，并通过操作系统和其他软件完成某项任务。

例如，一个程序可以搜索文件系统并找到所有包含特定关键字的文件，或者实现一个图形界面，以便用户可以轻松地管理其文件。程序可以是简单的脚本，也可以是复杂的应用程序。

3.1.2 程序的编写

1. 需求分析

“大道至简”，其实计算机编程与日常生活处理事情类似。如日常生活中，买了 12 元的苹果，15 元的香蕉。现在计算一下总的消费，处理步骤是：首先要分别记录苹果、香蕉每一项的花费，再计算总的消费，苹果＋香蕉＝12＋15＝27 元。这个过程涉及的实体对象是苹果、香

① 计算机程序是指用一种特定的编程语言编写的指令集合，用来控制计算机执行某些任务。程序的语言可以是低级语言，如汇编语言，也可以是高级语言，如 C++或 Python。

蕉；任务是计算求和。

2. 编码设计

如果将上述处理步骤翻译为 Python 程序，代码为：

```
apple = 12
banana = 15
total = apple + banana
print('总费用：',total)
```

将代码保存为 s.py，然后运行：python s.py，即可得到运算结果。

3.1.3 程序的运行

Python 程序的运行过程是这样的：Python 解释器会自动在其源码所在的目录下，创建“_pycache_”文件夹，将源代码（s.py 文件）编译得到“s.cpython-38.pyc”字节码文件。然后在 Python 虚拟机[①](Python Virtual Machine，PVM)中一条一条执行字节码指令，从而完成程序的执行（见图 3.1）。因此，Python 源程序不需要编译成二进制代码，它可以直接从源代码运行程序。上述程序运行得到结果为：“总费用：27”。

图 3.1　Python 程序执行过程

3.1.4 程序的文件类型

在 Python 应用程序开发以及执行过程中，需要编写或自动生成几种类型格式的文件，常见的文件类型如下。

（1）*.py：.py 是最常见的 Python 文件类型，它包含 Python 代码，可以通过解释器直接运行或作为模块导入其他 Python 代码中。可以通过任何文本编辑器编写 Python 脚本文件。

（2）*.pyw：.pyw 实际上就是 .py 文件，只不过在 Windows 系统中运行时不会出现命令行窗口，而是在后台静默运行。这通常用于编写 GUI 应用程序，让用户不会看到命令行窗口的弹出，而是直接打开应用程序的主窗口。

（3）*.pyc 或.pyo：Python 编译文件是 Python 源代码的编译版本。在运行 Python 程序时，解释器会将源代码编译为字节码形式，存储在 .pyc 或 .pyo 文件中以便下次使用。Python 编译文件通常用于加速 Python 程序的启动时间，可以通过运行 Python 脚本文件或在 Python 解释器中导入模块来生成 Python 编译文件。

（4）*.pyx：.pyx 是使用 Cython 编写的 Python 扩展模块的源代码文件，可以包含 Python 代码和一些特殊的语法，例如 C 扩展类型和声明。.pyx 文件需要通过 Cython 或 Pyrex 工具进行编译，生成对应的 C 或 C++代码，然后使用 C/C++编译器将其编译成动态链接库（.pyd）或静态库（.a 或 .so）。

（5）*.pyd：.pyd 是一种共享库文件，通常是用 C 或 C++编写的 Python 模块的编译结果。可以通过将这些模块编译为 .pyd 文件，然后在 Python 中使用它们。

（6）*.whl：Python 打包文件是包含 Python 模块和相关文件的压缩文件，可以使用这些文件将 Python 模块分发到其他机器或共享给其他人使用。可以使用 Python 的打包工具

---

① Python 虚拟机（Python Virtual Machine，PVM）是一种解释器，用于执行 Python 程序，它会将 Python 代码翻译成字节码，然后执行该代码。

