数据结构与算法B

Python 基础

目录

- Python 语言概述
- Python 编程基础
- Python 容器
- Python 面向对象编程（OOP）
- 其他 Python 特性与技巧

Python语言概述

- 为什么选择Python
  - 十大最流行的计算机语言之一
  - 语法简洁，极大地提高了生产力
  - 跨平台，代码可读性高
  - Python的解释器是开源的，可以自由访问、下载和修改

TIOBE
the software quality company

About us | Knowledge | News | Coding Standards | TIOBE Index | Contact

Products | Quality Models | Markets | Schedule a demo

Feb 2025	Feb 2024	Change	Programming Language	Ratings	Change
1	1		Python	23.88%	+8.72%
2	3	↑	C++	11.37%	+0.84%
3	4	↑	Java	10.66%	+1.79%
4	2	↓	C	9.84%	-1.14%
5	5		C#	4.12%	-3.41%
6	6		JavaScript	3.78%	+0.61%

Python语言概述

- Python已广泛应用于：
  - 大型网站：YouTube、Google、豆瓣、果壳网、NASA、Django
  - 图像多媒体：GIMP、Blender、Industrial Light & Magic
  - 系统文件：Dropbox、 BitTorrent、 Ubuntu Software Center
  - 科学计算/大数据：MySQL Workbench、numpy、pandas
  - 人工智能：pytorch, tensorflow

Python的历史

- 1989年12月，荷兰程序员吉多·范罗苏姆（Guido van Rossum）为了打发圣诞节假期，受ABC语言启发，希望创建一种更简洁、易读的语言，开发了Python
  - Python名称来自于他喜欢的一个情景剧Monty Python's Flying Circus

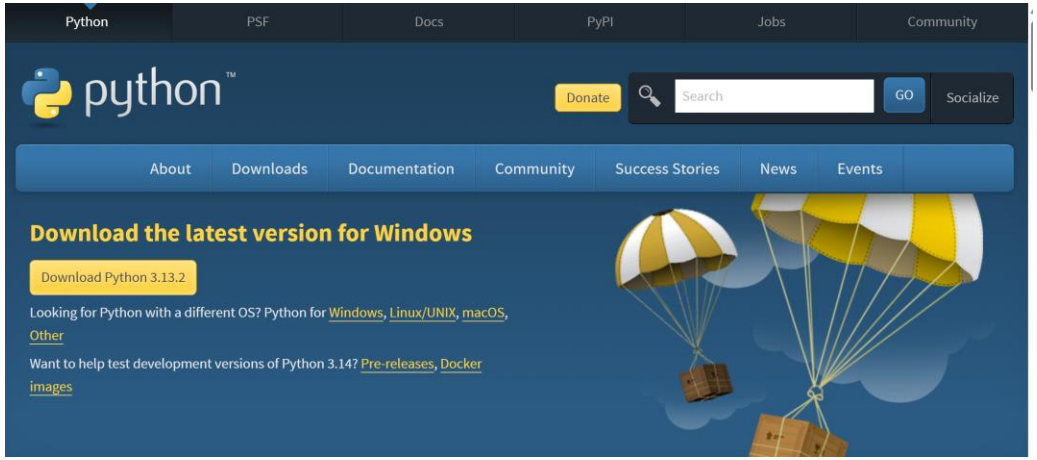
Python的历史

- 版本1.x：支持异常处理、函数定义，开发了核心数据结构
- 版本 2.x： 支持列表解析、垃圾收集器和Unicode编码
- 版本3.x：不向后兼容2.x，扫除了编程结构和模块上的冗余和重复

- Python3的改变：
  - 修改语法：使print()成为内置函数
  - 改进了Python2中input()函数
  - 统一字符编码
  - 更新了模块：删除了部分过时的模块或函数
  - 添加了一些新的模块
  - 数据结构dict（字典）性能的优化
  - 不再区分整数和长整数，统一为int
  - 两个整数相除，返回得到浮点数

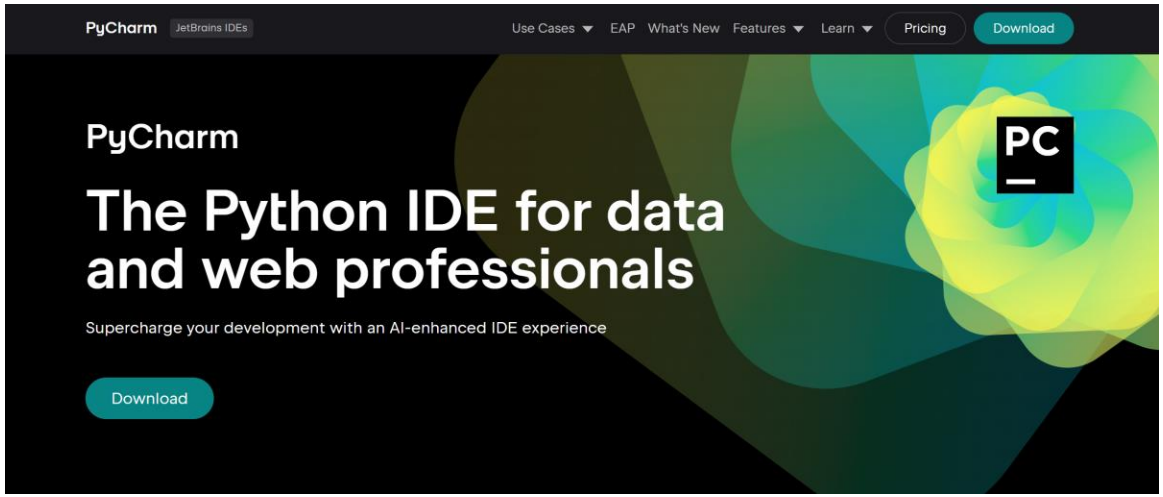
获取Python安装包

- Python官方下载页：http://www.python.org/download/
- 不同操作系统下的安装包格式
  - Linux/Unix：Source/release
  - Mac OS X：Mac OS X 64-bit/32-bit
  - Windows：zip/executable/web-based



