

# Python学习笔记1

---

## 一、基础知识

---

### 1. 注释

注释：对代码进行解释和说明

#### 单行注释

```
# 这是一个注释
```

其他用途

```
# -*-coding:utf-8_*_      # 表示指定文件编码格式  
#!/python(python解释器的路径)  # 表示操作系统直接执行文件，选择的解释器
```

#### 多行注释

```
"""  
这是  
多行  
注释  
"""  
  
'''  
这是  
多行  
注释  
'''
```

### 2. 变量

变量：可以发生改变的一个量。变量是用来区分不同数据的，可以指向一个内存空间，帮我们存储一些数据

```
a = 1000  
print(a)    # 这里是1000  
a = 104  
print(a)    # 这里是104  
# 发生了变化
```

## 变量命名规范：

- 必须是数字、字母或下划线组成
- 不能是数字开头，更不能是纯数字
- 不能用 python 的关键字
- 不要用中文
- 不要太长
- 要有意义
- 推荐使用下划线命名或者驼峰命名

```
a = 1
a1 = 1
a_1 = 1
_ = 1
```

## 3. 数据类型\*

数据类型：区分不同的数据。不同的数据类型应该有不同的操作

- 数字：+\*/
  - 整数：int
  - 小数：float
- 文字：展示
- 布尔：条件判断

### int 整数

```
a = 10
b = 20
print(a + b)    # 加法运算
```

### float 小数

```
a = 1.112
b = 0.21
print(a + b)
```

### str 文字(字符串)

展示方式：

- 单引号：''
- 双引号：""
- 三单引号：''' '''
- 三双引号：""" """

操作：

- `+`: 左右两端**必须**是字符串
- `*`: 一个字符串**只能**乘以一个数字, 表示字符串**重复的次数**

```
a = "hello"
name = "李华"
print(a + "," + name)    # 加法
print(a * 5)             # 乘法
```

## bool 布尔条件判断

布尔值主要有两个 `true` 和 `false`, 表示真和假

- `true` 真命题
- `false` 假命题

```
print(100 > 30) # true
print(100 < 10) # false
```

## 转义字符

`\n`: 换行

`\t`: 制表位, 相当于一个 `tab` 键

## print 结束符

例:

```
print("hello",end = "\n")    # 默认

print("hello",end = "\t")
print("你好")
# hello      你好

print("hello",end = "...")
print("你好")
# hello...你好
```

## 4. 用户交互

### 设计一个简单计算器

变量 = `input(提示语)`

```
num1 = input("请输入第一个数字: ")
num2 = input("请输入第二个数字: ")
print(int(num1) + int(num2))
```

**注意：** `input` 得到的结果**一定是字符串**

可以使用以下方式查看变量的数据类型：

```
num1 = input("请输入第一个数字：")  
print(type(num1))
```

## 5. `if` 条件判断

`if` 语句的语法规则：

### 第一种

```
if 条件：  
    代码
```

- 如果条件成立就执行里面的代码
- 如果不成立，就不执行

例：

```
money = 500 # 修改此处数字，进行测试  
if money > 300:  
    print("可以去吃大餐")  
print("钱不够，吃不了大餐")
```

### 第二种

```
if 条件：  
    代码1  
else:  
    代码2
```

如果条件成立，执行代码1；不成立，执行代码2

例：

```
money = int(input("输入你的余额："))  
if money > 300:  
    print("可以去吃大餐")  
else:  
    print("钱不够，吃不了大餐")
```

### 第三种

```
if 条件:
    if 条件:
        代码1
    else:
        代码2
else:
    代码2
```

if可以嵌套使用

例:

```
money = int(input("输入你的余额: "))
if money >= 500:
    if money >= 1000:
        print("可以去吃大餐")
    else:
        print("只能去吃点简单的")
else:
    print("钱不够, 回家吃")
```

### 第四种

```
if 条件:
    代码1
elif 条件:
    代码2
....
else:
    代码3
```

多条件判断

例:

```
money = int(input("输入你的余额: "))
if money >= 1000:
    print("可以去吃大餐")
elif money >= 500:
    print("只能去吃点简单的")
else:
    print("钱不够, 回家吃")
```

## 6. while 循环语句

可以让我们的代码重复的去执行

### while 循环

```
while 条件:  
    代码
```

判断条件是否为真，如果真则执行代码。然后再次判断条件，直到条件为假才会结束循环。

例：

```
while True:      # 死循环  
    print("喷死你")
```

实战1，输出从1-100

```
a = 1  
while a <= 100:  
    print(a)  
    a = a + 1
```

实战2，从1加到100

```
a = 1  
b = 0  
while a <= 100:  
    b = b + a # 累加  
    a = a + 1  
print(b)
```

## 7. break和continue

**break**：让当前的循环立即停止

例：

```
while True:  
    content = input("请输入你想说的话(q结束): ")  
    if content == "q":      # "=="判断左右是否一致  
        break      # 结束循环  
    print("昵称: ",content)
```

**continue**：结束当前本次循环，继续执行下一次循环

例：

```
a = 1
while a <= 10:
    if a == 3:
        a += 1
        continue
    print(a)
    a += 1
```

## 8. for 循环

for 循环，用于可迭代对象

比如：字符串是可迭代的

### for 循环语法

```
for 变量 in 可迭代:
    代码1
```

作用：把可迭代的東西中的每一項拿出來，挨個賦值給變量，每一次賦值都要執行一次循環體(代碼)

例：

```
name = "Hello,My name is Python"
for name2 in name:
    print(name2)
```

for 循环想要计数，必须借助于 range

### range

语法作用：

range(n)：从0数到n，不包含n

range(m , n)：从m数到n，不包含n

range(m , n , s)：从m数到n，不包含n，每次间隔s

## 9. 运算符

### 算数运算符

+: 加      1+1=2

-: 减      1-1=0

\*: 乘      1\*1=1

/: 除      10/2=5

//：整除  $9//2=2$

%：取余  $9\%4=1$

\*\*：指数  $2**4=2*2*2*2=16$

()：小括号  $(1+2)*3=9$

**注：**混合运算优先级顺序：() 高于 \*\*、/、//、% 高于 +、-

## 赋值运算符

| 运算符 | 描述 | 实例                |
|-----|----|-------------------|
| =   | 赋值 | 将=右侧的结果赋值给等号左侧的变量 |

- 单个变量赋值

例：

```
num = 1
print(num)
```

- 多个变量赋值

例：

```
a,b,c = 1,2,3
print(a)
print(b)
print(c)
```

- 多变量赋相同值

例：

```
a = b = 10
print(a)
print(b)
```

## 复合赋值运算符

| 运算符              | 描述      | 实例                                      |
|------------------|---------|-----------------------------------------|
| <code>+=</code>  | 加法赋值运算符 | <code>c += a ===&gt; c = c + a</code>   |
| <code>-=</code>  | 减法赋值运算符 | <code>c -= a ===&gt; c = c - a</code>   |
| <code>*=</code>  | 算符      | <code>c *= a ===&gt; c = c * a</code>   |
| <code>/=</code>  | 除法赋值运算符 | <code>c /= a ===&gt; c = c / a</code>   |
| <code>//=</code> | 整除赋值运算符 | <code>c //= a ===&gt; c = c // a</code> |
| <code>%=</code>  | 取余赋值运算符 | <code>c %= a ===&gt; c = c % a</code>   |
| <code>**=</code> | 幂赋值运算符  | <code>c **= a ===&gt; c = c ** a</code> |

**注：**先算复合赋值运算符右面的表达式，再算复合赋值运算

### 比较运算符

比较运算符也叫关系运算符，通常用来判断

| 运算符                | 描述                                              | 实例                                    |
|--------------------|-------------------------------------------------|---------------------------------------|
| <code>==</code>    | 判断相等。如果两个操作数的结果相等，则条件结果为真(True)，否则条件结果为假(False) | 如a=3,b=3,则(a == b)为True               |
| <code>!=</code>    | 不等于。如果两个操作数的结果不相等，则条件为真(True),否则条件结果为假          | 如a=3,b=3,则(a==b)为真，若a=3,b=4,则(a!=b)为假 |
| <code>&gt;</code>  | 大于                                              | a=4,b=3,a>b, 为真，否则为假                  |
| <code>&lt;</code>  | 小于                                              | a=4,b=5,a<b,为真，否则为假                   |
| <code>&gt;=</code> | 大于等于                                            |                                       |
| <code>&lt;=</code> | 小于等于                                            |                                       |

### 逻辑运算符

| 运算符 | 逻辑表达式   | 描述  |
|-----|---------|-----|
| and | x and y | 与运算 |
| or  | x or y  | 或运算 |
| not | not x   | 非运算 |

例：

```
# and
a = 3
b = 5
c = 6
print((a < b) and (b < c))

# or
print((a > b) or (b < c))

# not
print((not (a > b)))
```

## 拓展

数字之间的逻辑运算

例：

```
a = 0
b = 1
c = 2
# and: 只要有一个数字为0，则结果为0，否则为最后一个非0数字
print(a and b) # 0
print(c and b) # 1

# or
print(a or b) # 1
print(a or c) # 2
print(b or c) # 1
```

## 10. 随机数

```
import random
# 使用方式
random.randint(开始, 结束)
```

## 实战

猜拳小游戏：

```
# 导入随机数random模块
import random

user = int(input("请您输入您要出的类型(石头=0, 剪刀=1, 布=2): "))
print(f"你出的是{user}")
pc = random.randint(0,2)
print(f"电脑出的是{pc}")
if ((user == 0)and (pc == 1))or((user == 1)and (pc == 2))or((user == 2)and (pc == 0)):
    print("恭喜你, 你赢了! ")
elif user == pc:
    print("平局! ")
else:
    print("很遗憾, 你输了! ")
```

## 12. 三目运算符(三元运算)

三目运算符也叫三元运算符或三元表达式

语法:

条件成立执行的表达式 **if** 条件 **else** 条件不成立执行的表达式

例:

```
a = 1
b = 2
c = a if a > b else b
print(c)
# 2
```

## 11. 补充

**pass**

代码占位, 跳过此次执行

例:

```
a = 10
if a > 100:
    pass
```

## 九九乘法表分步

打印一行\*

```
a = 0
while a < 5:
    print("*",end = "")
    a += 1
```

打印5行\*

```
b = 0
while b < 5:
    a = 0
    while a < 5:
        print("*",end = "")
        a += 1
    print()
    b += 1
```

