魔法方法(了解)

```
魔法方法:很魔幻
格式:__xxxx__(self)
xxxx有很多,如:
add 加法(两个对象相加时调用)
lt  小于(两个对象比较时调用)
str 字符串(打印对象时调用)
....

class Maker():
    def __add__(self,other):#self=m,other=m2
        print(self.age)
        print(other.kk)
        print("当2个对象相加时,我会被调用")
        return self.age+other.kk


m=Maker()
m.age=20
m2=Maker()
m2.kk=30
print(m+m2)#两个对象相加

class Maker():
    def __lt__(self,other):#self=m,other=m2
        print(self.age)
        print(other.kk)
        print("当2个对象小于比较时,我会被调用")
        return self.age+other.kk


m=Maker()
m.age=20
m2=Maker()
m2.kk=30
print(m<m2)#两个对象比较

class Maker():
    def __str__(self):
        print("当直接打印对象时,会调用我")
        return "aaa"


m=Maker()
print(m)

#案例:定义一个魔法方法,比较两个对象的大小
```

面向对象的三大特征介绍

```
1.封装,继承,多态
2.封装:把属性和方法封装在一起,并赋予权限
2.继承:子类继承父类,那么子类有父类的属性和方法
3.多态:同一操作作用于不同对象，可以有不同的解释，产生不同的执行结果

```

封装(重点)

```
1.封装的概念:把属性和方法封装在一起,并赋予权限
2.作用:保证内部的高内聚性和与外部的低耦合性
3.私有成员:
    1.外部不能直接访问,内部可以直接访问
    2.在属性或方法名前面加2个下划线就表示该成员是私有的

#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

#封装是把属性和方法写到类中,并赋予权限

class Maker():
    __age=10,#私有类属性

    def getAge(self):
        return self.__age

    def setAge(self,age):
        #判断修改的年龄是否合法
        if age>=1 and age<=140:
            self.__age=age

m=Maker()
# print(m.age)#当类属性为私有时,类外面不能直接操作类属性
# m.age=1000
print(m.getAge())
m.setAge(30)
print(m.getAge())




#案例:设计一个类,有私有对象属性name,age,设计类方法,有r权限就可以读取这两个私有属性,
有w权限就可以修改这两个属性
```

继承介绍(重点)

```
1.类与类之间才能继承
2.作用:简化代码,代码复用
3.名词解释:父类(基类,超类),子类(派生类)
4.子类的小括号中写了哪个类,就表示继承了谁(认谁做爹了)
5.子类拥有父类的所有属性和方法(私有的除外)
6.继承有单继承和多继承
7.object是所有类的父类，如果一个类没有显式指明它的父类，则默认为object

```

单继承(重点)

```
单继承,就是子类继承一个父类(儿子只有一个爹)
格式:
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

#父类
class Father():
    m="1千万",
    __mytype="小三",

    def mytest01(self):
        print("有钱")

    def mytest02(self):
        print("有房")

    def mytest03(self):
        print("有颜值")

#子类
class Son(Father):
    def myfunc(self):
        print(self.m)
        # print(self.__mytype)#报错,不能继承父类私有成员

s=Son()
s.mytest01()
s.mytest02()
s.mytest03()
print(s.m)
# print(s.__type)
```

```
s.myfunc()



#案例:定义一个动物类,属性有name,age.方法有会走move,会叫spake.定义一个猫和狗类,继承动物类
#通过猫对象调用move和spake方法,分别是age岁的name猫在走,age岁的name猫在叫,狗也一样
```

多继承

```
多继承,就是子类有多个父类(儿子有多个爹)
继承规则： Python允许多继承。调用顺序：从左到右，先深度再广度
不建议使用多继承,多继承会增加代码的复杂性
格式:
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
class Maker():
    BMW="奔驰"

class Father(Maker):
    name="Father",
    # BMW="宝马1",

    def mytest01(self):
        print("我是Father的mytest01")



class Father2():
    name2="Father2",
    BMW = "宝马",

    def mytest02(self):
        print("我是Father2的mytest02")

class Son(Father,Father2):
    pass

s=Son()
s.mytest01()
s.mytest02()
print(s.name)
print(s.name2)
print(s.BMW)#?这个宝马是哪个爹的?不清楚

```

继承构造函数和析构函数问题(重点)

```
1.子类不写构造函数,那么会默认调用从父类继承过来的构造函数
2.如果重写了__init__ 时，要继承父类的构造方法，可以在子类构造函数中使用 super 关键字或父类名
3.析构和构造一样

#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
class Father():
    def __init__(self,name):
        print("我是Father的构造函数")


class Son(Father):
    def __init__(self):
        # # super(Son,self).__init__("kk")#调用从父类继承的构造函数
        # Father.__init__(self,"kk")#调用从父类继承的构造函数
        print("我是Son的构造函数")


s=Son()
#注意,当子类没有写自己的构造函数,那么就要调用从父类继承过来的构造函数,要注意父类的构造函数是否有参数



#案例:测试一下析构函数是怎么样的?
```

子类调用父类同名方法

```
1.当子类的函数和父类的函数同名时,在子类的同名函数中使用super()关键字
格式:
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

class Father():
    def mytest(self):
        print("Father")

class Son(Father):
    def mytest(self):
        super().mytest()#调用父类的mytest函数
        print("Son")
```

```
19
20 s=Son()
21 #当父类和子类有同名函数时,子类对象,先调用自己的函数
22 s.mytest()
23
```

