BDA_Ch2

2019 年 10 月 10 日

In [ ]:
```
print('hello world')
```

In [ ]:
```
word_count = 711
if word_count > 800:
    print("字数达到要求")
else:
    print("字数不达标")
```

In [ ]:
```
# 封装
class Student:
    name = '' #name 属性是公开的
    __score = -1 #score 属性前面有 __ 符号，表示是私有属性

    # 初始化方法，在类实例化的时候会首先调用这个方法
    def __init__(self, name, score):
        self.name = name
        self.__score = score

    # 由于 score 是私有属性，因此需要通过类内部的方法来进行访问
    def get_score(self):
        return self.__score

    def is_qualified(self):
        if self.__score > 90:
            print('优秀')
        else:
            print('继续努力')

s1 = Student('张三', 85)
s1.is_qualified()
print('学生姓名为：' + s1.name)#name 属性是公开的，所以可以直接访问
s1.name = '张麻子'
print('学生姓名改为：' + s1.name)# 甚至可以直接改名字
print('学生成绩为：' + str(s1.get_score()))#score 属性是私有的，可以通过类内部的方法来调用访问
print('学生成绩为：' + str(s1.__score))# 因为无法访问 score 属性，所以会报错
# 但是在 Python 中并没有办法真正限制，如下方式就可以直接调用（格式为：对象._ 类 __ 属性名）
print('学生成绩为：' + str(s1._Student__score))
s1._Student__score = 90# 修改值也没问题
print('学生成绩为：' + str(s1._Student__score))
```

In [ ]:
```python
# 继承
class Student:
    _name = ''#name 前面加 _，类和子类都可以调用

    def __init__(self, name):
        self._name = name

    def workday_act(self):
        print(self._name + '工作日上课')

    def weekend_act(self):
        print(self._name + '周末休息')


class UGStudent(Student):
    # 重写父类的方法
    def weekend_act(self):
        print(self._name + '周末出去玩')


class PhD(Student):
    # 重写父类的方法
    def weekend_act(self):
        print(self._name + '作为博士生，周末看文献、跑数据当作休息')


ug1 = UGStudent('张三')
ug1.workday_act()
ug1.weekend_act()
phd1 = PhD('李四')
phd1.workday_act()
phd1.weekend_act()
```

In [ ]:
```python
# 多态
class PhD:
    _name = ''
    def __init__(self, name):
        self._name = name

    def research(self):
        pass

    @staticmethod # 静态方法，需要通过类名来调用这个方法
    def phd_research(obj):
        obj.research()


class MathPhD(PhD):
```

```
        def research(self):
            print(self._name + '正在推公式')


    class ChemicalPhD(PhD):
        def research(self):
            print(self._name + '正在刷试管')


    class ManagementPhD(PhD):
        def research(self):
            print(self._name + '正在编故事')


    phd1 = MathPhD('李四')
    phd2 = ChemicalPhD('王五')
    phd3 = ManagementPhD('小六')

    PhD.phd_research(phd1)
    PhD.phd_research(phd2)
    PhD.phd_research(phd3)
```

In [ ]:
```
x = 1;print(x)# 两个逻辑行在同一个物理行中，第二个逻辑行后可以不标注分号
x = 'That\'s great'
print(x)
x = r'That\'s great'
print(x)
x = '{0} is good'.format(10)
print(x)
```

In [ ]:
```
print(10==10==10)
print((10==10)==10)
```

In [ ]:
```
def add(a,b):
    c = a + b
    return c

# 调用函数 add
print(add(3,4))
```

In [ ]:
```
x = 50# 全局变量 x

def func(x):
    print('未定义局部变量前 x={0}'.format(x))
    x = 2# 定义了一个局部变量 x
    print('x={0}'.format(x))
    x = 30
    print('更改局部变量 x={0}'.format(x))
```

```
    func(x)
    print('全局 x={0}'.format(x))
```

In [ ]:
```
# 列表
list1 = ['physics', 'chemistry', 1997, 2000]
print(list1)
print(list1[1])# 序号为 1 的元素
print(list1.index(1997))# 找出列表中第一个匹配的位置，不存在就会报错
print(1998 in list1)# 判断 1998 是否在列表中
print(1998 not in list1)# 判断 1998 是否不在列表中
print(list1[1:3])# 序号从 1 开始到 3 为止（不包括 3）的元素
print(list1[-1])# 倒数第一个元素
print(list1[-2])# 倒数第二个元素，以此类推
print(len(list1))#list1 的长度，即元素数量
list1.append('math')# 列表末尾插入'math'
print(list1)
list1.insert(1,2000)# 在位置 1 处插入 2000
print(list1)
list1.pop()# 删除最后一个元素
print(list1)
list1.pop(0)# 删除位置为 0 的元素
print(list1)
# 按顺序遍历整个列表
for x in list1:
    print(x)
# 第二种遍历方法，相对来说比较低效
for i in range(len(list1)):
    print(list1[i])
```