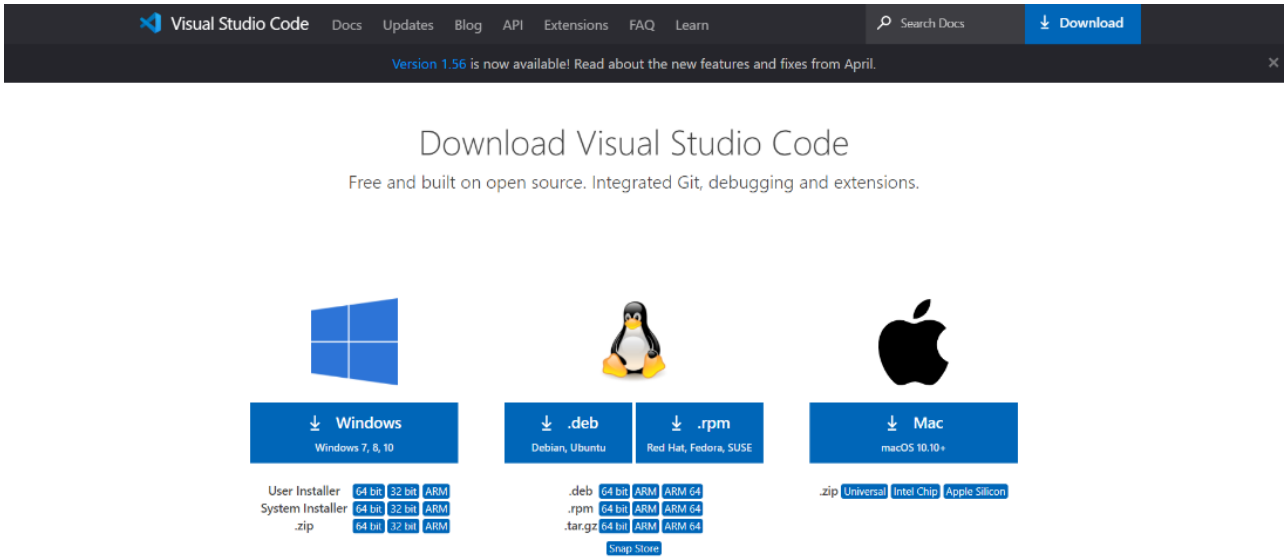
Python编程环境：PyCharm

- 集成开发环境–PyCharm：https://www.jetbrains.com.cn/en-us/pycharm/
- 由JetBrains公司打造的一款Python IDE （集成开发环境），它带有一整套可以帮助 用户在使用Python语言开发时提高其效率的工具。



Python编程环境：VSCode

- 集成开发环境–Vscode：https://code.visualstudio.com/Download
- Python 编程的最好搭档—VSCode 详细指南：https://zhuanlan.zhihu.com/p/112431369

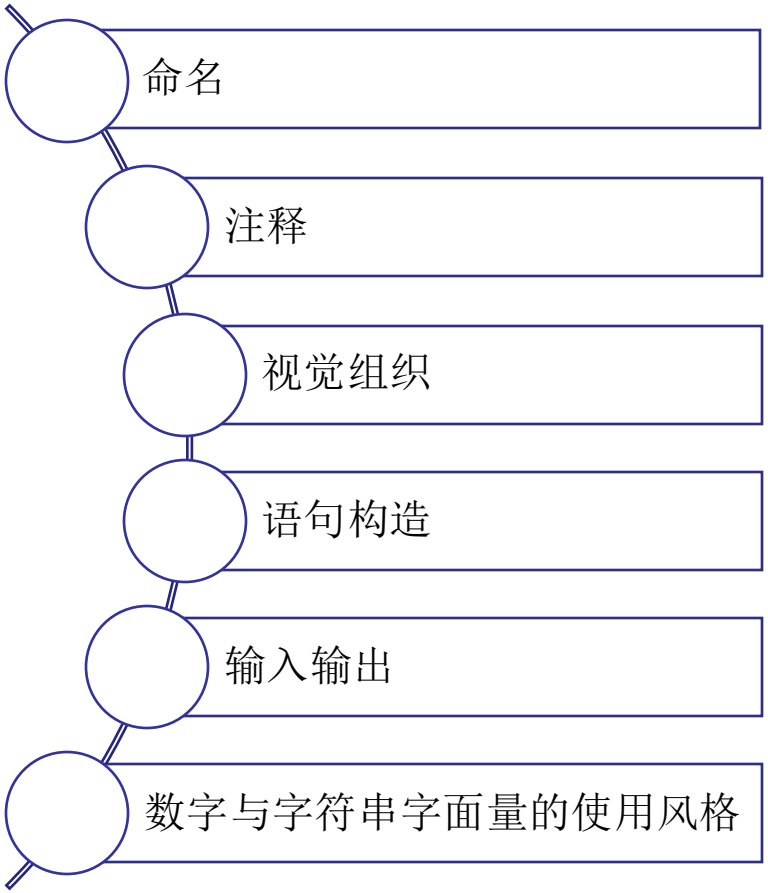


Python程序设计

- Python程序的风格：优雅、明确、简单
- Python程序的强制缩进规范完成了关键部分
  - 否则Python解释器产生解释错误
- 我们还需要使用良好的代码风格和编程规范