重写

```
1 1.什么叫重写,当子类定义了父类的同名函数时,就叫子类重写了父类的函数
2 2.有什么用:当从父类继承过来的方法功能不够用时,可以在子类中重写该方法
3
4 #!/usr/bin/env python
5 # -*- coding:utf-8 -*-
6 #====#====#====#====
7 #Author:
8 #CreatDate:
9 #Version:
10 #====#====#====#====
11
12 class Father():
13     def myadd(self,a,b):
14         return a+b
15
16
17 class Son(Father):
18     def myadd(self,a,b):
19         print("我是有新功能的")
20         return a+b
21
22 s=Son()
23 k=s.myadd(10,20)
24 print(k)
25
```

多态(重点)

```
1 1.多态:同一操作作用于不同对象，可以有不同的解释，产生不同的执行结果
2 2.鸭子类型:当看到一只鸟走起来像鸭子、游泳起来像鸭子、叫起来也像鸭子，那么这只鸟就可以被称为鸭子
3 3.多个类中的成员函数名字要一样
4
5 #!/usr/bin/env python
6 # -*- coding:utf-8 -*-
7 #====#====#====#====
8 #Author:
9 #CreatDate:
10 #Version:
11 #====#====#====#====
12
13 class Dog():
```

```
    def spake(self):
        print("狗在叫")

class Cat():
    def spake(self):
        print("猫在叫")

class tiger():
    def spake(self):
        print("老虎在叫")
class Maker():
    def spake(self):
        print("天在看,人在叫")
#同一个操作
def mytest(obj):
    obj.spake()

# 1.多态:同一操作作用于不同对象，可以有不同的解释，产生不同的执行结果
d=Dog()
c=Cat()
t=tiger()
m=Maker()

#作用于不同对
mytest(d)
mytest(c)
mytest(t)
mytest(m)

#不动前面的代码的基础上,增加新的功能



#案例:写一个操作(函数),传入三个不同的对象,打印不同的颜色

```

错误和异常

```
错误：通常是指代码中的语法错误，一般初级程序员很常见或者说很容易犯。
异常:即便Python程序的语法是正确的，在运行它的时候，也有可能发生错误。运行期间检测到的错误被称为异常
pycharm中错误常有箭头指示出来,异常没有
```

异常的捕获

```
1.捕获异常的格式:
try:#有可能出现异常的代码写在try里
    print('1111111')
    # print(a)
    print(1 / 0)
    # print(1 + 'a')
```

```
    print('2222222')
except Exception as e:#捕获到了异常,执行except里的代码
    print(e)
    print("I'm except")
说明:Exception常规错误的基类,把异常的基本信息存储到e中

2.异常处理-except分支else:没有异常执行else下面的代码

3.finally语句:有没有异常都要执行这下面的代码
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
def mytest(a,b):
    return a/b

try:
    mytest(10,0)
except Exception as e:
    print(e)
    # raise #抛出异常
else:
    print("没有异常就执行这")
finally:
    print("有没有异常都要执行这里的代码")




4.抛出异常:
    1.raise关键字

5.异常要处理,不管是谁
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
def mytest(a,b):
    return a/b
def myfunc():
    try:
        mytest(10,0)
    except Exception as e:
        # print(e)
        raise #抛出异常
    else:
```

```
        print("没有异常就执行这")
    finally:
        print("有没有异常都要执行这里的代码")

#这里处理myfunc函数抛出的异常
try:
    myfunc()
except Exception as ee:
    print(ee)

6.自定义异常:
    1.要继承Exception常规错误的基类

#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====

class ShowInputOut(Exception):
    def __init__(self,len,flag):
        super().__init__()
        self.len=len
        self.flag=flag




def main():
    try:
        s=input("请输入-->")
        if len(s)<3:
            raise ShowInputOut(len(s),3)
    except ShowInputOut as e:
        print("你输入长度为%d,长度必须为%d"%(e.len,e.flag))
    else:
        print("没有发生异常")

main()
```

OS模块(重点)

```
import sys
print('===Python import mode===')
print ('命令行参数为:')
for i in sys.argv:
    print (i)
print ('\n python 路径为',sys.path)
```

```

from sys import argv,path  #  导入特定的成员
print('===python from import===')
print('path:',path) # 因为已经导入path成员，所以此处引用时不需要加sys.path

import os
    #   path = os.getcwd()#获取当前执行文件的绝对路径
    #   path = os.path.abspath('.')#获取当前路径
    #     # path = os.path.abspath('..')#获取当前的父路径
#!/usr/bin/env python
# -*- coding:utf-8 -*-
#====#====#====#====
#Author:
#CreatDate:
#Version:
#====#====#====#====
import os
print(os.getcwd())#E:\cn202305\10day

print(os.path.abspath(".."))#E:\cn202305
#获取当前路径下所有文件名
mylist=os.listdir(os.getcwd())
print(mylist)
#修改文件名
os.rename(os.getcwd()+'/1.txt',os.getcwd()+'/2.txt')

案例:批量修改文件名
import os
funFlag = 1 # 1表示添加标志,2表示删除标志
folderName = './renameDir/'
#获取指定路径下所有文件的名字
dirList = os.listdir(folderName)
#遍历输出所有文件的名字
for name in dirList:
    print(name)
    if funFlag==1:
        newName='[山哥出品]-'+name
    elif funFlag==2:
        num=len('[山哥出品]-')
        newName=name[num:]
    print(newName)
    os.rename(folderName+name,folderName+newName)
```

;