**一、每创建一次对象就要做一次的事情统统放进\_\_init\_\_(self)函数里面**

class ICBC:

    """

        工商银行

    """

    # 类变量：总行的钱

    total\_money = 1000000

    # 类方法：对类变量的操作

    @classmethod

    def print\_total\_money(cls):

        # print("工商银行的钱:", ICBC.total\_money)

        print("工商银行的钱:", cls.total\_money)

**@classmethod**

**def print\_total\_money(cls):**

**print("工商银行的钱："，cls.money)**

    def \_\_init\_\_(self, name, money=0):

        # 实例变量：每个银行的信息

        self.name = name

        self.money = money

**ICBC.total\_money -= money#每建立对象就要进行的运算也适合放进\_\_init\_\_里面，直接用形参计算**

xd = ICBC("西单支行", 1000000)

trt = ICBC("陶然亭支行", 400000)

# print("总行的钱,还有:", ICBC.total\_money)  # 建议通过类访问

**ICBC.print\_total\_money()# print\_total\_money(ICBC)**

**二、数据存储方式的升级：**

**1、把嵌套字典变成变成由对象组成的列表，列表用来改动整体属性（顺序）**

**3、不用重复输入键了，输入更便捷，看上去也更简洁**

"""

    封装数据

"""

# 面向过程,没有对数据封装.

# dict\_commodity\_infos = {

#     1001: {"name": "屠龙刀", "price": 10000},

#     1002: {"name": "倚天剑", "price": 10000},

#     1003: {"name": "金箍棒", "price": 52100},

#     1004: {"name": "口罩", "price": 20},

#     1005: {"name": "酒精", "price": 30},

# }

# list\_commodity\_infos = list(dict\_commodity\_infos.items())

# for r in range(len(list\_commodity\_infos) - 1):

#     for c in range(r + 1, len(list\_commodity\_infos)):

#         if list\_commodity\_infos[r][1]["price"] > list\_commodity\_infos[c][1]["price"]:

#             list\_commodity\_infos[r], list\_commodity\_infos[c] = list\_commodity\_infos[c], list\_commodity\_infos[r]

# dict\_commodity\_infos = dict(list\_commodity\_infos)

# print(dict\_commodity\_infos)

# 优势1：将数据与对数据的操作相关联。

class CommodityInfo:

    """

        对商品信息的抽象

    """

    def \_\_init\_\_(self, cid, name="", price=0.0):

        self.cid = cid

        self.name = name

        self.price = price

    def print\_self(self):

        print("编号是%d,名称是%s,单价是%f" % (self.cid, self.name, self.price))

list\_commoditys = [

    CommodityInfo(1001, "屠龙刀", 10000),

    CommodityInfo(1002, "倚天剑", 30000),

    CommodityInfo(1003, "游戏机", 20000),

]

# 优势2：代码可读性更高（类是对象的模板）。

for r in range(len(list\_commoditys) - 1):

    for c in range(r + 1, len(list\_commoditys)):

        if list\_commoditys[r].price > list\_commoditys[c].price:

            list\_commoditys[r], list\_commoditys[c] = list\_commoditys[c], list\_commoditys[r]

for item in list\_commoditys:

    print(item.price)

**利用继承（保证大框架下所有人步调必须一致）和重写函数（相同函数名，不同执行代码）构建一个调用框架**

"""

    手雷爆炸了,伤害玩家生命(血量减少,闪现红屏),伤害敌人生命(血量减少,头顶爆字)

    要求：

        可能还增加其他事物，但是不能修改手雷代码.

            房子、鸭子...

"""

class Granade:

    """

        手雷

    """

    def explode(self, target):

        # 调用的是攻击目标

        # 传入的是一种攻击目标

**if isinstance(target, AttackTarget):**

**target.damage()**

class AttackTarget:

    """

        攻击目标

    """

    def damage(self):

        pass

# ----------------------------------

class Player(**AttackTarget**):

**def damage(self):**

        print("扣血")

        print("闪现红屏")

class Enemy(**AttackTarge**t):

**def damage(self):**

        print("扣血")

        print("头顶爆字")

g01 = Granade()

p01 = Player()

e01 = Enemy()

g01.explode(e01)