■字面量

字面量就是代码中被写下来的，固定的值。

例如：12.5， “这是字符串”, -53 等等

Python中常用的6种值（数据）的类型

|  |  |  |
| --- | --- | --- |
| **类型** | **描述** | **说明** |
| 数字（Number） | 整数（int） | 整数（int） 如：10、 -10 |
| 浮点数（float） | 浮点数（float） 如：13.14、 -13.14 |
| 复数（complex） | 复数（complex） 如：4+3j 以j结尾表示复数 |
| 布尔（bool） | 布尔（bool）True(1)、 False(0) |
| 字符串（String） | 描述文本的数据类型 | 字符串由任意数量的字符组成 |
| 列表（List） | 有序的可变序列 | 可有序记录一堆数据集合 |
| 元组（Tuple） | 有序的不可变序列 | 可有序记录一堆不可变的数据集合 |
| 集合（Set） | 无序不重复集合 | 可无序记录一堆不重复的数据集合 |
| 字典（Dictionary） | 无序键值对集合 | 可无序记录一堆Key-Value型的数据集合 |

■注释

单行注释用井号加空格后跟注释的内容

多行注释用”””

# 这是单行注释  
  
"""  
  
这是多行注释  
这是多行注释  
  
"""

●pycharm注释快捷键为 ctel + /

■数据类型

●type() 查看数据类型

print(type("abc"))  
# 输出 <class 'str'>  
  
print(type(123))  
# 输出 <class 'int'>  
  
print(type(2.16))  
# 输出 <class 'float'>

●数据类型的转换

|  |  |
| --- | --- |
| **语句（函数）** | **说明** |
| Int(x) | 将x转换为一个整数 |
| float(x) | 将x转换为一个浮点数 |
| str(x) | 将x转换为字符串 |

■标识符

标识符是用户在编程时所使用的一些列名字，用来给变量，方法，类等命名。例如变量名，方法名，类的名字等等，都叫做标识符。

●标识符的命名规则

只允许出现四类：

英文、中文、数字、下划线

主意：

数字不可以开头

英文大小写敏感

不推荐使用中文

不能使用以下关键字

False, True, None, and, as, assert, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

●变量名的命名规范（规则必须遵守，规范推荐遵守）

英文字母全小写

使用下划线

见名知意

■运算符

数字运算符

|  |  |  |
| --- | --- | --- |
| **运算符** | **描述** | **实例** |
| + | 加 | 两个对象相加 a + b ，例如 1 + 1 输出2 |
| - | 减 | 得到负数或是一个数减去另一个数 a - b 或 -10 |
| \* | 乘 | 两个数相乘或是返回一个被重复若干次的字符串 a \* b |
| / | 除 | b / a, 例如 8/4 输出2 |
| // | 取整除 | 返回商的整数部分 9//2 ,例如 9.0 // 2.0 输出结果 4.0 |
| % | 取余 | 返回除法的余数 b % a，例如 8 // 3 输出结果 2 |
| \*\* | 指数 | a \*\* b 为a的b次方， 例如 10 \*\* 2 输出100 |

赋值运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| = | 赋值运算符 | 把 = 号右边的结果赋给左边的变量，例如 num = 1 + 2 \* 3 结果num被赋值7 |

复合赋值运算符

|  |  |  |
| --- | --- | --- |
| 运算符 | 描述 | 实例 |
| += | 加法赋值运算符 | c += a 等效于 c = c + a |
| -= | 减法赋值运算符 | c -= a 等效于 c = c - a |
| \*= | 乘法赋值运算符 | c \*= a 等效于 c = c \* a |
| /= | 除法赋值运算符 | c /= a 等效于 c = c / a |
| %= | 取摸赋值运算符 | c %= a 等效于 c = c % a |
| \*\*= | 幂赋值运算符 | c \*\*= a 等效于 c = c \*\* a |
| //= | 取整除赋值运算符 | c /= a 等效于 c = c // a |

■字符串

●字符串的定义

字符串的三种定义方式：

# 单引号定义法  
str1 = '字符串1'  
  
# 双引号定义法  
str2 = "字符串2"  
  
# 三引号定义法  
str3 = """  
字符串3a  
字符串3b  
字符串3c  
"""

●字符串引号的嵌套

# 单引号嵌套双引号  
str1 = '字符串1“字符串1”字符串1'  
  
# 双引号嵌套单引号  
str2 = "字符串2'字符串2'字符串2"  
  
# 使用转义字符\来嵌套  
str3a = "字符串3a\"字符串3a\"字符串3a"  
str3b = '字符串3b\'字符串3b\'字符串3b'

●字符串的拼接

用加号，用法基本和js一样，但是不同类型的数据和字符串拼接时需要先转换成字符串，否则会报错

例如

a = "金额"  
b = 22  
print(a + str(b))  
# 直接用print(a + b)则会报错

●字符串格式化

◎使用%占位

name = "张三"  
age = 22  
message = "我的名字是%s" % name  
message2 = "我的名字是%s，我的年龄是%s岁" % (name, age)  
print(message)  
# 输出：我的名字是张三  
print(message2)  
# 输出：我的名字是张三，我的年龄是22岁

Python支持很多数据类型占位，最常用的如以下三类：

|  |  |
| --- | --- |
| 格式符号 | 转化 |
| %s | 将内容转换成字符串，放入占位位置 |
| %d | 将内容转换成整数，放入占位位置 |
| %f | 将内容转换成浮点型，放入占位位置 |

name = "张三"  
age = 22  
money = 15.56  
message = "我的名字是%s，我的年龄是%d岁，我的存款有%f元" % (name, age, money)  
print(message)  
# 输出：我的名字是张三，我的年龄是22岁，我的存款有15.560000元

●字符串格式化的数字精度控制

可以使用辅助符号”m.n”来控制数据的宽度和精度。

m, 控制宽度，如设置的宽度小于数字自身则不生效。

.n, 控制小数点精度，会进行四舍五入

示例：

%5d 表示将整数的宽度控制在5位，如数字11，就会变成，空格+空格+空格+11，用三个空格补足宽度。

%7.2f 表示将浮点数的宽度控制为5位，将小数点精度设置为2 小数点和小数部分也算入宽度计算。如11.345 ，就会变成，空格+空格+11.35，用两个空格补足宽度，小数部分限制2位精度后，四舍五入。

price1 = 11  
price2 = 11.345  
message = "苹果的价格是%5d元每斤，桔子的价格是%7.2f元每斤。" % (price1, price2)  
  
print(message)  
# 输出：苹果的价格是 11元每斤，桔子的价格是 11.35元每斤。

●字符串格式化的快速方法

格式：

f”内容{变量}”

注：快速方式没有精度控制，原样输出。

name = "张三"  
age = 22  
money = 16.7898  
  
message = f"我的名字是{name}，我的年龄是{age}岁，我的存款有{money}元。"  
print(message)  
# 输出：我的名字是张三，我的年龄是22岁，我的存款有16.7898元。

●在无需使用变量进行数据存储的时候，可以直接格式化表达式，简化代码

◎

表达式：一条具有明确执行结果的代码语句

◎示例：

print('我买了两个苹果，每个三元，一共花了%d元。' % (2 \* 3))  
# 输出：我买了两个苹果，每个三元，一共花了6元。  
  
print('一万块钱花一年，平均每天能花%.3f元。' %(10000 / 365))  
# 输出：一万块钱花一年，平均每天能花27.397元。  
  
print('字符串在python中的类型是%s' % (type('abc')))  
# 输出：字符串在python中的类型是<class 'str'>

■print语句

输出，多条print一般默认为换行。加上end=’’则为不换行

print('a')  
print('b')  
print('c', end='')  
print('d', end='')  
  
"""  
输出结果为：  
a  
b  
cd  
"""

■制表符

◎\t

在字符串中，有一个特殊符号 \t 效果等同于在键盘上按下 tab键。它可以让多行字符串进行对齐。

print('tom is cool')  
print('alice is cool')  
"""  
输出：  
tom is cool  
alice is cool  
"""  
  
print('tom \tis cool')  
print('alice \tis cool')  
  
"""  
输出：  
tom is cool  
alice is cool  
"""

◎换行用\n

print('第一行\n第二行\n第三行')  
  
"""  
输出：  
第一行  
第二行  
第三行  
"""

■input语句

# input获取的键盘输入信息是字符串，如需要其它类型需要进行转换  
a = input('请输入金额！')  
print(type(a))  
print('输入的金额是：' + a)  
  
"""  
结果：  
请输入金额！<class 'str'>  
输入的金额是：22  
"""

■if 语句

●bool类型

Python的布尔类型开头要大写

True

False

●比较运算符

比较运算得到的结果是bool值

Python的比较运算符有以下六种：

|  |  |
| --- | --- |
| 运算符 | 描述 |
| == | 判断内容是否相等，是为True，否为False |
| != | 判断内容是否不相等，是为True，否为False |
| > | 判断运算符左侧内容是否大于右侧，是为True，否为False |
| < | 判断运算符左侧内容是否小于右侧，是为True，否为False |
| >= | 判断运算符左侧内容是否大于等于右侧，，是为True，否为False |
| <= | 判断运算符左侧内容是否小于等于右侧，，是为True，否为False |

●if语句基本用法

age = 19  
  
if age >= 18:  
 print('我成年了！')  
 print('即将迎来大学生活！')

●if else的用法

age = int(input('请输入年龄：'))  
  
if age >= 18:  
 print('我成年了！')  
 print('即将迎来大学生活！')  
else:  
 print('哦哈，我还小。')  
 print('时间过的好慢。')

●else if 的用法

age = int(input('请输入年龄：'))  
level = int(input('请输入等级：'))  
mark = int(input('请输入分数：'))  
  
if age >= 18:  
 print('您的年龄符合要求。')  
 # 如果这个条件成立，则不执行下面的elif和else判断  
elif level > 5:  
 print('您的等级符合要求')  
elif mark > 80:  
 print('您的分数符合要求')  
else:  
 print('对不起，您所有指标均不合格。')

●if 语句的嵌套使用

age = int(input('请输入年龄：'))  
  
if age >= 18:  
 print('您的年龄符合要求，还需要验证您的vip等级。')  
 # 当满足年龄判断后，才会出现第二个input输入  
 if int(input('请输入等级：')) > 5:  
 print('您的等级合格。')  
 else:  
 print('您的等级不合格。')  
else:  
 print('您的年龄不符合要求。')

■while循环

●while循环的基本用法

i = 0  
  
while i < 10:  
 print(i)  
 i += 1  
  
# 输出结果，打印了0-9

案例1，求1到100累加的和

sum = 0  
i = 1  
  
while i <= 100:  
 sum += i  
 i += 1  
  
print(f"1到100累加的和是：{sum}")

# 输出结果：1到100累加的和是：5050

案例2

# 获取范围在1-100的随机数字  
import random  
num = random.randint(1, 100)  
  
# 定义一个变量，记录总共猜测了多少次  
count = 0  
  
# 通过一个布尔类型的变量，做循环时否继续的标记  
flag = True  
while flag:  
 guess\_num = int(input("请输入你猜测的数字："))  
 count += 1  
 if guess\_num == num:  
 print("猜中了！")  
 # 将flag设置为False终止循环  
 flag = False  
 else:  
 if guess\_num > num:  
 print("你猜的大了！")  
 else:  
 print("你猜的小了！")  
  
print(f"你总共猜测了{count}次")

●While循环的嵌套使用

案例：每天给小美送10只花之后表白一次，循环100天后追求成功。

# 外层循环  
i = 1  
while i <= 100:  
 print(f"今天是第{i}天，准备表白......")  
  
 # 内层循环  
 j = 1  
 while j <= 10:  
 print(f"送给小美第{j}只玫瑰花！")  
 j += 1  
  
 print("小美，我喜欢你！")  
 i += 1  
  
print(f"坚持到第{i - 1}天，追求成功！")

●综合练习

绘制九九乘法表：

# 外层循环  
i = 1  
while i <= 100:  
 print(f"今天是第{i}天，准备表白......")  
  
 # 内层循环  
 j = 1  
 while j <= 10:  
 print(f"送给小美第{j}只玫瑰花！")  
 j += 1  
  
 print("小美，我喜欢你！")  
 i += 1  
  
print(f"坚持到第{i - 1}天，追求成功！")