打印阶梯型\*(联动行列)

```
b = 0
while b < 5:
    a = 0
    while a <= b:
        print("*",end = "")
        a += 1
    print()
    b += 1
```

实战九九乘法表:

```
b = 1
while b <= 9:
    # 一行的表达式
    a = 1
    while a <= b:
        print(f"{a} * {b} = {a * b}",end = "\t")
        a += 1
    print()
    b += 1
```

## 二、基础数据类型详解

---

## 1. int、float、bool

### int：整数

作用：加减乘除、大小比较

```
a = 10
b = 1
```

### float：小数、浮点数

```
print(10/3)
# 3.3333333333333335
```

**注：**小数范围是无限的，整数在某一个特定区间内是可以表示清楚，计算机表示一个小数是会有误差的

### bool：布尔值

作用：用来做条件判断

取值范围：True、False

## 基础数据类型之间的转换

```
a = 20 # 这里是str类型
b = int(a) # 把a转换为int类型
print(type(b))
c = bool(b)
print(type(c)) # 把int, 转化为bool
print(c) # True
d = 0
e = bool(d)
print(type(e))
print(e) # False
s = "我"
i = ""
print(bool(s), bool(i))
```

**注：**所有非0的数字都是True，只有bool等于0时，才为False；所有非空字符串都是True，只有空字符串是false

综上所述，在python的基本数据类型中，表示空的东西都是false，不空的都是True

例：

```
while 1:
    content = input("请输入你要发送的内容：")
    if content:
        print(content)
    else:
        break
```

## 2. str 字符串

### 字符串的格式化

例：“我叫 xxx，我住在 xxx，今年 xx 岁”

```
name = input("请输入你的名字：")
address= input("亲输入你的住址：")
age = int(input("请输入你的年龄："))
# %s: 字符串的占位
# %d: int的占位
# %f: 占位小数
s = "我叫%s，我住在%s，今年%d岁" % (name,address,age)
# 简化
s1 = "我叫{},我住在{},今年{}岁".format(name,address,age)
# 情况2
# f-string
s2 = f"我叫{name}我叫{name}我叫{name}我叫{name}我住在{address}我叫{name}我叫{name}"
print(s)
print(s1)
print(s2)
```

### 字符串的索引和切片

#### 索引

按照位置提取元素

例：

```
a = "我叫周杰伦"
print(a[3])
# 杰
print(a[0])
# 我
print(a[-1])
# 伦
```

#### 切片

从字符串中提取一部分内容

例：

```
s = "我叫周杰伦，你叫周润发吗？"
print(s[0:6])
print(s[:6])    # 如果start是从头开始进行切片，可以省略
print(s[6:])    # 从start开始一直截取到末尾
print(s[:])# 如果左右两端都是空白，表示从头到结尾
```

**注：**切片拿不到第二个位置的元素

语法：

`s[start:end]`：从start开始到end切片，但取不到end

## 倒数切片

```
print(s[-3:-1])
print(s[-1:-3]) # 无结果
```

**注：**切片目前只能从左往右

## 切片·步长

可以给切片添加步长，来控制切片的方向

例：

```
s = "我爱你"
# 使输出结果为：你爱我
print(s[::-1]) # -表示，从右往左
```

语法：

`s[start:end:step]`：从start切到end，每个step个元素出来一个元素

例：

```
s = "123456789"
print(s[2:9:2])
print(s[-1:-9:-2])
```

## 字符串的常规操作

字符串的操作一般不会对原字符串产生影响，一般是返回一个新的字符串

### 字符串大小写转换

```
# 把字符串第一个字符大写
s = "python"
s1 = s.capitalize()
print(s1)

# 把句子中的每一个单词的首字母都大写
s = "I have a dream!"
s1 = s.title()
print(s1)

# 把所有单词都变成小写
s = "I HAVE A DREAM!"
s1 = s.lower()
print(s1)

# 把所有单词都变成大写
s = "i have a dream!"
s1 = s.upper()
print(s1)
```

如何忽略大小写来进行判断 **重点：** `upper()`

例:

```
yanzheng_code = "sdff"
user_input = input(f"请输入验证码 ({yanzheng_code}): ")
if yanzheng_code.upper() == user_input.upper():
    print("验证成功!")
else:
    print("验证失败, 请重新验证!")
```

## 替换和切割 (重点)

`strip()`: 去除字符串左右两端的空白符 (空格、\t、\n)

例:

```
s = "    我是    周    成功    我没    钱    "
s1 = s.strip()
print(s1)
```

## 实战:

```
username = input("请输入用户名: ")
password = input("请输入密码: ")
if username == "admin":
    if password == "123456":
        print("登录成功!")
    else:
        print("登录失败!")
else:
    print("登录失败!")
```

`replace(old, new)`: 字符串替换

```
s = "12345,98"
s1 = s.replace(",98","678")
print(s1)
```

## 替换空格

```
s = "hello my name is python"
s1 = s.replace(" ", "") # 去掉所有空格
print(s1)
```

## 字符串切割

`split(用什么切割)`: 字符串切割, 用什么切, 就会损失谁

```
s = "python_java_C++_C#_javascript"
s1 = s.split("_")
print(s1)
```

**注:** 切割之后的结果会放在列表中

重点: `replace()`, `split()`, `strip()` ==> 记住

## 字符串对齐

`ljust()`: 返回一个原字符串左对齐, 并使用指定字符(默认空格)填充至对应长度的新字符串

语法:

```
字符串序列.ljust(长度, 填充字符)
```

例:

```
a = "hello"
# 设置左对齐
b = a.ljust(10, ".")
print(b)
# 设置右对齐
c = a.rjust(10, ".")
print(c)
# 中对齐
d = a.center(10, ".")
print(d)
```

## 查找和判断

### 查找

```
# find()
s = "你好啊, 我叫周润发"
s1 = s.find("周润发") # 如果返回-1就是没有该字符串出现
print(s1)

# index()
s2 = s.index("周润发") # 如果报错就是没有
print(s2)

# in
print("周润发" in s) # 判断是否存在
print("周润发" not in s) # 判断是否不存在
```

### 判断

```
name = input("请输入你的名字: ")
if name.startswith("张"): # 判断字符串是否以xxx开头, endswith(): 判断结尾是xxx
    print("你姓张")
else:
    print("你不姓张")

# isalpha: 判断字符串是否为纯字母(有空格也是False)
a = "admin"
print(a.isalpha())

# 判断输入的是否为数字
money = input("输入你有多少钱: ")
```

```

if money.isdigit():      # isdigit(): 整数, isdecimal(): 小数
    money = int(money)
    print("可以花钱了")
else:
    print("对不起, 输入错误!")

# isalnum: 数字或字母或组合都返回True
b = "abd123"
print(b.isalnum())

# isspace: 空白
c = "    "
print(c.isspace())

```

**重点:** `startswith()`, `isdigit()`, `in`, `not in`, `find()` ==> 记住

## 补充和总结

`len()`: 长度

```

s = "hello"
print(len(s))    # 计算字符长度

```

`join()`: 将列表中的字符串用指定的内容连在一起

```

s = "python_java_C++_C#_javascript"
s1 = s.split("_")
print(s1)
# 相反
s2 = ['python', 'java', 'C++', 'C#', 'javascript']
# 用下划线把字符连起来
s3 = "_".join(s2)
print(s3)

```

## 总结

1. `f"{变量}"`: 格式化字符串
2. 索引和切片:
  1. 索引: 是从0开始的 `[]`
  2. 切片: `s[start: end: step]`: end位置的数据, 永远拿不到
3. 相关操作
 

字符串操作对原字符串是不发生改变的

  1. `upper()`: 需要忽略大小写的时候用到
  2. `strip()`: 去掉字符串左右两端的空白符 (空格、\t、\n)
  3. `replace()`: 字符串替换
  4. `split()`: 对字符串进行切割
  5. `join()`: 拼接一个列表中的内容, 成为一个新的字符串
  6. `startswith()`: 判断字符串是否以xxx开头
  7. `len()`: 字符串长度(内置函数)

字符串的循环和遍历

```
for c in s:
    print(c) # 获取字符串中的每一个字符
```

关于 in

1. 判断xxx是否在xxxx中出现
2. for循环

### 3. 列表

定义：能装东西的东西

在python中用 [] 来表示一个列表，列表中的元素通过 . 隔开

```
a = ["hello", 12, "old".[1,2,3,True]]
```

#### 特性

1. 也像字符串一样也有索引和切片

```
a = ["金毛", "泰迪", "拉布拉多", "杜宾"]
print(a[0])
print(a[1:3])
print(a[::-1])
```

2. 可以用for循环进行遍历

```
a = ["金毛", "泰迪", "拉布拉多", "杜宾"]
for item in a:
    print(item)
```

3. 用 len() 获取列表长度

```
a = ["金毛", "泰迪", "拉布拉多", "杜宾"]
print(len(a))
```

4. index()：查找是否有这个内容，没有就报错

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
print(lst.index("小舞"))
```

5. count()：计算指定数据在列表中出现的次数

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
print(lst.count("唐三")) # 1
```

6. 判断是否存在

in：判断指定数据在某个列表序列

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
print("唐三" in lst)
```

not in: 判断指定数据不在某个列表序列

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']  
print("朱竹清" not in lst)
```

## 实战

```
# 查找账号是否存在  
user_name = ["张元英", "张无忌", "马云", "马化腾"]  
name = input("请输入你要查询的账户名: ")  
if name in user_name:  
    print(f"{name}这个账户已存在!")  
else:  
    print(f"{name}这个账户不存在, 请前往注册")
```

## 列表的增删改查(\*)

### 1. 向列表中添加内容

`.append`: 追加

例:

```
lst = []  
lst.append("唐三")  
print(lst)
```

`.insert`: 插入

语法: `lst.insert(要插入的位置, 要插入的内容)`

例:

```
lst = ["唐三", "小舞"]  
lst.insert(0, "玉小刚")  
print(lst)
```

`.extend`: 可以合并两个列表, 批量添加

例:

```
lst = ["唐三", "小舞"]  
lst.extend(["玉小刚", "奥斯卡", "戴沐白"])  
print(lst)
```

### 2. 删除列表中的内容

`del`

语法:

`del 目标`

例:

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
del lst
print(lst)    # 报错

# 第二种用法
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
del lst[0]
print(lst)
```

`pop`

例:

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
dele = lst.pop(1)
print(lst)
print(dele)    # 给出被删除的索引, 返回被删除的元素
```