（如 setuptools 或 distutils）创建打包文件。

其中，使用频率最高的是 Python 源程序（*.py）文件。初学者应能够迅速识别上述 Python 相关的不同的文件类型及其作用。

3.2 Python 源程序的组成

3.2.1 程序代码示例

再来看一个稍复杂的 Python 源程序示例。该程序的功能是：打印当前日期时间、调用函数 hi()显示字符串信息。

【例 3.1】 Python 程序代码示例。

```
# -*- coding: utf-8 -*-                    ➤ 源文件编码格式声明
"""                                        ➤ 注释信息
Created on Tue Feb 25 09:07:47 2020
@author: Administrator
"""
import datetime                            ➤ 引入外部模块(库)
# 获取当前时间                             ➤ 注释信息
t = datetime.datetime.now()                ➤ 源代码

def hi(name):                              ➤ 定义 hi 函数
    """                                    ➤ 注释及帮助信息,可以用__doc__获取字符串内容
     Parameters
     ----------
    f : list
        files and directories list
    Returns
     -------
    flpy : list
        list only contains python files.
    """
   return 'hello:' + str(name)             ➤ 函数返回值

if __name__ == "__main__":                 ➤ 判断是否做主模块运行
    print(t)                               ➤ 程序源代码
    print(hi('James'))
```

从上面的演示代码可以看出：一个 Python 源程序通常由源码格式声明、注释、引入外部模块（库）、变量、函数、类等语句组成，每一条语句由 Python 的语法规则构成。

将上面的程序代码保存到 c:\mycode\demo.py 文件，在操作系统提示符下输入：python c:\mycode\demo.py，得到的程序运行结果如下。

```
2020-04-03 11:58:19.445626
hello:James
```

3.2.2 编码格式声明

Python 源程序默认为 UTF-8 编码[①]格式文本文件，当然也支持其他编码格式，如在 Windows 操作系统下的记事本中编辑 Python 源程序，可以选择 ANSI、Unicode、Unicode big

① UTF-8 是一种变长的编码方式，它可以使用 1～4 字节来表示一个 Unicode 字符。它使用最少的字节数来表示常用字符，同时又可以表示所有 Unicode 字符。它兼容 ASCII 编码，并支持多语言。

endian、UTF-8 类型的编码格式(见图 3.2)。

图 3.2 源程序的编码格式

如果 Python 源码文件使用非 UTF-8 类型的编码,必须在源程序的第一行声明文件的编码格式,如中文编码:# -*- coding: cp-936 -*-。

3.2.3 注释与文档属性

Python 程序的注释语句用于备注程序代码功能和逻辑关系、算法的编写思路,便于程序的后期维护等。另外,符合 Python 规范的程序注释,可自动生成程序的帮助文档。Python 有单行和多行注释两种方式。

(1) 单行注释:单行注释以 # 开头。

(2) 多行注释:多行注释可以用成对的三个单引号 ''' 或三个双引号 """来标注。

文档字符串(docString)是一种多行注释,它作为模块、函数、类或方法定义中的第一条注释语句出现,Python 的 PEP257(https://www.python.org/dev/peps/pep-0257/)规定了文档字符串的标准格式。文档字符串会自动解析成为该函数或对象的特殊属性__doc__,__doc__属性可以被 Python 程序访问,还可以直接由相关工具生成 html 等格式的文档。

以下程序演示多行注释。

【例 3.2】 查看__doc__属性的代码示例。

```
通过运行代码: print(hi.__doc__)
或者在 Python 提示符 ">>>" 下输入 help(hi),会打印如下信息:
Help on function hi in module __main__:
hi(name)
     Parameters
    -----------
    name : string
        a string,such as name
    Returns
    --------
    flpy : string
        hello + string
```

3.2.4 程序中的变量

变量是程序中用来储存数据的容器。在 Python 中,变量是没有类型限制的,可以存储任何类型的数据,Python 源程序中的数据、函数、类等都可以定义为变量,变量通过标识符来命

名。与其他编程语言不同，Python 没有声明变量的命令，变量在使用时，无须声明变量类型，Python 系统会自动根据变量值的类型，确定变量的类型。首次为其赋值时，才会创建变量，变量不需要使用任何特定类型声明，可以任意复制并更改其类型。