■for循环

For循环也叫遍历循环。主要用于遍历序列

◎while和for循环的区别

while循环的循环条件是自定义的，自行控制循环条件

for循环时一种“轮询”机制，是对一批内容进行“逐个处理”

●for循环的基本用法

for处理的数据集通常为序列。

序列类型指，其内容可以一个个依次取出的一种类型，包括：

字符串

列表

元组

以下例子使用for循环遍历一个字符串

name = 'abcdefg'  
for x in name:  
 print(x)  
"""  
输出结果：  
a  
b  
c  
d  
e  
f  
g  
  
"""

●range语句

range语句用于获得一个数字序列

# range语法1，获取一个从0开始，到5结束的数字序列，不包含5本身  
for x in range(5):  
 print(x)  
# 输出：01234  
  
# range语法2，获取一个从5开始到10结束的数字序列，不包含10本身  
for x in range(5, 10):  
 print(x)  
# 输出：56789  
  
# range语法3，获取一个从5开始到10结束的数字序列，不包含10本身，数字之间步长为2  
# 第三个参数，步长，不传参默认为1  
for x in range(5, 10, 2):  
 print(x)  
# 输出：579

●for循环临时变量的作用域

for i in range(5): # i 为for循环的临时变量，其作用范围（作用域）只限定在for循环内部  
 print(i)  
  
print(i) # 在for循环外部访问i时，虽然不报错，但编程规范上不允许，不建议这么做。  
  
"""  
输出：  
0  
1  
2  
3  
4  
4  
"""

解决方法，先定义一个变量i

i = 0  
for i in range(5):  
 print(i)  
  
print(i)  
  
"""  
输出：  
0  
1  
2  
3  
4  
4  
"""

●for循环的嵌套使用

案例：每天给小美送10只花之后表白一次，循环100天后追求成功。

i = 1  
for i in range(1, 101):  
 print(f"今天是向小美表白的第{i}天，坚持！")  
 for j in range(1, 11):  
 print(f"送给小美的第{j}朵玫瑰花")  
 print(f"小美，我喜欢你（第{i}天的表白结束）")  
  
print(f"第{i}天，追求成功！")

案例：用for循环打印99乘法表

# 通过外层循环控制函数  
for i in range(1, 10):  
 # 通过内层循环控制每一行的数据  
 for j in range(1, i + 1):  
 # 在内层循环中输出每一行的内容  
 print(f"{j} \* {i} = {j \* i}\t", end='')  
  
 # 外层循环可以通过print输出一个回车符  
 print()

●break和continue关键字控制循环

◎continue

可以用于for和while循环

作用为中断本次循环，直接进入下一次循环

for i in range(1, 6):  
 print('a')  
 continue  
 print('b') # 这行代码不会被执行  
  
"""  
输出：  
a  
a  
a  
a  
a  
"""

◎break

可以用于for和while循环

作用为直接结束循环

for i in range(1, 6):  
 print('a')  
 break  
 print('b')  
  
  
"""  
输出：  
a  
"""

●综合案例

公司账户有1万元，给20名员工发工资。

判断员工绩效分（1-10分），低于5则不发工资。大于等于5则发放1000元。

工资发完为止。

money = 10000  
# for循环对员工发放工资  
for i in range(1, 21):  
 import random  
 score = random.randint(1, 10)  
  
 if score < 5:  
 print(f"员工{i}绩效分{score},不满足，不发工资，下一位")  
 # continue跳过发放  
 continue  
  
 # 要判余额足不足  
 if money >= 1000:  
 money -= 1000  
 print(f"员工{i},满足条件发放工资1000，工资账户余额：{money}")  
 else:  
 print(f"余额不足，当前余额：{money}元，不足以发工资。")  
 # break结束发放  
 break

■函数

函数：是组织好的，可重复使用的，用来实现特定功能的代码段。

函数有内置函数和自定义函数。

Python的内置函数，比如len()用来计算长度

a = 'abcde'  
print(len(a))  
  
# 输出结果：5

●函数基本的用法

def 函数名（传入参数）：

函数体

Return 返回值

参数如不需要，可以省略。

返回值如不需要，可以省略。

函数必须先定义，后使用。

函数的基本定义和执行方法：

# 定义一个函数  
def say\_hi():  
 print('hello!')  
  
# 执行函数  
say\_hi()

●函数的参数

参数的作用是，在函数运行时，接受外部传入的数据。

函数定义中的参数，称为形式参数（形参）

函数调用中的参数，称为实际参数（实参）

函数的参数不限数量，使用逗号分隔开。

实参要和形参的位置一一对应。

def add(x, y):  
 print(f'计算结果为：{x + y}')  
  
add(2, 5)  
  
# 输出为：  
# 计算结果为：7

●函数的返回值

函数的返回值，是函数在执行完成后，返回给调用者的结果。

函数体在遇到return后就结束了，return后的代码不会执行。

def add(x, y):  
 result = x + y  
 return result  
 print('我不会被执行。')  
  
  
r = add(3, 5)  
print(r)  
  
# 输出为：  
# 计算结果为：8， return后的print不被执行

●None

◎

Python中有一个特殊的字面量：None，其类型是：<class ‘NoneType’>

None 表示空的，无实际意义的意思。

◎

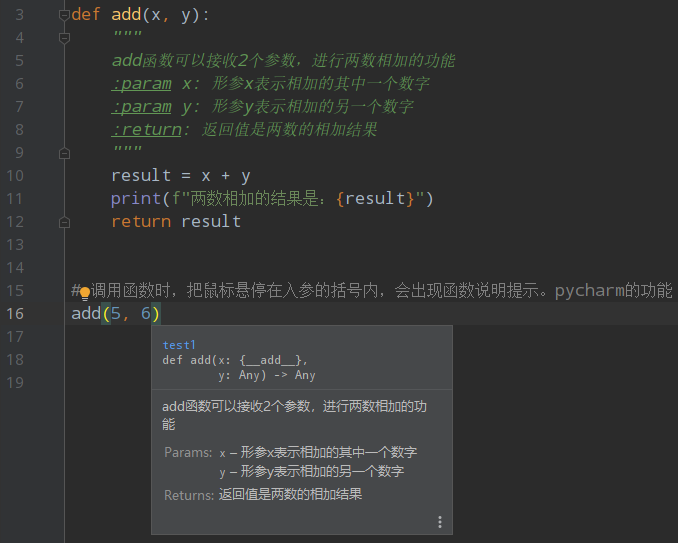
无返回值的函数，实际上返回的是None

def say\_hello():  
 print("Hello...")  
  
# 使用变量接收say\_hello函数的返回值  
result = say\_hello()  
  
print(result)  
# 输出：None  
print(type(result))  
# 输出：<class 'NoneType'>  
  
  
# 也可以主动使用return返回None  
def say\_hello2():  
 print("Hello2...")  
 return None  
  
result2 = say\_hello2()  
  
print(result2)  
# 输出：None  
print(type(result2))  
# 输出：<class 'NoneType'>

●函数的说明文档

在函数内部，函数体的开头，添加多行注释。格式如下

def add(x, y):  
 *"""  
 add函数可以接收2个参数，进行两数相加的功能* ***:param*** *x: 形参x表示相加的其中一个数字* ***:param*** *y: 形参y表示相加的另一个数字* ***:return****: 返回值是两数的相加结果  
 """* result = x + y  
 print(f"两数相加的结果是：{result}")  
 return result  
  
  
# 调用函数时，把鼠标悬停在入参的括号内，会出现函数说明提示。pycharm的功能  
add(5, 6)



●函数的嵌套

函数a中执行到调用函数b的语句，会将函数b全部执行完成后，再继续执行函数a的剩余内容。

def b():  
 print('b')  
  
  
def a():  
 print(1)  
 b()  
 print(2)  
  
  
a()  
"""  
输出：  
1  
b  
2  
"""

●变量的作用域

变量作用域指的是变量的作用范围（变量在哪里可用，在哪里不可用）

变量作用域主要分为两种：局部变量和全局变量

局部变量是定义在函数体内的变量，其只在函数体内部生效。

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

def testa():  
 num = 100  
 print(num)  
  
  
testa()  
# 输出：100  
print(num)  
# 报错：NameError: name 'num' is not defined.因为num是函数testa的局部变量，在函数外无法访问

●global关键字

num = 200  
  
  
def test\_a():  
 print(f"函数test\_a内部访问全局变量num的值是:{num}")  
  
  
def test\_b():  
 num = 500  
 print(f"函数test\_b定义了局部变量num,因此内部访问的num是局部变量，值是：{num}")  
  
  
test\_a()  
# 输出：函数test\_a内部访问全局变量num的值是:200  
test\_b()  
# 输出：函数test\_b定义了局部变量num,因此内部访问的num是局部变量，值是：500  
print(num)  
# 输出：200  
# test\_b内部定义的局部变量num，不会影响全局变量，因此全局变量num的值仍然是200

要想在函数内部改变全局变量，可以用global关键字

num = 200  
  
  
def test\_a():  
 print(f"函数test\_a内部访问全局变量num的值是:{num}")  
  
  
def test\_b():  
 global num  
 num = 500  
 print(f"函数test\_b使用global改变了全局变量num,值是：{num}")  
  
  
test\_a()  
# 输出：函数test\_a内部访问全局变量num的值是:200  
test\_b()  
# 输出：函数test\_b定义了局部变量num,因此内部访问的num是局部变量，值是：500  
print(num)  
# 输出：500  
# test\_b内部使用global改变了全局变量num，因此全局变量num的值是500

●函数的综合案例

# 定义全局变量  
money = 5000000  
name = None  
  
# 要求客户输入姓名  
name = input("请输入您的姓名：")  
  
  
# 定义查询函数  
def query(show\_header):  
 if show\_header:  
 print("---查询余额---")  
 print(f"{name}，您好，您的余额为：{money}元")  
  
  
# 定义存款函数  
def saving(num):  
 global money  
 money += num  
 print("---存款---")  
 print(f"{name}，您好，您的存款{num}元成功。")  
  
 # 调用query函数查询余额  
 query(False)  
  
  
# 定义取款函数  
def get\_money(num):  
 global money  
 money -= num  
 print("---取款---")  
 print(f"{name}，您好，您取款{num}元成功。")  
  
 # 调用query函数查询余额  
 query(False)  
  
  
# 定义主菜单函数  
def main():  
 print("---主菜单---")  
 print(f"{name}，您好，欢迎来到本银行ATM。请选择操作：")  
 print("查询余额\t[输入1]")  
 print("存款\t[输入2]")  
 print("取款\t[输入3]")  
 print("退出\t[输入4]")  
 return input("请输入您的选择：")  
  
  
# 设置无限循环，确保程序不退出  
while True:  
 keyboard\_input = main()  
 if keyboard\_input == "1":  
 query(True)  
 continue  
 elif keyboard\_input == "2":  
 num = int(input("您想要存多少钱？请输入："))  
 saving(num)  
 continue  
 elif keyboard\_input == "3":  
 num = int(input("您想取多少钱？请输入："))  
 get\_money(num)  
 continue  
 else:  
 print("程序退出啦")  
 break

■数据容器

一种可以容纳多份数据的数据类型，容纳的每一份数据称之为1个元素。每一个元素，可以是任意类型的数据，如字符串、数字、布尔等。

数据容器分为5类：列表（list）、元组（tuple）、字符串（str）、集合（set）、字典（dict）

■列表list

●列表的特点：

可以容纳多个元素

可以容纳不同类型的元素（混装）

数据是有序存储的（有下标序号）

允许重复数据存在

可以修改（增加或删除元素等）

●list的定义

List的定义方法和js大致相同，定义空list时还可以用list()

list\_1 = []  
list\_2 = list()  
list\_3 = ['abc', 56, "速的"]  
  
print(type(list\_1))  
print(type(list\_2))  
print(type(list\_3))  
  
"""  
输出结果：  
<class 'list'>  
<class 'list'>  
<class 'list'>  
"""

●list的下标索引

list\_3 = ['abc', 56, "速的"]  
  
# 正向索引，同JS方法  
print(list\_3[0])  
print(list\_3[1])  
print(list\_3[2])  
  
"""  
输出结果：  
abc  
56  
速的  
"""  
  
