浦东新区2023年Python教师培训项目

第4课-Python中的类与对象

课程概要

本节课主要内容继续学习面向对象的三大特征，类的继承，学习并使用turtle，掌握turtle常见函数。

知识点

面向对象中有两个概念非常重要，那就是类和对象。同时，这两个概念也非常抽象，为了便于理解，本节课将类类比成制造飞机的图纸、创造动物的图纸等等，将对象类比成了根据图纸创造出来的飞机、狗、老虎等。我们在一步步地编写Animal类、Tiger类、Dog类的代码和创建tiger对象、dog对象的同时，对类和对象理解也在逐渐地加深，这非常有利于之后更好地应用类和对象去进行程序设计。

继承也是本节课讲解的重点。作为面向对象的三大特征之一，继承能够解决特定问题，提高代码的复用性。

场景重现：

类和对象：类是具有相同特征的一类事物；这类事物的特征被称为属性，属性是静态的，一般由名词组成；这类事物的行为被称为方法，方法是动态的，一般由动词组成。对象是根据类创造出来的一个真正的事物，它具有类的属性和方法。

例如，如果我们想要造一架飞机，那么我们就需要先绘制出飞机图纸。在确定了飞机的功能：用于作战还是用于载客，还是用于货物运输之后，就要把这些体现在图纸上；另外图纸上也要包含飞机的机翼、引擎等信息。当图纸画好了之后，就可以根据它创造出一架和我们的设想一样的飞机。

在飞机的例子中，图纸可以看做类；图纸中包含的机翼、引擎等信息可以看做类的属性；图纸中体现的飞机的用途可以看做类的方法。由图纸造出的飞机可以看做对象。

```
class AirPlane():
    def __init__(self):
        self.wing = 2
        self.engine = 2
        self.wheel = 18
    def fly(self):
        print("我是一架飞机，我要准备起飞啦")
airOne = AirPlane()
airOne.fly()
```

技能小贴士：

在创建类和对象的时候，有以下注意点：

在创建类时，定义属性：

1、一般情况下，属性都要在__init__(self):内定义

2、__init__()前后都是两个英文下划线

3、属性前一定要添加 self.

在创建类时，定义方法：

1、 方法的第一个参数是 self

对象调用方法时：

1、可以没有参数

场景重现：

设计“动物图纸”：我们尝试了设计了一份动物图纸，并通过它创造了真实的动物--旺财；之后为了生成一个老虎对象，我们修改了这份动物图纸。不过，依据这样的操作，为了生成不同的动物对象，就要不断地修改动物图纸。这样太麻烦了。

为了解决上述问题，我们想到了一个办法：将动物对应的属性以参数的形式传入。也就是说，将 name、weight、height 这些属性以参数的形式存放在__init__()方法中，在方法内将这些值传递给 self.name、self.weight、self.height

```
class Animal():
    def __init__(self,name,weight,height):
        self.name = name
        self.weight = weight
        self.height = height
tiger = Animal("泰格",300,150)
```

技能小贴士：

__init__()方法称为构造方法，用来初始化新创建的对象的属性。这个方法在对象被创建时自动调用。

例如，在上述代码中，在创建了新的对象 tiger，并填入参数"泰格",300,150 之后，这些参数会自动传入__init__()方法中。__init__()方法就会对对象的属性进行初始化，之后，tiger 这个对象的属性就成为：self.name: "泰格"，self.weight:300，self.height:150

因为创建了对象后，程序会自动调用__init__()方法，所以，对象属性的定义放在__init__()方法中

场景重现：

显示“旺财”或“泰格”的体重、身高：为了实现打印出动物对象的名字、身高、体重的功能，我们在类中定义了一个show()函数。

为了在创建“泰格”或“旺财”后，就能显示一次它们的体重和身高信息； 每一次"泰格"和"旺财"进食或运动后，显示一次它们的体重和身高信息，我们采取了如下办法：在Animal类的内部，在定义的__init__()方法中、eat()方法中、run()方法中，都添加了一句self.show()语句。

```
class Animal:
    def __init__(self,name,weight,height):
    ......
    self.show()

    def eat(self):
    ......
    self.show()

    def run(self):
    ......
    self.show()

    def show(self):
    ......
```

技能小贴士：

上述代码，之所以可以在每一个方法中添加一个 self.show()，是因为 self 表示的就是创建的动物对象，self.show()表示的就是动物对象调用 show()方法。以 tiger 对象为例：当创建了 tiger 之后，self 代表的就是 tiger，self.show()代表的就是 tiger 对象可以调用 show()方法。

self 就是对象本身，通过打印 self 与对象，可以发现：这两者的打印结果是一致的，说明 self 就是创建出来的对象。
创建的是哪个对象，__init__()函数或其它带有 self 参数的函数内的 self 代表的就是哪个对象

场景重现：

继承：在实际情况中，狗和老虎的进食量肯定是不同的；而由于 tiger 对象和 wangcai 对象调用的是同一个类的 eat()方法，所以这两个对象的进食量是一样的。这不符合逻辑。所以，我们采用了两种方法去解决这个问题，其中一个是：各自定义动物的类。