例如：x＝8，x 是变量，8 是变量的值，数据类型为整数。

x＝print，x 是变量，print 是变量的值，数据类型为函数。

变量 x 初始值为整数 8，然后又赋值为函数 print。

3.2.5 程序中的代码块

Python 程序代码块是指具有一定逻辑功能的 n 行代码语句，Python 使用相同的缩进空格、连续 n 行代码来表示同一级别代码块。

（1）缩进空格：缩进的空格数是可变的，但是同级别的一个代码块的语句必须包含相同的缩进空格数，通常采用 4 个空格表示一个级别缩进。

【例 3.3】 Python 语言代码块缩进代码示例。

```
if True:
    print("True")
else:
    print("False")
```

如果同级别代码块缩进数的空格数不一致，会导致运行错误：

```
if True:
    print("Answer")
    print("True")
else:
    print("Answer")
  print("False")       # 缩进不一致，会导致运行错误
  File "<tokenize>", line 6
    print("False")
    ^
IndentationError: unindent does not match any outer indentation level
```

（2）分行语句：Python 通常是一行写完一条语句，但如果语句很长，可以使用反斜杠（\）来实现将一行长的语句分成多行语句。

【例 3.4】 Python 语言一行长代码，采用换行写法示例。

```
total = "A33"\
        "B44"\
        "C55"
print(total)
#在 [], {}, 或 () 中的多行语句，不需要使用反斜杠(\)，例如：
total = ['item_one', 'item_two', 'item_three',
         'item_four', 'item_five']
A33B44C55
```

进一步，如果将代码块按照一定格式组织并命名，就可以定义为函数（第 6 章将详细介绍函数的编写），如上述代码中的：

```
def hi(name):
……
```

3.3 Python 的标识符

3.3.1 标识符的作用

标识符(Identifier)是指用来标识某个实体的一个符号,在计算机编程语言中,标识符是用户编程时使用的名字,用于给变量、常量、函数、类等命名。标识符通常由字母、数字以及其他字符构成。上面程序中的变量 t、函数 hi、类名 datetime.datetime、模块名 datetime 都是标识符。

3.3.2 命名规则

Python 标识符有比较严格的命名规则,规则如下:

(1) 标识符不能与关键字相同。

(2) 标识符由字母、数字、下画线组成,但第一个字符不能是数字。标识符的第一个字符必须是字母(ASCII 字符或 Unicode 字符)或下画线。标识符对英文的大小写敏感。标识符的其他部分可以由字符、下画线、数字组成,标识符的长度没有限制。

可以使用非 ASCII 作标识符,如中文作为变量名或函数名。"_ko""中国""disk"等是合法的标识符; 3ks、in、+wps 等是不合法的标识符。

Python 建议使用驼峰命名法[①],大写字母开头的标识符用于类名,小写字母开头的标识符用于函数名和变量名。

3.3.3 特殊标识符

Python 有其专用的下画线标识符,"_"(1 个下画线)或 "__"(2 个下画线)可作为变量标识符的前缀和后缀来标识特殊变量,表 3.1 所示为 Python 中下画线标识符。

表 3.1 Python 中下画线标识符

类型	描 述	含 义	示 例
_xxx	1 个下画线开始	保护变量或方法,仅限类对象自己和子类对象对该变量或方法访问	_foo
__xxx	2 个下画线开始	私有成员,仅限其所在类访问	__foo
__xxx__	2 个下画线开头、结尾	通常是 Python 定义的系统变量或方法	__init__()、__main__

一般来讲,变量名_xxx 被看作"私有的",在模块或类外不可以使用。当变量是私有的时候,用_xxx 来表示变量是很好的习惯。

由于下画线对 Python 解释器有特殊的含义,建议初学者在不明确下画线的含义时,避免用下画线开头或结尾的标识符作为变量名。

3.4 Python 内置要素

3.4.1 关键词

Python 语言的关键词是构造 Python 逻辑程序代码的核心要素,它与用户定义的变量或函数组合构成程序语句代码。关键词可以按类别分为:常量、逻辑运算、程序流控制、异常与上下文处理、函数相关、模块与类管理(见图 3.3)6 大类。