# 反向索引，用-1表示最后一个元素，-2表示倒数第二个元素，以此类推  
print(list\_3[-1])  
print(list\_3[-2])  
print(list\_3[-3])  
  
"""  
输出结果：  
速的  
56  
abc  
"""

嵌套的索引

list\_5 = [[1, 2, 4], [4, 5, 6]]  
  
print(list\_5[1][2])  
  
# 输出结果为6

●列表的方法

◎方法的概念

函数是一个封装的代码单元，可以提供特定功能。

在Python中，如果将函数定义为class(类)的成员，那么函数会被称为，方法。

方法和函数一样，有传入参数，有返回值，只是方法的使用格式不同。

def add(x, y):  
 return x + y  
  
class Student:  
 def add2(self, x, y):  
 return x + y  
  
# 如上，add是一个函数，add2是Student类的一个方法。  
  
  
# 函数的调用方法  
print(add(1, 2))  
# 输出：3  
  
# 方法的调用方法  
student = Student() # 先将Student类实例化成一个实例对象并存入student变量  
print(student.add2(1, 2)) # 使用实例化对象加点加方法名的方式调用  
# 输出：3

◎列表的方法

|  |  |  |
| --- | --- | --- |
| **编号** | **使用方式** | **作用** |
| 1 | 列表.append（元素） | 向列表中追加一个元素 |
| 2 | 列表.extend（容器） | 将数据容器的内容依次取出，追加到列表尾部 |
| 3 | 列表.insert（下标, 元素） | 在指定下标处，插入指定的元素 |
| 4 | del 列表[下标] | 删除列表指定下标元素 |
| 5 | 列表.pop（下标） | 删除列表指定下标元素 |
| 6 | 列表.remove（元素） | 从前向后，删除此元素第一个匹配项 |
| 7 | 列表.clear() | 清空列表 |
| 8 | 列表.count（元素） | 统计此元素在列表中出现的次数 |
| 9 | 列表.index（元素） | 查找指定元素在列表的下标，找不到报错ValueError |
| 10 | len（列表） | 统计容器内有多少元素 |

mylist = ['元素1', '元素2', '元素3']  
  
  
# .index方法查询元素的下标  
# 如查找的元素不存在，会报错  
  
index = mylist.index('元素2')  
print(index)  
# 输出结果为1  
  
  
# .insert方法插入元素  
# .insert(下标, 元素) 在指定的下标位置插入指定的元素，原先位置的元素会后移  
  
mylist.insert(2, '元素2b')  
print(mylist)  
# 输出结果为['元素1', '元素2', '元素2b', '元素3']  
  
  
# .append方法在列表末尾追加一个元素  
mylist.append('元素4')  
print(mylist)  
# 输出结果为['元素1', '元素2', '元素2b', '元素3', '元素4']  
  
  
# .expend方法在列表末尾追加一批元素  
mylist.extend(['元素5', '元素6', '元素7'])  
print(mylist)  
# 输出结果为['元素1', '元素2', '元素2b', '元素3', '元素4', '元素5', '元素6', '元素7']  
  
  
# del删除列表的一个元素  
del mylist[2]  
print(mylist)  
# 输出结果为['元素1', '元素2', '元素3', '元素4', '元素5', '元素6', '元素7']  
  
# .pop删除列表的一个元素，并得到这个元素的返回值  
item6 = mylist.pop(6)  
print(mylist)  
print(item6)  
"""  
输出结果为：  
['元素1', '元素2', '元素3', '元素4', '元素5', '元素6']  
元素7  
"""  
  
# .remove删除指定元素，从前到后搜索列表，将找到的第一个匹配元素删除  
mylist.remove('元素4')  
print(mylist)  
# 输出结果为['元素1', '元素2', '元素3', '元素5', '元素6']  
  
# .clear清空整个列表  
mylist.clear()  
print(mylist)  
# 输出结果为[]  
  
# .count统计列表内指定元素的数量  
mylist3 = [1, 2, 3, 4, 5, 3]  
print(mylist3.count(3))  
# 输出结果为2（被统计的元素3在列表中有2个）  
  
# len函数统计列表总共的长度  
print(len(mylist3))  
# 输出结果为6

●用while循环来遍历列表

将容器内的元素依次取出进行处理的行为，称之为遍历，或者迭代。

my\_list = [1, 2, 3, 4, 5]  
  
index = 0  
while index < len(my\_list):  
 element = my\_list[index]  
 print(f"列表的元素：{element}")  
 index += 1  
  
"""  
输出：  
列表的元素：1  
列表的元素：2  
列表的元素：3  
列表的元素：4  
列表的元素：5  
"""

●用for循环来遍历列表

my\_list = [1, 2, 3, 4, 5]  
  
for element in my\_list:  
 print(f"列表的元素：{element}")  
  
"""  
输出：  
列表的元素：1  
列表的元素：2  
列表的元素：3  
列表的元素：4  
列表的元素：5  
"""

◎while循环和for循环的对比

在循环控制上：

while循环可以自定循环条件，并自行控制。

for循环不可以自定循环条件，只可以一个个从容器内取出数据。

在无限循环上：

while循环可以通过条件控制做到无限循环。

for循环理论上不可以，因为被遍历的容器不是无限的。

在使用场景上：

while循环适用于任何想要循环的场景。

for循环适用于遍历数据容器的场景或简单的固定次数循环场景。

■元组

元组和列表的区别是：

列表可以修改，元组不能修改。

可以理解为元组就是只读的list

◎元组的定义

元组的定义使用小括号，使用逗号隔开各个元素，元素可以是不同的数据类型。

t1 = (1, 2, 3, 4, 5)  
t2 = ()  
t3 = tuple()  
  
print(type(t1))  
print(type(t2))  
print(type(t3))  
# 均输出：<class 'tuple'>

# 定义只有一个元素的元组，直接定义会被默认类型为字符串。需要加一个逗号  
a = ('abc')  
print(type(a))  
# 输出结果为<class 'str'>  
b = ('abc', )  
print(type(b))  
# 输出结果为<class 'tuple'>

◎获取元组内的元素，方法和列表一样

t1 = ((1, 2, 3), (4, 5, 6))  
print(t1[1][2])  
# 输出结果为6

◎元组的操作。

由于元组不可修改，因此元组的操作要比列表少很多。

|  |  |  |
| --- | --- | --- |
| **编号** | **方法** | **作用** |
| 1 | index() | 查找某个数据，如果数据存在返回对应的下标，否则报错 |
| 2 | count() | 统计某个数据在元组出现的次数 |
| 3 | len(元组) | 统计元组内的元素个数 |

# 根据下标索引取出数据  
t1 = (1, 'hi', 3)  
print(t1[1])  
# 输出：hi  
  
  
# 根据index()查找特定元素的第一个匹配项  
t2 = (1, 2, 'hi', 4, 5, 'hi')  
print(t2.index('hi'))  
# 输出：2  
  
  
# 统计某个元素在元组内出现的次数  
t3 = (1, 2, 'hi', 4, 5, 'hi')  
print(t3.count('hi'))  
# 输出：2  
  
  
# 统计元组内的元素个数  
print(len(t3))  
# 输出：6

■字符串

字符串是字符的容器

●字符串也可以用下标来取值

s1 = "abcde"  
print(s1[2])  
# 输出结果为c

●字符串是无法修改的

无法通过指定下标或者追加等方式进行修改。

s1 = "abcde"  
print(s1[2])  
# 输出结果为c  
  
s1[2] = 'f'  
# 修改下标2会报错

●字符传的方法

|  |  |  |
| --- | --- | --- |
| 编号 | 操作 | 说明 |
| 1 | 字符串[下标] | 根据下标索引取出特定位置字符 |
| 2 | 字符串.index(字符串) | 查找给定字符的第一个匹配的下标 |
| 3 | 字符串.replace(字符串1, 字符串2) | 将字符串内的全部字符串1，替换为字符串2  不会修改元字符串，而是得到一个新字符串 |
| 4 | 字符串.split(字符串) | 按照给定字符串，对字符串进行分隔  不会修改元字符串，而是得到一个新字符串 |
| 5 | 字符串.strip() | 移除收尾的空格和换行符或指定字符串 |
| 6 | 字符串.count(字符串) | 统计字符串内某字符串的出现次数 |
| 7 | Len(字符串) | 统计字符串的字符个数 |

◎.index

# .index方法获取元素的位置  
  
s1 = "abcde"  
  
print(s1.index('c'))  
# 输出结果为s1中c的位置 2  
  
s2 = "i am c a chinese"  
print(s2.index('chinese'))  
# 输出结果为s2中chinese的起始位置 9

◎.replace

s1 = 'i am chinese'  
s2 = s1.replace('chinese', 'american')  
print(s1)  
# 输出为 i am chinese 因为s1不能被修改  
print(s2)  
# 输出为新的字符串s2 i am american

◎.split

s1 = 'i a\*m chinese'  
s2 = s1.split()  
print(s2)  
# split不带参数，默认为按照空格截取  
# 输出结果为['i', 'a\*m', 'chinese']  
  
s3 = s1.split('\*')  
print(s3)  
# split带参数，则按参数指定的字符截取  
# 输出结果为['i a', 'm chinese']

◎.strip

my\_str = " itheima and itcast "  
print(my\_str.strip())  
# 输出：itheima and itcast  
  
my\_str2 = "12itheima and itcast21"  
print(my\_str2.strip("12"))  
# 注意，传入的是12，其实就是“1”和“2”都会被移除，是按单个字符。  
# 输出：itheima and itcast

◎count和len

s1 = 'abcadeafg'  
print(s1.count('a'))  
# 输出3（字符串s1中字符a出现的次数）  
print(len(s1))  
# 输出9（字符串s1的长度）

●字符串的遍历

my\_str = 'abcde'  
  
# 使用while循环遍历  
index = 0  
while index < len(my\_str):  
 print(my\_str[index])  
 index += 1  
# 输出：abcde  
  
  
# 使用for循环遍历  
for i in my\_str:  
 print(i)  
# 输出：abcde

■序列的切片

●序列包括：列表、元组、字符串

序列是指：内容连续、有序，可使用下标索引的一类数据容器。

列表、元组、字符串，均可以视为序列。

●

切片：从一个序列中，取出一个子序列。

序列支持切片，即：列表、元组、字符串，均支持切片操作

切片语法：

序列[起始下标: 结束下标: 步长]

表示从序列中，从指定位置开始，依次取出元素，到指定位置结束，得到一个新序列：

起始下标表示从何处开始，可以留空，留空视作从头开始。

结束下标（不含）表示何处结束，可以留空，留空视作截取到结尾。

步长表示，依次取元素的间隔。步长为负数表示反向取。

# 对list进行切片，从1开始，3结束，步长1  
my\_list = [0, 1, 2, 3, 4, 5, 6]  
result1 = my\_list[1:4] # 步长默认是1，所以可以省略不写  
print(f"结果1：{result1}")  
# 输出：结果1：[1, 2, 3]  
  
  
# 对tuple进行切片，从头开始，到最后结束，步长1  
my\_tuple = (0, 1, 2, 3, 4, 5, 6)  
result2 = my\_tuple[:] # 起始和结束不写表示从头开始到尾。步长1也省略  
print(f"结果2：{result2}")  
# 输出：结果2：(0, 1, 2, 3, 4, 5, 6)  
  