参考：孙艳春，黄罡，邓水光．**软件工程：经典、现代和前沿**．北京大学出版社，2024年2月出版．

Python程序设计：代码风格包含的要素




```
class Solution:
    def returnString(self, strs):
        if len(strs)==0:return"";ans=strs[0]
        for i in range(1,len(strs)):
            j = 0
            while j<len(ans) and j<len(strs[i]):
                if ans[j]≠strs[i][j]: break; j += 1;
            ans = ans[:j];
            if ans == "": return ans
        return ans
```

Python程序设计：命名

```
def exit_loop(should_exit: bool):
    if should_exit:
        return
```

出了什么问题？

Python程序设计：命名

命名：符合语义

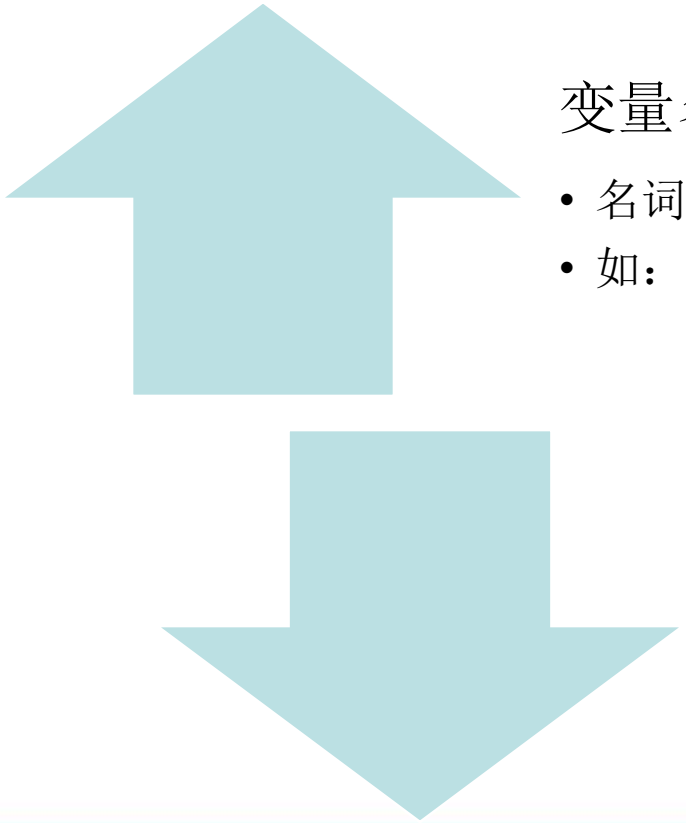
变量名
“是什么”

函数名
“做什么”

命名应具体

Python程序设计：命名

命名怎样符合语义？



变量名

- 名词或名词化的动词，保持单复数的正确
- 如：egg, paper_index…

函数/方法名

- 动词+名词/形容词，考虑在命名中适当反映函数的返回值类型
- 如：ask_questions, search_max_number…

Python程序设计：命名

- 建议使用下划线命名法，例如module_name
- 对于异常和类名，建议使用大驼峰命名法，例如ClassName
- 对于全局变量，建议使用全大写字母的下划线命名法，例如GLOBAL_VAR_NAME
- 尽可能避免双下划线开头和结尾的变量命名，因为此形式可能是Python的保留字，例如__init__
- Python依赖于变量的命名格式实现封装，一般来说，正常的变量命名可见性是public，以单下划线开头的变量名的可见性是protected，以双下划线开头的变量的可见性是private

Python程序设计：注释

何时需要注释？

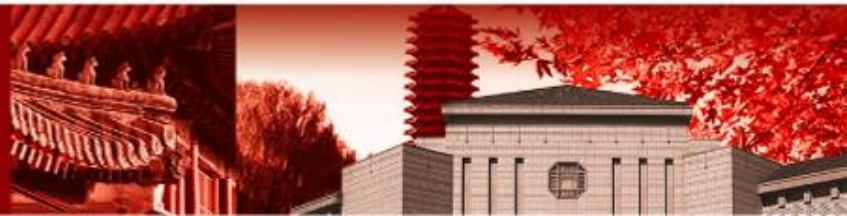
Python程序设计：注释

应该使用注释的场景

- 总结代码，以便读者快速理解代码；
- 标记一些内容，例如，强调代码的重要部分、编写者的联系信息等；
- 说明代码的意图或设计，例如，在类或函数前的块注释指出随后代码的设计思路或意图解决的问题；
- 说明代码无法表述的信息，例如，不明显的算法选择、版本信息或法律声明等。

注释的错误使用场景

- 说明编码的一些技巧；
- 利用注释说明代码实现的整个过程。



Python程序设计：注释规范-函数或方法

- 函数或方法的文档字符串主要用于描述函数或方法的功能以及输入输出格式，通常不需要描述“怎么做”
- 对于外部不可见、非常短小或者简洁明了的函数或方法，文档字符串可以省略
- 包含
  - –Args：列出参数名字以及对该参数的描述
  - –Returns（Yields用于生成器）：描述返回值的类型和语义
  - –Raises：列出与接口有关的所有异常