---

① 驼峰命名法主要用于类名和变量名,它有助于代码可读性和可维护性。小写字母开头的标识符用于函数名和变量名,大写字母开头的标识符用于类名。还有一种命名规范叫作下画线命名法(snake_case),在这种命名法中,单词之间用下画线隔开。

图 3.3　Python 语言关键词分类

这些关键词是 Python 的保留字，只能在特定的上下文中使用，例如，if 关键词用于条件语句，def 关键词用于定义函数，class 关键词用于定义类。

关键词可以在 Python 中使用 import keyword 获取，keyword.kwlist 可以获取 Python 的所有关键词。

【例 3.5】 查看当前 Python 版本所有关键词，统计关键词数量。

```
>>> import keyword
>>> keyword.kwlist                    #关键词列表
>>> len(keyword.kwlist)               #统计关键词的个数
```

上述代码运行结果：

```
['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']
35
```

可以看到，Python 共有 35 个关键词（注：Python 3.10 后新增的 match-case 关键词暂未列入其中），有许多关键词与其他编程语言是相同的，如关键词'assert'，'break'，'class'，'continue'，'else'，'except'，'finally'，'for'，'if'，'import'，'return'，'try'，'while'等与 Java 语言的关键词是相同的，而且含义和用法是一样的。

3.4.2　内置常量

常量是不变的量，可以在程序中直接使用，不需要自己定义，如日常生活中，一提到圆周率，人们会自然想到 3.14。Python 中有一些内置常量，这些常量是 Python 编程语言的一部分，可以直接使用。Python 定义了 6 个常量（见图 3.4），使用最频繁的 3 个常量是 False、True、None，其余 3 个是 Ellipsis、__debug__、NotImplemented。

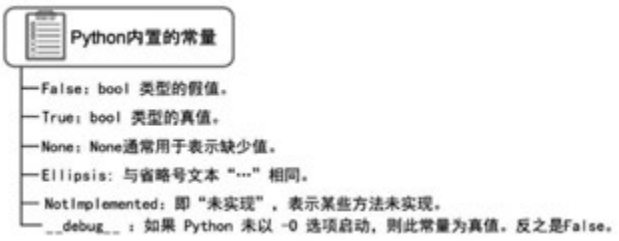

图 3.4　Python 内置的 6 个常量

3.4.3 内置函数

Python 语言核心部分内置了 69 个内置函数(也称内部函数),在 Python 程序中可以直接调用这些函数(https://docs.python.org/3.9/library/functions.html),这些函数是 Python 编程语言的一部分,可以直接使用。Python 内置函数分为:数字相关、数字运算、编码相关、容器相关、对象相关、系统函数、变量相关 7 个大类(见图 3.5)。

图 3.5 Python 语言内置函数及分类

- 用户自定义的标识符(变量名、函数、类名等)应避免与内置函数名相同。否则,内置函数会被用户定义的同名标识符覆盖。
- 可以在 Python 提示符“>>>”下,输入 dir(__builtins__),可显示 Python 内置名字空间中的函数名、常量和关键字。

由于 Python 系统内置函数比较多,初学者一次全部掌握有困难。这里推荐读者先掌握以下使用频率最高的 6 个内置函数:输入输出函数、对象相关函数。这些函数之间的逻辑关系见图 3.6。