  
# 对str进行切片，从头开始，最后结束，步长2  
my\_str = "0123456"  
result3 = my\_str[::2]  
print(f"结果3：{result3}")  
# 输出：结果3：0246  
  
  
# 对str进行切片，从头到尾，步长-1  
result4 = my\_str[::-1]  
print(f"结果4：{result4}")  
# 输出：结果4：6543210  
  
  
# 对list进行切片，从3开始，到1结束，步长-1  
result5 = my\_list[3:1:-1]  
print(f"结果5：{result5}")  
# 输出：结果5：[3, 2]  
  
  
# 对tuple进行切片，从头到尾，步长-2  
result6 = my\_tuple[::-2]  
print(f"结果6：{result6}")  
# 输出：结果6：(6, 4, 2, 0)

■集合

●集合的内容无需，不重复

●集合的定义

set1 = {'a', 'b', 'a', 'c', 'd', 'e'}  
set2 = {} # 空集合只能用set()定义，如果是{},则是空字典  
set3 = set()  
  
print(type(set1))  
# 输出：<class 'set'>  
print(type(set2))  
# 输出：<class 'dict'>  
print(type(set3))  
# 输出：<class 'set'>  
  
print(set1)  
# 输出：{'b', 'a', 'e', 'c', 'd'}  
# 可以看出是无序的存储  
# 重复的'a'被去重了

●集合的操作

◎因为集合是无序的，所以不支持下标访问

◎.add添加元素

set1 = {'a', 'b', 'c'}  
  
set1.add('a') # 添加已有的元素，无效  
set1.add('d')  
  
print(set1)  
# 输出：{'c', 'a', 'd', 'b'}

◎.remove移除元素

set1 = {'a', 'b', 'c'}  
  
set1.remove('b')  
  
print(set1)  
# 输出：{'a', 'c'}

◎.pop

# 集合.pop()，从集合中随机取出一个元素  
# 会得到这个元素的结果  
# 集合本身被修改，该元素被移除  
  
  
set1 = {'a', 'b', 'c'}  
  
item1 = set1.pop()  
  
print(set1)  
# 输出：{'b', 'c'}  
  
print(item1)  
# 输出：a

◎.clear清空集合

set1 = {'a', 'b', 'c'}  
  
set1.clear()  
  
print(set1)  
# 输出：set()

◎.difference取两个集合的差集

# 语法：集合1.difference(集合2)  
# 功能：取出集合1和集合2的差集（集合1有而集合2没有的），生成一个新集合。原集合1和集合2不变  
  
set1 = {1, 2, 3}  
set2 = {1, 5, 6}  
set3 = set1.difference(set2)  
  
print(set3)  
# 得到新集合set3  
# 输出：{2, 3}  
print(set1)  
# 原set1集合不变  
# 输出：{1, 2, 3}  
print(set2)  
# 原set2集合不变  
# 输出：{1, 5, 6}

◎.difference\_update消除两个集合的差集

# 语法：集合1.difference\_update(集合2)  
# 功能：对比集合1和集合2，在集合1内删除和集合2相同的元素。集合1被修改，集合2不变。  
  
set1 = {1, 2, 3}  
set2 = {1, 5, 6}  
set1.difference\_update(set2)  
  
print(set1)  
# 输出：{2, 3}  
print(set2)  
# 集合set2不变  
# 输出：{1, 5, 6}

◎.union合并两个集合

# 语法：集合1.union(集合2)  
# 功能：将集合1和集合2组合成新集合。集合1和集合2不变  
  
set1 = {1, 2, 3}  
set2 = {1, 5, 6}  
set3 = set1.union(set2)  
  
print(set3)  
# 输出：{1, 2, 3, 5, 6}  
print(set1)  
# 原set1不变  
# 输出：{1, 2, 3}  
print(set2)  
# 原set1不变  
# 输出：{1, 5, 6}

◎len()获取集合内元素个数

set1 = {1, 2, 3}  
num = len(set1)  
  
print(num)  
# 输出：3

●集合的遍历

集合不支持下标，因此不能用while循环遍历。但是可以用for循环遍历

set1 = {1, 2, 3, 4, 5}  
for element in set1:  
 print(element)  
  
  
"""  
输出：  
1  
2  
3  
4  
5  
"""

集合操作汇总

|  |  |  |
| --- | --- | --- |
| 编号 | 操作 | 说明 |
| 1 | 集合.add（元素） | 集合内添加一个元素 |
| 2 | 集合.remove（元素） | 移除集合内指定元素 |
| 3 | 集合.pop() | 从集合中随机取出一个元素 |
| 4 | 集合.clear() | 将集合清空 |
| 5 | 集合1.difference（集合2） | 得到一个新集合，内含2个集合的差集，原有的两个集合不变 |
| 6 | 集合1.difference\_update（集合2） | 在集合1中删除集合2中存在的元素，集合1被修改，集合2不变 |
| 7 | 集合1.union（集合2） | 得到1个新集合，内含2个集合的全部元素，原有的两个集合不变 |
| 8 | len（集合） | 得到一个整数，记录集合的元素数量 |

■字典

●字典的定义

字典存储的元素是一个个的：键值对

dict1 = {'tom': 22, 'jerry': 17, 'alice': 19, 'tom': 23}  
dict2 = {}  
dict3 = dict()  
  
print(type(dict1))  
# 输出：<class 'dict'>  
print(type(dict2))  
# 输出：<class 'dict'>  
print(type(dict3))  
# 输出：<class 'dict'>  
print(dict1)  
# 重复的键被去重，值取的是后一个的值，两个tom,取的后一个23的  
# 输出：{'tom': 23, 'jerry': 17, 'alice': 19}

●通过key来取字典内的元素的值

字典和集合一样，不可以使用下标索引。但是字典可以通过key来取得对应的value

dict1 = {'tom': 22, 'jerry': 17, 'alice': 19, 'tom': 23}  
  
print(dict1['jerry'])  
# 输出：17

●字典的嵌套

dict1 = {  
 "张三": {  
 "语文": 99,  
 "数学": 98,  
 "英语": 97  
 }, "李四": {  
 "语文": 89,  
 "数学": 88,  
 "英语": 87  
 }, "王五": {  
 "语文": 98,  
 "数学": 97,  
 "英语": 96  
 },  
}  
  
print(dict1['李四']['数学'])  
# 输出：88

●字典的操作

◎新增和更新

dict1 = {"tom": 22, "jerry": 18}  
  
print(dict1)  
# 输出：{'tom': 22, 'jerry': 18}  
  
# 新增元素  
dict1['alice'] = 19 # 新增一个元素  
print(dict1)  
# 输出：{'tom': 22, 'jerry': 18, 'alice': 19}  
  
# 更新元素  
dict1['jerry'] = 17 # 修改一个元素的值  
print(dict1)  
# 输出：{'tom': 22, 'jerry': 17, 'alice': 19}

◎.pop删除元素

# 语法：字典.pop(key)  
# 结果：获得指定key的value，原字典被修改，指定key的数据被删除  
  
dict1 = {"tom": 22, "jerry": 18, "alice": 19}  
value = dict1.pop("jerry")  
  
print(value)  
# 输出：18  
print(dict1)  
# 输出：{'tom': 22, 'alice': 19}

◎.clear清空字典

dict1 = {"tom": 22, "jerry": 18, "alice": 19}  
dict1.clear()  
  
print(dict1)  
# 输出：{}

◎.keys获取字典中全部的key,然后通过for循环遍历字典

dict1 = {"tom": 22, "jerry": 18, "alice": 19}  
keys = dict1.keys()  
  
print(keys)  
# 输出：dict\_keys(['tom', 'jerry', 'alice'])  
  
for key in keys:  
 print(dict1[key])  
  
"""  
输出：  
22  
18  
19  
"""

◎直接用for循环遍历字典

dict1 = {"tom": 22, "jerry": 18, "alice": 19}  
  
  
for itemname in dict1:  
 print(itemname)  
 print(dict1[itemname])  
  
"""  
输出：  
tom  
22  
jerry  
18  
alice  
19  
"""

◎len统计字典内元素数量

dict1 = {"tom": 22, "jerry": 18, "alice": 19}  
  
print(len(dict1))  
# 输出：3

总结

|  |  |  |
| --- | --- | --- |
| 编号 | 操作 | 说明 |
| 1 | 字典[key] | 获取指定key对应的value值 |
| 2 | 字典[key]=value | 添加或更新键值对 |
| 3 | 字典.pop(key) | 取出key对应的value并在字典内删除此键值对 |
| 4 | 字典.clear() | 清空字典 |
| 5 | 字典.keys() | 获取字典的全部key，可用于for循环遍历字典 |
| 6 | Len(字典) | 计算字典内的元素数量 |

■数据容器的总结

数据容器可以从以下视角进行简单的分类：

|  |
| --- |
| 是否支持下标索引： |
| 支持：序列类型（列表、元组、字符串）  不支持：非序列类型（集合、字典） |
| 是否支持重复元素： |
| 支持：序列类型（列表、元组、字符串）  不支持：非序列类型（集合、字典） |
| 是否可以修改 |
| 可以：列表、集合、字典  不可以：元组、字符串 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 列表 | 元组 | 字符串 | 集合 | 字典 |
| 元素数量 | 支持多个 | 支持多个 | 支持多个 | 支持多个 | 支持多个 |
| 元素类型 | 任意 | 任意 | 仅字符 | 任意 | Key:除字典外任意类型  Value:任意 |
| 下标索引 | 支持 | 支持 | 支持 | 不支持 | 不支持 |
| 重复元素 | 支持 | 支持 | 支持 | 不支持 | 不支持 |
| 可修改性 | 可 | 不可 | 不可 | 可 | 可 |
| 数据有序 | 是 | 是 | 是 | 否 | 否 |
| 使用场景 | 可修改、可重复的一批数据记录场景 | 不可修改、可重复的一批数据记录场景 | 一串字符的记录场景 | 不可重复的数据记录场景 | 以key索引value的数据记录场景 |

●数据容器的通用操作

◎遍历

5类数据容器都支持for循环遍历。

序列类型（列表、元组、字符串）支持while循环。

非序列类型（集合、字典）不支持while循环，因为它们无法下标索引。

◎数据容器还可以通用非常多的功能方法，例如len, max, min

my\_list = [1, 2, 3]  
my\_tuple = (1, 2, 3, 4, 5)  
my\_str = "abcdef"  
  
# len(容器)统计容器的元素个数  
print(len(my\_list))  
# 输出：3  
print(len(my\_tuple))  
# 输出：5  
print(len(my\_str))  
# 输出：6  
  
  
# max(容器)统计容器的最大元素  
print(max(my\_list))  
# 输出：3  
print(max(my\_tuple))  
# 输出：5  
print(max(my\_str))  
# 输出：f  
  
  
# min(容器)统计容器的最小元素  
print(min(my\_list))  
# 输出：1  
print(min(my\_tuple))  
# 输出：1  
print(min(my\_str))  
# 输出：a

◎容器转换

|  |  |
| --- | --- |
| list(容器)  将给定容器转换为列表 | str(容器)  将给定容器转换为字符串 |
| tuple(容器)  将给定容器转换为元组 | set(容器)  将给定容器转换为集合 |

my\_list = [1, 2, 3, 4, 5]  
my\_tuple = (1, 2, 3, 4, 5)  
my\_str = "abcdefg"  
my\_set = {1, 2, 3, 4, 5}  
my\_dict = {"key1": 1, "key2": 2, "key3": 3, "key4": 4, "key5": 5}  
  
  
# list(容器) 将容器转换成列表  
print(f"列表转列表的结果是：{list(my\_list)}")  
print(f"元组转列表的结果是：{list(my\_tuple)}")  
print(f"字符串转列表的结果是：{list(my\_str)}")  
print(f"集合转列表的结果是：{list(my\_set)}")  
print(f"字典转列表的结果是：{list(my\_dict)}")  
  
"""  
输出：  
列表转列表的结果是：[1, 2, 3, 4, 5]  
元组转列表的结果是：[1, 2, 3, 4, 5]  
字符串转列表的结果是：['a', 'b', 'c', 'd', 'e', 'f', 'g']  
集合转列表的结果是：[1, 2, 3, 4, 5]  
字典转列表的结果是：['key1', 'key2', 'key3', 'key4', 'key5']  
"""  
  
  
# tuple(容器) 将容器转换成元组  
print(f"列表转元组的结果是：{tuple(my\_list)}")  
print(f"元组转元组的结果是：{tuple(my\_tuple)}")  
print(f"字符串转元组的结果是：{tuple(my\_str)}")  
print(f"集合转元组的结果是：{tuple(my\_set)}")  
print(f"字典转元组的结果是：{tuple(my\_dict)}")  
  
"""  
输出：  
列表转元组的结果是：(1, 2, 3, 4, 5)  
元组转元组的结果是：(1, 2, 3, 4, 5)  
字符串转元组的结果是：('a', 'b', 'c', 'd', 'e', 'f', 'g')  
集合转元组的结果是：(1, 2, 3, 4, 5)  
字典转元组的结果是：('key1', 'key2', 'key3', 'key4', 'key5')  
"""  
  
  
# str(容器) 将容器转换成字符串  
print(f"列表转字符串的结果是：{str(my\_list)}")  
print(f"元组转字符串的结果是：{str(my\_tuple)}")  
print(f"字符串转字符串的结果是：{str(my\_str)}")  
print(f"集合转字符串的结果是：{str(my\_set)}")  
print(f"字典转字符串的结果是：{str(my\_dict)}")  
  