Python程序设计：注释规范-函数或方法

```
def fetch_smalltable_rows(table_handle: smalltable.Table,
                          keys: Sequence[Union[bytes, str]],
                          require_all_keys: bool = False,
) -> Mapping[bytes, Tuple[str]]:
    """Fetches rows from a Smalltable.

    Retrieves rows pertaining to the given keys from the Table instance
    represented by table_handle.  String keys will be UTF-8 encoded.

    Args:
        table_handle: An open smalltable.Table instance.
        keys: A sequence of strings representing the key of each table
        row to fetch.  String keys will be UTF-8 encoded.
        require_all_keys: Optional; If require_all_keys is True only
        rows with values set for all keys will be returned.

    Returns:
        A dict mapping keys to the corresponding table row data
        fetched. Each row is represented as a tuple of strings. For
        example:

        {b'Serak': ('Rigel VII', 'Preparer'),
        b'Zim': ('Irk', 'Invader'),
        b'Lrrr': ('Omicron Persei 8', 'Emperor')}

        Returned keys are always bytes.  If a key from the keys argument is
        missing from the dictionary, then that row was not found in the
        table (and require_all_keys must have been False).

    Raises:
        IOError: An error occurred accessing the smalltable.
    """
```

Python程序设计：注释规范-类

- 类在其定义下应当有一个用于描述该类功能的文档字符串
- 如果该类具有公共属性（Attributes），那么文档字符串也应当有一个属性（Attributes）段，类似于函数或方法中的Args段

```
class myClass(object):
    """Summary of myClass here.

    Longer class information...
    Longer class information...

    Attributes:
        bias: A additional add number on input value
        edit: A bollean indicating if we modify the input value
    """

    def __init__(self, edit=False):
        pass
```

Python程序设计：注释规范-行注释

- 注释的目的是解释代码的功能，即“做什么”，而不应解释代码如何实现这样的功能，即“怎么做”
- 为了提高可读性，行注释至少需要离开代码句2个空格

```
# We use weighted dictionary search to find the position of
# i in the array. We infer the position based on the maximum
# num and array size in the array, and then perform a binary
# search to obtain the exact number.

if i & (i - 1) = 1: # True if i is not 0 or a power of 2.
    pass
```

Python程序设计：视觉组织

缩进

空格

换行

Python程序设计：缩进

- 规范的缩进可以让代码阅读者快速定位程序的嵌套层次
- 规范的缩进可以帮助编写代码的开发人员快速定位一些错误，如“遗漏end”类型的错误
  - 在开发过程中，可能有时会遗漏和左括号对应的右括号的错误
  - 规范的缩进能让我们更容易在视觉上找到语言中对应的括号，从而更好地避免这种错误

```
def longest_common_prefix(strs):
    ans = strs[0]
    for i in range(1, len(strs)):
        j = 0
        while j < len(ans) and j < len(strs[i]):
            if ans[j] != strs[i][j]:
                break
            j += 1
        ans = ans[:j]
        if ans == "":
            return ans
    return ans
```

Python程序设计：缩进规范

- 制表符（tab）在各类编辑器中的宽度不同, 因此不推荐使用tab键或者tab和空格混合模式进行缩进，推荐使用4个空格缩进
- 除去Python编程语言要求的缩进方案之外，对于行连接的语句，例如长函数名的参数，字典变量的内容等，我们应当注意使用规范的缩进来对齐它们的内部元素

Python程序设计：空格

- 赋值等双目运算符前后加一个空格
- 行内逗号、分号、问号等后加一个空格（这一部分和英文写作的要求是十分类似的）
- 括号前后是否加空格都可以，但在一个软件项目中，要保持统一风格

```
sum+=num>=0?(num +((1<<20)+(num&1))):(num+((1<<20)+(num&2)));
```

VS

```
sum += num >= 0 ? (num + ((1 << 20) + (num & 1))) : (num + ((1 << 20) + (num & 2)));
```

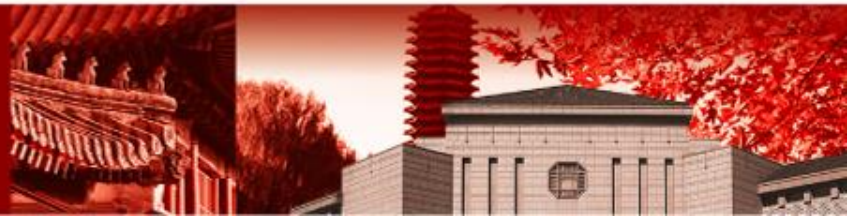
Python程序设计：换行

- 一般会在一行写一个语句，然后进行换行
- 在一个程序块中，逻辑区别较大的代码段可以用空行进行分隔，例如输入和输出处理部分
- 用一个（或多个）空行分隔函数、类等定义，方便代码阅读

```
def calculate(num):
    if num ⩾ 0:
        temp = (1 << 20) + (num & 1)
        return num + temp
    else:
        temp = (1 << 20) + (num & 2)
        return num + temp
```

VS

```
def calculate(num):
    if num ⩾ 0:
        temp = (1 << 20) + (num & 1); return num + temp
    else:
        temp = (1 << 20) + (num & 2); return num + temp
```