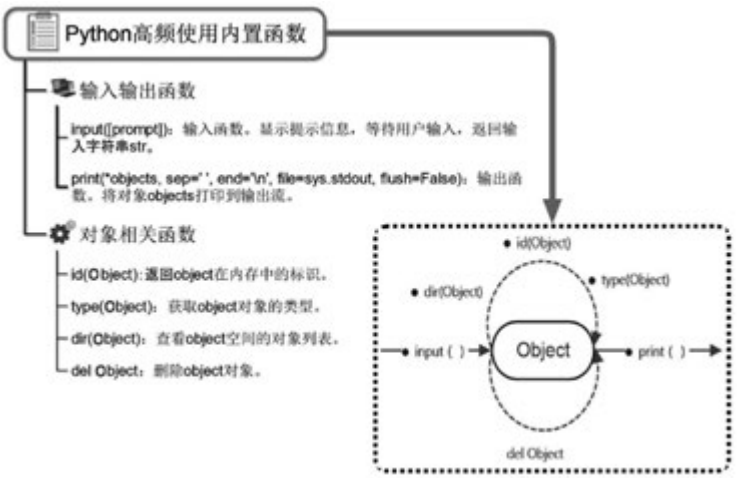

图 3.6 使用频率较高的 6 个常用内置函数

3.4.4 名字空间

在 Python 中，名字空间(Name space)是用来组织名称(变量名、函数名、类名等)的容器。在 Python 程序中，变量、函数、类等都被分配到相关的名字空间，每一个标识符都会属于特定的名字空间，并占据特定的内存位置，可以在相应名字空间中来查找到这些标识符并使用。Python 有 3 个级别的名字空间，即局部、全局、内置名字空间(见图 3.7)。

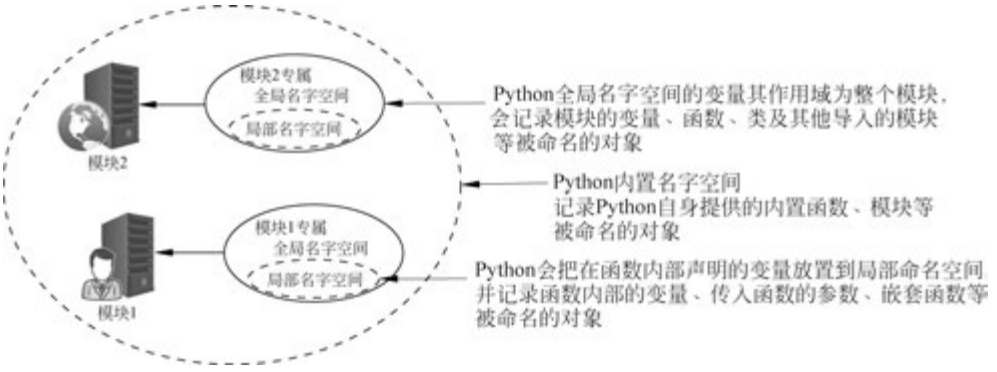

图 3.7 Python 的名字空间及作用范围

(1) 局部名字空间：Python 会把在函数内部声明的变量放置到局部命名空间，并记录函数内部的变量、传入函数的参数、嵌套函数等被命名的对象。

(2) 全局名字空间：Python 全局名字空间的变量其作用域为整个模块，会记录模块的变量、函数、类及其他导入的模块等被命名的对象。

(3) 内置名字空间：记录 Python 自身提供的内置函数、模块等被命名的对象。

在 Python 程序执行过程中，会有局部名字空间、全局名字空间和内置名字空间 3 个空间同时存在的情况。Python 会按照“局部”→“全局”→“内置”名字空间的顺序对变量、函数、类进行查找，如果找到了就可以使用，否则将引发一个 NameError 异常。若 Python 同一个名字空间中出现同名变量、函数、类等标识符，则后出现的同名变量、函数、类将覆盖先前的。

综上所述，Python 名字空间除了容纳标识符以外，还有两个作用：

(1) 区分对象的作用域；

(2) 防止同名变量、函数、类等名字冲突。

【例 3.6】 查看定义一个变量前后名字空间内容的变化。

查看名字空间的命令函数是 dir()，如查看定义 x=100 前后，Python 名字空间内容的变化情况。

(1) 定义变量前，查看名字空间内的对象：

```
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
```

(2) 定义变量后，查看名字空间内的对象：

```
>>> x = 100
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__', 'x']
```

可以看到变量'x'已经占据了当前的名字空间的一个位置。

如果释放变量、函数、类等的名字空间，直接使用 del 删除即可。

【例 3.7】 查看删除变量后名字空间的内容变化。

清理、删除对象 x。

```
del x
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__']
```

可以看到执行了'del x'命令后，'x'所占据的名字空间已经释放。

【例 3.8】 测试使用未定义的变量引发 NameError 异常。

```
>>> aa
Traceback (most recent call last):
  File "<pyshell#3>", line 1, in <module>
    aa
NameError: name 'aa' is not defined
```