`.remove`: 删除列表中的某个元素(\*)

例:

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
lst.remove("唐三")
print(lst)
```

`clear()`: 清空列表

例:

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
lst.clear()
print(lst)
```

### 3. 修改列表中的内容

例:

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
lst[1] = "朱竹清"
print(lst)
```

### 4. 排序

逆序 `reverse`

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
lst.reverse()
print(lst)
# ['戴沐白', '奥斯卡', '玉小刚', '小舞', '唐三']
```

`sort`: 排序, 升序和降序

- 升序 `reverse = False`

```
lst = [1,5,2,6,3,4,7]
lst.sort()
print(lst)
# 或者
lst.sort(reverse = False)
print(lst)
```

○ 降序 reverse=True

```
lst = [1,5,2,6,3,4,7]
lst.sort(reverse = True)
print(lst)
```

## 5. copy 复制

例:

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
lst1 = lst.copy()
print(lst)
print(lst1)
```

## 列表的遍历

while 的列表遍历

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
a = 0
while a < len(lst):
    print(lst[a])
    a += 1
```

for 对列表遍历

```
lst = ['唐三', '小舞', '玉小刚', '奥斯卡', '戴沐白']
for a in lst:
    print(a)
```

## 列表嵌套

列表中包含其他的子列表

```
name_list = [["小红", "小明", "小张"], ["张校长", "李老师", "王老师"]]
```

查找子列表中的数据

```
name_list = [["小红", "小明", "小张"], ["张校长", "李老师", "王老师"]]
# 查找小张所在的列表
print(name_list[0])
print(name_list[0][1])
```

## 综合应用-随机分配办公室

三个办公室，8位老师

```
import random

# 准备数据
teacher_name = ['wang', 'zhang', 'li', 'dong', 'qian', 'xiao', 'tang', 'huang']
office_home = [[], [], []]

# 将老师放到办公室-- 遍历全部老师
for name in teacher_name:
    # 创建随机数
    num = random.randint(0, 2)
    name1 = office_home[num].append(name)

a = 1
# 遍历office_home中的数据
for office in office_home:
    print(f"办公室{a}的人数是{len(office)},老师分别是：")
    for name2 in office:
        print(name2)
    a += 1
```

## 总结

列表的格式

```
["数据1", "数据2", "数据3"]
```

常用操作方法

- `index()`：检查是否存在
- `len()`：计算列表长度
- `append()`：往列表添加内容
- `pop()`：删除列表中的内容
- `remove()`：按照指定数据删除

列表嵌套

```
name_list = [["小红", "小明", "小张"], ["张校长", "李老师", "王老师"]]
# 查找小张所在的列表
print(name_list[0])
print(name_list[0][1])
```

## 4. 元组

### 特性

元组可以存储多个数据，但元组中的数据不能修改

```
a = (1, 2, 3)
print(type(a))
```

## 定义元组

```
# 多个数据元组
t1 = (1,2,3)
# 单个数据元组
t2 = (10,)
```

## 常见操作

元组数据不支持修改，只支持查找，具体如下：

### 索引和查找

```
a = (1,2,3)
print(a[0])
```

index()

```
a = (1,2,3)
print(a.index(1))
```

count()：统计出现次数

```
a = (1,2,3,1)
print(a.count(1))
```

len：统计数据个数

```
a = (1,2,3,1)
print(len(a))
```

## 元组数据的修改操作

```
a = (1,2,3,1,["aa","bb","cc"])
a[-1][0] = "root"
print(a)
```

## 5. 字典

### 特性

字典里的数据是以**键值对**形式出现的，即字典不支持索引，无论后期数据如何变化，只要按照对应的键的名字查找数据即可

语法：

```
# 有数据字典
a = {'name': 'tom', "age": 20, 'gender': '男'}

# 空字典
b = {}

c = dict()
```

## 常见操作

### 1. 增加和修改

写法：字典序列[key] = 值

```
a = {'name': 'tom', "age": 20, 'gender': '男'}
# 如果key存在则修改key的值
a["age"] = 18
print(a)

# 如果key不存在字典中，则新增
a["money"] = 1000000
print(a)
```

**注：**如果key存在则修改这个key对应的值；如果key不存在则新增此键值对

### 2. 删除

`del()` / `del`：删除字典或删除字典指定键值对

```
a = {'name': 'tom', "age": 20, 'gender': '男'}
# del 删除字典
b = a.copy()
del(b)
print(b)
# del 删除键值对
del a['name']
print(a)

# del()
del(a['age'])
print(a)
```

`clear()`：清空字典

```
a = {'name': 'tom', "age": 20, 'gender': '男'}
a.clear()
print(a)
```

### 3. 查找

key值查找

```
a = {'name': 'tom', "age": 20, 'gender': '男'}
print(a['name'])
print(a['id']) # 如果key值在字典中没有，则报错
```

`get()`

语法:

字典序列.`get(key, 默认值)`

**注:** 如果当前查找的key不存在则返回第二个参数(默认值), 如果省略第二个参数, 则返回None

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
print(a.get('name')) # tom
print(a.get('id', "不存在")) # 不存在
print(a.get('id')) # None
```

`keys()`: 获取字典中所有的key名, 返回可迭代对象

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
print(a.keys())
# dict_keys(['name', 'age', 'gender'])
```

`values`: 获取字典中的所有value, 返回可迭代对象

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
print(a.values())
# dict_values(['tom', 20, '男'])
```

`items`: 获取字典中所有的键值对, 返回可迭代对象, 里面的数据是元组, 元组数据1是字典的key, 元组数据2是字典key的值

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
print(a.items())
# dict_items([('name', 'tom'), ('age', 20), ('gender', '男')])
```

## 字典的循环遍历

### 1. 遍历字典的key

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
for key in a.keys():
    print(key)
```

### 2. 遍历字典的value

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
for value in a.values():
    print(value)
```

### 3. 遍历字典的元素

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
for item in a.items():
    print(item)
```

#### 4. 遍历字典的键值对

```
a = {'name': 'tom', 'age': 20, 'gender': '男'}
for key, value in a.items():
    print(f"{key} = {value}")
```

## 6. 集合

### 创建集合

创建集合使用 `{}` 或 `set()`，但是如果要创建空集合只能使用 `set()`，因为 `{}` 只能用来创建空字典

```
# 创建有数据的集合
a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
print(a)

# 集合数据有去重功能
a1 = {1, 2, 1, 2, 66, 48, 14, 15, 44, 66}
print(a1)

a3 = set("abcdefg")
print(a3)

# 创建空集合
a4 = set()
print(type(a4))
```

### 操作方法

#### 1. 增加数据

`add()`：追加单一数据

```
a1 = {1, 2, 1, 2, 66, 48, 14, 15, 44, 66}
a1.add(5)
print(a1)
```

`update()`：追加数据序列

```
a1 = {1, 2, 1, 2, 66, 48, 14, 15, 44, 66}
a1.update([10, 20, 30, 40])
print(a1)
```

#### 2. 删除数据

`remove()`：删除集合中的指定数据，如果数据不存在则报错

```
a1 = {1, 2, 1, 2, 66, 48, 14, 15, 44, 66}
a1.remove(1)
print(a1)
```

`discard()`：删除指定数据，如果数据不存在也不会报错

```
a1 = {1,2,1,2,66,48,14,15,44,66}  
a1.discard(2)  
print(a1)
```

`pop()`：随机删除集合中的某个数据，并返回这个数据

```
a1 = {1,2,1,2,66,48,14,15,44,66}  
a2 = a1.pop()  
print(a2)  
print(a1)
```

### 3. 查找数据

`in`：判断数据在集合序列

```
a1 = {1,2,1,2,66,48,14,15,44,66}  
print(1 in a1)
```

`not in`：判断数据不在集合数列

```
a1 = {1,2,1,2,66,48,14,15,44,66}  
print(1 not in a1)
```

## (\*) 记忆

| 数据类型        | 增                                                                                                                   | 删                                                                                                                                                          | 改            | 查                                                                               | 其他                                                                                                                                  |
|-------------|---------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------|---------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| 列表<br>list  | <code>.append(值)</code> : 追加<br><code>.insert(位置, 值)</code> : 插入<br><code>.extend([要合并/加入的值])</code> : 将两个列表合并/批量添加 | <code>.pop(位置)</code> : 删除指定位置的值<br><code>.clear()</code> : 清空列表<br><code>.remove(值)</code> : 删除指定的值, 如果数据不存在则报错<br><code>del 目标/目标[]</code> : 删除列表/或列表中的值 | 列表名[位置] = 新值 | 列表名[位置]: 查询列表中的值<br>列表名[位置][位置]: 查找子列表中的值                                       | <code>.copy()</code> : 复制列表<br><code>.reverse()</code> : 逆序<br><code>.sort(reverse = True/False)</code> : 排序, True为降序, False为升序     |
| 元组<br>tuple | 无                                                                                                                   | 无                                                                                                                                                          | 无            | 无                                                                               |                                                                                                                                     |
| 字典<br>dict  | 字典名[key] = 新值: 如果key存在则修改对应的值, 不存在则增加键值对                                                                            | <code>del 字典名/字典名[key]</code> : 删除字典/键值对<br><code>.clear()</code> : 清空字典                                                                                   | 同增           | 字典名[key]: 如果key值在字典中没有则报错<br><code>.get(key, 返回值)</code> : 如果查找的key不存在, 则返回None | <code>.keys()</code> : 获取所有key, 返回可迭代对象<br><code>.values()</code> : 获取所有values, 返回可迭代对象<br><code>.items()</code> : 获取所有键值对, 返回可迭代对象 |
| 集合 set      | <code>add(追加的值)</code> : 追加, 追加单一数据<br><code>.update([多个值])</code> : 追加数据序列                                         | <code>.remove(值)</code> : 删除指定的值, 如果数据不存在则报错<br><code>.pop(位置)</code> : 随机删除指定位置的值<br><code>.discard(值)</code> : 删除指定数据, 数据不存在也不报错                         | 同增           | 值 in 集合名: 判断值是否在集合内<br>值 not in 集合名: 判断值是否不在集合内                                 |                                                                                                                                     |

### 三、公共操作

#### 1. 运算符

| 运算符    | 描述      | 支持的容器类型      |
|--------|---------|--------------|
| +      | 合并      | 字符串、列表、元组    |
| *      | 复制      | 字符串、列表、元组    |
| in     | 元素是否存在  | 字符串、列表、元组、字典 |
| not in | 元素是否不存在 | 字符串、列表、元组、字典 |

### 1. +

