

人工智能程序设计

M1 Python程序设计基础

5 面向对象程序设计

张 莉



面向对象

```
>>> x = len(lst)
>>> lst_new = sorted(lst)
```

```
>>> s = s.lower()
>>> lst.sort()
```

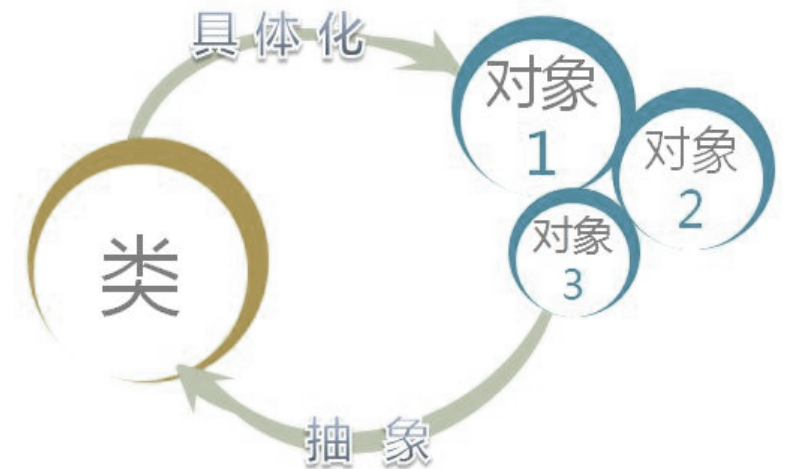
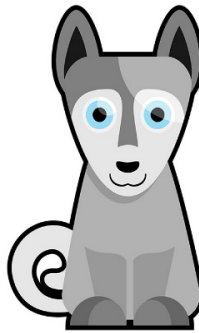


```
class Dog(object):  
    "define Dog class"  
    def __init__(self, name):  
        self.name = name  
    def greet(self):  
        print("Hi, I am {}".format(self.name))  
  
if __name__ == "__main__":  
    dog = Dog("Paul")  
    dog.greet()
```

