第7章 面向对象的程序设计与Python生态

本章教学目标：

- 理解面向对象的概念，理解类与实例、属性和方法。
- 了解如何创建类、创建子类以及创建类实例。
- 初步理解Python面向对象的特征。
- 理解Python的程序管理结构。
- 掌握库、包和模块的引用方法。
- 熟悉Python的生态，掌握第三方库的获取和安装方法。
- 了解Python程序编译方法。

7.1 面向对象的概念

- 类（Class）

对具有相同属性和方法的一组对象的描述或定义

- 对象（Object）

现实世界中可以明确标识的一个实体，类的实例

- 实例（Instance）

其含义与对象基本一致。创建一个新对象的过程称为实例化(Instantiation)，这个新对象称为这个类的一个实例

- 标识（Identity）

每个实例对象的唯一标识

7.1 面向对象的概念

- 实例属性（Instance Attribute）

特定对象所具有的一组属性的集合

- 实例方法（Instance Method）

对于特定对象实例的一条或者多条属性的操作函数集合

- 类属性（Class Attribute）：

属于一个类中所有对象的属性，不会只在某个实例上发生变化

- 类方法（Class Method）

无须指定实例就能够工作的从属于类的函数

已学的类（类型）

- 类（类型）

  int、 float 、bool 、string

  list 、tuple 、dict 、set

- 对象举例

  3，[5.7,’a’]，{1:4,2:5}

type()判断对象的类

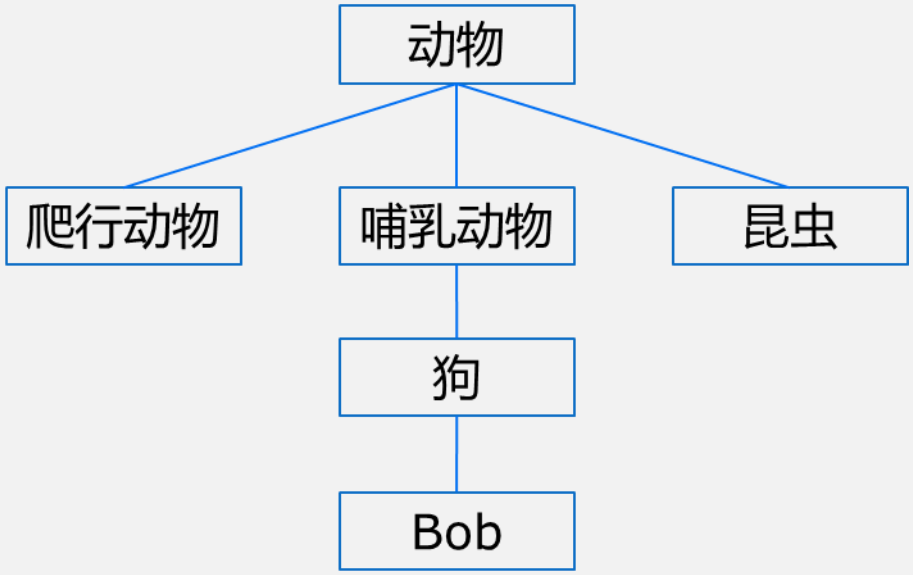
isinstance()测试一个对象是否为某个类的实例

7.1 面向对象的概念

在Python语言里，对象其实是一个指向一个数据结构的指针，这个特定数据结构里有属性、有方法。从面向对象的概念来讲，对象是类的一个实例。

对象可以使用从属于该对象的变量存储数据，属于一个对象或类的变量称为特性。特性有两种类型：属于每个实例/类的对象或者属于类本身，它们分别被称为实例变量和类变量，而所有的变量都可以被称为对象

对象也可以使用属于类的函数具有的功能，这样的函数被称为类的方法，特性和方法可以合称为类的属性。

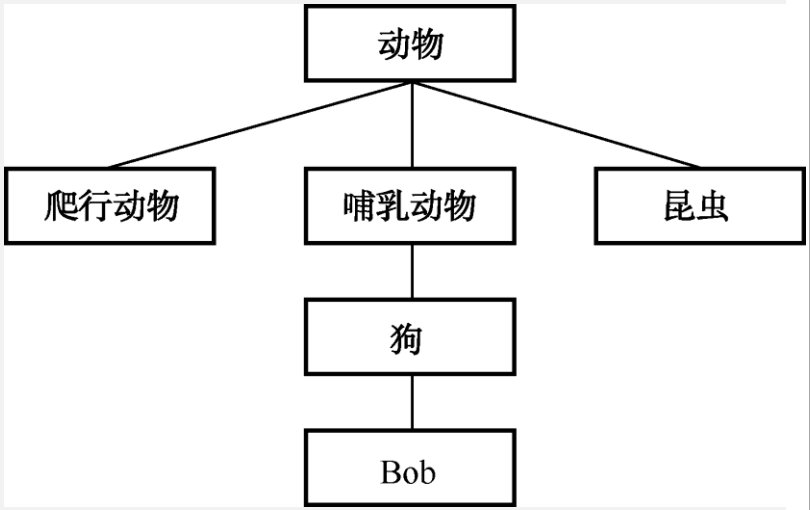


7.2 类与实例

- 创建类和子类

使用class关键字创建，类的属性和方法被列在一个缩进块中。

```
class Animals:
    pass
class Mammals(Animals):
    pass
class Dog(Mammals):
    pass
```