"""  
输出：  
列表转字符串的结果是：[1, 2, 3, 4, 5]  
元组转字符串的结果是：(1, 2, 3, 4, 5)  
字符串转字符串的结果是：abcdefg  
集合转字符串的结果是：{1, 2, 3, 4, 5}  
字典转字符串的结果是：{'key1': 1, 'key2': 2, 'key3': 3, 'key4': 4, 'key5': 5}  
"""  
  
  
# set(容器) 将容器转换成集合  
print(f"列表转集合的结果是：{set(my\_list)}")  
print(f"元组转集合的结果是：{set(my\_tuple)}")  
print(f"字符串转集合的结果是：{set(my\_str)}")  
print(f"集合转集合的结果是：{set(my\_set)}")  
print(f"字典转集合的结果是：{set(my\_dict)}")  
  
"""  
输出：  
列表转集合的结果是：{1, 2, 3, 4, 5}  
元组转集合的结果是：{1, 2, 3, 4, 5}  
字符串转集合的结果是：{'a', 'e', 'g', 'b', 'f', 'd', 'c'}  
集合转集合的结果是：{1, 2, 3, 4, 5}  
字典转集合的结果是：{'key3', 'key4', 'key5', 'key2', 'key1'}  
"""

◎容器排序

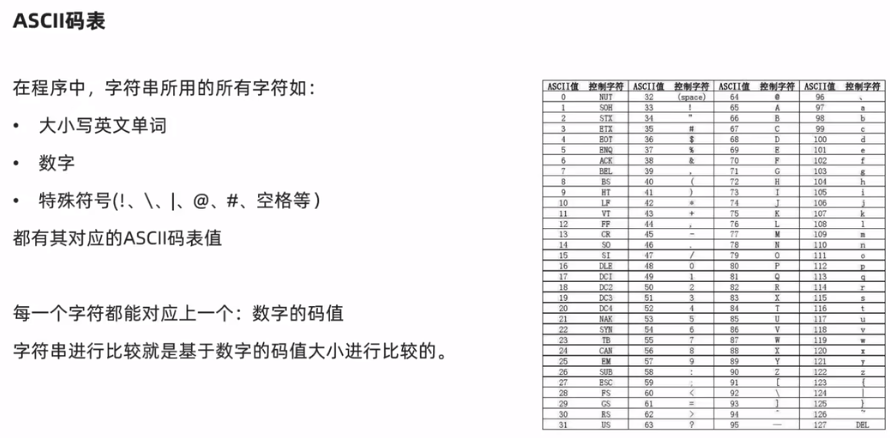
使用.sorted排序后的结果是一个列表对象。

# 语法：sorted(容器, [reverse=True])  
# 功能：将给定容器进行排序。reverse默认为False(正序)，设为True为倒序  
  
my\_list = [3, 1, 2, 5, 4]  
my\_tuple = (3, 1, 2, 5, 4)  
my\_str = "bdcefga"  
my\_set = {3, 1, 2, 5, 4}  
my\_dict = {"key3": 1, "key1": 2, "key2": 3, "key5": 4, "key4": 5}  
  
print(f"列表对象的排序结果：{sorted(my\_list)}")  
print(f"元组对象的排序结果：{sorted(my\_tuple)}")  
print(f"字符串对象的排序结果：{sorted(my\_str)}")  
print(f"集合对象的排序结果：{sorted(my\_set)}")  
print(f"字典对象的排序结果：{sorted(my\_dict)}")  
  
"""  
输出：  
列表对象的排序结果：[1, 2, 3, 4, 5]  
元组对象的排序结果：[1, 2, 3, 4, 5]  
字符串对象的排序结果：['a', 'b', 'c', 'd', 'e', 'f', 'g']  
集合对象的排序结果：[1, 2, 3, 4, 5]  
字典对象的排序结果：['key1', 'key2', 'key3', 'key4', 'key5']  
"""  
  
  
print(f"列表对象的反向排序结果：{sorted(my\_list, reverse=True)}")  
print(f"元组对象的反向排序结果：{sorted(my\_tuple, reverse=True)}")  
print(f"字符串对象的反向排序结果：{sorted(my\_str, reverse=True)}")  
print(f"集合对象的反向排序结果：{sorted(my\_set, reverse=True)}")  
print(f"字典对象的反向排序结果：{sorted(my\_dict, reverse=True)}")  
  
"""  
输出：  
列表对象的反向排序结果：[5, 4, 3, 2, 1]  
元组对象的反向排序结果：[5, 4, 3, 2, 1]  
字符串对象的反向排序结果：['g', 'f', 'e', 'd', 'c', 'b', 'a']  
集合对象的反向排序结果：[5, 4, 3, 2, 1]  
字典对象的反向排序结果：['key5', 'key4', 'key3', 'key2', 'key1']  
"""

容器通用功能总览

|  |  |
| --- | --- |
| 功能 | 描述 |
| 通用for循环 | 遍历容器（字典是遍历key） |
| max() | 容器内最大元素 |
| min() | 容器内最小元素 |
| len() | 容器元素个数 |
| list() | 转换为列表 |
| tuple() | 转换为元组 |
| str() | 转换为字符串 |
| set() | 转换为集合 |
| Sorted(序列, [reverse=True]) | 排序，reverse=True降序，False升序 |

■拓展讲解，字符串大小的比较



A 对应码表65

a 对应码表97

因此 a > A

字符串比较，从头到尾，一位位进行比较，其中一位大，后面的就不比了。

■函数进阶

●函数的多个返回值

# 如果函数要返回多个值，可以用逗号隔开  
def f():  
 a = 'abc'  
 b = 22  
 return a, b  
  
  
# 接收返回多个值的函数，用多个变量，逗号隔开，位置对应函数返回值的位置  
x, y = f()  
print(x)  
print(y)  
"""  
输出：  
abc  
22  
"""

●函数的多种传参方式

函数四种常见参数：

位置参数

关键字参数

不定长参数

缺省参数

◎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  
# 传递的参数和定义的参数顺序及个数必须一致  
def user\_info(name, age, gender):  
 print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')  
  
  
user\_info('TOM', 20, '男')  
# 输出：您的名字是TOM, 年龄是20, 性别是男

◎关键字参数

# 关键字参数：函数调用时通过“键=值”形式传递参数  
# 可以让函数更加清晰，易于使用，同时也清除了参数的顺序需求  
  
def user\_info(name, age, gender):  
 print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')  
  
  
user\_info(name="张三", age=20, gender="男")  
# 输出：您的名字是张三, 年龄是20, 性别是男  
  
  
# 可以不按固定顺序传参  
user\_info(age=19, gender="女", name="李四")  
# 输出：您的名字是李四, 年龄是19, 性别是女  
  
  
# 可以和位置参数混用，位置参数必须在前，且匹配原定参数顺序  
user\_info("王五", gender="男", age=22)  
# 输出：您的名字是王五, 年龄是22, 性别是男

◎缺省参数

# 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  
# 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  
# 作用：当调用函数时没有传递参数，就会使用默认的值  
  
def user\_info(name, age, gender='男'):  
 print(f'您的名字是{name}, 年龄是{age}, 性别是{gender}')  
  
  
user\_info(name="张三", age=20)  
# 输出：您的名字是张三, 年龄是20, 性别是男

◎不定长参数

位置传递，参数内得到一个元组

# 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会传递多少个参数（不传参数也可以）  
# 函数内接收到的入参是一个元组  
  
def user\_info(\*args):  
 print(args)  
 print(type(args))  
  
  
user\_info('tom', 18)  
"""  
输出：  
('tom', 18)  
<class 'tuple'>  
"""  
  
user\_info()  
"""  
输出：  
()  
<class 'tuple'>  
"""

◎关键字传递，参数内拿到一个字典

# 关键字传递  
# 函数内接收到的入参是一个字典  
  
def user\_info(\*\*kwargs):  
 print(kwargs)  
 print(type(kwargs))  
  
  
user\_info(name='tom', age=18)  
"""  
输出：  
{'name': 'tom', 'age': 18}  
<class 'dict'>  
"""

●将函数作为参数传递

将函数作为参数传递，传入的是计算逻辑，而非数据

def a(x):  
 r = x(1, 2)  
 return r  
  
  
def b(x, y):  
 return x + y  
  
  
print(a(b))  
# 输出结果为3

●匿名函数

注意：lambda匿名函数只能写一行，无法写多行

# def a(x):  
# r = x(1, 2)  
# return r  
#  
#  
# def b(x, y):  
# return x + y  
#  
#  
# print(a(b))  
# 输出结果为3  
  
# 以上可以改写为如下，将函数b的定义省略，直接将函数b用lambda方法写到入参里  
def a(x):  
 r = x(1, 2)  
 return r  
  
  
print(a(lambda x, y: x + y))  
# 输出：3

■文件操作

●文件编码概念

◎什么是编码

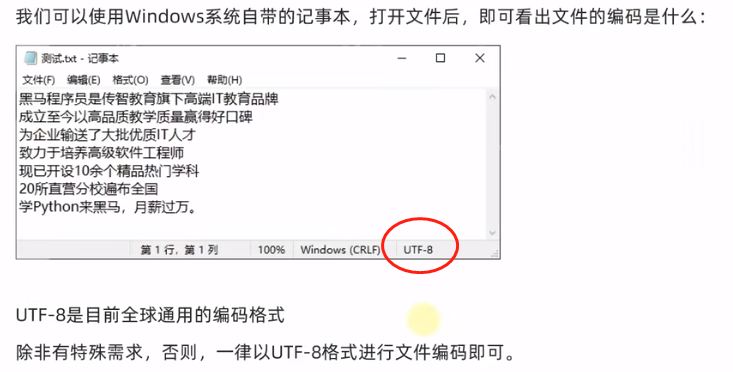
编码就是一种规则集合，记录了内容和二进制间进行互相转换的逻辑。编码格式有许多种，最常用的是UTF-8编码格式。

◎为什么要使用编码

计算机只识别0和1，所以需要编码格式将内容翻译成0和1的二进制码才能保持在计算机中。同时也需要编码格式将0和1的二进制码反向翻译回人类识别的内容。

◎常见编码有UTF-8、GBK、Big5等。

UTF-8是目前全球同用的编码格式。除非有特殊需求，否则一律以UTF-8格式进行文件编码即可。



●打开文件

# open()打开文件函数  
# 语法：open(name, mode, encoding)  
# 作用：打开一个已存在文件，或者创建一个新文件  
# name:要打开的文件路径和文件名  
# mode:设置打开文件的模式（访问模式）：只读、写入、追加等。  
# r：以只读方式打开文件。文件的指针将会放在文件的开头。这是默认模式。如果该文件不存在，将会报错  
# w：打开一个文件只用于写入。如果该文件已存在则代开文件，并从头开始编辑，原有内容会被删除。如果该文件不存在，创建新文件。  
# a：打开一个文件用于追加。如果该文件已存在，新的内容将会被写入到已有内容的末尾。如果该文件不存在，创建新文件。  
# encoding:编码格式（推荐使用UTF-8）  
  
  
# 注意：文件路径要不能直接用\,要用\\进行转义。不写路径默认文件和.py文件同一目录下  
f = open('E:\\test1\\t2.txt', 'w', encoding="UTF-8")  
print(type(f))  
# 此时的f是open函数的文件对象。  
# 输出：<class '\_io.TextIOWrapper'>  
print(f)  
# 输出：<\_io.TextIOWrapper name='E:\\test1\\t2.txt' mode='w' encoding='utf-8'>

●操作文件 读

◎.read读取.read(num)传入的参数是要读取的字节数

f = open('E:\\test1\\t1.txt', 'r', encoding="UTF-8")  
print(f.read(10))  
# 参数是读取内容的长度，单位是字节  
# 输出：这里是第一行的内容。  
  
print(f.read())  
# .read()不传入参数则是读取文件的全部内容  
# 由于刚才已经读了10个字节，现在指针指到第11个字节的位置，所以读取的内容从第11个字节开始  
"""  
输出：  
这里是第二行的内容。  
这里是第三行的内容。  
这里是第四行的内容。  
这里是第五行的内容。  
这里是第六行的内容。  
"""

