人工智能程序设计

计算机学院

李晶晶

華南師範大學
South China Normal University

问题

- 判断密码强度
  - 判断标准（规则）：
    - ✓密码长度必须大于等于8个字符
    - ✓字符串中是否含有字母
    - ✓字符串中是否含有数字

问题

- 分析问题的计算部分：根据判断规则判断密码的强度。
- 确定功能：
  - 输入：密码字符串
  - 处理：逐条规则进行判断
  - 输出：密码强度判断结果
- 实现方法：
  - 定义一个password工具类，包括密码、密码强度等属性，包括密码强度判断方法以及其他相关辅助方法。

目录

- 面向对象编程
- 类与对象
- 迭代对象
- 继承
- 封装
- 多态

面向对象概述

- 对象的静态特征称为“属性”，对象的动态特征称为“方法”
- “对象=属性+方法”

面向对象概述

- 在程序中使用对象来映射现实中的事物，使用对象间的关系来描述事物之间的联系，这种思想就是面向对象。
- 面向过程（procedure-oriented）：是一种以事件为中心的编程。
- 面向对象（Object-Oriented）：一种以事物为中心的编程思想。

面向对象概述

- 分别使用面向过程和面向对象来实现五子棋

编程思想	实现步骤	特点
面向过程	✓ 开始游戏 ✓ 黑子先走 ✓ 绘制画面 ✓ 轮到白子 ✓ 绘制画面 ✓ 判断输赢 ✓ 返回步骤2 ✓ 输出最后结果	先分析解决问题的步骤 使用函数把这些步骤依次实现 使用时需要逐个调用函数
面向对象	✓ 黑白双方：这两方的行为一样 ✓ 棋盘系统：负责绘制画面 ✓ 规则系统：负责判断诸如犯规、输赢等。	把解决问题的事物分为多个对象 对象具备解决问题过程中的行为

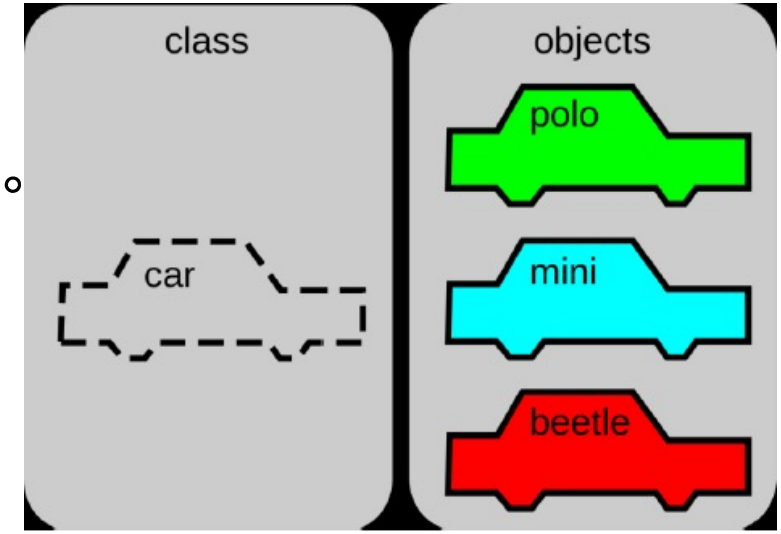
面向对象概述

- 若加入悔棋功能，面向过程和面向对象，分别怎么实现呢？

面向过程	面向对象
从输入到判断到显示的一系列步骤都需要改动	只需要改动棋盘对象就可以 更简便！

面向对象概述

- 类和对象的关系
  - 面向对象编程有两个非常重要的概念：类和对象。
  - 对象是面向对象编程的核心。
  - 具有相似特征和行为的事物的集合统称为类
  - 对象是根据类创建的，一个类可以对应多个对象。



面向对象概述

- Python中对象的概念很广泛，Python中的一切内容都可以称为对象，除了数字、字符串、列表、元组、字典、集合、range对象、zip对象等等，函数也是对象，类也是对象。
- 创建类时用变量形式表示的对象属性称为数据成员，用函数形式表示的对象行为称为成员方法，成员属性和成员方法统称为类的成员