Python程序设计：使用EditorConfig

- 对于代码的格式性约束都靠　本描述会很麻烦；
- EditorConfig 提出了　个代码格式性约束的描述　式，并且让它可以被　多数代码编辑器所识别；
- 只需要在工程的根目录的创建 .editorconfig 就可以让所有协作者按照同一个格式性约束开发了，编辑器会自动适应这种格式。

Python程序设计：语句构造风格

- 避免子程序过长
- 避免嵌套过深
- 避免语句过长
- 尽量一行一条语句
- 优先使用清晰的结构构造语句

Python程序设计：输入输出风格

- **输入**
  - –检查输入项组合的合理性，避免不合常理的输入项组合；
  - –允许更加自由的输入，例如支持输入中的换行；
  - –对于输入数据要进行检查，筛除不合法的输入，或给出报警提示；
  - –在交互式场景下，对用户下一步的输入给出提示信息。

- **输出**
  - –检验输出数据，避免无意义的输出；
  - –为输出数据加注释，并设计报表格式，便于用户快速了解。

Python程序设计：数字与字符串字面量的使用风格

- 魔法数字/串
  - 将其写入单独的配置文件，让它们作为常量存在
  - 再单独引入，并使用全大写命名表示

魔法数字：就是你在代码里直接写上的字符串或者数字的字面量

demo.py

```
if category == 0:
    print('该课程为语文')
if category == 1:
    print('该课程为数学')
if category == 2:
    print('该课程为英语')
```

config.py

```
CAT_CHINESE = 0
CAT_MATH = 1
CAT_ENGLISH = 2
```

demo.py

```
from config import CAT_CHINESE, CAT_MATH, CAT_ENGLISH

if category == CAT_CHINESE:
    print('该课程为语文')
if category == CAT_MATH:
    print('该课程为数学')
if category == CAT_ENGLISH:
    print('该课程为英语')
```

Python的基本数据类型：int

- 整数类型 int
- 最大的特点是不限制大小，无论多复杂的算式都直接得到结果
- 常见的运算

运算符	功能	备注
m + n	加法	
m - n	减法	
m * n	乘法	
m // n	整数除法	结果是商的整数部分
m / n	除法	“真”除法，得到小数
m % n	求余数	
divmod(m, n)	求整数除法和余数	会得到两个整数，一个是m//n，另一个是 m % n
m ** n	求乘方	整数m的n次方
abs(m)	求绝对值	

Python的基本数据类型：int

- 两个整数的比较

`m == n`	相等比较	m 是否等于 n
`m > n`	大于比较	m 是否大于 n
`m >= n`	大于或等于比较	m 是否大于或者等于 n
`m < n`	小于比较	m 是否小于 n
`m <= n`	小于或等于比较	m 是否小于或者等于 n

- 连续比较判断

```
>>> 7 > 3 >= 3
True
>>> 12 < 23 < 22
False
```

Python的基本数据类型：float

- 浮点数类型 float
  - –运算操作与整数类似
  - –浮点数受到17位有效数字的限制，并且存在精度误差
  - –判断两个浮点数相等，通常不能直接使用 == 来判断，实际上等于一个存在微小误差的值
    - 例如：4.2+2.1不等于6.3
  - –判断两数之差是否小于一个阈值（例如 eps = 1e-9 等）

```
>>> 355/113
3.1415929203539825
>>> 3.1415926535897932
384626
3.141592653589793
>>> 123412342341234231
2341234.0
1.2341234234123424e+24
>>> 0.0000012312312312
3123123
1.2312312312312312e-06
>>> 4.2 + 2.1 == 6.3
False
>>> 4.2 + 2.1
6.300000000000001
```

Python的基本数据类型：str

- 字符串类型 str
  - 文字字符的序列串（string）
  - 包含拉丁字母、数字、标点符号、 特殊符号，以及各种语言文字
  - 特殊字符用转义符号 “\”表示
- 用双引号或者单引号都可以表示字符串， 但必须成对
- 多行字符串用三个连续单引号表示

```
>>> "abc"
'abc'
>>> 'abc'
'abc'
>>> '''abc def
ghi jk'''
'abc def\nghi jk'
```

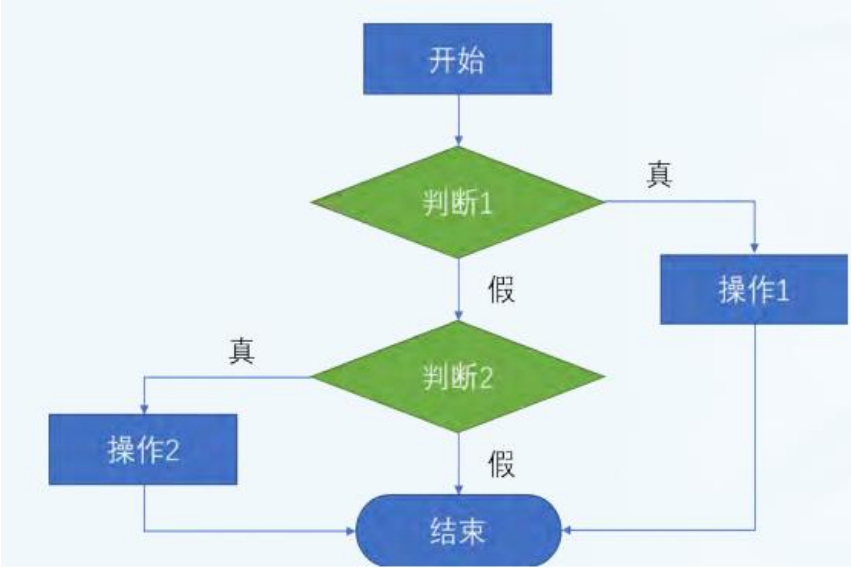
转义字符	描述
在行尾时	续行符
\\	反斜杠符号
\’	单引号
\’’	双引号
\a	响铃
\b	退格
\e	转义
\000	空
\n	换行
\v	纵向制表符
\t	横向制表符
\r	回车
\f	换页
\oyy	八进制数yy表示的转义符，例如\o12表示换行，参见ASCII码表
\xyy	十六进制数yy代表的字符，例如\x0a表示换行，参见ASCII码表
\other	其他的字符以普通格式输出