上述代码出错原因是在所有的名字空间中找不到 aa 的定义。

3.4.5 综合案例：探究变量在名字空间内的变化

【例 3.9】 已知苹果花费 12 元，香蕉花费 15 元，求总费用。请综合使用函数 dir()、id()、type()、input()、print()、del()来计算总费用，并查看保存总和的变量 id、数据类型，运算前后名字空间变量情况。

```
print('程序运行前名字空间情况',dir())
apple = 12
banana = 15
total = apple + banana
print(f'{id(total) = },{type(total) = }')
msg = input('输入名称: ')
print(msg,total)
print('程序运行后名字空间情况',dir())
del msg,total,banana,apple
print('删除使用变量后名字空间情况',dir())
```

运行结果：

```
程序运行前名字空间情况 ['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__']
```

解释：可以看到程序在运行前，名字空间内没有自定义的变量。

```
id(total) = 2320094397488,type(total) = <class 'int'>
输入名称：总费用
总费用 27
程序运行后名字空间情况 ['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'apple', 'banana', 'msg', 'total']
```

解释：可以看到程序在运行后，名字空间内增加了自定义的'apple'，'banana'，'msg'，'total'变量。

```
删除使用变量后名字空间情况 ['__annotations__', '__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__']
```

解释：代码 del msg,total,banana,apple,删除了这些变量。

3.5 Python 代码的组织与使用

编写好的代码如何被其他程序调用？如何使用别人编写的代码？

Python 代码通常使用以下几种方式组织。

(1) 函数：可以被多次调用的代码块。

(2) 类：提供了对象的抽象,可以被继承和实例化。

(3) 模块：一个独立的文件,包含了可以被其他代码复用的函数、变量、类。

(4) 包：一组相关的模块组成的目录,可以在需要时被导入和使用。

3.5.1 模块

在 Python 语言中,模块(Module)是一组定义的函数、变量和类的集合。模块可以帮助组织代码,将代码分成不同的部分,并使用不同的模块实现不同的功能。模块可以在多个程序中使用,这样可以避免代码冗余,使代码更简洁、易读。可以使用 Python 内置的模块,如 math、os 等,也可以创建自己的模块,以实现特定的功能。

1. 什么是模块

Python 程序源码可以包含变量、常量、注释、函数、类、导入语句等,将程序源码保存到文件,就得到了一个 Python 源程序(如例 3.1 的 c:\demo\demo.py 文件),一个 Python 源程序就是一个 Python 模块。

2. 模块的检索路径设置

代码通常是通过模块或包来组织管理与使用的。在 Python 中使用其他模块时,即使这些模块与你的代码在同一个目录下,如果没有将这些模块所在的目录添加到 PYTHONPATH 或者 sys.path 中,Python 也无法找到这些模块。如果要在其他程序中使用该模块定义的函数或变量,首先要在环境变量 PYTHONPATH 中配置该模块的检索路径(见图 3.8)。

图 3.8 PYTHONPATH 的配置

另一种添加模块检索路径方法是使用代码来实现：

```
import sys
sys.path.append('your/module/path')
```

3. 模块的使用

在使用 import 导入该模块后,导入的模块中的函数、变量和类可以通过“模块名.函数名”的方式访问。

(1) 方法 1：使用 import...

【例 3.10】 使用 import demo 导入模块，并查看导入 demo 模块后的名字空间。

```
>>> import demo
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__spec__',
'demo']
```

可以看到当前名字空间中多了一个“demo”对象。

【例 3.11】 调用 demo 模块内的函数案例。

调用 demo 名字空间下的函数，前面必须加名字空间名称 demo。

```
>>> demo.hi('james')
'hello:james'
```

总结：“import demo”是将“demo.py”模块内的变量、函数导入“demo”名字空间下，使用变量或函数时要加 demo 名字空间名。

(2) 方法 2：使用 from...import...