创建类

- 类的定义

class 类名:

类的属性

类的方法

根据类创建对象的语法格式如下：

对象名 = 类名()

要想给对象添加属性，可以通过如下方式：

对象名.新的属性名 = 值

创建类

```
class Cat():
    """一次模拟猫咪的尝试"""
    name = 'maomi'
    age = 3
    def sleep(self):
        """猫咪睡觉"""
        print('%d岁的%s正在沙发上睡懒觉。 '%(self.age, self.name))
    def eat(self, food):
        """猫咪吃东西"""
        self.food = food
        print("%d岁的%s在吃%s" %(self.age, self.name, self.food))
```

绑定self

- 在方法的列表中，第1个参数永远都是self。
- self的字面意思是自己，我们可以把它当做C++里面的this指针理解，表示的是对象自身。
- 在类的实例方法中访问实例属性时需要以self为前缀
- 当某个对象调用方法的时候，Python解释器会把这个对象作为第1个参数自动传给self，开发者只需要传递后面的参数就可以了。
- 如果在外部通过类名调用对象方法则需要显式为self参数传值

绑定self

- self代表当前对象的地址，能避免非限定调用时找不到访问对象或变量。

```
#<例5.1:绑定self>
class Cat():
    def sleep(self):
        print(self)

new_cat = Cat()
print(new_cat.sleep())
```

绑定self

- self名称也不是必须的
- 在python中，self不是关键字

```
#<例5.2:self名称>
class Test:
    def prt(my_address):
        print(my_address)
        print(my_address.__class__)

t = Test()
t.prt()
```

绑定self

```
#<self作用实例程序>
class Dog:
    def setOwner(self,name):     #方法1：设置对象的主人姓名
        self.owner=name
    def getOwner(self):          #方法2：获取对象的主人姓名
        print('我的主人是：%s'%self.owner)
    def run(self):               #方法3
        pass

a = Dog()                        #创建了小狗对象a
a.setOwner("小红")               #小红认领了小狗a
b = Dog()                        #创建了小狗对象b
b.setOwner("兰兰")               #兰兰认领了小狗b
a.getOwner()    #输出：我的主人是：小红
b.getOwner()    #输出：我的主人是：兰兰
```

pass 是Python提供的一个关键字，它代表空语句，目的是是为了保持程序结构的完整性

类的专有方法

- 通常用双下划线“_”开头和结尾
- 访问类或者对象的属性和方法，通过点号操作，object.attribute
- 可以对属性的修改和增加

类的专有方法

类的专有方法	功能	类的专有方法	功能
__init__	构造函数，在生成对象时调用	__call__	函数调用
__del__	析构函数，释放对象时使用	__add__	加运算
__repr__	打印，转换	__sub__	减运算
__setitem__	按照索引赋值	__mul__	乘运算
__getitem__	按照索引获取值	__div__	除运算
__len__	获得长度	__mod__	求余运算
__cmp__	比较运算	__pow__	乘方

上机实践

- 1）使用class语句创建Car类并命名，添加车轮数和颜色两个属性
- 2）使用def函数定义getCarInfo函数，添加参数name，用print函数输出“name 有%d个车轮，颜色是%s”
- 3）使用def函数定义run函数，用print函数输出“车行驶在学习的大道上”。
- 4）调用Car类赋值于new_car
- 5）对new_car调用getCarInfo函数和run函数，打印调用后的结果。

创建对象

- 类的实例化，创建的对象称为实例对象
- Python类的构造函数
  - _init_()
  - 一般用来为数据成员设置初值或进行其他必要的初始化工作
  - 在创建对象时被自动调用和执行
- 如果用户没有设计构造函数，Python将提供一个默认的构造函数用来进行必要的初始化工作

创建对象

```
class Cat():
    """模拟猫咪的尝试"""
    #构造器方法
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def sleep(self):
        """猫咪睡觉"""
        print('%d岁的%s正在沙发上睡懒觉。'%(self.age, self.name))
    def eat(self, food):
        """猫咪吃东西"""
        self.food = food
        print("%d岁的%s在吃%s" %(self.age, self.name, self.food))
```

