Python及其应用

主讲人：钱惠敏
E-mail: amandaqian@hhu.edu.cn

第7讲 面向对象编程

7.1 面向对象编程

7.2 类

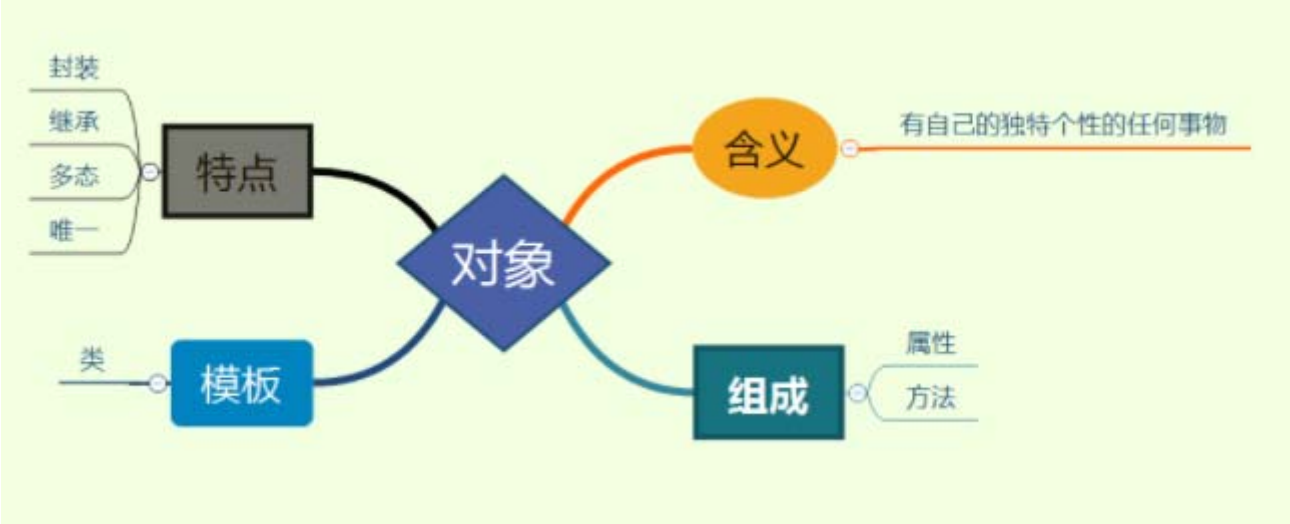
7.3 对象

7.4 类的创建

7.5 类的继承

7.1 面向对象编程

➢Python语言是一种面向对象编程语言（Object-Oriented Programming Language），简称为**OOP**语言

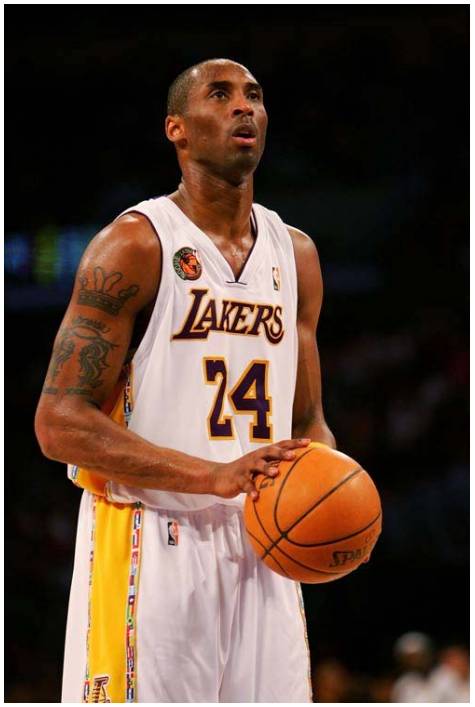


7.1 面向对象编程（续1）

- 无论是变量还是函数，它们都是属于某一个特定的类（class）。类中的对象叫做该类的**实例（instance）**。
- 特定类的所有对象都与**方法（method）**相关联，这些方法类似于一个函数，我们编写一次之后，可以重复的调用。
- 一旦某个对象属于某个特定类之后，该对象就可以使用该类的所有方法。

7.2 类

- 物以类聚，类是许多具有相似特征的事物的集合，特征就是属性和方法



- 一个NBA球员
  - 组成元素
    - 头、五官、四肢
    - 性别、血型、身高、名字
    - 所在球队、球衣号码
  - 行为
    - 吃喝
    - 跑步、跳跃
    - 运球、投篮、传球、抢断、传球