在各自定义了 Tiger 与 Dog 类之后，我们发现，这两份代码除了 eat()方法的代码不同之外，其他的都相同，代码太冗余了。为了解决代码冗余问题，我们使用了继承：让 Tiger 类继承 Animal 类的属性和方法；但是原来的 Tiger 类与 Dog 类的 eat()方法一样的问题还是没有解决。为了解决它，我们在 Tiger 类中，又定义了新的 eat()方法；之后，tiger 对象调用就是 Tiger 类中的 eat()方法。

```
class Tiger(Animal):
    def eat(self):
        print("开吃啦 biajibiaji...")
        self.weight += 5
        self.height += 3
        self.show()

tiger.Animal("泰格",300,150)
tiger.eat()
tiger.run()
```

技能小贴士：

继承是面向对象的三大特征之一。关于继承，你需要注意这些：

1、Python 中的继承是类与类之间的继承

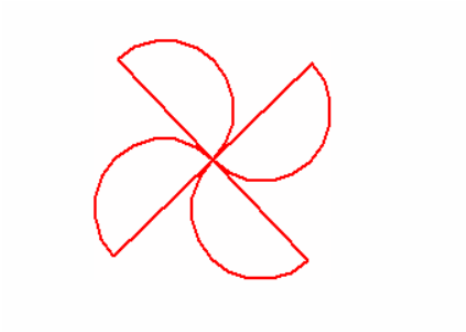
2、 继承后，子类拥有父类所有的属性和方法

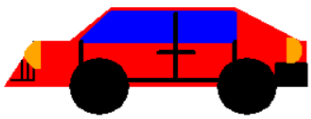
3、当子类继承父类，子类的对象调用方法时：如果子类没有实现该方法，会直接调用父类的方法；如果子类已经实现了该方法，会调用子类的方法

4、继承的优点是减少代码量；提高代码的复用性；只需要修改部分代码，就能实现区别性功能

turtle库

turtle的英文意思是“ 海龟 ”，今天学习的turtle库也叫作海龟绘图库，是Python语言中一个很流行的绘制图像的函数库。想象一下一只小海龟在海面上游泳，海龟游过的轨迹变成了一幅幅有趣的图案。





turtle常见函数

turtle.forward(distance) 向当前画笔方向移动distance像素长度

turtle.backward(distance) 向当前画笔相反方向移动distance像素长度

turtle.right(degree) 顺时针移动degree°

turtle.left(degree) 逆时针移动degree°

turtle.pendown() 移动时绘制图形，缺省时也为绘制

turtle.goto(x,y) 将画笔移动到坐标为x,y的位置

turtle.penup() 提起笔移动，不绘制图形，用于另起一个地方绘制

turtle.circle() 画圆，半径为正(负)，表示圆心在画笔的左边(右边)画圆

setx( ) 将当前x轴移动到指定位置

sety( ) 将当前y轴移动到指定位置

setheading(angle) 设置当前朝向为angle角度

home() 设置当前画笔位置为原点，朝向东。

dot(r) 绘制一个指定直径和颜色的圆点

turtle.fillcolor(colorstring) 绘制图形的填充颜色

turtle.color(color1, color2) 同时设置pencolor=color1, fillcolor=color2

turtle.filling() 返回当前是否在填充状态

turtle.begin_fill() 准备开始填充图形

turtle.end_fill() 填充完成

turtle.hideturtle() 隐藏画笔的turtle形状

turtle.showturtle() 显示画笔的turtle形状

turtle.clear() 清空turtle窗口，但是turtle的位置和状态不会改变

turtle.reset() 清空窗口，重置turtle状态为起始状态

turtle.undo() 撤销上一个turtle动作

turtle.isvisible() 返回当前turtle是否可见

stamp() 复制当前图形

turtle.write(s [,font=("font-name",font_size,"font_type")])