删除对象

- Python中类的析构函数
  - __del__()
  - 一般用来释放对象占用的资源
  - 在Python删除对象和收回对象空间时被自动调用和执行
- 如果用户没有编写析构函数，Python将提供一个默认的析构函数进行必要的清理工作。

删除对象

```
#<例5.3:析构函数>
class Animal():
    #构造方法
    def __init__(self):
        print('----构造方法被调用————————')
    #析构方法
    def __del__(self):
        print('----析构方法被调用————————')

cat = Animal()
print(cat)
del cat
print(cat)
```

对象的属性和方法

```
#<例5.4:对象的属性和方法>
class Cat():
    #构造器方法
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def sleep(self):
        print('%d岁的%s正在沙发上睡懒觉。
'%(self.age, self.name))
    def eat(self, food):
        self.food = food
        print("%d岁的%s在吃%s" %(self.age,
self.name, self.food))
cat1 = Cat('Tom', 3)
cat2 = Cat('Jack', 4)
print('Cat1的名字为：', cat1.name)
print('Cat2的名字为：', cat2.name)
print(cat1.sleep())
print(cat2.eat('fish'))
```

对象的方法

- 方法引用
  - 在创建具体实例对象时程序才会为对象的每个数据属性和方法分配内存
  - 对象名.成员

```
cat1 = Cat('Tom', 3)
cat2 = Cat('Jack', 4)
print('Cat1的名字为：', cat1.name)
print('Cat2的名字为：', cat2.name)
print(cat1.sleep())
print(cat2.eat('fish'))
```

```
cat1 = Cat('Tom', 3)
sleep = cat1.sleep
print(sleep())
cat2 = Cat('Jack', 4)
eat = cat2.eat
print(eat('fish'))
```

公有和私有

- Python并没有对私有成员提供严格的访问保护机制。
  - 定义类成员时，如果成员名以两个或多个下划线开头但不以两个或多个下划线结束则表示是私有成员
  - 私有成员在类的外部不能直接访问，需要通过调用对象的公开成员方法来访问，也可以通过Python支持的特殊方式来访问
- 公有成员既可以在类的内部进行访问，也可以在外部程序中使用。

公有和私有

- 公有属性：可以通过“对象名.成员”的方式来进行访问。

```
#<程序：公有属性name>
class Dog:
    name='大黄'  #属性
mydog=Dog()
print(mydog.name) #输出：大黄
```

公有和私有

- 私有属性：Python定义私有类型的方式很简单，只需在变量名前加上“__”(两个下划线)。

```
#<程序：私有属性>
class Dog:
    __name='大黄'  #私有属性
```

公有和私有

- 访问私有属性：在类的外部不能直接访问，那我们可以利用类内部的方法，通过“self.私有成员”进行访问。

```
#<程序：访问私有属性>
class Dog:
    __name='大黄'  #私有属性
    def getName(self): #方法：获取私有属性值
        return self.__name
mydog=Dog()
print(mydog.getName()) #输出：大黄
```

公有和私有

- 私有方法：私有方法与私有属性类似，它的名字也是以两个下划线开始的，可以在类的内部通过“self.私有成员” 来访问

公有和私有

```
#<例5.5：私有方法的使用>
class Dog:
    __age=1
    def getAge(self):
        return self.__age
    def __changeAge(self,age):#私有方法
        self.__age=age
    def needChange(self,age):
        if type(age)==int:
            self.__changeAge(age)
        else:print("年龄更改不合法")
mydog=Dog()
print(mydog.getAge()) #输出：小狗的年龄为 1
mydog.needChange("毛毛")  #输出：年龄更改不合法
print(mydog.getAge())  #输出：小狗的年龄为 1
```