```
str1 = 'aa'
str2 = 'bb'

list1 = [1,2]
list2 = [3,4]

t1 = (1,2)
t2 = (10,20)

# +
print(str1 + str2)
print(list1 + list2)
print(t1 + t2)
```

### 2. \*

```
str1 = 'aa'
str2 = 'bb'

list1 = [1,2]
list2 = [3,4]

t1 = (1,2)
t2 = (10,20)

# +
print(str1 * 2)
print(list1 * 2)
print(t1 * 2)
```

### 3. in

```
str1 = 'aa'
str2 = 'bb'

list1 = [1,2]
list2 = [3,4]

t1 = (1,2)
t2 = (10,20)

dict1 = {'a':1, 'b':2}
```

```
dict2 = {'c':3,'d':4}

# +
print('aa' in str1)
print(1 in list1)
print(1 in t1)
print('a' in dict1)
```

#### 4. not in

```
str1 = 'aa'
str2 = 'bb'

list1 = [1,2]
list2 = [3,4]

t1 = (1,2)
t2 = (10,20)

dict1 = {'a':1,'b':2}
dict2 = {'c':3,'d':4}

# +
print('aa' not in str1)
print(1 not in list1)
print(1 not in t1)
print('a' not in dict1)
```

## 2. 公共方法

| 函数                                    | 描述                                                         |
|---------------------------------------|------------------------------------------------------------|
| <code>len()</code>                    | 计算容器中元素个数                                                  |
| <code>del</code> 或 <code>del()</code> | 删除                                                         |
| <code>max()</code>                    | 返回容器中元素的最大值                                                |
| <code>min()</code>                    | 返回容器中元素的最小值                                                |
| <code>range(start,end,step)</code>    | 生成从start到end的数字，步长为step，供for循环使用                           |
| <code>enumerate()</code>              | 用于将一个可遍历的数据对象(如列表，元组或字符串)组合为一个索引序列，同时列出数据和数据下标，一般用于for循环当中 |

#### 1. len()

```
str1 = 'aa'
list1 = [1,2]
t1 = (1,2)
dict1 = {'a':1, 'b':2}

print(len(str1))
print(len(list1))
print(len(t1))
print(len(dict1))
```

## 2. del/del()

```
str1 = 'aa'
list1 = [1,2]
t1 = (1,2)
dict1 = {'a':1, 'b':2}

del str1
del list1
del t1
del dict1
```

## 3. max()

```
str1 = 'abcdef'
list1 = [1,2,4,5,8,9,7]

print(max(str1))
print(max(list1))
```

## 4. min()

```
str1 = 'abcdef'
list1 = [1,2,4,5,8,9,7]

print(min(str1))
print(min(list1))
```

## 5. range()

```
# 输出1-9
for i in range(1,10):
    print(i,end = "")
print()

# 输出13579
for i in range(1,10,2):
    print(i,end = "")
print()

# 输出0-9
for i in range(10):
    print(i,end = "")
```

## 6. enumerate()

语法:

```
enumerate(可遍历对象, start=0)
```

**注:** `start` 参数用来设置遍历数据下标的起始值, 默认为0

```
list1 = [1,2,3,5,45,4545,116]
for i in enumerate(list1):
    print(i)

for i in enumerate(list1,start=1):
    print(i)
```

## 3. 容器类型转换

### 1. tuple()

作用: 将某个序列转换成元组

```
list1 = [1,2,3,5,45,4545,116]
s1 = {1,5,845,84654,4541}
t1 = (11,5454,8487,45651)

print(tuple(list1))
print(tuple(t1))
```

### 2. list()

作用: 将某个序列转换为列表

```
list1 = [1,2,3,5,45,4545,116]
s1 = {1,5,845,84654,4541}
t1 = (11,5454,8487,45651)

print(list(s1))
print(list(t1))
```

### 3. set

作用：将某个序列转换为集合

```
list1 = [1,2,3,5,45,4545,116]
s1 = {1,5,845,84654,4541}
t1 = (11,5454,8487,45651)

print(set(list1))
print(set(s1))
```

## 四、推导式

### 1. 列表推导式

作用：用一个表达式创建一个**有规律**的列表，或控制一个有规律的列表

列表推导式有叫列表生成式

#### 1. 实战：创建一个0-10的列表

```
# while实现
list1 = []
i = 0
while i <= 10:
    list1.append(i)
    i += 1
print(list1)

# for循环实现
list2 = []
for i in range(11):
    list2.append(i)
    i += 1
print(list2)

# 列表推导式实现
list3 = [i for i in range(11)]
print(list3)
```

#### 2. 创建带 if 的列表推导式

要求：创建一个从0-10的偶数列表

```
# range步长实现
list1 = [i for i in range(0,11,2)]
print(list1)

# 使用if实现
list2 = [i for i in range(11) if i % 2 == 0]
print(list2)
```

#### 3. 多个 for 循环实现列表推导式

需求：创建列表如下：

```
[(1, 0), (1, 1), (1, 2), (2, 0), (2, 1), (2, 2)]
```

代码如下：

```
# 使用for循环实现
list1 = []
for i in range(1,3):
    for j in range(3):
        list1.append((i,j))

print(list1)

# 使用多个for循环推导式实现
list2 = [(i,j) for i in range(1,3) for j in range(3)]
print(list2)
```

所以，多 `for` 的列表推导式等同于 `for` 循环嵌套

## 2. 字典推导式

1. 思考：如果有以下两个列表

```
list1 = ['name', 'age', 'gender']
list2 = ['tom', 20, 'man']
```

如何快速合并为一个字典，代码如下：

```
list1 = ['name', 'age', 'gender']
list2 = ['tom', 20, 'man']
dict1 = {list1[i]:list2[i] for i in range(len(list1))}
print(dict1)
```

2. 创建一个字典：字典的keys 1-5数字，value是这个数字的2次方，代码如下：

```
dict1 = {i:i**2 for i in range(1,6)}
print(dict1)
```

3. 将两个列表合并为一个字典：

```
list1 = ['name', 'age', 'gender']
list2 = ['tom', 20, 'man']
dict1 = {list1[i]:list2[i] for i in range(len(list1))}
print(dict1)
```

**注：**

1. 如果，两个列表数据相同，`len()` 统计任何一个列表的长度都可以
2. 如果两个列表的数据不同，`len()` 统计多的列表数据个数会报错，`len()` 统计列表数据个数小的不会报错
4. 提取字典中的目标数据

```
counts = {'MBP': 268, 'HP': 125, 'DELL': 201, 'Lenovo': 199, 'acer': 99}
count1 = {key:value for key,value in counts.items() if value >= 200}
print(count1)
```

### 3. 集合推导式

1. 需求：创建一个集合，数据为下方列表的2次方

```
list1 = [1,1,2,3]
```

代码如下：

```
list1 = [1,1,2,3]
# set1 = {i**2 for i in list1}
set1 = set(i**2 for i in list1)
print(set1)
```

## 五、函数

### 1. 函数的作用

函数就是将一段具有独立功能的代码块整合到一个整体并命名，在需要的位置调用这个名称即可完成对应需求

- 函数在开发过程中，可以更高效的实现代码重用

### 2. 函数的使用步骤

1. 定义函数

```
def 函数名(参数):
    代码1
    代码2
    .....
```

2. 调用函数

```
函数名(参数)
```

注：

1. 不同的需求，参数可有可无
2. 在python中，函数必须先定义后使用

3. 快速体验

需求：复现ATM取钱功能

1. 搭建整体框架(复现需求)

```
print("密码正确，登录成功！")
# 显示"选择功能"界面
print("余额查询完毕！")
# 显示"选择功能"界面
print("成功提现2000元！")
# 显示"选择功能"界面
print("退卡成功！")
```

## 2. 确定"选择功能"界面内容

```
print("余额查询")
print("提现2000元")
print("退卡")
```

## 3. 完整代码：

```
def money():
    print("*****选择功能*****")
    print("1. 余额查询")
    print("2. 提现2000元")
    print("3. 退卡")
    print("*****选择功能*****")

print("密码正确，登录成功！")
# 显示"选择功能"界面
money()
print("您的余额是：800000元")
# 显示"选择功能"界面
money()
print("成功提现2000元！")
money()
# 显示"选择功能"界面
print("退卡成功！")
```

# 3. 函数的参数作用

## 1. 函数：固定1,2做加法

```
def num1():
    a = 1 + 2
    print(a)

num1()
```

## 2. 参数形式传入真实数据做加法运算

```
# 定义函数时同时定义了接收用户数据的参数a和b，a和b是形参
def num1(a,b):
    c = a + b
    print(c)

# 调用函数时传入了真实的数据10和20，真实数据为实参
num1(10,20)
```

## 4. 函数的返回值作用

```
def buy():
    return '烟'

good = buy()
print(good)
# 烟
```

`return` 作用：

- 负责函数返回值
- 退出当前函数：导致 `return` 下方的所有代码(函数体内部)不执行

### 1. 应用

需求：制作一个计算器，计算两个数字之和，并保存结果

```
def sum_num(num1, num2):
    return num1 + num2

sum1 = sum_num(int(input("输入数字1: ")),int(input("输入数字2: ")))
print(sum1)
```

## 5. 函数的说明文档

### 1. 语法

定义函数的说明文档

```
def 函数名():
    """说明文档的位置"""    # 必须放在第一行
    代码1
    代码2
    .....
```

查看函数说明文档

```
help(函数名)
```

实战：

```
def sum_num(a,b):  
    """求和函数"""  
    return a+b  
  
help(sum_num)
```

函数说明文档的高级使用

```
def sum_num(a,b):  
    """  
    求和函数sum_num(a,b)  
    :param a:数据1  
    :param b:数据2  
    :return:返回值  
    """  
    return a+b  
  
help(sum_num)
```

## 6. 函数嵌套

函数嵌套是指一个函数里面又调用了另外的一个函数

例:

```
def test1():  
    print("-----")  
  
def test2():  
    print("test1")  
    test1()  
    print("test2")  
  
test2()
```

## 7. 打印图形

1. 打印一条横线

```
def print_line():  
    print("-" * 20)  
  
print_line()
```

2. 打印多条横线

```
def print_line():
    print("-" * 20)

def print_not(num):
    i = 0
    while i < num:
        print_line()
        i += 1

print_not(10)
```

### 3. 函数计算

要求：先计算求三个数的和，再求三个数的平均值

```
# 求三个数之和
def sum_num(a, b, c):
    return a + b + c

# 求三个数的平均值
def average_num(a, b, c):
    # 先求和再除3
    return sum_num(a, b, c) / 3

print(sum_num(1, 2, 3))
print(average_num(1, 2, 3))
```

## 六、函数二

### 1. 变量作用域

变量作用域指的是变量生效的范围，主要分为两类：**局部变量**和**全局变量**

#### 1. 局部变量

局部变量是定义在函数体内部的变量，即只在函数体内部生效

