

## BDA\_Ch2

2019 年 10 月 17 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

            def __init__(self, name, score):
                self.name = name
                self.__score = score

            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 [ ]: # 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 [ ]: # 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])

```

```

In [ ]: # 字典
dict1 = {'a':1,'b':2,'c':3}
print(dict1)
print(dict1['a'])# 访问 key 为 'a' 的元素
print('e' in dict1)# 判断 dict1 是否存在键 'e'
dict1['a'] = 'string'# 修改 key 为 'a' 的元素的值
print(dict1['a'])
dict1['d'] = [10,'hello']# 插入一个元素
print(dict1)# 输出结果可以看到, 字典的元素排列是无序的
print(dict1['d'][1])
del dict1['b']# 删除一个元素
print(dict1)
for k,v in dict1.items():
    print('key:{0},value:{1}'.format(k,v))
for k in dict1:
    print('key:{0},value:{1}'.format(k,dict1[k]))

```

```

In [ ]: # 系统输入输出
print(input())
str = input('Please enter a number:')
print('You entered: ' + str)
if int(str) > 0:
    print('大于 0')

```

```

In [ ]: # 文件输入输出
f = open('file_in.txt',encoding='utf-8')# 因为文件中包含中文, 所以需要指定编码 utf-8
f_content = f.read()# 一次性读入所有的内容
print(f_content)
print('-----')
f.close()# 关闭文件, 释放占用
# 一行一行读入
f = open('file_in.txt',encoding='utf-8')# 再次读入文件
while True:
    line = f.readline()
    if not line:
        break
    print(line)
f.close()# 关闭文件, 释放占用

# 输出文件
log_f = open('file_out.log','w',encoding='utf-8')
log_f.write(f_content)

```

```
log_f.close()
```

```
In [ ]: # 异常处理
```

```
try:
    f = open('no_file')
except SyntaxError:
    # 语法异常则...
    print('syntax error')
except IOError:
    # 如果发生文件输入输出异常, 则...
    print('no such file')
finally:
    # 不管异常发生与否, 最终执行...
    print('end')
print('go on')# 继续执行后需命令
# 注意: except 类型可以不指定, finally 语句也不是必须存在的
```

```
In [15]: # 正则表达式
```

```
import re # 导入正则匹配库
```

```
str = '''初步核算, 全年国内生产总值 900309 亿元, 比上年增长 6.6%。其中, 第一产业增加值 64734 亿元, 增长 3.5%
; 第二产业增加值 366001 亿元, 增长 5.8%; 第三产业增加值 469575 亿元, 增长 7.6%
。第一产业增加值占国内生产总值的比重为 7.2%, 第二产业增加值比重为 40.7%
, 第三产业增加值比重为 52.2%。'''
```

```
p1 = re.compile('增加值 (\d+) 亿元.*? 增长 (\d+.\d+)%')
result1 = re.findall(p1, str)
p2 = re.compile('比重为 (\d+.\d+)%')
result2 = re.findall(p2, str)
for i in range(len(result1)):
    print('{0}\t{1}%\t{2}%'.format(result1[i][0], result1[i][1], result2[i]))
```

64734	3.5%	7.2%
366001	5.8%	40.7%
469575	7.6%	52.2%

```
In [38]: # 正则表达式
```

```
import re
```

```
# 贪婪匹配和非贪婪匹配
```

```
# 找出字符串中介于‘总票房’和‘亿元’之间的内容
```

```
str = '哪吒总票房 47 亿元; 战狼总票房 57 亿元。'
```

```
p1 = re.compile('总票房 (.*) 亿元')# 贪婪匹配,* 匹配的字符越多越好
```

```
print(p1.findall(str))
```

```
p2 = re.compile('总票房 (.*)? 亿元')# 加上问号, 表示非贪婪匹配, * 匹配的字符越少越好 (次数则不限)
```

```
print(p2.findall(str))
```

```
print(p2.search(str))#search() 返回第一次成功匹配的位置
```

```
print(p2.search(str).span())# 再用 span() 提取具体位置, span 返回一个 tuple
```

```
print(p2.match(str))
```

```
#match() 尝试从字符串的起始位置匹配一个模式, 如果不是起始位置匹配成功的话, match() 就返回 none
```

```
print(re.match('哪吒', str))#match() 的另外一种使用形式
```

```
# 逻辑或
```

```

# 找出所有姓陈或者李的姓名
str = '''陈思敏
      陈泽权
      陈梓奕
      程青竹
      凡东华
      方丹峰
      高思怡
      韩倩
      黄铄源
      姜永超
      蒋伟枫
      赖劲圯
      李梦杰
      李竹馨
      林菲菲
      林敏宏
      '''

p1 = re.compile('(陈.{1,2}| 李.{1,2})')
# 匹配陈或者李开头、后面跟着 1-2 个任意字符的字符串，注意：换行符除外
print(p1.findall(str))

```

```
['47 亿元；战狼总票房 57']
```

```
['47', '57']
```

```
<re.Match object; span=(2, 9), match='总票房 47 亿元'>
```

```
(2, 9)
```

```
None
```

```
<re.Match object; span=(0, 2), match='哪吒'>
```

```
['陈思敏', '陈泽权', '陈梓奕', '李梦杰', '李竹馨']
```