7.2 类（续2）

高中生类

属性：姓名、年龄、性别、分数

方法：对属性进行查询或修改、与其他同类发生一些关系、运算等等

7.3 对象

高中生类

高中生1

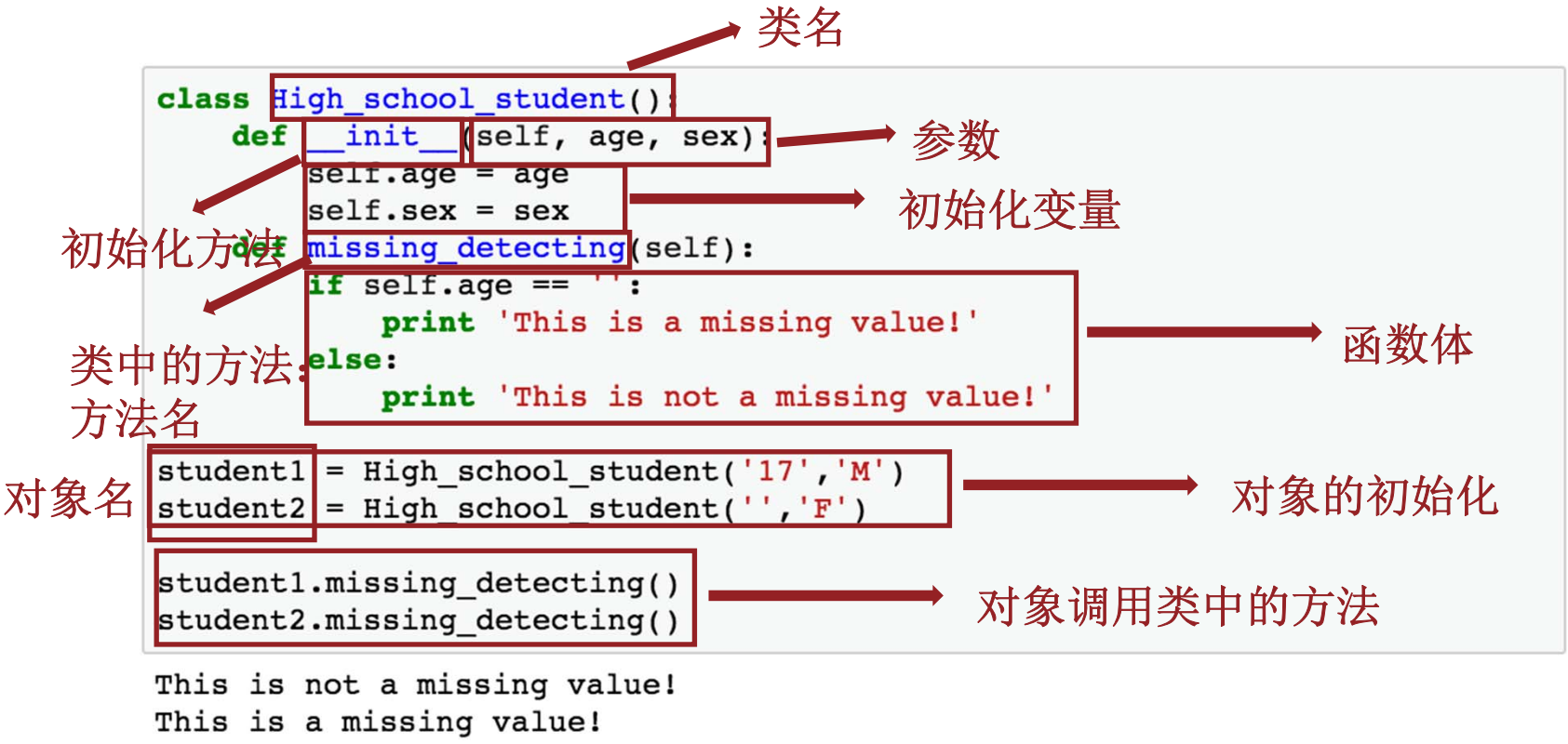
对象的属性
姓名：王耀华
年龄：18
性别：男
成绩：3.51
……

共有的方法：
改变姓名、改变成绩等等

高中生2

对象的属性
姓名：刘美丽
年龄：19
性别：女
成绩：3.78
……

➢ 代码中的类和对象


类名
class High_school_student():
def __init__(self, age, sex):
参数
self.age = age
self.sex = sex
初始化变量
初始化方法
def missing_detecting(self):
if self.age == '':
print 'This is a missing value!'
else:
print 'This is not a missing value!'
函数体
类中的方法:
方法名
对象名
student1 = High_school_student('17','M')
student2 = High_school_student('','F')
对象的初始化
student1.missing_detecting()
student2.missing_detecting()
对象调用类中的方法
This is not a missing value!
This is a missing value!