上机实践

- 1）用Class语句创建Car类
- 2）将Car类实例化，添加newWheelNum和newColor两个属性
- 3）使用def函数定义run函数，用print函数输出“车在跑，目标：夏威夷”
- 4）用def__del__定义析构方法，用print函数输出“---析构方法被调用---”
- 5）调用Car类，创建对象并命名为BMW，并且可以赋值参数，比如BMW=Car（4，’green’)。
- 6）访问对象属性，调用run函数，并用print函数输出
- 7）用析构方法删除BMW，并查看对象是否被删除

迭代器

- 实现了__iter__()的对象是可迭代对象（iterable）
  - 使用内置函数iter(iterable)可以返回可迭代对象iterable的迭代器
- 实现了__next()__的对象是迭代器
  - 使用内置函数next()可以依次返回迭代器对象的下一个项目值，如果没有新项目，则将导致stopiteration
- 迭代器是一个可以记住遍历的位置的对象
- 迭代器只能往前，不会后退

迭代器

```
>>>L=[1,2,3]
>>>it = iter(L)
>>>it
<list_iterator at 0x107671470>
>>>next(it)
1
>>>next(it)
2
>>>next(it)
3
>>>next(it)
Traceback (most recent call last):
  File File "<ipython-input-16-2cdb14c0d4d6>", line 1, in
<module>
    next(it)
StopIteration
```

自定义可迭代对象和迭代器

- 迭代器对象必须实现两个方法：__iter__()和__next()__，二者合称为迭代器协议。__iter__()用于返回对象本身，以方便for语句进行迭代，__next()__用于返回下一元素
- 声明一个类，定义__iter__方法和__next()__。创建该类的对象，即是可迭代对象，也是迭代

自定义可迭代对象和迭代器

```
#<例5.6:定义类fib，迭代器>
class Fib:
    def __init__(self):
        self.a,self.b = 0,1 #前两项值
    def __next__(self):
        self.a,self.b = self.b,self.a+self.b
        return self.a    #f(n)=f(n-1)+f(n-2)
    def __iter__(self):
        return self
#测试代码
fibs = Fib()
for f in fibs:
    if f < 1000: print(f, end=',')
    else: break
```

生成器(Generator)

- 生成器函数使用yield语句返回一个值，然后保存当前函数整个执行状态，等待下一次调用
- 生成器函数是一个迭代器，是可迭代对象，支持迭代
- 调用时返回生成器对象

生成器

```
>>> def gentripls(n):
        for i in range(n):
            yield i*3
>>> f = gentripls(10)
>>> f
<generator object gentripls at 0x000001ED6C57DA20>
>>> i=iter(f)  #通过内置函数iter获得iterator
>>> next(i)   #通过内置函数next获得下一个项目：
0
>>> next(i)   #通过内置函数next获得下一个项目：
3
>>> for t in f: print(t, end=' ')
6 9 12 15 18 21 24 27
```

生成器

```
#<例5.7:生成器函数创建斐波那契数列>
def fib():
    a,b = 0,1     #前两项值
    while 1:
        a,b = b,a+b
        yield a  #f(n)=f(n-1)+f(n-2)
#测试代码
for f in fib():
    if f < 1000: print(f, end=',')
    else: break
```

生成器

- 生成器表达式
  - 类似于列表解析式，不过用圆括号括起来
  - 返回的是一个可迭代对象

```
gen = (i*2 for i in range(1, 10))
print(gen)
for e in gen:
    print(e, end=",")
```

面向对象三大特性

- 封装
  - 在面向对象的编程语言中“封装”就是将抽象得到的属性和行为相结合，形成一个有机的整体（即类）

面向对象三大特性

```
#<例5.8：Room类>
class Room: #类的设计者：定义一个房间的类
    def __init__(self,n,o,l,w,h):
        self.name = n
        self.owner = o
        self.length = l  #房间的长
        self.width = w   #房间的宽
        self.height = h  #房间的高
#类的使用者
r1 = Room("客厅","阿珍",20,30,9)
def area(room):
  print("{0}的{1}面积是{2}".format(room.owner,room.name,
     room.length*room.width))
area(r1)  #输出：阿珍的客厅面积是600
```