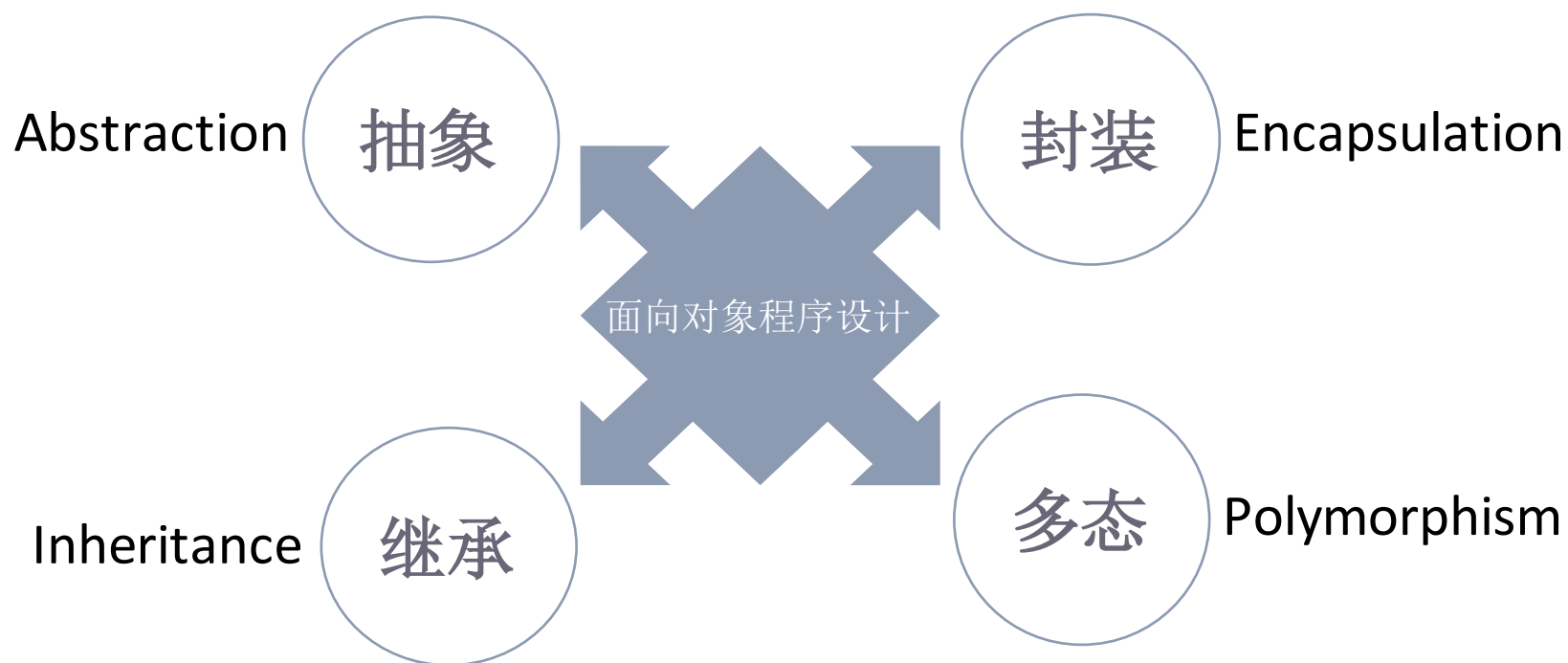
1 人工智能程序设计 面向对象程序设计基本概念

面向对象程序设计

- 对象（实例）
 - 由数据及能对其实施的操作所构成的封装体
- 类
 - 类描述了对应的特征（数据和操作）



面向对象程序设计（OOP）



面向对象程序设计的基本特征

- **抽象 (Abstraction) 与封装 (Encapsulation)**
 - **抽象**是指对现实世界问题和实体的本质表现；问题分解成数据和数据上的操作
 - **封装**是将程序具体的实现细节进行隐藏的一种机制
- **多态 (Polymorphism) 与绑定 (Binding)**
 - **多态**指一个事物有多种不同的解释，根据传递参数的不同执行不同的函数或操作不同的代码
 - **绑定**是指在具体某次使用多态元素时确定使用的是哪一种形式
- **继承 (Inheritance)**
 - 新创建的类的一些特征 (包括属性和方法) 可以从其他已有的类获得

2 人工智能程序设计 类与对象

类的定义和方法1



```
class Dog:
    '''define Dog class'''
    counter = 0
    def greet(self):
        print('Hi')
```



```
class MyDate:
    '''
    this is a very simple example class
    '''
    pass
```

类的定义和方法1



The diagram illustrates the components of a Python class definition. It shows the code `class ClassName:` followed by a docstring `"""类文档字符串"""` and a class body. Three numbered annotations are present: 1. A blue circle with the number 1 points to the class name `ClassName`. 2. A green circle with the number 2 points to the docstring `"""类文档字符串"""`. 3. A blue circle with the number 3 points to the class body. A speech bubble containing the word `File` is positioned above the class name.

```
class ClassName:  
    """类文档字符串"""  
    类体
```

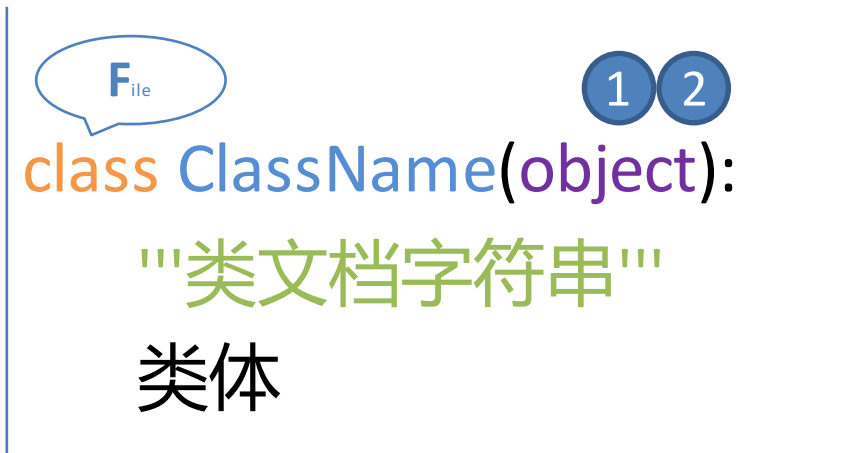
- ① **类名**, 类的名称
- ② **类文档字符串**, 提供查询时的帮助信息
- ③ **类体**, 定义一些类的属性和方法