```
def test1():
    a = 199
    print(a)

test1() # 199
print(a) # 报错
```

变量 `a` 是定义在 `test1` 函数内部的变量，在函数外部访问则**立即报错**

局部变量的作用，在函数体内部，临时保存数据，即当函数调用完成后，则销毁局部变量

#### 2. 全局变量

全局变量指的是在函数体内、都能生效的变量。

```
a = 199
def test1():
    print(a)

test1() # 199
print(a) # 199
```

### 3. 修改全局变量

需求：b函数想要a的取值为200

```
a = 100

def test1():
    print(a)

def test2():
    # 使用global声明a是全局变量
    global a
    a = 200
    print(a)

test1() # 100
test2() # 200
print(a) # 200
```

注：

1. 如果在函数里面直接把 `a = 200` 赋值，此时修改的是局部变量，不影响全局变量的赋值结果
2. 如果想要使 `test2` 中的 `a = 200` 变为全局，则需要使用 `global` 定义 `a` 为全局函数

## 2. 多函数程序执行流程

一般在实际开发过程中一个程序往往由多个函数组成(类)，并且多个函数共享某些数据

共用全局变量

```
a = 100

def test1():
    # 修改全局变量
    global a
    a = 199

def test2():
    # 调用test1中修改后的变量a
    print(a)

test1()
test2() # 199
```

### 3. 函数的返回值

返回值作为参数传递

```
def test1():  
    return 100  
  
def test2(num):  
    print(num)  
  
result = test1()  
print(result)    # 100  
# 自测  
test2(test1())  # 100
```

多返回值:

```
def test1():  
    return 100, 200  
    # return (100, 200)  
    # return [100, 200]  
    # return {"name": "tom"}  
  
result = test1()  
print(result)  # (100, 200), 返回元组
```

`return`后可以直书写 元组、列表、字典 返回多个值

### 4. 函数的参数

#### 1. 位置参数

位置参数: 调用函数时根据函数定义的参数位置来传递参数

```
def user_info(name, age, gender):  
    print(f'你的名字是{name}, 你今年{age}岁, 是个{gender}生。')  
  
user_info('tom', 22, '男')
```

**注:** 传递和定义参数的顺序及个数必须一致

#### 2. 关键字参数

关键字参数: 函数调用时通过"键 = 值"形式加以指定, 可以让函数更加清晰、容易使用, 同时也清除了参数的顺序需求

```
def user_info(name, age, gender):  
    print(f'你的名字是{name}, 你今年{age}岁, 是个{gender}生。')  
  
user_info('tom', age = "18", gender = "男")  
user_info('tom', gender = "男", age = "18")
```

**注:** 函数调用时, 如果有位置参数时, 位置参数必须在关键字参数的前面, 但关键字参数之间不存在先后顺序。

### 3. 缺省参数

缺省参数：也叫默认参数，用于定义函数，为参数提供默认值，调用函数时可不传该默认参数的值（注意：所有位置参数必须出现在默认参数前，包括函数定义和调用）

```
def user_info(name, age, gender = "男"):
    print(f'你的名字是{name}，你今年{age}岁，是个{gender}生。')

user_info(name = 'roes', age = 18, gender = "女")    # 修改默认值
user_info(name = "tom", age = 20)
```

**注：**函数调用时，缺省参数传值则修改默认值，否则使用这个默认值

### 4. 不定长参数

不定长参数：也叫可变参数，用于不确定调用的时候回传递多个参数（不传参也可以）的场景。此时可用包裹（`packing`）位置参数，或者包裹关键字参数，来进行参数传递，会显得非常方便

- 包裹位置传递

```
def user_info(*args):
    print(args)

user_info('tom', 18) # ('tom', 18)
user_info('tom', 18, "男") # ('tom', 18, '男')
user_info() # ()
```

**注：**传进所有的参数都会被 `args` 变量收集，它会更具传进参数的位置合并为一个元组

`args` 是元组类型，这就是包裹位置传递。

- 包裹关键字传递

```
def user_info(**kwargs):
    print(kwargs)

user_info(name = "tom", age = 20) # {'name': 'tom', 'age': 20}
user_info() # {}
```

## 5. 拆包和交换两个变量值

### 1. 拆包

- 拆包：元组

```
def return_num():
    return 10, 20

num1, num2 = return_num()
print(num1) # 10
print(num2) # 20
```

- 拆包：字典

```
dict1 = {'name': 'tom', 'age': 18}

a, b = dict1

# 对字典进行拆包，取出来的是字典的key
print(a)
print(b)

# 想取value的值则需要
print(dict1[a])
print(dict1[b])
```

## 2. 交换变量值

需求：有变量 `a = 10` 和 `b = 20`，交换两个变量的值

### ◦ 方法一

借助第三变量存储数据

```
a = 10
b = 20

# 定义中间变量
c = 0
# 将a的数据传入c中，此时c=10
c = a
# 修改a的值为b，此时a=20
a = b
# 修改b的值为c，此时b=10
b = c

print(a)    # 20
print(b)    # 10
```

### ◦ 方法二

```
a, b = 10, 20
a, b = b, a
print(a)    # 20
print(b)    # 10
```

# 6. 引用

## 1. 了解引用

在python中值是靠引用来传递的

我们可以用 `id()` 来判断两个变量是否为同一个值的引用。我们可以将id理解为那块内存的地址标识

- 不可变： `int`

```

# 不可变: int
a = 1
b = a

print(id(a))    # 140718934881192
print(id(b))    # 140718934881192

# 修改a的数据, 再测id值
a = 2
print(b)        # 1
print(id(a))    # 140718977676232
print(id(b))    # 140718977676232

```

- 列表-可变类型

```

aa = [1,4]
bb = aa

print(bb)    # [1, 4]
print(id(aa))    # 1760859696576
print(id(bb))    # 1760859696576

aa.append(9)
print(bb)    # [1, 4, 9]
print(id(aa))    # 1760859696576
print(id(bb))    # 1760859696576

```

- 引用当做实参

需求: 引用是否可以当做实参

```

def num1(a):
    print(a)
    print(id(a))

    a += a
    print(a)
    print(id(a))

b = 5
num1(b)
print('*' * 25)
c = [1, 2, 3]
num1(c)
"""
5
140718977676328
10
140718977676488
*****

[1, 2, 3]
3091934129600
[1, 2, 3, 1, 2, 3]
3091934129600

```

## 7. 可变和不可变类型

所谓可变类型与不可变类型是指：数据能够直接进行修改，如果能直接修改那么就是可变，否则是不可变

- 可变类型
  - 列表
  - 字典
  - 集合
- 不可变类型
  - 整型
  - 浮点型
  - 字符串
  - 元组

## 8. 总结

- 变量作用域
  - 全局：函数体内外都能生效
  - 局部：当前函数体内部生效
- 函数多返回值写法

```
return 表达式1, 表达式2...
```

- 函数的参数
  - 位置参数
    - 形参和实参的个数书写顺序必须一致
  - 关键字参数
    - 写法： `key = value`
    - 特点：形参和实参的书写顺序可以不一致，关键字参数必须书写在位置参数的后面
  - 缺省参数
    - 缺省参数就是默认参数
    - 写法： `key = value`
  - 不定长位置参数
    - 收集所有位置参数，返回一个元组
  - 不定长关键字参数
    - 收集所有关键字参数，返回一个字典
- 引用：Python中，数据的传递都是通过引用

## 七、面向对象基础

# 1. 理解面向对象

面向对象是一种抽象化的编程思想，很多编程语言都有的一种思想

总结：面向对象就是将编程当成是一个事物，对外界来说，事物是直接使用的，不用管他内部的情况，而编程就是设置事物能够做什么事

## 2. 类和对象

在面向对象编程过程中，有两个重要的组成部分：类和对象

类和对象的关系：用类去创建一个对象

### 理解类和对象

#### 1. 类

类是一系列具有相同特征和行为的事物的统称，是一个抽象的概念，不是真实存在的事物

- 特征即是属性
- 行为即是方法

#### 2. 对象

对象是类创建出来的真实存在的事物，例如：洗衣机

注：开发中，先有类，在有对象

#### 3. 面向对象实现方法

##### 1. 定义类

Python2 中类分为：经典类和新式类

语法：

```
class 类名():  
    代码  
    ....
```

注：类名要满足标识符命名规则，同时遵循大驼峰命名习惯

需求：洗衣机，功能：能洗衣服

```
# 需求：洗衣机，功能：能洗衣服  
# 定义洗衣机类  
class xiyiji():  
    def xiyi(self):  
        print('能洗衣服')  
  
# 创建对象  
haier =xiyiji()  
# 验证成果  
print(haier)      # 成功创建类  
  
# 使用xiyi功能，--实例方法/对象方法-- ：对象名.xiyi()  
haier.xiyi()
```

##### 2. 创建对象

对象又名实例

语法：

```
对象名 = 类名()
```

体验：

```
# 创建对象
haier = xiyiji()
# 验证成果
print(haier)    # 成功创建类
```

### 3. self

self 指的是调用该函数的对象

```
# 需求：洗衣机，功能：能洗衣服
# 定义类
class xiyiji():
    def xiyi(self):
        print('能洗衣服')    # 能洗衣服
        print(self) # <__main__.Xiyiji object at 0x000001612EE54C20>

haier = xiyiji()
print(haier)
# <__main__.Xiyiji object at 0x000001612EE54C20>

haier.xiyi()

# 由于打印对象和打印self得到的内存地址相同，所以self指的是调用该函数的对象
```

实战：

一个类是否能创建多个对象？

多个对象都调用函数的时候，self的地址是否相同

```
# 需求：洗衣机，功能：能洗衣服
# 定义类
class xiyiji():
    def xiyi(self):
        print('能洗衣服')    # 能洗衣服
        print(self)