面向对象三大特性

- 封装
  - 封装可以简化编程，使用者不必了解具体的实现细节。
  - 封装带来的另一个好处是增强安全性。
  - 封装也提供了良好的可扩展性

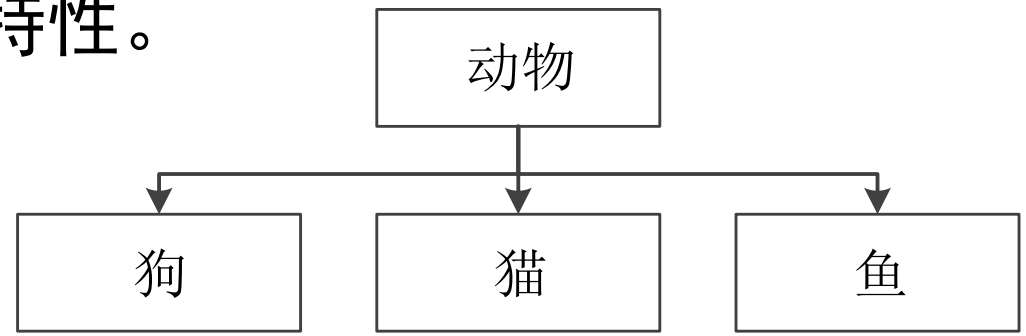
```
#<例5.9：对Room类的方法进行封装>
#类的设计者
class Room: #定义一个房间的类
    def __init__(self,n,o,l,w,h):
        self.name = n
        self.owner = o
        self.length = l  #房间的长
        self.width = w   #房间的宽
        self.height = h  #房间的高
    def area(self):          #求房间的平方的功能
        print("{0}的{1}面积
{2}".format(self.owner,self.name,
self.length*self.width))
#类的使用者
r1 = Room("客厅","阿珍",20,30,9)
r1.area()   #输出：阿珍的客厅面积是600
```

面向对象三大特性

```
#<例5.10：对Room类中的方法进行扩展>
#类的设计者，轻松的扩展了功能，而类的使用者不需要改变自己的代码
class Room: #定义一个房间的类
    def __init__(self,n,o,l,w,h):
        self.name = n
        self.owner = o
        self.__length = l #房间的长
        self.__width = w  #房间的宽
        self.__height = h #房间的高
    def area(self): #此时我们增加了求体积
        print("{0}的{1}面积是{2}".format(self.owner,self.name,self.__length*self.__width))
        print("体积是{0}".format(self.__length*self.__width*self.__height))
#类的使用者
r1 = Room("客厅","阿珍",20,30,9)
r1.area()#输出：阿珍的客厅面积是600 体积是5400
```

面向对象三大特性

- 继承
  - 被继承的类称为父类，继承者称为子类。
  - 子类是父类的特殊化，子类继承了父类的特性，同时可以对继承到的特性进行更改，也可以拥有父类没有的特性。



面向对象三大特性

- 继承
  - 若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法，这称之为方法重写

```
#<例5.11：继承示例>
class Animal:
    def run(self):
        print("run为父类的方法")
class Dog(Animal):
    def run(self):
        print("run为子类的方法")
    def eat(self):
        print("子类特有的方法")
class Cat(Animal):
    pass
#继承测试
cat=Cat()
cat.run()  #输出：run为父类的方法
dog=Dog()
dog.run()  #输出：run为子类的方法
dog.eat()  #输出：子类特有的方法
```

面向对象三大特性

```
#<例5.12：Animal类的继承问题演示>
import random as r
class Animal:
    def __init__(self):
        self.x=r.randint(0,10)
        self.y=r.randint(0,10)
    def run(self):
        self.x+=1
        print("我的位置是：",self.x,self.y)
```

```
class Cat(Animal):
    pass
class Dog(Animal):
    def __init__(self):
        self.hungry=True
    def eat(self):
        if self.hungry:
            print("我正在吃东西")
            self.hungry=False
        else:print("我不想吃东西")
```

```
#继承测试
cat=Cat()
cat.run();cat.run()
dog=Dog()
dog.eat();dog.run()
```

