day4

一、今日内容

a. 面向过程编程与面向对象编程
b. 类和对象
c. 封装 -- 魔法方法
d. 继承
e. 多态
f. 设计模式 -- 单例模式

二、昨日复习

a. 列表的查找
    i. in : 判断 element是否在列表中
    ii. not in: 判断element 是否不在列表中
    iii. count : 统计列表中某个element出现的次数
    iv. index: 查找元素在列表中出现的位置
b. 列表排序
c. 数据类型 --- 元组
    i. 概念: 元组是一个容器 (int, list, tuple, string,bool),元组是不可以修改的,所以元组没有增删改
    ii. 访问
        1. 下标
        2. 切片
    iii. 查找
        1. in , not in index count
d. 数据类型 -- 字典
    i. 概念: 字典是一个容器. {key: value, key:value}
    ii. 本质: 红黑树
    iii. 字典的访问
        1. dict[key]: 当key值不存在的时候,会报错
        2. dict.get(key), 当key值不存在的时候,会返回值 None
    iv. 增删改查
        1. 增加: dict[new_key] = value
        2. 删除: del dict[key]
        3. 修改: dict[old_key] = value
        4. 清空: dict.clear()
        5. 查找: in not in
            a. len()
            b. keys()

c. values()

d. items()

e. 函数

i. 概念: 将专门的功能(一段代码)封装起来.本质: 方便代码复用

ii. 函数定义

```
def 函数名(形式参数列表):
    code
    return
函数调用:
函数名(实际参数列表)
形式参数是实际参数的一份临时拷贝
```

iii. 函数返回值: python中支持一个函数返回值多个值

iv. 局部变量与全局变量

1. 局部变量: 函数内部定义的变量
2. 全局变量: 函数外部定义的变量全局变量
3. global : 声明使用的变量是全局变量

v. 递归函数

1. 概念:函数调用函数本身
2. 结束条件: 每递归一次,都要向结束条件靠近
3. 时间复杂度: 函数复杂度 *递归的次数

三、面向对象编程

3.1 面向对象与面向过程

案例： 菜鸟买电脑

方法1：

1. 上网查找自己喜欢的电脑： Macpro 32G 1TB 1.8W
2. go to中关村，无法分辨真伪。 随便找一家购买
3. 导购推荐了一款电脑，也是Mac系统，同样的配置，只要1W
4. 砍价半小时，成交价： 9999
5. 回家一看，各种问题

方案2：

1. 表哥：电脑天才， 将买电脑这件事情，委托给表哥
2. 付款，等待收货

方案1：强调的是过程，所有的事情，都有自己完成。典型面向过程： C语言

方案2：将事情委托给擅长做这件事情的人来完成.强调的结果.典型面向对象语言: C++ python Java

3.2 面向对象相关概念

i. 类: 类型, 表示一类物体,是一个统称. 是一个泛指

ii. 对象: 类的具体实例化. 是一个特指.

3.3 类的组成

python类: 将属性与方法封装到一起.

类由三部分组成:

1. 类名 : 一般情况下,大驼峰命名
2. 属性 : 一般由一组数据组成
3. 方法 : 通常是有一组函数组成

3.4 植物大战僵尸

向日葵类:

i. 类名: sunflower

ii. 属性: 一组数据; hp 图片

iii. 方法: shake product

3.5 类的定义

```
# 新式类
"""
新式类格式:
class 类名(object):
```

```
    def 函数名(self):
        pass
    code
注:
    object: python当中,最顶级的父类.
    self: 对象本身,类似于C++ this指针.

"""
# 定义一个向日葵的类
class SunFlower(object):

    # 定义属性
    info = "this class is SunFlower!"

    # 定义方法
    def shake(self):

        print("向日葵左右摇晃!!")

    def product(self):

        print("向日葵产生一个阳光!!")
```

3.6 对象的定义

```
# 对象是类的具体实例化:   类是创建对象的一个模板
# 对象创建格式:  对象名 = 类名()
# 一个类可以创建无数个对象.
flower = SunFlower()
print("flower :", flower)
flower.product()
```

3.7 定义属性


```python
# 新式类
"""
新式类格式:
class 类名(object):

    def 函数名(self):
        pass
    code
注:
    object: python当中,最顶级的父类.
    self: 对象本身,类似于C++ this指针.

"""

# 定义一个向日葵的类
class SunFlower(object):

    # 定义属性
    info = "this class is SunFlower!"
    name = "向日葵"

    # 定义方法
    def shake(self):

        print("向日葵左右摇晃!!")

    def product(self):
        print("self : ", self)
        print("向日葵产生一个阳光!!")