# haier1
haier1 = xiyiji()
haier1.xiyi()    # <__main__.Xiyiji object at 0x000001CCAE6B4C20>
# haier2
haier2 = xiyiji()
haier2.xiyi()    # <__main__.Xiyiji object at 0x000001CCAE6B4C20>
```

结论：一个类可以创建多个对象，多个对象都调用函数时，self 地址不同

### 3. 添加和获取对象属性

属性即是特征，比如：洗衣机的长宽高和重量...

对象数学既可以在类外面添加和获取，也能在类里面添加和获取

#### 1. 类外面添加对象属性

语法：

```
对象名.属性名 = 值
```

体验：

```
class xiyiji():
    def xiyi(self):
        print('能洗衣服')    # 能洗衣服
        print(self)

haier = xiyiji()
# 添加属性
haier.width = 10
haier.height = 20
```

#### 2. 类外面获取对象属性

语法：

```
对象名.属性名
```

体验：

```
class xiyiji():
    def xiyi(self):
        print('能洗衣服')    # 能洗衣服
        print(self)

haier = xiyiji()
# 添加属性
haier.width = 100
haier.height = 200
# 获取属性
print(f'haier洗衣机的宽是{haier.width}cm，高是{haier.height}cm')
```

#### 3. 类里面获取对象属性

语法：

```
self.属性名
```

体验：

```

class xiyiji():
    def xiyi(self):
        # 在类中获取实例属性
        print(f'haier洗衣机的宽是{self.width}cm，高是{self.height}cm')

haier = xiyiji()
# 添加属性
haier.width = 100
haier.height = 200

haier.xiyi()

```

## 4. 魔法方法

在python中，`__xx__()` 的函数叫做魔法方法，指的是具有特殊功能的函数

### 1. `__init__()`

`__init__()` 方法的作用：初始化对象

```

class washer():
    # 定义__init__(),添加实例属性
    def __init__(self):
        # 添加实例属性
        self.width = 100
        self.height = 200

    def print_info(self):
        print(f"洗衣机宽{self.width}cm，高{self.height}cm")

haier = washer()
haier.print_info()

```

注：

1. `__init__()` 方法，在创建一个对象时默认被调用，不需要手动调用
2. `__init__()` 中的 `self` 参数，不需要开发者传递，python解释器会自动把当前的对象引用传递过去

### 2. 带参数的 `__init__()`

```

class washer():
    def __init__(self,width,height):
        # 添加实例属性
        self.width = width
        self.height = height

    def print_info(self):
        print(f"洗衣机宽{self.width}cm，高{self.height}cm")

# 定义两个对象，输入两个不同的实参，看两个对象是否一致
medi = washer(10,19)
medi.print_info()

haier = washer(20,30)
haier.print_info()

```

### 3. `__str__()`

当使用 `print` 输出对象的时候，默认打印对象的内存地址。如果类定义了 `__str__()` 方法，那么就会打印从这个方法中 `return` 的数据

简易理解：如果不定义 `__str__()` 的话，`print(对象名)` 时打印出的则是，对象的地址

`<__main__.Washer object at 0x0000025836674c20>`，如果定义了 `__str__()` 的话，将地址转换成 `__str__()` 下的值

```
class Washer():
    def __init__(self,width,height):
        self.width = width
        self.height = height

    def __str__(self):
        return '海尔洗衣机使用说明'

haier = Washer(20,30)
print(haier)      # 海尔洗衣机使用说明
```

### 4. `__del__()`

当删除对象时，python解释器也会调用 `__del__()` 方法

```
class Washer():
    def __init__(self,width,height):
        self.width = width
        self.height = height

    def __del__(self):
        print(f'{self.width},{self.height},已被删除')

haier = Washer(10,20)
del haier
```

## 5. 综合应用

### 烤地瓜

#### 1. 需求

被烤的时间和对应的地瓜状况：

0-3分钟：生的

3-5分钟：半生不熟

5-8分钟：熟的

超过8分钟：烤糊了

#### 2. 添加的调料

用户按照自己的意愿添加调料

#### 3. 步骤分析

##### 定义类

- 地瓜的属性

- 被烤的时间
- 地瓜的状态
- 添加的调料
- 地瓜的方法
  - 被烤
    - 用户根据自己的意愿控制每次烤的时间
    - 判断地瓜被烤的总时间在哪个区间，修改地瓜状态
  - 添加调料
    - 用户根据意愿设定添加的调料
    - 将用户添加的调料储存
- 显示对象信息

```
# 定义类：初始化属性，被烤和添加调料的方法、显示对象信息的str
class SweetPotato:
    # 定义初始化属性
    def __init__(self):
        # 被烤的时间
        self.cook_time = 0
        # 地瓜的状态
        self.cook_status = '生的'
        # 调料列表
        self.condiments = []

    def cook(self, time):
        """开始烤地瓜"""
        self.cook_time += time
        if 0 <= self.cook_time < 3:
            self.cook_status = "生的"
        elif 3 <= self.cook_time < 5:
            self.cook_status = '半生不熟'
        elif 5 <= self.cook_time < 8:
            self.cook_status = '熟的'
        elif 8 < self.cook_time:
            self.cook_status = '烤糊了'

    def add_condiments(self, condiments):
        """添加调料"""
        self.condiments.append(condiments)

    def __str__(self):
        return f'烤了{self.cook_time}分钟,现在是{self.cook_status},添加的调料有{self.condiments}'

# 创建对象并调用实例方法
laowang = SweetPotato()

laowang.cook(4)
laowang.add_condiments('辣椒')
laowang.add_condiments('酱油')
print(laowang)
```

## 搬家具

### 1. 需求

将小于房子剩余面积的家具摆到房子中

## 2. 步骤分析

需求涉及两个事物：房子和家具，故涉及两个类，房子类和家居类

### 1. 定义类

- 房子类
  - 实例属性
    - 房子的地理位置
    - 房子占地面积
    - 房子剩余面积
    - 房子内家具列表
  - 实例方法
    - 容纳家具
  - 显示房屋信息
- 家具类
  - 家具名称
  - 家具占地面积

### 2. 代码实现

```
# 定义家具类
class Furniture():
    def __init__(self, name, area):
        # 家具名称
        self.name = name
        # 家具占地面积
        self.area = area

# 房子类
class Home():
    def __init__(self, address, area):
        # 地理位置
        self.address = address
        # 房屋面积
        self.area = area
        # 剩余面积
        self.free_area = area
        # 家居列表
        self.furniture = []

    def __str__(self):
        return f'房子坐落于{self.address}, 占地面积{self.area}平方米, 剩余面积{self.free_area}平方米, 家具有{self.furniture}'

    def add_furniture(self, item):
        """容纳家具"""
        if self.free_area >= item.area:
            self.furniture.append(item.name)
            # 搬入家具后, 房屋剩余面积 = 之前剩余面积 - 家具面积
            self.free_area -= item.area
        else:
            print('家具太大, 剩余面积不够, 无法容纳')
```

```
bad = Furniture('床', 6)
sofa = Furniture('沙发', 10)

changes = Home('长沙市', 198)
print(changes)

changes.add_furniture(sofa)
changes.add_furniture(bad)
print(changes)
```

## 6. 总结

- 面向对象重要组成部分

- 类

- 创建类

```
class 类名():
    代码
    .....
```

- 对象

```
对象名 = 类名
```

- 添加对象属性

- 类外面

```
对象名.属性名 = 值
```

- 类里面

```
self.属性名 = 值
```

- 获取对象属性

- 类外面

```
对象名.属性名
```

- 类里面

```
self.属性名
```

- 魔法方法

- `__init__()`: 初始化
  - `__str__()`: 输出对象信息
  - `__del__()`: 删除对象时调用

## 八、面向对象--继承

### 1. 继承的概念

生活中的继承，一般指的是子女继承父辈的财产

- 拓展1：经典类或旧式类

不由任意内置类型派生出的类，称之为经典类

```
class 类名:
    代码
    .....
```

- 拓展2新式类

```
class 类名(object):    # object: 默认值，可省略
    代码
    ...
```

Python面向对象的继承指的是多个类之间的所属关系，即子类继承父类的所有属性和方法，如下：

```
# 父类A
class A(object):
    def __init__(self):
        self.num = 1
    def info_print(self):
        print(self.num)

# 子类B
class B(A):
    pass

num = B()
num.info_print()    # 1
```

在Python中，所有类默认继承object类，object类是顶级类或基类，其他子类叫做派生类

### 2. 单继承

故事：一个煎饼果子老师傅，研发了一套精湛的摊煎饼果子的技术，师傅要把这个技术传给他的徒弟

分析：徒弟是不是要继承师傅的所有技术

```
# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(Master):
```

```
pass

# 创建对象
disciple = Disciple()
# 对象访问实例属性
disciple.make_cake()
```

### 3. 多继承

所谓多继承意思就是一个类同时继承了多个父类

```
# 徒弟太想进步了，又TMD报了个牛马煎饼培训班
# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = '[牛马煎饼果子配方]'
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(Master, School):
    pass

laow = Disciple()
print(laow.kongfu) # [古法煎饼果子配方]
laow.make_cake() # 运用[古法煎饼果子配方]制作煎饼果子
```

**注：**当一个类有多个父类的时候，默认使用第一个父类的同名属性和方法

### 4. 子类重写父类的同名属性和方法

故事：老王掌握了师傅和培训的技术后，自己潜心钻研出了自己的独门配方

```
# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = '[牛马煎饼果子配方]'
    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(Master, School):
    def __init__(self):
        self.kongfu = '[独家配方]'
```

```

def make_cake(self):
    print(f'运用{self.kongfu}制作煎饼果子')

laow = Disciple()
print(laow.kongfu) # [独家配方]
laow.make_cake()   # 运用[独家配方]制作煎饼果子

```

结论：如果子类 and 父类拥有同名属性和方法，子类创建对象调用属性和方法的时候，调用到的是子类里面的同名属性和方法

## (\*) 扩展

`__mro__` 继承层级顺序

语法：

```
print(类名.__mro__)
```

## 5. 子类调用父类的同名属性的方法

有很多牛马都希望吃到古法和黑马技术的煎饼果子

```

# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = '[牛马煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(Master, School):
    def __init__(self):
        self.kongfu = '[独家配方]'

    def make_cake(self):
        # 加自己初始化的原因：如果不加自己的初始化，kongfu属性就是上一次调用的init内的属性
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

# 调用父类init魔法方法和make_cake方法从而实现调用父类属性和方法
def make_master_cake(self):
    Master.__init__(self)
    Master.make_cake(self)

def make_school_cake(self):
    School.__init__(self)
    School.make_cake(self)

```