面向对象三大特性

- 解决方法
  - 使用super()并不需要传递参数，而用父类类名的方法需要在调用的函数中传入self参数。这是因为，super在调用的时候已经知道调用的对象是什么了，因此不需要再将self传入

```
class Dog(Animal):
    def __init__(self):
        Animal.__init__(self)
        self.hungry=True
```

```
class Dog(Animal):
    def __init__(self):
        super().__init__()
        self.hungry=True
```

面向对象三大特性

- 继承特点
  - 基类初始化__int__不会被自动调用。如果希望子类调用基类的__int__方法，需要在子类的__int__方法中显示调用它。
  - 在调用基类方法时，需要加上基类的类名前缀，且带上self参数变量，或者引入super()机制调用父类中的方法
  - Python总是首先查找对应类的方法，如果在子类没有对应的方法，才会在继承链的基类中按顺序查找
  - 子类不能访问基类的私有成员

面向对象三大特性

- 继承
  - 单继承
  - 多继承（多个父类）

    class 子类名（父类名1，父类名2，...）

```
#<例5.13:多继承>
class A():
    def __init__(self):
        print('  -->input A')
        print('  <--output A')
class B(A):
    def __init__(self):
        print('  -->input B')
        A.__init__(self)
        print('  <--output B')
class C(A):
    def __init__(self):
        print('  -->input C')
        A.__init__(self)
        print('  <--output C')
class D(B, C):
    def __init__(self):
        print('  -->input D')
        B.__init__(self)
        C.__init__(self)
        print('  <--output D')
```

面向对象三大特性

- 测试“是”关系
  - issubclass

    用于判断一个类是否是从另一个类继承而来的。

```
print(issubclass(B,D))
```

  - Isinstance

    用于判断一个对象是否与一个指定类型满足“是”关系

```
s = D()
print(isinstance(s,D))
```

面向对象三大特性

- 多态

```
#<例5.14:对象中方法的多态表现>
from random import choice
class Dog:
    def move(self):
        print("飞快地跑！")
class Cat:
    def move(self):
        print("慢悠悠地走。")
dog=Dog()
cat=Cat()
obj=choice([dog,cat]) #choice函数实现从列表中随机选择一个元素
print(type(obj))  #type函数可以查看对象类型
obj.move()#若obj为Cat，输出慢悠悠地走，若obj为Dog，输出飞快地跑！
```

面向对象三大特性

- 函数和运算符的多态表现

```
#<例5.15:“+”的多态表现>
a=1
b=2
print(a+b) #输出：3
a="Hello "
b="world!"
print(a+b) #输出：Hello world!
```

```
#<count函数多态>
from random import choice
x=choice(["Hello
world!",["o","a",1,2]])
print(x.count("o"))
```

面向对象三大特性

- 运算符的重载
  - 对于一个可以重载的运算符，object类都定义了一个内置方法，如__add__对应加法（+）运算符、__mul__对应乘法（*）运算符。通过重写这些方法，能够定义一个给定运算符如何处理自定义类对象。

面向对象三大特性

- 运算符的重载的限制
  - 不能改变运算符的优先级
  - 不能修改右结合性和左结合性
  - 一个运算符的操作数个数无法修改
  - 无法创新的运算符，即只能重载已有的运算符
  - 一个运算符如何处理内置类型对象，这一点无法更改
  - 运算符重载中，至少有一个操作数应该是自定义类的对象

面向对象三大特性

- Python和其他静态形态检查类的语言（如C++等）不同，在参数传递时不管参数是什么类型，都会按照原有的代码顺序执行，这就很可能会出现因传递的参数类型错误而导致程序崩溃的现象

代码演示

- 分析问题的计算部分：根据判断规则判断密码的强度。
- 确定功能：
  - 输入：密码字符串
  - 处理：逐条规则进行判断
  - 输出：密码强度判断结果
- 实现方法：
  - 定义一个password工具类，包括密码、密码强度等属性，包括密码强度判断方法以及其他相关辅助方法。

小结

- 类与对象
- 迭代对象
- 继承
- 封装
- 多态