类的定义与方法2



```
class Dog(object):  
    '''define Dog class'''  
    counter = 0  
    def greet(self):  
        print('Hi')
```

类的定义和方法2



The diagram illustrates the components of a Python class definition. A speech bubble labeled 'File' points to the 'class' keyword. Two numbered circles, '1' and '2', point to the 'ClassName' and 'object' parameters respectively. The class body is shown as a multi-line string followed by the class body text.

```
class ClassName(object):  
    """类文档字符串"""  
    类体
```

- ① **父类**, 可选, 指定从某个已定义的类继承
- ② **object**, 万类之源

实例

- 类实例化的形式

变量 = 类名(<参数>)

- 创建实例后，可以使用实例调用方法
- 类方法的第一个参数总是self，指向实例本身，Python自动将对象作为第一个参数传入方法中

File

Filename: object.py

```
class Dog(object):  
    "define Dog class"  
    def setName(self, name):  
        self.name = name  
    def greet(self):  
        print("Hi, I am %s." % self.name)  
if __name__ == "__main__":  
    dog = Dog()  
    dog.setName("Paul")  
    print(dog.name)  
    dog.greet()
```

__init__() 方法

- __init__()方法永远会在对象创建完成后被Python自动调用
- 在对象创建后被Python自动调用的第一个方法
- 和其他方法一样，实例对象本身会作为self参数传递



```
# Filename: init.py
class Dog(object):
    '''define Dog Class'''
    def __init__(self, name):
        self.name = name
    def greet(self):
        print("Hi, I am %s." % self.name)
if __name__ == "__main__":
    dog = Dog("Paul")
    dog.greet()
```

实例属性 (Instance Attributes)

- 实例属性创建时间: 定义类时或实例创建之后
- 所有实例属性保存在名为 `__dict__` 的内嵌属性里

```
self.name = name
```



```
>>> class Date:
    pass

>>> curDate = Date()
>>> curDate.month = 6
>>> curDate.day = 1
>>> curDate.__dict__
{'day': 1, 'month': 6}
```



Filename: classatr.py

```
class Dog(object):  
    "define Dog class"  
    counter = 0  
    def __init__(self, name):  
        self.name = name  
        Dog.counter += 1  
    def greet(self):  
        print("Hi, I am {:s}, my number is  
              {:d}".format(self.name, Dog.counter))  
if __name__ == '__main__':  
    dog1 = Dog("Zara")  
    dog1.greet()  
    dog2 = Dog("Paul")  
    dog2.greet()
```