7.4 类的创建

- 用 class 类名 即可以创建一个类
- 在类名的程序块中可以定义这个类的属性、方法等等

```
class High_school_student():
    def __init__(self):
        self.age = 18.0
        self.sex = 'M'
        self.gradyear = 2006
```

7.4 类的创建（续1）

➢ 创建实例

创建相应的实例

```
student_a = High_school_student()
print student_a
print High_school_student
```

查看实例属性

```
student_a.age
```

7.4 类的创建（续2）

➢ 初始化__init__方法

✓__init__是一种特殊的方法，使得实例一开始就拥有类模板所具有的属性

✓self参数是类中函数定义时必须使用的参数，并且永远是第一个参数，该参数表示创建的实例本身

```
class High_school_student():
    def __init__(self, age, sex):
        self.age = age
        self.sex = sex
```

```
class High_school_student():
    def __init__(self, age, sex):
        self.age = age
        self.sex = sex
```

➢ 正确地初始化对象法

一旦“__init__”存在，除了"self"之外的参数，那么在创建实例"student_a"时，我们就需要传入相应的参数值（不过"self"不需要传入数值，Python解释器会自动将实例变量传入）

```
student_a = High_school_student()
```

```
student_a = High_school_student(18.0, 'M')
print student_a.age
print student_a.sex
```

```
18.0
M
```

7.4 类的创建（续3）

- 在__init__方法中导入数据

✓在每一次创建实例的时候，自动导入数据，并且定义和处理与数据有关

```
import csv
class High_school_student():
    def __init__(self):
        f = open('teenager_sns.csv')
        csvreader = csv.reader(f)
        teenager_sns = list(csvreader)
        # 删除第一个元素，即变量名列表元素
        del teenager_sns[0]
        self.teenager_sns = teenager_sns
    def missing_detecting(self, age, sex):
        if age == '' or sex == 'NA':
            missing = True
        else:
            missing = False
        return missing
```

7.4 类的创建（续4）

➢ 成员的私有化

✓封装性：一个对象的成员属性要得到一定程度的保护。例如，要对一个对象的成员属性进行修改或访问，就必须通过对象允许保护对象，使程序不易出错。

```
class High_school_student():
    def __init__(self):
        self.age = 18.0
        self.sex = 'M'

student_a = High_school_student()
print student_a.age
student_a.age = 18.8
print student_a.age
```

High_school_student类没有进行有效的封装!

```
18.0
18.8
```

7.4 类的创建（续5）

- 成员的私有化

```
class High_school_student():
    def __init__(self):
        self.__age = 18.0
        self.__sex = 'M'

student_a = High_school_student()
student_a.__age
```

```
---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call last)

<ipython-input-44-5796a68eb152> in <module>()
      5
      6 student_a = High_school_student()
----> 7 student_a.__age


AttributeError: High_school_student instance has no attribute '__age'
```

7.4 类的创建（续6）

- 成员的私有化

方法的私有化：在

被内部访问，不能

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-50-39ff3ef81fac> in <module>()
     11
     12 student_a = High_school_student()
---> 13 student_a.__missing_detecting

AttributeError: High_school_student instance has no attribute '__missing_detecting'
```

```
class High_school_student():
    def __init__(self):
        self.age = 18.0
        self.sex = 'M'
    def __missing_detecting(self):
        if self.age == '' or self.sex == 'NA':
            missing = True
        else:
            missing = False
        return missing

student_a = High_school_student()
student_a.__missing_detecting
```

7.5 类的继承

- 类的继承好比孩子与父母之间的继承关系一样，孩子拥有父母所拥有的许多特性。在编程语言中，如果一个新的类继承另外一个类，那么这个新的类成为子类（Subclass），被子类继承的类称为父类。
- 我们可以把更加一般、范围大的类的属性在父类定义，把更加具体、范围小的特点在子类定义。

7.5 类的继承（续1）

- 定义子类

在定义好High_school_student后，可以定义两个子类Male_student与female_student继承它

```
class High_school_student():
    def __init__(self):
        f = open('teenager_sns.csv')
        csvreader = csv.reader(f)
        teenager_sns = list(csvreader)
        # 删除第一个元素，即变量名列表元素
        del teenager_sns[0]
        self.teenager_sns = teenager_sns

    def missing_detecting(self, age, sex):
        if age == '' or sex == 'NA':
            missing = True
        else:
            missing = False
        return missing
```

父类的定义

子类继承父类
没有定义专属于子类的属性/方法

```
class Male_student(High_school_student):
        pass

class Female_student(High_school_student):
        pass
```