```

```
31
32 # 对象是类的具体实例化:    类是创建对象的一个模板
33 # 对象创建格式:  对象名 = 类名()
34 # 一个类可以创建无数个对象.
35 flower = SunFlower()
36
37 # # 对象添加属性 通过 对象.属性 = value
38 # flower.hp = 100
39 #
40 # # 打印属性
41 # print("name is %s, hp is %d" % (flower.name, flower.hp))
42 # SunFlower.product(flower)
43
44 flower.product()
```

3.8魔法方法

i. __init__方法: 不是创建对象的方法,而是初始化对象的方法.这个函数无须手动调用,函数是自动调用的.

ii. __del__方法: 不是删除对的方法,而是在删除对象之前,会自动调用__del__函数, 回收资源.

iii. __str__ 方法: 返回一个字符串,当调用print进行打印对象的时候,会自动打印出 __str_的返回值

```
1  # 新式类
2  """
3  新式类格式:
4  class 类名(object):
5
6      def 函数名(self):
7          pass
8      code
9  注:
10     object: python当中,最顶级的父类.
11     self: 对象本身,类似于C++ this指针.
12
13 """
14
15 # 定义一个向日葵的类
16 class SunFlower(object):
17
18     # 定义属性
```

```
    info = "this class is SunFlower!"
    name = "向日葵"

    # 定义魔法方法 __init__
    def __init__(self, hp, mp): # 初始化对象
        print("---------init start --------------")
        self.hp = hp
        self.mp = mp
        # 产生了文件资源
        self.file = open("./hello.c", "w")
        print("---------init end------------------")

    # 定义方法
    def shake(self):

        print("向日葵左右摇晃!!")

    def product(self):
        print("self : ", self)
        print("向日葵产生一个阳光!!")

    # 定义__del__函数
    def __del__(self):
        print("--------------回收资源开始----------------")
        self.file.close()
        print("--------------回收资源完毕----------------")

    # 定义一个__str__方法
    def __str__(self):

        msg = self.name + "hp : " + str(self.hp) + "mp : " + str(self.mp)

        return msg


# 创建对象 # SunFlower() 创建对象 __init__会自动执行.
flower = SunFlower(100, 200)
print(flower)
# flower.name = "hahahah"
# # 修改类属性
# SunFlower.name = "my name is 向日葵"
# flower_v2 = SunFlower(10, 20)
# print("flower.name = %s, flower.hp = %d, flower.mp = %d" %(flower.name, flower.hp, flo
# print("flower_v2.name = %s, flower_v2.hp = %d, flower_v2.mp = %d" %(flower_v2.name, fl
```

```
63
64 # # 回收对象 del
65 # del flower
66 #
67 # # input 阻断程序结束
68 # input()
69
```

3.9 封装

封装: 将属性与方法封装到同一个类中.

__init__ __del__ __str__

3.10 课堂练习

```
1  class SweetPotato(object):
2      """这是烤地瓜的类"""
3
4      # 定义初始化方法
5      def __init__(self):
6          self.cookedLevel = 0
7          self.cookedString = "生的"
8          self.condiments = []
9
10     # 定制print时的显示内容
11     def __str__(self):
12         msg = self.cookedString + " 地瓜"
13         if len(self.condiments) > 0:
14             msg = msg + "("
15
16             for temp in self.condiments:
17                 msg = msg + temp + ", "
18             msg = msg.strip(", ")
19
20             msg = msg + ")"
21         return msg
22
23     # 烤地瓜方法
24     def cook(self, time):
25         self.cookedLevel += time
26         if self.cookedLevel > 8:
27             self.cookedString = "烤成灰了"
```

```
        elif self.cookedLevel > 5:
            self.cookedString = "烤好了"
        elif self.cookedLevel > 3:
            self.cookedString = "半生不熟"
        else:
            self.cookedString = "生的"

    # 添加配料
    def addCondiments(self, condiments):
        self.condiments.append(condiments
```

四、继承

4.1 概念

继承是类与类之间的关系。 本质：类级别的代码复用。

类的继承是指的： 子类继承父类的属性与方法。

Father: 父类　Children: 子类

```
class Father(object):

    # 属性
    money = 100

    # 定义了一个花钱的方法
    def cost(self, cost_money):

        self.money -= cost_money

    # 查看剩余余额
    def show(self):

        print("您的余额: ", self.money)


# 继承是指的子类继承父类的属性与方法
class Children(Father):

    pass

# 通过Children类创建一个对象
child = Children()
```

```
child.cost(1)
child.show()
```