类属性应用



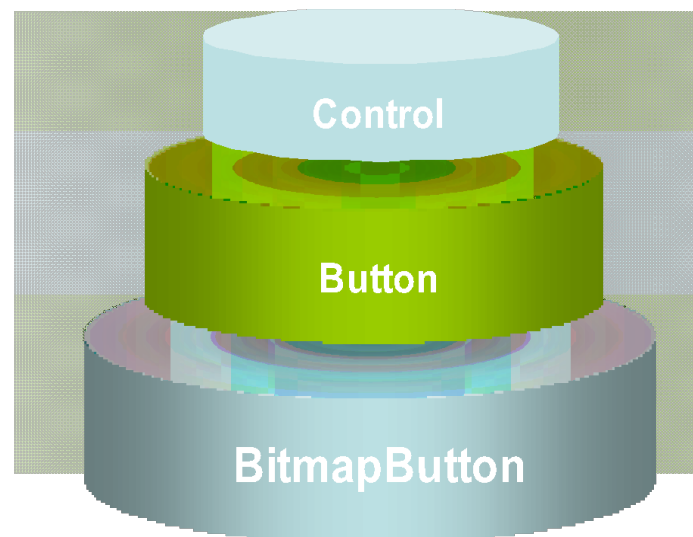
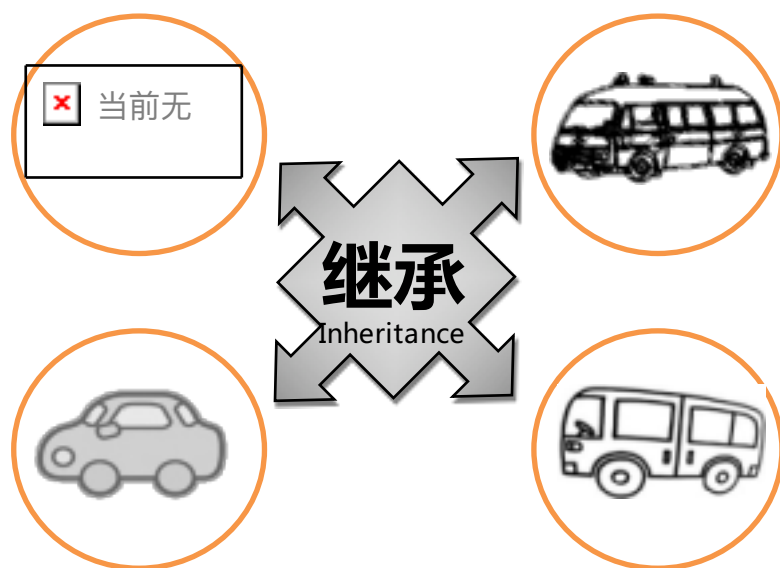
Hi, I am Zara, my number is 1
Hi, I am Paul, my number is 2

类属性 *counter* 用作追踪
已创建实例的计数器

3 人工智能程序设计

类的继承和方法重写

父类（基类）子类（派生类）



子类的定义



Filename: subclass.py

```
class Dog(object):
    "define Dog class"
    counter = 0
    def __init__(self, name):
        self.name = name
        Dog.counter += 1
    def greet(self):
        print("Hi, I am {s}, my number is
              {d}".format(self.name, Dog.counter))
```

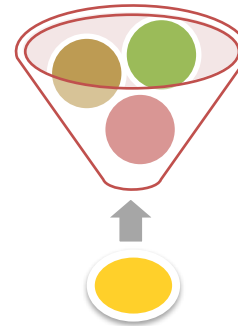
```
class BarkingDog(Dog):
    "define subclass BarkingDog"
    def bark(self):
        print("barking")

if __name__ == '__main__':
    dog = BarkingDog("Zoe")
    dog.greet()
    dog.bark()
```

子类的定义

```
class SubClassName (ParentClass1[, ParentClass2, ...]):  
    '''类文档字符串'''  
    类体
```

单
继
承



多
继
承

子类重写(overriding)1

File

Filename: override1.py

```
class Dog(object):
    "define Dog class"
    counter = 0
    def __init__(self, name):
        self.name = name
        Dog.counter += 1
    def greet(self):
        print("Hi, I am {s}, my number is
              {d}".format(self.name, Dog.counter))
```

```
class BarkingDog (Dog):
    "define subclass BarkingDog"
    def greet(self):
        "initial subclass"
        print("Woof! I am {s}, my number is
              {d}".format(self.name, Dog.counter))
    if __name__ == '__main__':
        dog = BarkingDog("Zoe")
        dog.greet()
```