另一种导入方式是：将 demo.py 模块内的变量、函数导入全局名字空间内。导入后可以直接调用导入的变量和函数。

【例 3.12】 使用 from demo import * 导入模块，并查看导入后的名字空间。

```
>>> from demo import *
>>> dir()
['Info', '__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'datetime', 'hi', 't']
>>> hi('james')
'hello:james'
```

总结：“from demo import *”是将“demo”模块内的变量、函数或类等导入全局名字空间内，导入后可以直接使用相关变量或函数。通过使用模块，可以更好地组织代码，便于代码维护、扩展和共享代码，因此，使用模块是高效编写 Python 代码的重要方法。

3.5.2 常用的模块

1. 常用的内置模块

Python 有很多内置模块，它们提供了丰富的功能。下面是一些常用的内置模块的作用。

- sys：提供了许多变量和函数，用于操作系统，如命令行参数、标准输入输出、环境变量等。
- os：提供了许多与操作系统相关的函数，如文件/目录操作、进程管理、环境变量等。
- math：提供了数学运算函数，如三角函数、对数、幂、随机数等。
- random：提供了生成随机数的函数。
- datetime：提供了日期和时间处理的类和函数，如日期、时间、时间差、格式化等。
- time：提供了时间相关的函数，如时间戳、睡眠、计时等。
- json：提供了 JSON 数据处理的函数。
- pickle：提供了 Python 对象序列化和反序列化的函数。
- re：提供了正则表达式模块，用于文本模式匹配和替换。
- logging：提供了日志系统，用于记录程序运行信息。

- collections：提供了高效的集合容器，如 defaultdict、counter、ordereddict 等。
- heapq：提供了堆排序相关的函数。
- itertools：提供了迭代器函数，如 groupby、count、cycle 等。
- functools：提供了高阶函数，如 map、reduce、filter、partial 等。

这些模块提供了非常多且高效的函数，编程时可以拿来直接用，大大节省了编程量。建议读者在掌握核心的 Python 语法后，再熟悉这些模块。

2. 常用的第三方模块

Python 常用的第三方模块非常多，这里仅列出常用的数据分析与可视化第三方模块。

- NumPy：提供了高性能的数值计算工具，如数组、矩阵运算、线性代数等。
- Pandas：提供了强大的数据处理工具，如数据帧、时间序列、缺失值处理、合并等。
- Matplotlib：提供了丰富的可视化工具，如折线图、散点图、直方图、热图等。
- Seaborn：基于 Matplotlib 的可视化库，提供了更为美观的可视化效果。
- Statsmodels：提供了统计分析和建模功能，它包括了统计检验、线性回归、时间序列分析、状态空间模型等。这些功能可以用来做数据预处理、统计建模、结果预测等。
- Bokeh：提供了交互式可视化工具，可用于制作交互式网页图表。
- Plotly：提供了交互式可视化工具，可用于制作交互式网页图表。
- Scikit-learn：提供了机器学习工具，如分类、回归、聚类、降维等。
- TensorFlow：提供了深度学习工具，支持计算图、自动微分、分布式计算等。
- Keras：提供了深度学习高层次封装，简化了深度学习模型的构建和训练。
- PyTorch：提供了深度学习工具，类似 TensorFlow，用于计算图、自动微分、分布式计算等。

这些第三方模块提供了强大的数据分析与可视化，以及机器学习等方面的函数与软件框架。建议读者先熟悉 NumPy、Pandas、Seaborn 等基础模块，再去学习 Scikit-learn、TensorFlow 或 PyTorch 等机器学习模块。

3.5.3 包

Python 中的包(Package)是组织、管理源代码的一种方式。包就是一个容器，用来存放其他模块和子包。一个大的 Python 项目可能由不同开发人员分工完成，使用包来组织代码、管理代码可以避免出现模块名、变量名、函数名、类名等重名的问题，便于分工协作。Python 的包使用树状目录结构组织模块(见图 3.9)。

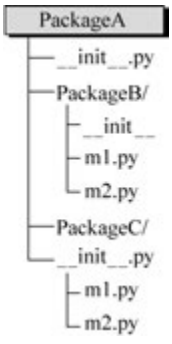

图 3.9 包结构

创建一个包需要以下步骤：

(1) 创建一个目录，用来存放包中的模块。将 Python 代码按照不同级别的目录存放代码。

(2) 在该目录下创建一个名为 __init__.py 的文件。该文件的存在标识则是一个包(该文件内容可以为空)，代表该目录下的 Python 文件使用包进行管理。