写文本，s为文本内容，font是字体的参数，分别为字体名称，大小和类型；font为可选项，font参数也是可选项

课堂案例

贪吃蛇

使用Python中的turtle模块完成贪吃蛇的案例。

```
import turtle
import time
import random

turtle.bgpic("背景.gif")
turtle.setup(700, 700)
turtle.tracer(False)

# 创建鸡蛋
turtle.register_shape("egg.gif")
egg = turtle.Turtle()
egg.shape("egg.gif")
egg.penup()
egg.goto(-200,-200)

# 创建小蛇
turtle.register_shape("head1.gif")
turtle.register_shape("head2.gif")
turtle.register_shape("head3.gif")
turtle.register_shape("head4.gif")
```

```
turtle.register_shape("body.gif")
snakeList = []
for i in range(5):
    t = turtle.Turtle()
    t.shape("turtle")
    t.setheading(180)
    t.penup()
    t.goto(i * 20, 0)  # 将小海龟一字排开
    if i == 0:
        t.shape("head1.gif")
    else:
        t.shape("body.gif")
    snakeList.append(t)

# 记录长度
word = turtle.Turtle()
word.penup()
word.goto(-270, 310)
word.color("white")
word.write("5", font=("微软雅黑", 20))
word.hideturtle()

# 刷新窗口
turtle.update()


# 蛇类
class Snake():
    def __init__(self, snakeList):
        self.snakeList = snakeList
        self.length = len(snakeList)

    # 向左移动
    def goLeft(self):
        # 获取列表中第一只小海龟的坐标
        x = self.snakeList[0].xcor()
        y = self.snakeList[0].ycor()
        # 删除列表中的最后一只小海龟
        t = self.snakeList.pop(-1)
        t.hideturtle()
        # 在列表的开头加入一只新的小海龟
        t1 = turtle.Turtle()
        t1.shape("head1.gif")
        self.snakeList.insert(0, t1)
        t1.penup()
        t1.goto(x - 20, y)
        # 将列表中第二只小海龟的形状设置为身体部分
        self.snakeList[1].shape("body.gif")

        turtle.update()

    # 向上移动
    def goUp(self):
```

```
        # 获取列表中第一只小海龟的坐标
        x = self.snakeList[0].xcor()
        y = self.snakeList[0].ycor()
        # 删除列表中的最后一只小海龟
        t = self.snakeList.pop(-1)
        t.hideturtle()
        # 在列表的开头加入一只新的小海龟
        t1 = turtle.Turtle()
        t1.shape("head2.gif")
        self.snakeList.insert(0, t1)
        t1.penup()
        t1.goto(x, y + 20)
        # 将列表中第二只小海龟的形状设置为身体部分
        self.snakeList[1].shape("body.gif")

        turtle.update()

    # 向右移动
    def goRight(self):
        # 获取列表中第一只小海龟的坐标
        x = self.snakeList[0].xcor()
        y = self.snakeList[0].ycor()
        # 删除列表中的最后一只小海龟
        t = self.snakeList.pop(-1)
        t.hideturtle()
        # 在列表的开头加入一只新的小海龟
        t1 = turtle.Turtle()
        t1.shape("head3.gif")
        self.snakeList.insert(0, t1)
        t1.penup()
        t1.goto(x + 20, y)
        # 将列表中第二只小海龟的形状设置为身体部分
        self.snakeList[1].shape("body.gif")

        turtle.update()

    # 向下移动
    def goDown(self):
        # 获取列表中第一只小海龟的坐标
        x = self.snakeList[0].xcor()
        y = self.snakeList[0].ycor()
        # 删除列表中的最后一只小海龟
        t = self.snakeList.pop(-1)
        t.hideturtle()
        # 在列表的开头加入一只新的小海龟
        t1 = turtle.Turtle()
        t1.shape("head4.gif")
        self.snakeList.insert(0, t1)
        t1.penup()
        t1.goto(x, y - 20)
        # 将列表中第二只小海龟的形状设置为身体部分
        self.snakeList[1].shape("body.gif")
        turtle.update()
```

```
    def eat(self):
        headX = self.snakeList[0].xcor()
        headY = self.snakeList[0].ycor()
        eggX = egg.xcor()
        eggY = egg.ycor()
        if headX == eggX and headY == eggY:
            eggX = random.randint(-17,17)*20
            eggY = random.randint(-17,17)*20
            egg.goto(eggX,eggY)
            t = turtle.Turtle()
            t.shape("body.gif")
            self.snakeList.append(t)
            self.length += 1
            word.clear()
            word.write(self.length, font=("微软雅黑", 20))
            turtle.update()


    def checkBound(self):
        headX = self.snakeList[0].xcor()
        headY = self.snakeList[0].ycor()
        if headX < -350 or headX > 350 or headY < -350 or headY >350:
            return True

    def checkBody(self):
        headX = self.snakeList[0].xcor()
        headY = self.snakeList[0].ycor()
        for i in range(1, len(self.snakeList)):
            if headX==self.snakeList[i].xcor() and headY==self.snakeList[i].ycor():
                return True

flag = "left"


def left():
    global flag
    if flag != "right":
        flag = "left"


def right():
    global flag
    if flag != "left":
        flag = "right"


def up():
    global flag
    if flag != "down":
        flag = "up"
```

```
def down():
    global flag
    if flag != "up":
        flag = "down"


turtle.listen()
turtle.onkeypress(left, "Left")
turtle.onkeypress(right, "Right")
turtle.onkeypress(up, "Up")
turtle.onkeypress(down, "Down")

s = Snake(snakeList)

while True:
    s.eat()
    if flag == "left":
        s.goLeft()
    elif flag == "right":
        s.goRight()
    elif flag == "up":
        s.goUp()
    elif flag == "down":
        s.goDown()
    time.sleep(0.1)

    if s.checkBound():
        break

    if s.checkBody():
        break

time.sleep(2)
turtle.bye()
```

相关学习网站

菜鸟教程：https://www.runoob.com/python3/python3-class.html

turtle：https://www.cnblogs.com/chen0307/articles/9645138.html