子类重写2



Filename: override2.py

```
class Dog(object):  
    "define Dog class"  
    def __init__(self, name):  
        self.name = name  
    ...
```

```
class BarkingDog (Dog):  
    "define subclass BarkingDog"  
    def __init__(self, name):  
        self.name = "Little "+name
```

```
class BarkingDog (Dog):  
    "define subclass BarkingDog"  
    def __init__(self, name):  
        self.name = name  
        print("My name is", self.name)
```

```
class BarkingDog (Dog):  
    "define subclass BarkingDog"  
    def __init__(self, name):  
        super().__init__(name)  
        print("My name is", self.name)
```

super()方法

```
class p_father(object):  
    def pnt(self, a, b):  
        self.a = a  
        self.b = b  
        print(self.a+self.b)
```

```
class p_child(p_father):  
    def pnt(self, a, b):  
        super().pnt(a, b)  
        print(self.a*self.b)
```

```
calc = p_child()  
calc.pnt(3, 5)
```

duck typing

```
class Duck:
    def quack(self):
        print("Quaaaaaack!")
    def feathers(self):
        print("The duck has white and gray feathers.")
class Person:
    def quack(self):
        print("The person imitates a duck.")
    def feathers(self):
        print("The person takes a feather from the ground and shows it.")
    def name(self):
        print("John Smith")

def in_the_forest(duck): # 关键：参数duck因为动态语言可以不指定类型
    duck.quack()
    duck.feathers()

def game():
    donald = Duck()
    john = Person()
    in_the_forest(donald) # 只要具有相应行为的对象均可作为参数进行调用
    in_the_forest(john)
```

game()

访问控制



```
>>> class P:
    def __init__(self, name):
        self.__name = name
```

```
>>> x = P('John')
```

```
>>> x.__name
```

Traceback (most recent call last):

File "<pyshell#0>", line 1, in <module>

x.__name

AttributeError: 'P' object has no attribute '__name'

__var属性会被
__classname__var替
换，将防止父类与
子类中的同名冲突

类内可见

```
class A:
    def __init__(self, name):
        self.__name = name
    def f1(self):
        print(self.__name)
```

```
class B(A):
    def f2(self):
        self.__name = 'Liu'
        # 父类子类同名不冲突
        print("Hello, {} {}".format(self._A__name, self.__name))
```

```
class D:
    def f4(self):
        print(self._A__name)
```

```
x = A('xiaoming')
print(x.__name) # 出错
print(x._A__name) # 用真正运行时替换的写法可以正常输出
x.f1()
y = B('xiaohong')
y.f2()
# 类内可见因此会产生AttributeError异常
z = D()
z.f4()
```

补充：是否支持 传统函数重载 (overloading) 方法？

```
class printf(object):  
    def pnt(self, a, b):  
        self.a = a  
        self.b = b  
        print(self.a+self.b)
```

```
def pnt(self, a):  
    self.a = a  
    print(self.a)
```

```
e = printf()  
e.pnt(3, 5)  
e.pnt(8)
```

OO例1

身体质量指数（BMI，Body Mass Index）是国际上常用的衡量人体肥胖程度和是否健康的重要标准，计算公式为：BMI=体重/身高的平方（国际单位kg/m²）。

- (1) 定义BMI类，将身高体重作为__init__()方法的参数，在__init__()方法中计算BMI指数，并使用printBMI()方法输出BMI指数（保留一位小数），使用本人身高体重数据实例化。

OO例1

- (2) 在上题的基础上定义ChinaBMI子类，根据BMI指数的中国参考标准，重写printBMI()方法，在输出BMI指数（保留一位小数）后输出BMI分类和相关疾病发病的危险性信息，使用本人身高体重数据实例化。

BMI 分类	中国参考标准	相关疾病发病的危险性
偏瘦	<18.5	低（但其它疾病危险性增加）
正常	18.5~23.9	平均水平
偏胖	24~26.9	增加
肥胖	27~29.9	中度增加
重度肥胖	≥30	严重增加

M1.5小结

01 面向对象程序设计基本概念

02 类与对象

03 类的继承和方法重写

04 常用类和实例相关内建函数