子类Mammals可以继承父类Animals的所有属性，同样，子类Dog也可以继承父类Mammals的所有属性。

__init__方法

- __init__方法是Python类中的一种特殊方法，方法名的开始和结束都是双下划线，该方法称为构造方法，当创建类的对象时，它被自动调用。
- __init__方法中可以声明类所产生的对象属性，并可为其赋初始值。该方法有一个特点，不能有返回值，因为它是用来构造对象的，调用后实例化了一个该类型的对象。

Self参数

- 类的实例方法按惯例有一个名为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。
- 在类的方法中访问实例变量（数据成员）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。

7.2 类与实例

- 增加属于类的对象实例

有一条具体存在的名叫Bob的狗，属于类Dog，是一个实例对象，定义如下：

Bob=Dog()

  - 【例7-1】 创建Dog类，并实例化对象Bob

- 用函数表示类的行为特征
  - 【例7-2】 为类和子类创建行为特征—函数。

7.3 面向对象的特征

- 封装

将抽象得到的数据和行为相结合，将基本类结构的细节隐藏起来，通过方法接口实现对实例变量的所有访问。

```
class Company:
    def __init__(self, companyname, leader):
        self.companyname = companyname
        self.leader = leader
if __name__ == "__main__":
    obj1 = Company("A","Kevin")
    obj2 = Company("B","Jone")
```

将公司名和领导人分别封装到对象obj1、obj2中self的companyname和leader属性中

- 【例7-3】封装及封装数据调用

  通过对象直接调用；通过self间接调用

7.3 面向对象的特征

- 继承

当已经存在一个类，需要另外再创建一个和已有类非常相似的类时，通常不必将同一段代码重复多次，而是用继承。在类上添加关联，使得位于下层的类可以“继承”位于关系上层的类的属性。继承有利于代码的复用和规模化。和其他语言不同的是，Python中的类还具有多继承的特性，即一个类可以有多个父类。

- 【例7-4】 继承一个父类。
- 【例7-5】 继承多个父类。

7.3 面向对象的特征

- 多态

即多种状态，是指在事先不知道对象类型的情况下，可以自动根据对象的不同类型，执行相应的操作。很多内建运算符以及函数、方法都能体现多态的性质。

  - 【例7-6】 函数的多态性举例。

7.4 Python程序的组织和管理

- 源程序和模块结构

```
"""This is Universe Module"""
```
（1）模块文档

```
import math
```
（2）模块导入

```
Light_speed = 2.9992458e8
```
（3）变量定义

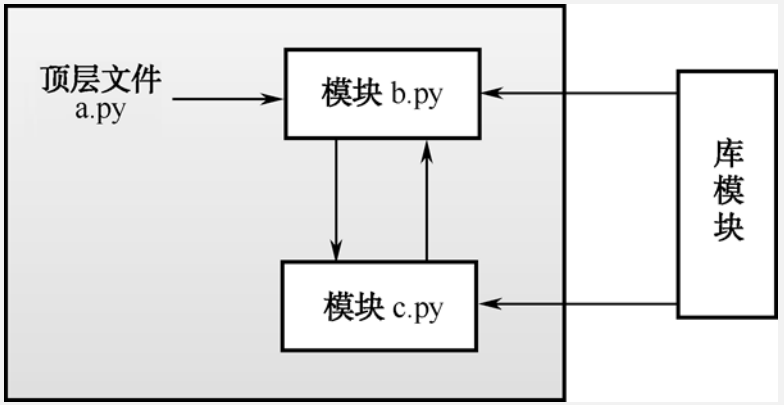
```
class UniverseClass(object):
    """Universe class"""
    pass
```
（4）类定义语句

```
def BigBang():
    """The beginning of all things"""
    ourUniverse = UniverseClass()
    print('Let there be light')
```
（5）函数定义语句

```
if __name__ == '__main__':
    BigBang()
```
（6）主程序

7.4 Python程序的组织和管理

通常Python程序的架构是指将一个完整的程序分割为源代码文件的集合以及将这些文件连接起来的方法。Python使用模块化的方法来组织其架构，一个Python程序就是一个模块化的系统，它有一个顶层文件和多个模块文件