```

laow = Disciple()
print(laow.kongfu) # 【独家配方】
laow.make_cake() # 运用【独家配方】制作煎饼果子
laow.make_master_cake() # 运用【古法煎饼果子配方】制作煎饼果子
laow.make_school_cake() # 运用【牛马煎饼果子配方】制作煎饼果子
laow.make_cake() # 运用【独家配方】制作煎饼果子

```

## 6. 多层继承

多年后，老王老了，要把技术全部传授给小刘

```

# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '【古法煎饼果子配方】'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = '【牛马煎饼果子配方】'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(Master, School):
    def __init__(self):
        self.kongfu = '【独家配方】'

    def make_cake(self):
        # 加自己初始化的原因：如果不加自己的初始化，kongfu属性就是上一次调用的init内的属性
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    # 调用父类init魔法方法和make_cake方法从而实现调用父类属性和方法
    def make_master_cake(self):
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)

class Tusun(Disciple):
    pass

xiaoлиu = Tusun()
xiaoлиu.make_cake()
xiaoлиu.make_master_cake()
xiaoлиu.make_school_cake()

```

## 7. `super()` 调用父类方法

方法一：如果定义类名修改，这里也要修改，麻烦；代码量庞大

```
# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(Master):
    def __init__(self):
        self.kongfu = '[牛马煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(School):
    def __init__(self):
        self.kongfu = '[独家配方]'

    def make_cake(self):
        # 加自己初始化的原因：如果不加自己的初始化，kongfu属性就是上一次调用的init内的属性
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    def make_master_cake(self):
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)

# 一次性调用全部父类
def make_master_school_cake(self):
    # 方法1：如果定义类名修改，这里也要修改，麻烦；代码量庞大
    School.__init__(self)
    School.make_cake(self)
    Master.__init__(self)
    Master.make_cake(self)

laow = Disciple()
laow.make_master_school_cake()
```

方法二： `super()`

2.1 语法：

```
super(当前类名, self).函数()
# super(School, self).__init__()
# super(School, self).make_cake()
```

代码实现：

```
# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(Master):
    def __init__(self):
        self.kongfu = '[牛马煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')
        # 2.1 super()
        super(School, self).__init__()
        super(School, self).make_cake()

# 徒弟类
class Disciple(School):
    def __init__(self):
        self.kongfu = '[独家配方]'

    def make_cake(self):
        # 加自己初始化的原因：如果不加自己的初始化，kongfu属性就是上一次调用的init内的属性
        self.__init__()
        print(f'运用{self.kongfu}制作煎饼果子')

    def make_master_cake(self):
        Master.__init__(self)
        Master.make_cake(self)

    def make_school_cake(self):
        School.__init__(self)
        School.make_cake(self)

# 一次性调用全部父类
def make_master_school_cake(self):
    ## 方法1：如果定义的类型修改，这里也要修改，麻烦；代码量庞大
    # School.__init__(self)
    # School.make_cake(self)
    # Master.__init__(self)
    # Master.make_cake(self)

    # 方法二：super()
    # 2.1 super(当前类名, self).函数()
    super(Disciple, self).__init__()
    super(Disciple, self).make_cake()

laow = Disciple()
laow.make_master_school_cake()
```

## 2.2 无参数 super()

语法:

```
super().函数()  
# super().__init__()  
# super().make_cake()
```

代码实现:

```
# 师傅类  
class Master(object):  
    def __init__(self):  
        self.kongfu = '[古法煎饼果子配方]'  
  
    def make_cake(self):  
        print(f'运用{self.kongfu}制作煎饼果子')  
  
class School(Master):  
    def __init__(self):  
        self.kongfu = '[牛马煎饼果子配方]'  
  
    def make_cake(self):  
        print(f'运用{self.kongfu}制作煎饼果子')  
        ## 2.1 super()  
        # super(School, self).__init__()  
        # super(School, self).make_cake()  
        # 无参数super()  
        super().__init__()  
        super().make_cake()  
  
# 徒弟类  
class Disciple(School):  
    def __init__(self):  
        self.kongfu = '[独家配方]'  
  
    def make_cake(self):  
        # 加自己初始化的原因: 如果不加自己的初始化, kongfu属性就是上一次调用的init内的属性  
        self.__init__()  
        print(f'运用{self.kongfu}制作煎饼果子')  
  
    def make_master_cake(self):  
        Master.__init__(self)  
        Master.make_cake(self)  
  
    def make_school_cake(self):  
        School.__init__(self)  
        School.make_cake(self)  
  
# 一次性调用全部父类  
def make_master_school_cake(self):  
    # 方法二: super()  
    ## 2.1 super(当前类名, self).函数()  
    # super(Disciple, self).__init__()  
    # super(Disciple, self).make_cake()  
  
    # 无参数super()
```

```

        super().__init__()
        super().make_cake()

laow = Disciple()
laow.make_master_school_cake()

```

**注：**使用 `super()` 可以自动查找父类。调用顺序遵循 `__mro__` 类属性的顺序。比较适合单继承使用

## 8. 私有属性和私有方法

### 定义私有属性和方法

老王把技术传给徒弟的同时，不想把自己的钱（20000个亿）继承给这个牛马徒弟，这个时候要为“钱”这个实例属性设置私有属性

设置私有权限的方法：在属性名和方法名前面加上两个下划线

```

class Disciple(School, Master):
    def __init__(self):
        self.kongfu = '[独家配方]'

    # 定义私有属性
    self.__money = 1000000

    # 定义私有方法
    def __info_print(self):
        print(self.kongfu)
        print(self.__money)

```

**注：**私有属性和方法只能在类里面访问和修改

### 获取和修改私有属性值

在python中，一般定义函数名 `get_xx` 用来获取私有属性，定义 `set_xx` 用来修改私有属性值

```

# 师傅类
class Master(object):
    def __init__(self):
        self.kongfu = '[古法煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

class School(object):
    def __init__(self):
        self.kongfu = '[牛马煎饼果子配方]'

    def make_cake(self):
        print(f'运用{self.kongfu}制作煎饼果子')

# 徒弟类
class Disciple(School, Master):
    def __init__(self):
        self.kongfu = '[独家配方]'

    # 定义私有属性

```

```

        self.__money = 1000000
# 定义私有方法
def __info_print(self):
    print(self.kongfu)
    print(self.__money)

# 获取私有属性
def get_money(self):
    return self.__money
# 修改私有属性值
def set_money(self):
    self.__money = 888

def make_cake(self):
    # 加自己初始化的原因: 如果不加自己的初始化, kongfu属性就是上一次调用的init内的属性
    self.__init__()
    print(f'运用{self.kongfu}制作煎饼果子')

def make_father_cake(self):
    School.__init__(self)
    School.make_cake(self)
    Master.__init__(self)
    Master.make_cake(self)

class Tusun(Disciple):
    pass

laow = Tusun()
print(laow.get_money()) # 1000000
laow.set_money()
print(laow.get_money())# 888

```

## 9. 总结

- 继承的特点
  - 子类默认拥有父类的所有属性和方法
  - 子类重写父类同名方法和属性
  - 子类调用父类同名方法和属性
- `super()` 方法快速调用父类方法
- 私有限制
  - 不能继承给子类的属性和方法需要添加私有权限
  - 语法

```

class 类名():
    # 私有属性
    __属性名 = 值

    # 私有办法
    def __函数名(self):
        代码

```

## 九、面向对象--其他

# 1. 面向对象三大特性

- 封装
  - 将属性和方法书写到类里面的操作即为封装
  - 封装可以为属性和方法添加私有权限
- 继承
  - 子类默认继承父类的所有属性和方法
  - 子类可以重写父类属性和方法
- 多态
  - 传入不同的对象，产生不同的结果

## 2. 多态

### 了解多态

多态指的是一类事物有多种形态，(一个抽象类有多个子类，因而多态的概念依赖于继承)。

- 定义：多态是一种使用对象的方式，子类重写父类方法，调用不同子类对象的相同父类方法，可以产生不同的执行结果
- 好处：调用灵活，有了多态，更容易编写出通用的代码，做出通用的编程，以适应需求的不断变化!
- 实现步骤：
  - 定义父类，并提供公共方法
  - 定义子类，并重写父类方法
  - 传递子类对象给调用者，可以看到不同子类执行效果不同

### 体验多态

```
class Dog(object):
    def work(self):
        pass

class ArmyDog(Dog):
    def work(self):
        print('追击敌人...')

class DrugDog(Dog):
    def work(self):
        print('追踪毒品...')

class Person(object):
    def work_dog(self, dog):
        dog.work()

ad = ArmyDog()
dd = DrugDog()

person = Person()
person.work_dog(ad) # 追击敌人...
person.work_dog(dd) # 追踪毒品...
```

## 3. 类属性和实例属性

### 类属性

#### 1. 设置和访问类属性

- 类属性就是**类对象**所拥有的属性，它被**该类的所有实例对象**所共有
- 类属性可以使用**类对象**或**实例对象**访问

```
class Dog(object):  
    num = 10  
  
laow = Dog()  
laox = Dog()  
  
print(Dog.num) # 10  
print(laow.num) # 10  
print(laox.num) # 10
```

#### 类属性的优点

- 记录的某项数据，始终保持一致时，则定义类属性
- 属性实例要求每个对象为其单独开辟一份内存空间来记录数据，而类属性为全类所共有，仅占用一份内存，更加节省空间

#### 2. 修改类属性

类属性只能通过类对象修改，不能通过实例对象修改，如果通过实例对象修改类属性，表示的是创建了一个实例属性

```
class Dog(object):  
    num = 10  
  
laow = Dog()  
laox = Dog()  
  
# 类修改：类.类属性=值  
Dog.num = 99  
print(Dog.num) # 99  
print(laow.num) # 99  
print(laox.num) # 99  
  
# 测试对象修改属性  
laow.num = 8  
print(Dog.num) # 99  
print(laow.num) # 8  
print(laox.num) # 99
```

## 4. 类方法和静态方法

### 类方法

#### 1. 类方法特点

- 需要用装饰器 `@classmethod` 来标识其为类方法，对于类方法，第一个参数必须是类对象，一般以 `cls` 作为第一个参数

## 2. 类方法的使用场景

- 当方法中需要使用对象(如访问私有类属性等)时, 定义类方法
- 类方法一般和类属性配合使用