◎.readlines读取全部行，封装到一个列表中

f = open('E:\\test1\\t1.txt', 'r', encoding="UTF-8")  
lines = f.readlines()  
print(type(lines))  
# 输出：<class 'list'>  
print(lines)  
# 输出：['这里是第一行的内容。\n', '这里是第二行的内容。\n', '这里是第三行的内容。\n', '这里是第四行的内容。\n', '这里是第五行的内容。\n', '这里是第六行的内容。\n']

◎.readline读取一行

f = open('E:\\test1\\t1.txt', 'r', encoding="UTF-8")  
line1 = f.readline()  
line2 = f.readline()  
line3 = f.readline()  
  
print(line1)  
# 输出：这里是第一行的内容。  
print(line2)  
# 输出：这里是第二行的内容。  
print(line3)  
# 输出：这里是第三行的内容。  
  
# 关闭文件  
f.close()

◎for循环读取

f = open("E:\\test1\\t1.txt", "r", encoding='UTF-8')  
for line in f:  
 print(line)  
  
"""  
输出：  
这里是第一行的内容。  
  
这里是第二行的内容。  
  
这里是第三行的内容。  
  
这里是第四行的内容。  
  
这里是第五行的内容。  
  
这里是第六行的内容。  
"""

◎关闭文件

使用完文件后，通过close关闭文件对象，也就是关闭对文件的占用。

如果不调用close，同时程序没有停止运行，那么这个文件将一直被python程序占用。

# 关闭文件  
f.close()

◎with open as f 打开文件并操作，完成代码块内的操作后，文件会自动被关闭，解除占用。

with open("E:\\test1\\t1.txt", "r", encoding='UTF-8') as f:  
 # 代码块里的语句被全部执行后，文件会自动执行.close关闭  
 for line in f:  
 print(line)

文件操作方法总结

|  |  |
| --- | --- |
| 操作 | 功能 |
| 文件对象= open(file, mode, encoding) | 打开文件获得文件对象 |
| 文件对象.read(num) | 读取指定长度字节。不指定长度则读取全部 |
| 文件对象.readline() | 读取一行 |
| 文件对象.readlines() | 读取全部行，得到列表 |
| for line in 文件对象 | for循环文件行,一次循环得到一行数据 |
| 文件对象.close() | 关闭文件对象 |
| with open() as f | 通过with open语法打开文件，可以自动关闭 |

●操作文件 写

◎w，写入操作

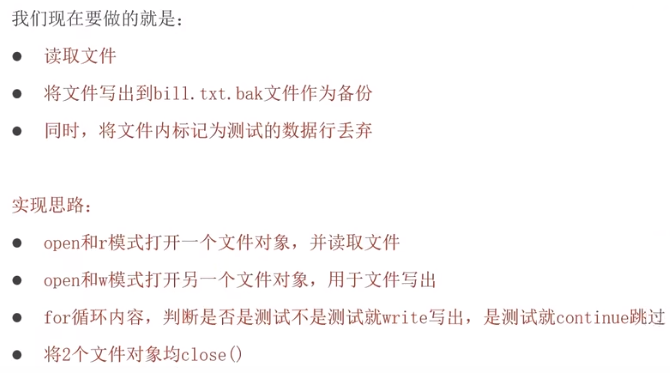
# 直接调用write，内容被写入内存中，成为缓冲区。  
# 当调用flush时，内容才真正被存入硬盘中。  
  
# 这里访问模式是w，文件中原内容将被新写入的内容覆盖  
f = open("E:\\test1\\t1.txt", "w", encoding='UTF-8')  
  
f.write("写入的第一行数据。\n")  
  
f.write("写入的第二行数据。\n")  
  
f.flush()  
  
f.close()

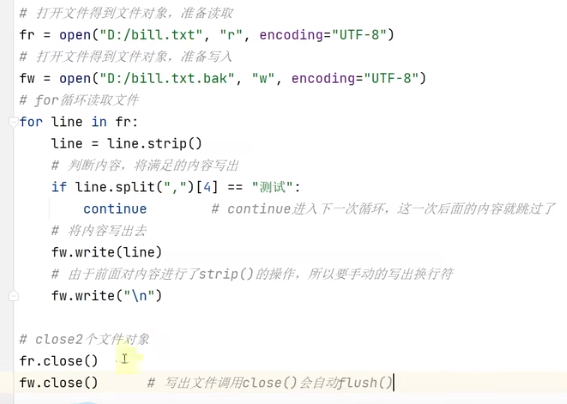
◎a，追加写入操作

# 直接调用write，内容被写入内存中，成为缓冲区。  
# 当调用flush时，内容才真正被存入硬盘中。  
  
# 这里访问模式是a，新写入的内容将追加到原有内容的末尾  
f = open("E:\\test1\\t1.txt", "a", encoding='UTF-8')  
  
f.write("写入的第三行数据。\n")  
  
f.write("写入的第四行数据。\n")  
  
f.flush()  
  
f.close()

●文件读写综合案例







■异常BUG

●捕获异常

◎捕获常规异常

# 捕获异常基本语法：  
# try:  
# 可能发生错误的代码  
# except:  
# 出现异常后执行的代码  
  
  
# 尝试打开t3.txt文件，如果文件不存在，也不会报错，使用except创建一个新文件  
try:  
 f = open('t3.txt', 'r', encoding="UTF-8")  
except:  
 f = open('t3.txt', 'w', encoding="UTF-8")

◎捕获制定异常

# 捕获指定异常  
# 如果执行代码的异常类型和要捕获的异常类型不一致，则无法捕获。  
# 一般捕获指定异常的try下方只放一行尝试执行的代码。  
  
try:  
 print(name)  
except NameError as e:  
 print(e)  
   
# 输出：name 'name' is not defined

◎捕获多个异常

# 捕获多个异常，可以把要捕获的异常类型的名字放到except后，使用元组的方式进行书写  
  
  
try:  
 print(1 / 0)  
except (NameError, ZeroDivisionError) as e:  
 print(e)  
  
# 输出：division by zero

◎捕获全部异常

# 捕获全部异常  
  
try:  
 print(1 / 0)  
except Exception as e:  
 print(e)  
  
# 输出：division by zero

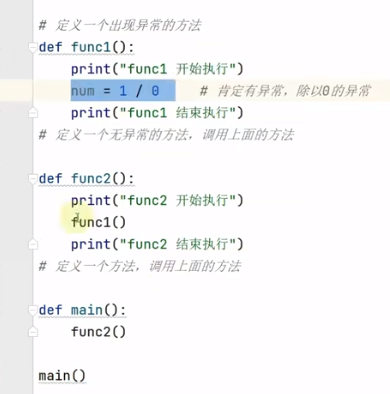
◎异常else用法

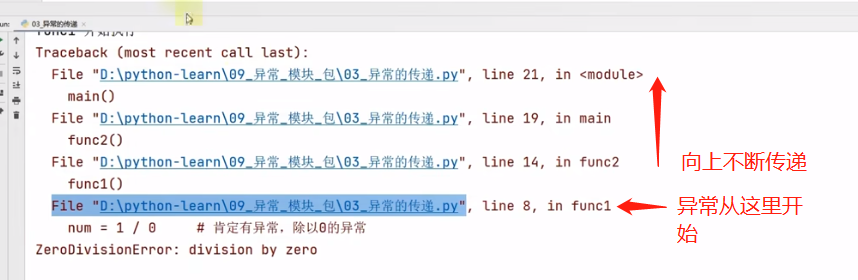
try:  
 print(1)  
except Exception as e:  
 print(e)  
else:  
 print("我是else,是没有异常的时候执行的代码")  
  
"""  
输出：  
1  
我是else,是没有异常的时候执行的代码  
"""

◎异常的finally

try:  
 print(1)  
except Exception as e:  
 print(e)  
else:  
 print("我是else,是没有异常的时候执行的代码")  
finally:  
 print("有没有异常都执行的代码。")  
  
"""  
输出：  
1  
我是else,是没有异常的时候执行的代码  
有没有异常都执行的代码。  
"""

◎异常的传递性





如下方式就不会报错了：

# 函数内的异常会向上传递到调用它的地方  
def f1():  
 print("我这里有一个错误。")  
 print(1 / 0)  
  
  
def f2():  
 print("函数f2开始")  
 f1()  
 print("函数f2结束")  
  
  
def main():  
 try:  
 f2()  
 except Exception as e:  
 print(e)  
  
  
main()  
"""  
输出：  
函数f2开始  
我这里有一个错误。  
division by zero  
"""

■模块

Python的模块（Module）,是一个python文件，以.py结尾。

模块能定义函数，类和变量，模块里也能包含可执行的代码。

●import的基本用法

# 基本语法：  
# import 模块名  
# import 模块名1，模块名2

# 使用 模块名.功能名()  
  
# 导入time模块，按住Ctrl在点击time可跳转到该模块的文件  
import time  
print("开始")  
time.sleep(5)  
print("结束")

●通过from来import具体功能。

# from import语法：  
# 导入：from 模块名 import 功能名  
# 使用：直接功能名()  
  
  
from time import sleep  
print("开始")  
sleep(5)  
print("结束")

# 导入全部功能用星号：  
# 导入：from 模块名 import \*  
# 使用：直接功能名  
  
  
from time import \*  
print("开始")  
sleep(5)  
print("结束")

●import一个模块，然后用as起个别名

# 导入模块并定义别名：  
# import 模块名 as 自定义别名  
# 使用 别名.功能名()  
  
  
import time as tt  
print("开始")  
tt.sleep(5)  
print("结束")

# 导入功能并定义别名：  
# from 模块名 import 功能名 as 自定义别名  
# 使用 别名()  
  
  
from time import sleep as sl  
print("开始")  
sl(5)  
print("结束")

●自定义模块

每个.py文件都可以作为一个模块，模块名就是文件名

模块1（my\_module1.py）的代码：

def my\_test(a, b):  
 print(a + b)

模块2（my\_module2.py）的代码：

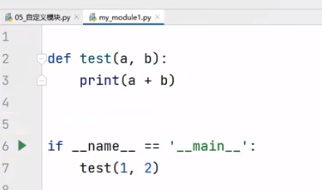
def my\_test(a, b):  
 print(a - b)

主文件

from my\_module1 import my\_test  
from my\_module2 import my\_test  
# 如果不同模块的方法名重复，则取最后导入的方法  
  
my\_test(2, 1)  
# 输出：1（后导入的模块2的方法生效）

●\_\_main\_\_的用法





◎在模块中，有时需要对已定义的函数进行测试，但是这会导致导入模块时测试函数自动被执行。

模块1（my\_module1.py）的代码：

def my\_test(a, b):  
 print(a + b)  
  
  
# 测试  
my\_test(2, 1)

主文件

from my\_module1 import my\_test  
  
my\_test(3, 1)  
"""  
输出：  
3 模块1里测试的代码被自动执行了  
4  
"""

◎解决方法

将测试内容写到 if \_\_name\_\_ == ‘\_\_main\_\_’的代码块内

模块1（my\_module1.py）的代码：

def my\_test(a, b):  
 print(a + b)  
  
  
# 测试  
if \_\_name\_\_ == '\_\_main\_\_':  
 my\_test(2, 1)

主文件

from my\_module1 import my\_test  
  
my\_test(3, 1)  
# 输出：4

* \_\_all\_\_的用法

模块1（my\_module1.py）的代码：

只有在\_\_all\_\_列表中的函数才能被导入页面使用

\_\_all\_\_ = ['my\_test1']  
  
def my\_test1(a, b):  
 print(a + b)  
  
  
def my\_test2(a, b):  
 print(a - b)

主文件

from my\_module1 import \*  
  
my\_test1(3, 2)  
# 输出：5  
  
my\_test2(3, 2)  
# 报错，因为此方法限制使用

■python包

创建一个文件夹，里面存放.py文件的模块。并有一个\_\_init\_\_.py文件。这就是一个python包了。

Pycharm里 new—python package可以自动创建一个带有\_\_init\_\_.py文件的包。

test\_package文件夹里module1.py文件

def say():  
 print('hello')