4.2 继承特性

```
""" 如何成为世界首富

<1>. 有一位老师傅,在煎饼果子界摸爬滚打了很多年,总结了一套<古法煎饼果子配方>。
<2>. 老师傅年事已高,希望把自己的煎饼果子技术传承下去, 主角: 大猫 --> 拜干爹 学习煎饼果子技术 ==
<3>. 大猫特别爱学习,希望在课余的事件,学习现代煎饼果子配方技术, go to 新东方烹饪学校 === 多继
<4>. 大猫掌握了古法煎饼果子,也学会了现代煎饼果子,大猫自己创建了一个猫式煎饼果子配方 == 子类中重
<5>. 大猫的猫氏煎饼果子配方的技术很好,但是有的人也想吃现代煎饼果子,和古法煎饼果子。 == 子类调用
<6>. 随着时间的增加, 大猫终于成为了世界首付. money = 1000
<7>. 大猫也老去了,大猫也希望把自己的技术传承下去, kitty
"""


# 创建一个老师傅的类
class Master(object):

    # 定义属性
    def __init__(self):

        self.kongfu = "古法煎饼果子配方"

    # 制作一份煎饼果子
    def make_cake(self):

        print("Master : 根据<%s>,制作一份煎饼果子." % self.kongfu)

    # 喝酒
    def drink(self):

        print("喝酒,我只喝台子! ! ")


print("---------------master begin ------------------------")
master = Master()  # master老师傅的对象
master.make_cake()
```

```python
print("---------------master end ------------------------")


# 定义一个学校的类
class School(object):

    def __init__(self):

        self.kongfu = "现代煎饼果子配方"

    def make_cake(self):

        print("School : 根据<%s>,制作一份煎饼果子." % self.kongfu)

    def smoking(self):

        print("吸烟,我只抽华子,一次两盒！！")


# 定义大猫的类
class Student(School,Master ):  # 多继承,一个类继承多个父类,多个父类之间使用','隔开

    def __init__(self):

        self.kongfu = "猫氏煎饼果子配方"
        # 添加属性money

        # 私有属性:  只能够在类的内部访问，不能在类的外部访问
        # 定义私有属性:  在属性前 添加__
        self.__money = 1000

    def make_cake(self):

        print("Cat : 根据<%s>,制作一份煎饼果子." % self.kongfu)

    def make_old_cake(self):

        # 子类调用父类的同名方法: 父类名.方法名(self)
        Master.__init__(self) # 先获取到父类的同名属性 kongfu = "古法煎饼果子配方"
        Master.make_cake(self)
        self.__init__() # 将 self.kongfu = "猫氏煎饼果子配方"

    def make_new_cake(self):

```

```
        School.__init__(self) # 获取现代煎饼果子配方技术
        School.make_cake(self)
        self.__init__()

    # 想要修改私有属性,必须在类的内部修改.
    # 私有方法 :  通过在方面前面添加 __ 两个下划线
    def __show(self):
        print("self.__money : ", self.__money)


    def cost(self, money):

        self.__money -= money
        self.__show()

print("---------------cat begin ------------------------")
cat = Student()
cat.smoking()
cat.drink()

# __mro__属性:标识类调用函数的顺序
print("Student.__mro__ :", Student.__mro__)
"""
(<class '__main__.Student'>, <class '__main__.Master'>,
<class '__main__.School'>, <class 'object'>)
调用函数的时候,先去Student这个里面寻找,Student没有 -->Master ....

"""
cat.make_cake()
cat.make_old_cake()
cat.make_new_cake()
print("---------------cat end ------------------------")

# 定义一个 kitty的类
class Kitty(Student):

    pass

print("---------------kitty begin ------------------------")
kitty = Kitty()
kitty.smoking()
kitty.drink()
kitty.make_new_cake()
kitty.make_old_cake()
```

```
kitty.make_cake()

# 获取到大猫的钱
# print("kitty.money : ", kitty.__money)
# kitty.__money -= 100
# print("kitty.money : ", kitty.__money)

kitty.cost(100)

print("---------------kitty end ------------------------")
```

五、多态

5.1 概念

函数的多种形态: 同一个函数,传递的参数对象不同，最终导致函数的运行结果也不相同。 多态 ---> 类与类之间的关系

```
# 定义一个animal的类
class Animal(object):

    # 定义一个call方法
    def call(self):

        print("这是动物叫声的类")


# 定义一个Dog类
class Dog(Animal):

    # 从写同名方法 call
    def call(self):

        print("旺旺旺! ! ")

# 定义一个Cat类
class Cat(Animal):

    # 从写方法
    def call(self):

        print("喵喵喵")
```

```
25
26 dog = Dog()
27 cat = Cat()
28
29 # 传递参数
30 def func(obj):
31
32     obj.call()
33
34 func(dog)
35 func(cat)
```

5.2 鸭子模型

鸭子模型： 我想要一只鸭子，但是身边没有鸭子，我有一只鸡，这只鸡走路想鸭子，叫声也像鸭子。我就认为他是一只鸭子。

```
1 def add(x, y):
2
3     return x  + y # 只要 x 与 y 可以相加
4
5 add(10, 20)
6 add("hell0", "world")
7
8
```

六、人工智能环境安装

```
1 # 安装人工智能环境
2 """
3 升级 pip
4 i. window + r --> 输入 cmd + 回车
5 ii. python -m pip install --upgrade pip
6
7 安装sklearn模块 ：  机器学习算法模块
8 i. window + r --> 输入 cmd + 回车
9 ii. pip install sklearn==0.0
10
11 安装pandas模块: 读取文件的模块
12 i. window + r --> 输入 cmd + 回车
```

```
ii. pip install pandas

安装 numpy模块: 处理数据的模块
i. window + r --> 输入 cmd + 回车
ii. pip install numpy

安装分词模块jieba: 中文分词模块
i. window + r --> 输入 cmd + 回车
ii. pip install jieba

检测软件是否安装成功
<1>. 方案1
i. window + r --> 输入 cmd + 回车
ii. pip list
"""
```