可以简单地把包理解为：按照不同级别的子目录来组织模块代码，每个目录下必须有一个__init__.py 文件。包的本质就是一个含有__init__.py 文件的文件夹。

➢ 使用 Python 包中的模块语法是通过目录的级别的方式来引用模块(如"PackageA.PackageB")。

➢ PackageB,PackageC 目录下均有同名 m1. py、m2. py 模块,可以通过包名和模块名来区别。如 import PackageA. PackageB. m1; import PackageA. PackageC. m1。

Python 会在 sys. path 指定的目录(环境变量 PYTHONPATH 所设定的目录)下搜索包的路径。包中的模块和函数,可以通过两种导入方式使用。

(1) import 包的名字。

(2) from 包的名字 import 函数名。

3.5.4 模块与包的区别

在 Python 中,模块和包都是用来组织代码的容器。但是,它们有一些区别:

- 模块是一个 Python 文件,其中包含了 Python 代码。模块可以被导入其他模块或脚本中使用。模块是一个 Python 脚本(源程序),扩展名为 .py 的文件,包含变量、函数、类等。
- 包是一个目录,其中包含了多个模块和其他子包。包可以被导入其他模块或脚本中使用。包是一个包含模块集合的目录,这个目录还包含一个 __init__. py 文件,该文件可以用于包的初始化或组织管理目录下的模块,Python 解释器通过它将该目录下的 .py 文件解析为一个包,导入该包时候会自动运行__init__. py 文件。
- 模块可以被单独使用,而包必须被导入其他模块或脚本中使用。

其实,大多数情况下将包、库、模块"混称"为模块。读者可以这样验证:尝试用代码调用一个没有安装的包,Python 控制台一律显示某模块没有安装,而不是某包没有安装。

```
import mxnet
Traceback (most recent call last):
  File "< pyshell # 1 >", line 1, in < module >
    import mxnet
ModuleNotFoundError: No module named 'mxnet'
```

3.6 本章小结

本章主要介绍了 Python 源程序的结构、编码、标识符、注释、Python 关键字、几个常见的内置函数、Python 模块、名字空间,以及使用包管理代码的机制。Python 作为一种编程语言,具有易于学习、灵活性和丰富的库的特性,程序可以通过模块和代码组织的方式来提高复用性和可维护性(见图 3.10)。

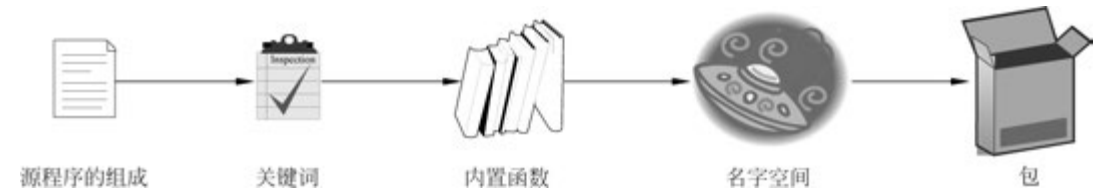

图 3.10 本章知识要点一览图

3.7 习题

1. Python 程序有哪些文件类型? 源程序由哪些组成?
2. Python 注释方法有哪几种? 文档属性是什么?
3. Python 内置常量有哪些? 什么是变量?

4. Python 常用的输入输出函数有哪些？

5. 如何查看对象的 id 和数据类型？dir()函数的作用是什么？

6. 如何使用别人编写好的模块中的函数？PYTHONPATH 的作用是什么？

7. Python 的名字空间有哪几类？各自的作用范围？

8. 任意定义一个变量并赋值，然后删除该变量。试分析：变量定义前后，名字空间的变化；删除变量后，名字空间的变化。

9. Python 关键词有哪些？编写一个程序打印 Python 语言所有关键词的程序 h.py。

10. 分析比较“import…”与“from…import…”两种方式引入 h.py 模块后，变量与函数使用方式的区别。