主文件，通过import导入

# 导入包里的模块  
import test\_package.module1  
  
# 使用包里的模块的函数  
test\_package.module1.say()  
# 输出：hello

主文件，通过from import导入

# 导入包里的模块  
from test\_package import module1  
  
# 使用包里的模块的函数  
module1.say()  
# 输出：hello

主文件，通过from import导入具体功能

# 导入包里的模块的函数  
from test\_package.module1 import say  
  
# 使用包里的模块的函数  
say()  
# 输出：hello

◎\_\_all\_\_要放到\_\_init\_\_.py文件里

# \_\_all\_\_要写在\_\_init\_\_.py文件中，对包里的所有模块进行控制  
\_\_all\_\_ = ["module1"]

●安装第三方包

◎

常用的第三方包

科学计算常用：numpy

数据分析常用：pandas

大数据计算常用：pyspark、apache-flink

图形可视化常用：matplotlib、pyecharts

人工只能常用：tensorflow

◎安装第三方包

使用python内置的pip程序

命令行输入：

pip install 包名称

直接安装速度慢，建议使用国内连接安装。

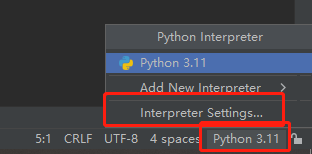
<https://pypi.tuna.tsinghua.edu.cn/simple>是清华大学提供的一个网站，可供pip程序下载第三方包

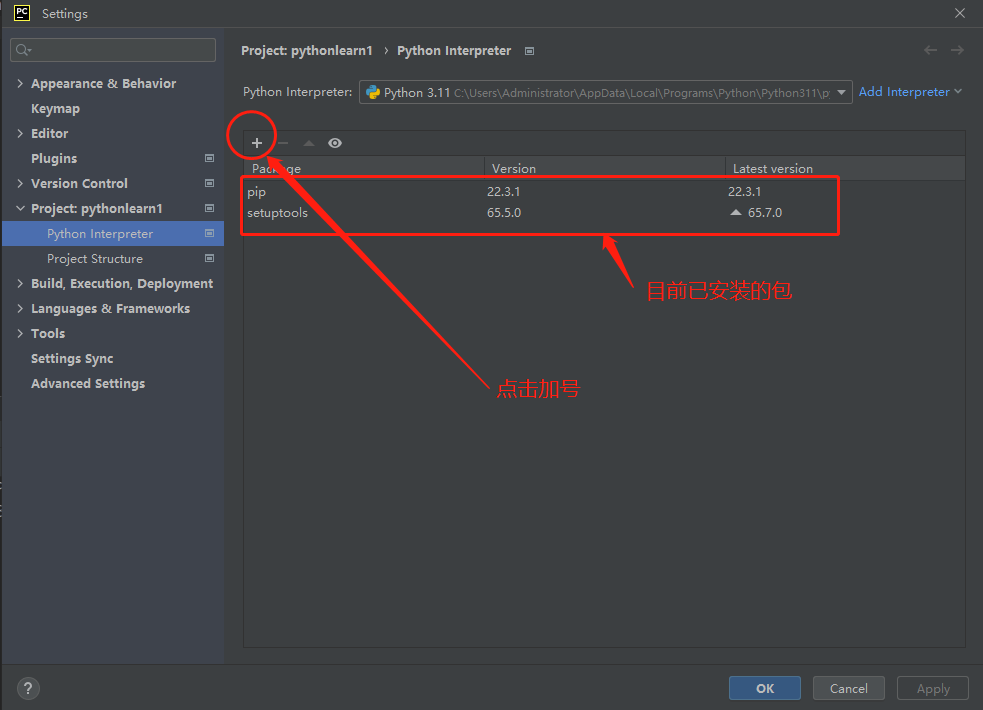
pip install -i <https://pypi.tuna.tsinghua.edu.cn/simple> 包名

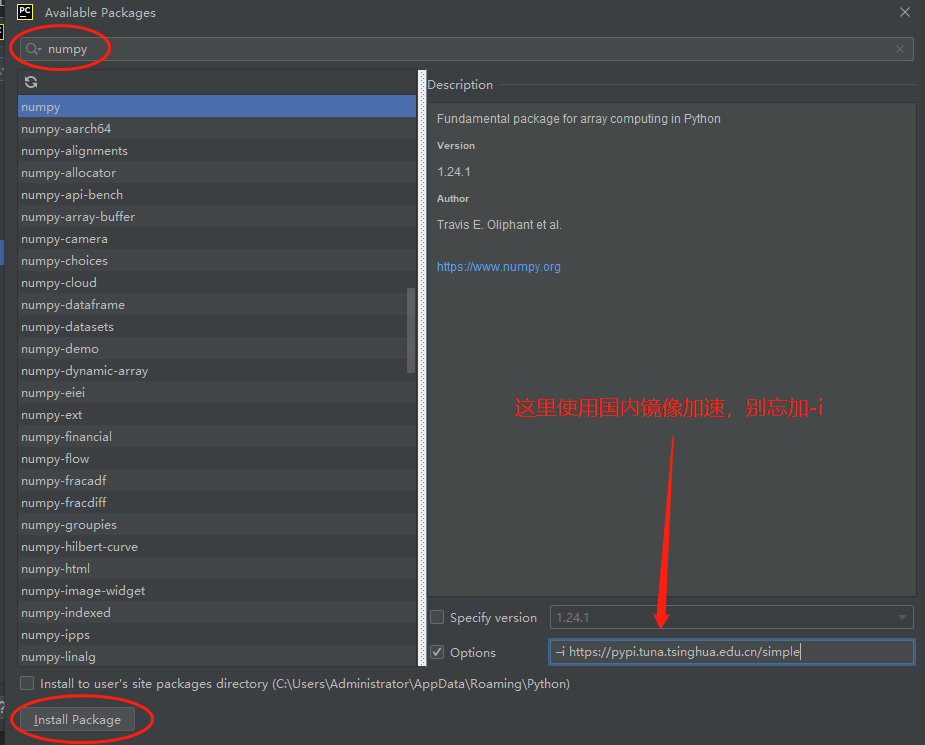
安装成功后在命令行的python解释器中输入import numpy来测试是否安装成功。如上图

◎使用pycharm安装第三方包

点右下角这里







■JSON

◎python转json

import json  
  
data = [  
 {  
 "name": "张三",  
 "age": 22  
 },  
 {  
 "name": "李四",  
 "age": 19  
 },  
 {  
 "name": "王五",  
 "age": 23  
 },  
]  
  
# 将jython数据转成json字符串  
json\_str = json.dumps(data)  
print(type(json\_str))  
print(json\_str)  
"""  
输出：  
<class 'str'>  
[{"name": "\u5f20\u4e09", "age": 22}, {"name": "\u674e\u56db", "age": 19}, {"name": "\u738b\u4e94", "age": 23}]  
"""

# 将字典转成json字符串,加入ensure\_ascii=False参数，中文就不会被编码了  
json\_str = json.dumps(data, ensure\_ascii=False)  
print(type(json\_str))  
print(json\_str)  
"""  
输出：  
<class 'str'>  
[{"name": "张三", "age": 22}, {"name": "李四", "age": 19}, {"name": "王五", "age": 23}]  
"""

◎json转python

# 将json字符串转换为python数据  
new\_data = json.loads(json\_str)  
print(type(new\_data))  
print(new\_data)  
"""  
输出：  
<class 'list'>  
[{'name': '张三', 'age': 22}, {'name': '李四', 'age': 19}, {'name': '王五', 'age': 23}]  
"""

◎懒人工具

<http://www.ab173.com/>

◎pyecharts官网

<https://pyecharts.org/#/zh-cn/>

■类和对象

◎

类里面可以定义变量和方法。

类的属性，即定义在类中的变量，叫成员变量。

类的行为，即定义在类中的函数，叫成员方法。

函数在外面叫函数，在类里面交方法。

◎

在类里的函数定义方法大致一样，只是多了一个self

Sele关键字是成员方法定义的时候，必须填写的。

它表示类对象自身

但我们使用类对象调用的方法时，self会自动被python传入

在方法内部，想要访问类的成员变量，必须使用self

# 类和成员方法的定义语法  
# class 类名称：  
# 成员变量  
#   
# def 成员方法(self, 参数列表)：  
# 成员方法体  
#   
# 类的实例化及成员属性和方法的调用  
# 对象 = 类名称()  
# 对象.成员变量  
# 对象.成员方法  
  
  
class Student:  
 name = None  
  
 def say\_hi(self):  
 print(f'大家好，我的名字是{self.name}。')  
  
 def say\_hi2(self, msg):  
 # 访问成员变量name需要用self.name，而访问msg是传入的参数，直接访问就行了  
 print(f'大家好，我的名字是{self.name}。{msg}')  
  
  
stu = Student() #将Student类实例化成实例对象储存到变量stu中  
stu.name = '张三'  
stu.say\_hi()  
stu.say\_hi2('请多多关照！')  
  
stu2 = Student()  
stu2.name = '李四'  
stu2.say\_hi()  
stu2.say\_hi2('互相帮助哈！')  
  
"""  
输出：  
大家好，我的名字是张三。  
大家好，我的名字是张三。请多多关照！  
大家好，我的名字是李四。  
大家好，我的名字是李四。互相帮助哈！  
"""

●构造方法

# python类可以使用 \_\_init\_\_()方法，称之为构造方法。  
# 可以实现：  
# 在创建类对象（构造类）的时候，会自动执行。  
# 在创建类对象（构造类）的时候，将传入参数自动传递给\_\_init\_\_方法使用。  
  
class Student:  
 name = None  
 age = None  
 tel = None  
  
 # \_\_init\_\_方法将在实例化类对象时自动执行  
 def \_\_init\_\_(self, name, age, tel):  
 self.name = name  
 self.age = age  
 self.tel = tel  
 print("Student类创建了一个对象")  
  
  
stu = Student("张三", 22, "18666666666")  
# 输出：Student类创建了一个对象（\_\_init\_\_方法被执行了）  
print(stu.name)  
# 输出：张三（实例化对象时的入参已经传递给了\_\_init\_\_方法）

●魔术方法

上文的\_\_init\_\_构造方法，是python的类的内置方法之一。

这些内置的类方法，各自有各自的特殊功能，被称为，魔术方法。

|  |  |
| --- | --- |
| 魔术方法 | \_\_init\_\_ 构造方法 |
| \_\_str\_\_ 字符串方法 |
| \_\_lt\_\_ 大于、小于符号比较 |
| \_\_le\_\_ 小于等于、大于等于符号比较 |
| \_\_eq\_\_ ==符号比较 |

◎字符串方法

class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
  
student = Student("张三", 22)  
print(student)  
print(str(student))  
"""  
输出：  
<\_\_main\_\_.Student object at 0x0000026D5A11F710>  
<\_\_main\_\_.Student object at 0x0000026D5A11F710>  
"""  
# 以上输出是内存地址

内存地址对我们来说没什么用，可以用\_\_str\_\_方法控制输出内容。

class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 def \_\_str\_\_(self):  
 return f"Student类对象，name={self.name}, age={self.age}"  
  
  
student = Student("张三", 22)  
print(student)  
print(str(student))  
"""  
输出：  
Student类对象，name=张三, age=22  
Student类对象，name=张三, age=22  
"""  
# 通过\_\_str\_\_控制的输出结果就清晰明了了

◎\_\_lt\_\_小于符号比较

class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 # 如果要比较两个实例化对象的大小，就要用\_\_lt\_\_进行设置，否则会报错  
 def \_\_lt\_\_(self, other):  
 return self.age < other.age  
  
  
student1 = Student("张三", 22)  
student2 = Student("李四", 19)  
print(student1 < student2)  
# 输出：False  
print(student1 > student2)  
# 输出：True

◎\_\_le\_\_小于等于比较

class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 # 如果要比较两个实例化对象的小于等于，就要用\_\_le\_\_进行设置，否则会报错  
 def \_\_le\_\_(self, other):  
 return self.age <= other.age  
  
  
student1 = Student("张三", 22)  
student2 = Student("李四", 19)  
print(student1 <= student2)  
# 输出：False  
print(student1 >= student2)  
# 输出：True

◎\_\_eq\_\_判断是否相等

class Student:  
 def \_\_init\_\_(self, name, age):  
 self.name = name  
 self.age = age  
  
 # 如果要比较两个实例化对象是否等于，就要用\_\_eq\_\_进行设置，否则会报错  
 def \_\_eq\_\_(self, other):  
 return self.age == other.age  
  
  
student1 = Student("张三", 22)  
student2 = Student("李四", 19)  
print(student1 == student2)  
# 输出：False