7.4 Python程序的组织和管理

- 模块的引用
  - import方式

    在当前的命名空间中建立了一个指向该模块的引用

    import <模块名>

    import <模块名> as <别名>

    使用模块中的函数：<模块名>.<函数名>

    <别名>.<函数名>
  - from 方式

    将模块中指定的属性或名称导入当前命名空间中

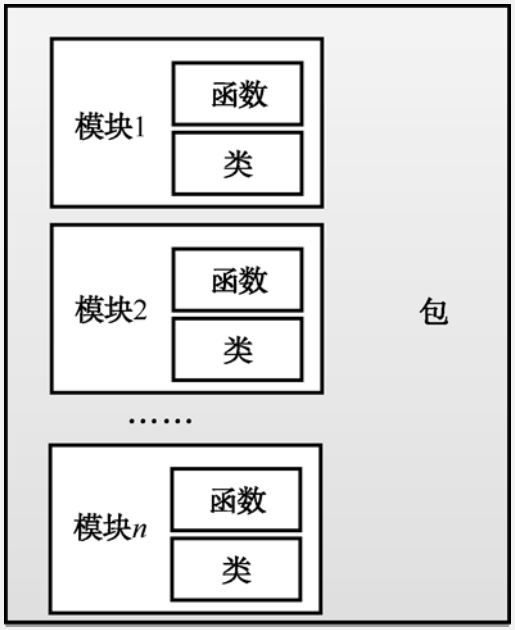
    from <模块名> import <函数名>

    from <模块名> import *

    直接使用函数

7.4 Python程序的组织和管理

- 包和库
  - 程序以模块化组织和管理，把函数或类定义在一个以.py为后缀的文件里，在需要的时候引用。
  - Python程序一般由包（package）、模块（module）和函数（function）三部分组成。其中，包是由一系列模块组成的集合，模块是处理某类问题的函数和类的集合
  - 一个包中可以包含多个模块，每个模块中可以包含多个函数与类，同时也可以有执行语句，每个包其实就是完成特定任务的工具箱。
  - 具体来说，一个.py文件可以被看作一个独立的模块，一个模块通常就是一个文件，因为模块是按照逻辑组织代码的方法，而文件是物理层存储模块的方式。



7.5 Python的生态

- 庞大的开源社区，良性的计算生态维持了涵盖各个领域的专业级免费共享代码复用
- 共享库索引
  - 官网：

    https://pypi.python.org

    https://pypi.org
  - 国内镜像：

    https://pypi.tuna.tsinghua.edu.cn/simple/

    http://mirrors.aliyun.com/pypi/simple/

    http://pypi.douban.com/simple/

    http://pypi.mirrors.ustc.edu.cn/simple/

7.5 Python的生态

- 内置库（标准库）
  - 与Python一起安装的类库
  - 模块对应的物理层结构是文件
  - 直接用import语句即可引用
- 第三方库
  - 来源于共享社区
  - 以包的形式发布
  - 需要安装后方能使用

7.5 Python的生态

- 包的管理

  包是通过目录结构组织的模块的集合

  - 包对应的物理层结构是文件夹
  - 文件夹中需要包含一个_init_.py文件
  - 通过import语句导入
  - 管理工具
    - distutils
    - setuptools
    - easy_install
    - pip
  - 安装包格式

  .exe　.zip或.tar.gz　.whl　.egg

7.5 Python的生态

- 第三方库的安装
  - 安装可执行安装文件（.exe）

直接运行，自动安装。

  - 安装带setup.py文件的安装包

将安装包解压缩后，用CMD进入命令提示符界面，进入setup.py文件所在目录，执行命令python setup.py install即可开始安装。

  - 安装（.whl）压缩包

用CMD进入命令提示符界面，执行命令：

<Python安装路径>\Scripts\pip install <.whl文件>

  - 安装（.egg）压缩包

用CMD进入命令提示符界面，执行命令：

<Python安装路径>\Scripts\easy_install <.egg文件>

  - 已知类库名称，在连网状态下直接安装

<Python安装路径>\ Scripts\easy_install <类库名称>

<Python安装路径>\ Scripts\pip install <类库名称>

7.5 Python的生态

管理员: 命令提示符

```
C:\WINDOWS\system32>pip install openpyxl
Collecting openpyxl
  Downloading openpyxl-3.0.3.tar.gz (172 kB)
     |                                | 81 kB 2.3 kB/s eta 0:00:3
     |                                | 92 kB 2.4 kB/s eta 0:00
     |                                | 102 kB 2.5 kB/s eta 0
     |                                | 112 kB 2.7 kB/s eta
     |                                | 122 kB 2.7 kB/s e
     |                                | 133 kB 2.8 kB/s
     |                                | 143 kB 3.1 kB
     |                                | 153 kB 2.8
     |                                | 163 kB 2.
     |                                | 172 kB 2
.3 kB/s
Collecting jdcal
  Downloading jdcal-1.4.1-py2.py3-none-any.whl (9.5 kB)
Collecting et_xmlfile
  Downloading et_xmlfile-1.0.1.tar.gz (8.4 kB)
Installing collected packages: jdcal, et-xmlfile, openpyxl
    Running setup.py install for et-xmlfile ... done
    Running setup.py install for openpyxl ... done
Successfully installed et-xmlfile-1.0.1 jdcal-1.4.1 openpyxl-3.0.3

C:\WINDOWS\system32>
```