```
class Dog(object):
    __num = 10

    @classmethod
    def return_num(cls):
        return cls.__num

laow = Dog()
result = laow.return_num()
print(result)    # 10
```

## 静态方法

### 1. 静态方法特点

- 需要通过装饰器 @staticmethod 来进行修饰, 静态方法既不需要传递类对象也不需要传递实例对象(形参没有 self/cls)
- 静态方法也能通过实例对象和类对象去访问。

### 2. 静态方法使用场景

- 当方法中既不需要使用实例对象(如实例对象, 实例属性), 也不需要使用类对象(如类属性、类方法、创建实例等)时, 定义静态方法
- 取消不需要的参数传递, 有利于减少不必要的内存占用和性能消耗

```
class Dog(object):
    __num = 10

    @staticmethod
    def info_print():
        print('这是一只狗! ')

laow = Dog()
laow.info_print()    # 这是一只狗!
Dog.info_print()     # 这是一只狗!
```

## 5. 总结

- 面向对象三大特点
  - 封装
  - 继承
  - 多态
- 类属性
  - 归属于类对象的属性, 所有对象共有的属性
- 实例属性
- 类方法

```
@classmethod
def xx():
    代码
```

- 静态方法

```
@staticmethod
def xx():
    代码
```

## 十、异常

### 1. 了解异常

当检测到一个错误时，解释器就无法执行了，反而出现了一些错误的提示，这就是所谓的异常

例：以 `r` 方式打开了一个不存在的文件

```
open('test.txt', 'r')    # r:只读
```

### 2. 异常的写法

#### 1. 语法

```
try:
    可能发生错误的代码
except:
    如果出现异常执行的代码
```

#### 2. 快速体验

需求：尝试以 `r` 模式打开文件，如果文件不存在，则以 `w` (只写/编辑)的方式打开

```
try:
    f= open('test.txt', 'r')
except:
    f= open('test.txt', 'w')
```

### 3. 捕获异常

捕获指定异常

#### 1. 语法：

```
try:
    可能发生错误的代码
except 异常类型:
    如果捕获到该异常类型执行的代码
```

#### 2. 快速体验

```
print(num) # NameError
```

```
try:
    print(num)
except NameError:
    print("错误!!")
```

#### 注意:

1. 如果尝试执行的代码的异常类型和要捕获的异常类型不一致，则无法捕获异常
2. 一般 `try` 下方只放一行尝试执行的代码

### 捕获多个指定异常

当捕获多个异常时，可以把要捕获的异常类型的名字，放到 `except` 后，并使用元组的方式进行书写

```
try:
    # print(num)
    print(1/0)
except (NameError, ZeroDivisionError):
    print("错误!!")
```

### 捕获异常描述信息

```
try:
    # print(num)
    print(1/0)
except (NameError, ZeroDivisionError) as e:
    print(e)
```

### 捕获所有异常

`Exception` 是所有程序异常类的父类

```
try:
    print(num)
except Exception as e:
    print(e)
```

## 4. 异常的 `else`

`else` 表示的是如果没有异常要执行的代码

```
try:
    print(1)
except Exception as e:
    print(e)
else:
    print('没有异常')
```

## 5. 异常的finally

finally 表示的是无论是否异常都要执行的代码，例如关闭文件

```
try:
    f = open('test.txt', 'r')
except Exception as e:
    f = open('test.txt', 'w')
else:
    print('没有异常')
finally:
    # 关闭文件.close()
    f.close()
```

## 6. 异常的传递

需求:

1. 尝试只读方式打开test.txt文件，如果文件存在则读取文件内容，文件不存在则提示用户即可。
2. 读取内容要求:尝试循环读取内容，读取过程中如果检测到用户意外终止程序，则 except 捕获异常并提示用户。

```
import time
try:
    f = open("test.txt")
    try:
        while True:
            content = f.readline() # 读取单行
            if len(content) == 0:
                break
            # 每读取一行延迟2秒
            time.sleep(2)
            print(content)
    except:
        print('程序被意外终止')
    finally:
        f.close()
except:
    print('没有这个文件')
```

## 7. 自定义异常

在Python中，抛出自定义异常的语法为 raise 异常类对象

需求：密码长度不足，则报异常(用户输入密码，如果输入的长度不足3位，则报错，即抛出自定义异常，并捕获该异常)。

```
# 1. 自定义异常类
class ShortInputError(Exception):
    def __init__(self, user_input_len, min_len):
        self.user_input_len = user_input_len
        self.min_len = min_len
    def __str__(self):
```

```

        return f'你输入的是{self.user_input_len}位字符，不能少于{self.min_len}位'
# 2.抛出异常
def user_input():
    try:
        con = input('输入你的密码: ')
        if len(con) < 3:
            raise ShortInputError(len(con), 3)
# 3.捕获该异常
    except ShortInputError as e:
        print(e)
    else:
        print('密码输入完成')

user_input()

```

## 8. 总结

- 异常语法

```

try:
    可能发生错误的代码
except:
    如果出现异常执行的代码
else:
    没有异常执行的代码
finally:
    无论是否异常都要执行的代码

```

- 捕获异常

```

except 异常类型:
    代码
except 异常类型 as xx:
    代码

```

- 自定义异常

```

# 1. 自定义异常类
class 异常类类名(Exception):
    代码
    # 设置抛出异常描述信息
    def __str__(self):
        return ...

# 2. 抛出异常
raise 异常类名()

# 捕获异常
except Exception...

```

## 十一、模块和包

# 1. 模块

Python 模块(Module), 是一个 Python 文件, 以 `.py` 结尾, 包含了 Python 对象定义和Python语句, 模块能定义函数, 类和变量, 模块里也能包含可执行的代码。

## 2. 导入模块

- `import` 模块名
- `from` 模块名 `import` 功能名
- `from` 模块名 `import*`
- `import` 模块名 `as` 别名
- `from` 模块名 `import` 功能名 `as` 别名

### 导入方式详解

#### 1. `import`

语法:

```
# 1. 导入模块
import 模块名 # 推荐
import 模块名1, 模块名2

# 2. 调用功能
模块名.功能名()
```

体验:

```
import math
# 开平方
print(math.sqrt(9)) # 3.0
```

#### 2. `from...import`

```
from 模块名 import 功能1...
```

体验:

```
from math import sqrt
print(sqrt(9))
```

#### 3. `from...import*`

语法

```
from 模块名 import * # 导入所有
```

体验:

```
from math import *
print(sqrt(9))
```

#### 4. as 定义别名

语法

```
# 模块定义别名
import 模块名 as 别名

# 功能定义别名
from 模块名 import 功能 as 别名
```

体验：

```
# 模块定义别名
import time as tt
tt.sleep(2)
print('hello world')

# 功能定义别名
from time import sleep as s1
s1(2)
print('hello world')
```

### 3. 制作模块

在Python中，每个Python文件都可以作为一个模块，模块的名字就是文件的名字。也就是说**自定义模块名必须要符合标识符命名规则**。

#### 1. 定义模块

新建来一个python文件，命名为 `my_module.py`，并定义 `testa` 函数

```
def testa(a,b):
    print(a + b)
```

#### 2. 测试模块

在实际开中，当一个开发人员编写完一个模块后，为了让模块能够在项目中达到想要的效果，这个开发人员会自行在 `.py` 文件中添加一些测试信息，例如，在 `my_module1.py` 文件中添加测试代码。

```
def testa(a,b):
    print(a + b)

# 测试信息
testa(1, 2)
```

此时，无论是当前文件，还是其他已经导入了该模块的文件，在运行的时候都会自动执行 `testA` 函数的调用。

解决办法如下：

```
def testA(a,b):
    print(a + b)

# 只在当前文件中调用该函数，其他导入的文件内不符合该条件，则不执行testA函数
if __name__ == '__main__':
    testA(2,2)
```

`__name__` 是系统变量，是模块的标识符，值是：如果是自身模块值是 `__main__`；否则是当前的模块名

```
print(__name__) # __main__
```

### 3. 模块的定位顺序

当导入一个模块，Python解析器对模块位置的搜索顺序是：

1. 当前目录
2. 如果不在当前目录，Python则搜索在 `shell` 变量 `PYTHONPATH` 下的每个目录。
3. 如果都找不到，Python会察看默认路径。UNIX 下，默认路径一般为 `/usr/local/lib/python/`

模块搜索路径存储在system模块的 `sys.path` 变量中。变量里包含当前目录，`PYTHONPATH` 和由安装过程决定的默认目录。

**注意：**

- 自己的文件名不要和已有的模块名重复，否则导致模块功能无法使用
- 使用 `from 模块名 import` 功能的时候，如果名字重复，调用到的是最后定义或导入的功能

## 4. `__all__`

如果一个模块文件中有 `__all__` 变量，当使用 `from xxx import *` 导入时，只能导入这个列表中的元素

- my\_module模块代码

```
__all__ = ['tastA']

def tastA():
    print('tastA')

def tastB():
    print('tastB')
```

- 导入模块的文件代码

```
from my_module import *

tastA()
tastB() # 报错
```

## 5. 包

包将有联系的模块组织在一起，即放到同一个文件夹下，并且在这个文件夹创建一个名字为 `__init__.py` 文件，那么这个文件夹就称之为包。

### 1. 制作包

[新建]-->[python软件包]-->输入包名-->[OK]-->新建功能模块(有联系的模块)

**注：**新建包后，包内部会自动创建 `__init__.py` 文件，这个文件控制着包的导入行为

### 2. 快速体验

1. 新建包 `mypackage`

2. 新建包内模块： `my_module`

3. 模块内代码如下

```
# my_module1
print(1)

def info_print1():
    print('my_module1')
```

```
# my_module2
print(2)

def info_print2():
    print('my_module2')
```

### 3. 导入包

方法一

```
import 包名.模块名

包名.模块名.功能
```

体验

```
import my.my_module

my.my_module.info_print1()
```

方法二

**注：**必须在 `__init__.py` 文件中添加 `__all__ = []`，控制允许导入的模块列表

```
from 包名 import *

模块名.功能
```

体验

```
# __init__.py
__all__ = ["my_module1"]
```

```
# my_module1.py
print(1)

def info_print1():
    print('my_module1')
```

```
# my_module2.py
print(2)

def info_print2():
    print('my_module2')
```

```
# 测试.py
from my import *
my_module1.info_print1()
```

## 6. 总结

- 导入模块方法

```
import 模块名

from 模块名 import 功能

from 模块名 import *
```

- 导入包

```
import 包名.模块名

from 包名 import *
```

- `__all__ = []`: 允许导入的模块或功能列表