Python的基本数据类型：str

- 常见的字符串操作
  - “加法”：将两个字符串进行连接，得到新的字符串。
    - “Python is ”+ “awesome” >> “Python is awesome”
  - “乘法”：将字符串重复若干次，生成新的字符串
    - “abc ” * 3 >> “abc abc abc ”
  - 获取字符串的长度：len()函数
  - 切片(slice)操作：s[start: end: step]
    - s = “hello world”
    - s[:5] >> “hello” s[6:11] >> “world”
    - s[::2] >> “hlowrd”

Python的基本数据类型：bool

- 布尔类型 bool
  - 逻辑值仅包括真(True) / 假(False)两个
  - 用来配合if / while等语句做条件判断
- 逻辑运算
  - 与：and，双目运算
  - 或：or，双目运算
  - 非：not，单目运算
  - 优先级not最高，and次之，or最低



Python的基本数据类型：bool

- 各种类型的真值：
  - –整数、浮点数

    0是“假”，所有非0的数值都是“真”
  - –字符串类型

    空串("")是“假”，所有非空串都是“真”
  - –所有序列类型（包括字符串）

    空序列是“假”，所有非空的序列都是“真”
  - –空值None

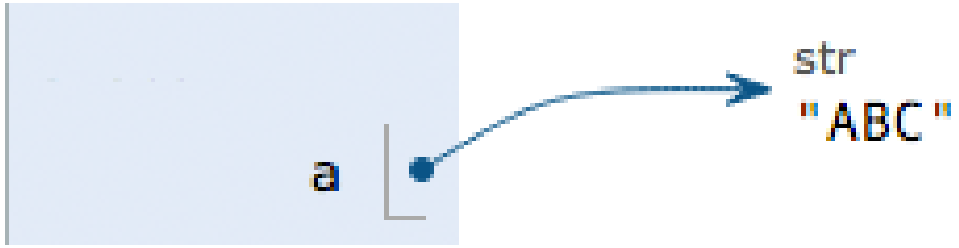
    表示“无意义”或“不知道”，也是“假”

Python的变量与引用

- 在python中，一个变量可以是任意数据类型
  - a = 1：变量a是一个整数
  - t_007 = 'T00'：变量t_007是字符串
- 可以把任意数据类型赋值给变量，同一个变量可以反复赋值，而且可以是不同类型的变量，例如：
  - a = 123　　# a是整数
  - print(a)　　# 123
  - a = 'ABC'　　# a变为字符串
  - print(a)　　# 'ABC'
- 这种变量本身类型不固定的语言称之为动态语言
- 这一机制为编程提供了极大灵活性

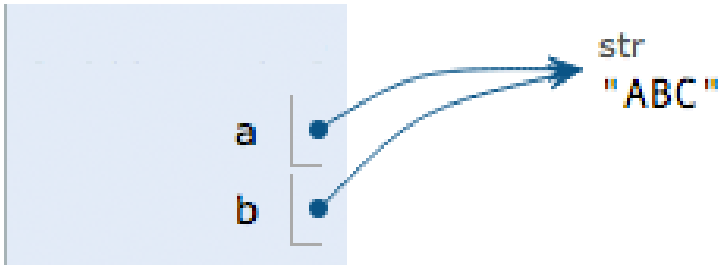
Python的变量与引用

- 理解Python中的变量表示“引用”
  - 对于C语言的使用者：指针
- 当我们运行 a = ‘ABC’ 时，Python解释器：
  - 在内存中创建了一个'ABC'的字符串；
  - 在内存中创建了一个名为a的变量，并把它指向'ABC'。



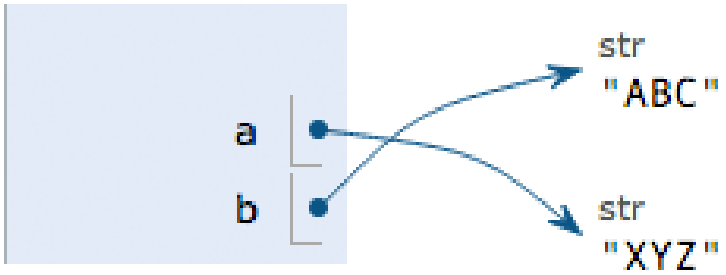
Python的变量与引用

- 再执行 b = a 时：



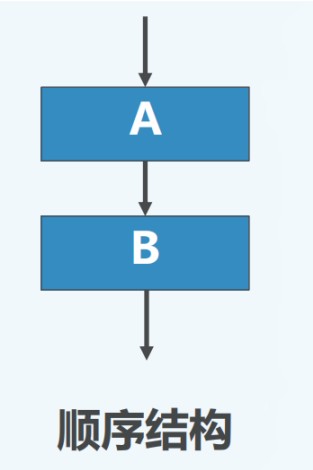

```
a = 'ABC'
b = a
a = 'XYZ'
print(b)
```