◎魔术方法小结

|  |  |
| --- | --- |
| 方法 | 功能 |
| \_\_init\_\_ | 构造方法，可用于创建类对象时设置初始化行为 |
| \_\_str\_\_ | 用于类的实例化对象转字符串的行为 |
| \_\_lt\_\_ | 用于类的两个实例化对象进行小于或大于比较 |
| \_\_le\_\_ | 用于类的两个实例化对象进行小于等于或大于等于比较 |
| \_\_eq\_\_ | 用于类的两个实例化对象进行判断是否相等 |

●封装

◎

面向对象编程，是许多编程语言都支持的一种编程思想。

简单理解是：基于模板（类）去创建实体（对象），使用对象完成功能开发。

面向对象包含三大主要特征：

封装

继承

多态

◎

封装就是将属性和行为，封装到类里，成为成员变量和成员方法。

◎私有成员

私有成员变量：变量名以\_\_开头

私有成员方法：方法名以\_\_开头

class Phone:  
 IMEI = None # 序列号  
 producer = None # 厂商  
  
 \_\_current\_voltage = None # 当前电压（私有属性）  
  
  
 def call\_by\_5g(self):  
 print("5g通话已开启")  
  
 def \_\_keep\_single\_core(self): # 私有方法  
 print("让cpu以单核模式运行以节省电量")  
  
  
phone = Phone()  
print(phone.producer)  
# 输出：None(可以访问)  
  
print(phone.\_\_current\_voltage)  
# 输出：AttributeError: 'Phone' object has no attribute '\_\_current\_voltage'.（不可访问）

●继承

◎单继承

# 继承基本语法  
# class 子类名（父类名）:  
# 子类内容体  
  
  
class Phone2021:  
 IMEI = None # 序列号  
 producer = None # 厂商  
  
 def call\_by\_4g(self):  
 print("4g通话")  
  
  
class Phone2022(Phone2021):  
 face\_id = True # 面部识别  
  
 def call\_by\_5g(self):  
 print("5g通话")  
  
  
phone = Phone2022()  
phone.call\_by\_4g()  
# 输出：4g通话  
phone.call\_by\_5g()  
# 输出：5g通话

◎多继承

# 多继承语法  
# class 子类名（父类1， 父类2， ...， 父类N）:  
# 子类内容体  
  
  
class Phone2021:  
 IMEI = None # 序列号  
 producer = None # 厂商  
  
 def call\_by\_4g(self):  
 print("4g通话")  
  
  
class Phone2022():  
 face\_id = True # 面部识别  
  
 def call\_by\_5g(self):  
 print("5g通话")  
  
  
class MyPhone(Phone2021, Phone2022):  
 pass  
  
  
my\_phone = MyPhone()  
my\_phone.call\_by\_4g()  
# 输出：4g通话  
my\_phone.call\_by\_5g()  
# 输出：5g通话

注：小知识pass 上图Myphone类里面不需要写什么方法，但什么都不写又会报错，因此写个pass即可。

◎复写

子类继承父类的成员属性核成员方法后，如果对其“不满意”，可以进行复写。

即：在子类中重新定义同名的属性和方法即可。

class Phone:  
 IMEI = None # 序列号  
 producer = None # 厂商  
  
 def call\_by\_5g(self):  
 print("5g通话")  
  
  
class MyPhone(Phone):  
 producer = "ITHEIMA" # 复写父类属性  
  
 def call\_by\_5g(self):  
 print("新的5G通话。") # 复写父类方法  
  
  
my\_phone = MyPhone()  
my\_phone.call\_by\_5g()  
# 输出：新的5g通话

◎复写后，调用父类同名成员

一旦复写父类成员，那么子类的对象调用成员的时候，就会调用复写后的新成员。

如果需要使用被复写的父类的成员，需要特殊的调用方式：

方式1：

调用父类成员

使用成员变量：父类名.成员变量

使用成员方法：父类名.成员方法

方法2：

使用super()调用父类成员

使用成员变量：super().成员变量

使用成员方法：super().成员方法

class Phone:  
 IMEI = None # 序列号  
 producer = None # 厂商  
 mark = "apple"  
  
 def call\_by\_5g(self):  
 print("5g通话")  
  
  
class MyPhone(Phone):  
 producer = "ITHEIMA" # 复写父类属性  
  
 def call\_by\_5g(self):  
 print("新的5G通话。") # 复写父类方法  
 print(f"父类的属性是：{Phone.mark}")  
 print(f"父类的属性使用super调用是：{super().producer}")  
  
  
my\_phone = MyPhone()  
my\_phone.call\_by\_5g()  
"""  
输出：  
新的5G通话。  
父类的属性是：apple  
父类的属性使用super调用是：None  
"""

●变量的类型注解

Python在3.5版本的时候引入了类型注解，以方便静态类型检查工具，IDE等第三方工具。

类型注解：在代码中涉及数据交互的地方，提供数据类型的注解（显示的说明）。

主要功能：

帮助第三方IDE工具（如pycharm）对代码进行类型推断，协助做代码提示。

帮助开发者自身对变量进行类型注释。

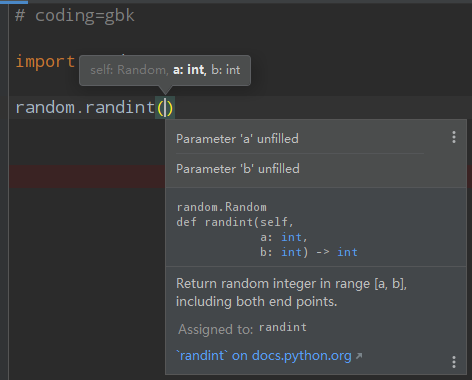
支持：

变量的类型注解

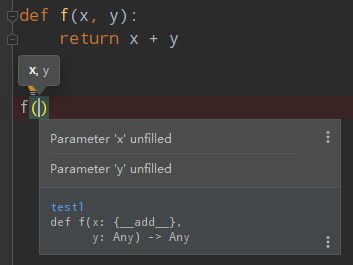
函数或方法形参列表和返回值的类型注解

◎

把光标停留在函数或方法的括号内，然后按ctrl + p



如果不加注解，就是如下样子



◎为变量设置类型注解

# 为变量设置类型注解的基本语法  
# 变量：类型  
  
  
# 基础数据类型注解：  
var\_1: int = 10  
var\_2: float = 3.1415926  
var\_3: bool = True  
var\_4: str = 'abc'  
  
  
# 基础容器类型注解：  
my\_list: list = [1, 2, 3]  
my\_tuple: tuple = (1, 2, 3)  
my\_str: str = 'abc'  
my\_set: set = {1, 2, 3}  
my\_dict: dict = {'key1': 1, 'key2': 2}  
  
  
# 容器类型详细注解：  
my\_list: list[int] = [1, 2, 3]  
my\_tuple: tuple[str, int, bool] = ('abc', 2, True)  
my\_set: set[int] = {1, 2, 3}  
my\_dict: dict[str, int] = {'key1': 'bbb', 'key2': 2}  
  
  
# 类对象类型注解：  
class Student:  
 pass  
  
  
stu: Student = Student()

import random  
  
# 即使不写注解也能明确知晓变量类型的情况，一般无需注解：  
# 例如  
a = 2  
  
# 无法直接看出变量类型之时，才会添加类型注解  
b: int = random.randint(1, 10)  
  
print(b)  
print(a)

◎函数和方法的类型注解

def add(x: int, y: int):  
 return x + y  
  
  
def func(data: list):  
 pass

◎函数和方法的返回值注解，用->

def add(x: int, y: int) -> int:  
 return x + y  
  
  
def func(data: list) -> list:  
 return data

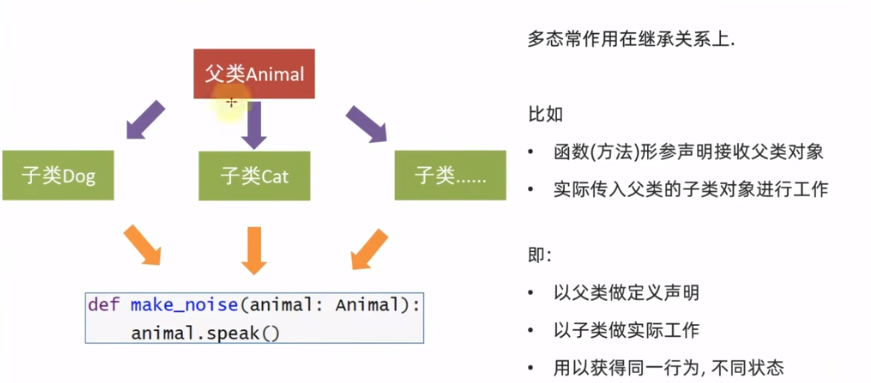
◎union类型的注解

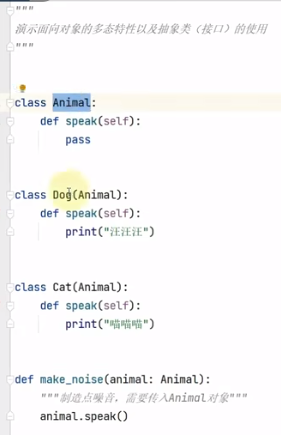
from typing import Union  
  
# Union[str, int]表示列表中可以出现str和int两种类型  
my\_list: list[Union[str, int]] = [1, 'abc', 2, 'def', 3, 4, 5]

@@@

●多态



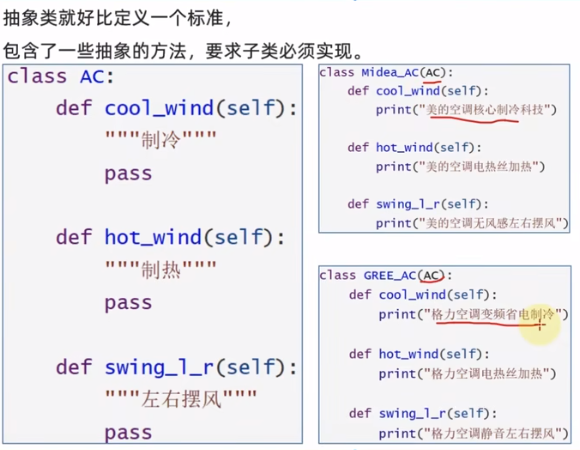




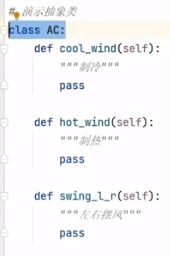


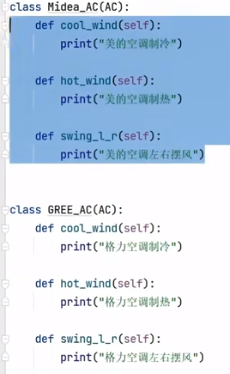
◎抽象类用来做顶层设计，一般不会构建抽象类，而是构建它的子类





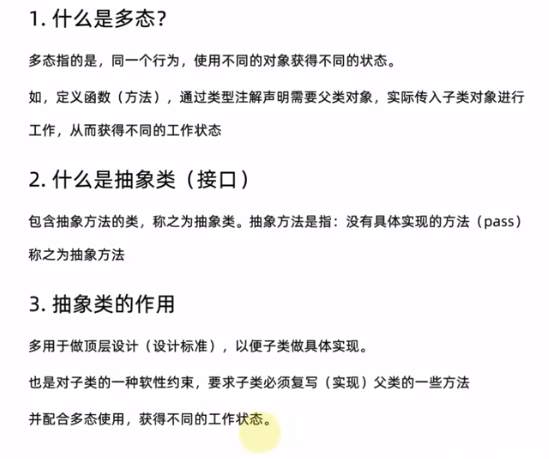








总结



■其他问题

●教学视频地址

<https://www.bilibili.com/video/BV1qW4y1a7fU?p=2&vd_source=0aaddd86e8bfefe5e362ccf93b4a9dc1>

●list中有中文会报错的问题，在文件开头加上这两行，就解决了。

#!usr/bin/env python3  
# -\*- coding:utf-8 -\*-

# coding=gbk

或者用

<https://blog.csdn.net/Li_Lei1/article/details/122418009>

●

