人工智能实践教程

从Python入门到机器学习

第一部分Python核心编程技术

第 3 章 Python 高级编程

- *所有代码及**ppt**均可以由以下链接下载*
- ***https://github.com/shao1chuan/pythonbook***
- ***https://gitee.com/shao1chuan/pythonbook***

面向对象高级特性

目录

Contents

01 类属性与实例属性

类属性与实例属性

类属性就是类对象所拥有的属性,它被所有类对象的实例对象所共有,在内存中只存在一个副本。

在前面的例子中我们接触到的就是**实例属性**(对象属性)，它不被所有类对象的实例对象所共有,在内存中的副本个数取决于对象个数。

02 类方法与静态方法

类方法与静态方法

类方法是类对象所拥有的方法,需要用修饰器一般以@classmethod来标识其为类方法,

1). 对于类方法,第一个参数必须是类对象,作为第一个参数

(cls是形参， 可以修改为其它变量名,但最好用'cls'了)

2). 能够通过实例对象和类对象去访问。

类方法与静态方法

静态方法需要用修饰器一般以@staticmethod来标识其为静态方法,

1). 静态方法不需要多定义参数

2). 能够通过实例对象和类对象去访问。

类方法与静态方法

静态方法需要用修饰器一般以@staticmethod来标识其为静态方法,

1). 静态方法不需要多定义参数

2). 能够通过实例对象和类对象去访问。

04 property类属性

property类属性

1). Python内置的@property装饰器就是负责把一个方法变成属性调用的;

2). @property本身又创建了另一个装饰器@state.setter，负责把一个setter方法变成属性赋值，于是，我们就拥有一个可控的属性操作.

3). @property广泛应用在类的定义中，可以让调用者写出简短的代码，同时保证对参数进行必要的检查，这样，程序运行时就减少了出错的可能性。

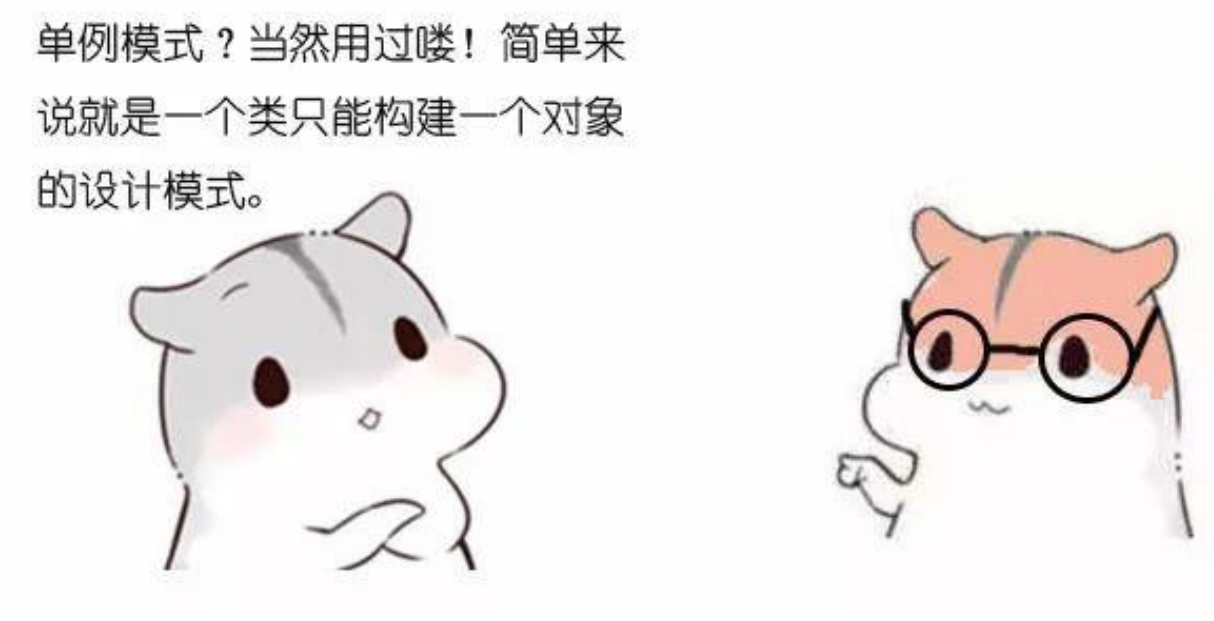
property类属性

```
# Read-only field accessors
@property
def year(self):
    """year (1-9999)"""
    return self._year

@property
def month(self):
    """month (1-12)"""
    return self._month

@property
def day(self):
    """day (1-31)"""
    return self._day
```

04 单例模式

单例模式

What?



单例模式

Why?

对于系统中的某些类来说，只有一个实例很重要，例如，一个系统中可以存在多个打印任务，但是只能有一个正在工作的任务；一个系统只能有一个窗口管理器或文件系统；一个系统只能有一个计时工具或ID(序号)生成器。如在Windows中就只能打开一个**任务管理器**。如果不使用机制对窗口对象进行唯一化，将弹出多个窗口，如果这些窗口显示的内容完全一致，则是重复对象，浪费内存资源；如果这些窗口显示的内容不一致，则意味着在某一瞬间系统有多个状态，与实际不符，也会给用户带来误解，不知道哪一个才是真实的状态。因此有时确保系统中某个对象的唯一性即一个类只能有一个实例非常重要

单例模式

How?

```
# 实例化一个单例
class Singleton(object):
    __instance = None
    def __new__(cls, age, name):
        #如果类数字能够__instance没有或者没有赋值
        #那么就创建一个对象,并且赋值为这个对象的引用,保证下次调用这个方法时
        #能够知道之前已经创建过对象了,这样就保证了只有1个对象
        if not cls.__instance:
            cls.__instance = object.__new__(cls)
        return cls.__instance
a = Singleton(18, "dongGe")
b = Singleton(8, "dongGe")
print(id(a))
print(id(b))
a.age = 19 #给a指向的对象添加一个属性
print(a.age)#获取a指向的对象的age属性
```

单例模式

How?

装饰器（decorator）可以动态地修改一个类或函数的功能。这里，我们也可以使用装饰器来装饰某个类，使其只能生成一个实例，代码如下：

```
from functools import wraps

def singleton(cls):
    instances = {}
    @wraps(cls)
    def getinstance(*args, **kw):
        if cls not in instances:
            instances[cls] = cls(*args, **kw)
        return instances[cls]
    return getinstance

@singleton
class MyClass(object):
    a = 1
```