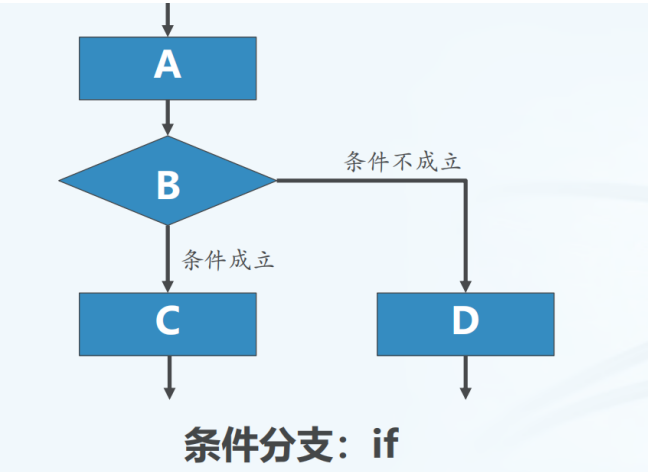
- 如果再执行 a = ‘XYZ’，是否会改变b？

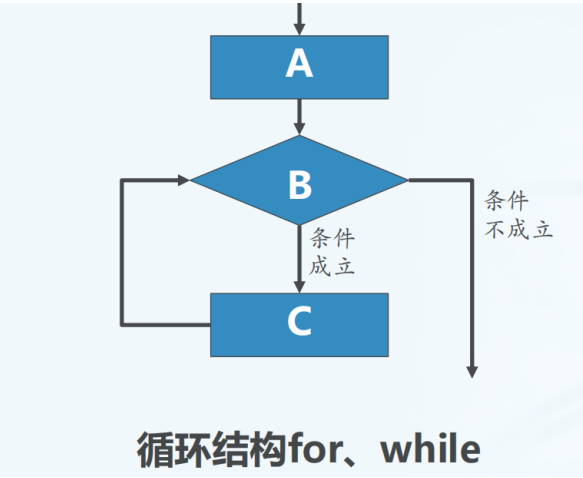


- 确保你理解这段代码的运行过程

Python中的控制流







与C语言等编程语言基本相同

Python中的函数

- 函数(function)：程序中实现明确功能的代码段可以封装成一个函数，以便复用（reuse）

定义函数

用def语句创建一个函数

用return关键字指定函数返回的值

```
def <函数名> (<参数表>):
    <缩进的代码段>
    return <函数返回值>
```

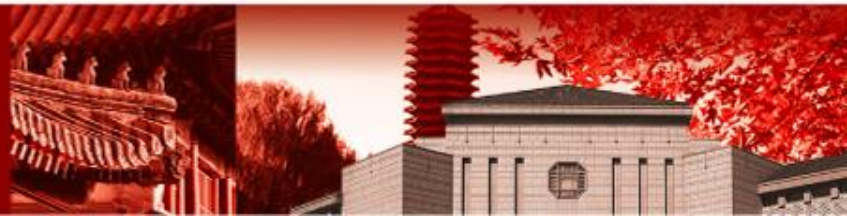
调用函数

```
<函数名>(<参数>)
```

注意括号！

无返回值：<函数名>(<参数表>)

返回值赋值：v = <函数名>(<参数表>)



Python中的函数

- 定义与调用函数示例

```
1   def sum_list(alist):  # 定义一个带参数的函数
2       sum_temp = 0
3       for i in alist:
4           sum_temp += i
5       return sum_temp  # 函数返回值
6
7
8   print(sum_list)  # 查看函数对象sum_list
9
10  my_list = [23, 45, 67, 89, 100]
11  # 调用函数，将返回值赋值给my_sum
12  my_sum = sum_list(my_list)
13  print("sum of my list:%d" % (my_sum,))
```

```
<function sum_list at 0x10067a620>
sum of my list:324
```

如果直接打印函数名，可以发现这是一个函数对象，也存储在内存中的某个位置

Python中的输入输出

- 输入：input(prompt)
  - 输入时显示提示信息prompt，由用户输入内容
- input()的返回值是字符串

```
>>> x = input('x:')
x:5
>>> y = input('y:')
y:6
>>> x + y
'56'
```

- 通过int()函数将input()返回值转换为数值类型

```
>>> x = input('x:')
x:5
>>> y = input('y:')
y:6
>>> int(x) + int(y)
11
```

Python中的输入输出

- 输出：print()函数，允许一次性输出多个值，自动换行

打印各变量的值输出

```
print([object,...][,sep=‘ ‘][,end=‘\n’]
      [,file=sys.stdout])
```

object 包含任意多个待输出的变量

- sep：表示变量之间用什么字符串隔开，缺省是空格
- end：表示以这个字符串结尾，缺省为换行
- file：指定了文本将要发送到的文件、标准流或其它类似的文件的对象；默认是sys.stdout

Python中的输入输出

- 格式化输出
  - 整数：%d
  - 字符串：%s
  - 浮点数：%f
  - 控制浮点位数：%.2f
  - 补齐长度：%4d或%04d

```
>>> yname = input ("Please input your name")
Please input your nameTom Hanks
>>> yname
'Tom Hanks'
>>> print (1, 23, 'Hello')
1 23 Hello
>>> print (1, 23, 'Hello', end='')
1 23 Hello
>>> print (1, 23, 'Hello', sep=',')
1,23,Hello
>>> '%d %s' % (23, 'Hello')
'23 Hello'
>>> '%d' % (23,)
'23'
>>> '(%4d):K:%s' % (12, 'Hello')
'(  12):K:Hello'
>>> '(%04d):K:%10s' % (12, 'Hello')
'(0012):K:     Hello'
```

Python中的容器：list

- Python内置的容器类型：list
  - list是一种有序的集合，可以随时添加和删除其中的元素。

```
>>> classmates = ['Michael', 'Bob', 'Tracy']
>>> classmates
['Michael', 'Bob', 'Tracy']
```

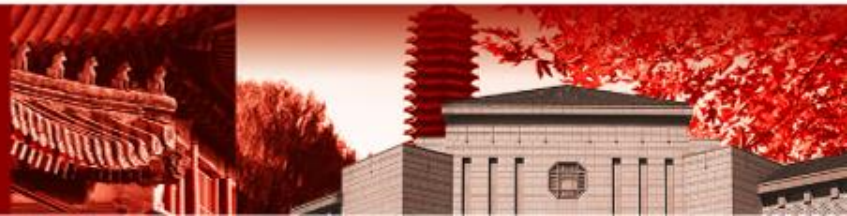
  - 用数字索引来访问list中每一个位置的元素，索引从0开始；len()可以获得列表长度

```
>>> classmates[0]
'Michael'
```

```
>>> len(classmates)
3
```

  - 使用负数索引：-1表示最后一个元素，-n表示第一个元素（n为元素数量）

```
>>> classmates[-3]
'Michael'
```

Python中的容器：list

- List的其他常用操作：
  - 可以向列表末尾追加元素（append）
  - 或者在指定位置插入/删除（insert/pop）
  - 列表中的元素还可以是另一个列表
  - 列表的加法：拼接两个列表生成新的列表
    - [‘Michael’, ‘Bob’] + [‘Adam’] >> [‘Michael’, ‘Bob’, ‘Adam’]
  - 列表的乘法：将列表重复若干倍生成新的列表
    - [‘Adam’] * 3 >> [‘Adam’, ‘Adam’, ‘Adam’]

Python中的容器：tuple

- tuple（元组）与列表的区别：不可变对象

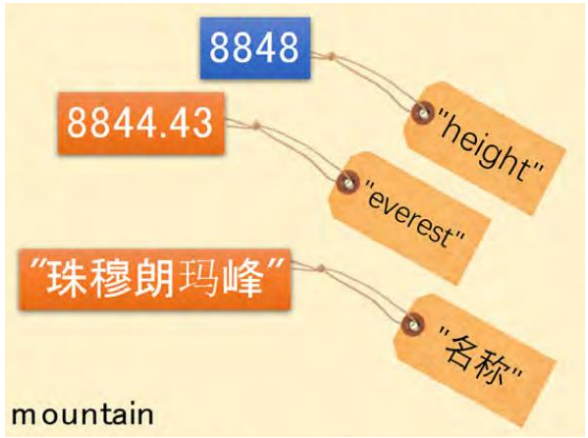
```
>>> classmates = ('Michael', 'Bob', 'Tracy')
```

- tuple一旦初始化就不能修改。但仍然可以使用索引访问
- 不可变对象有什么意义？
  - 代码的安全性
    - tuple作为函数参数防止意外修改
    - 在多线程程序中避免了多个线程同时修改列表内容引起的竞争
  - 换取更高的性能
    - list与tuple相比需要更多的存储空间来支持扩展性
    - 更快的访问速度与创建速度

Python中的容器：dict

- 字典：给数据贴上标签，就可以通过具有特定含义的名字或者别的记号来获取数据。

- 字典容器中保存着一系列的key-value对，通过键值key来索引元素value



Python中的容器：dict

- 创建与使用字典：例如，要根据学生姓名查询成绩

```
names = ['Michael', 'Bob', 'Tracy']
scores = [95, 75, 85]
```

使用列表存储
需要手动查找索引序号，再访问成绩

```
>>> d = {'Michael': 95, 'Bob': 75, 'Tracy': 85}
>>> d['Michael']
95
```

使用字典存储
直接根据姓名访问成绩

Python中的容器：set

- set和dict类似，也是一组key的集合，但不存储value。
- 由于key不能重复，所以，在set中，没有重复的key。
- 同样使用花括号 {} 来创建集合，或自动将list转换为集合（重复元素会被自动过滤）

```
>>> s = {1, 2, 3}
>>> s
{1, 2, 3}
```

```
>>> s = set([1, 2, 3])
>>> s
{1, 2, 3}
```

- 注意：set与dict均属于无序集合，即内部存放顺序与key的插入顺序无关

Python面向对象编程

- 面向对象的基本概念：万物皆对象
- Python中的所有事物都是以对象形式存在， 从简单的数值类型，到复杂的代码模块，都是对象
- 既表示客观世界问题空间中的某个具体事物
- 又表示软件系统解空间中的基本元素

Python面向对象编程

- 同一个例子：为了处理学生的成绩表，面向过程的程序可以使用dict与函数完成：

```
std1 = { 'name': 'Michael', 'score': 98 }
std2 = { 'name': 'Bob', 'score': 81 }
def print_score(std):
    print('%s: %s' % (std['name'], std['score']))
```

- 面向对象思想：
  - 学生就是对象（Object），属性包括姓名与分数（数据封装）
  - 向学生对象发送“打印成绩”的消息，对象自身完成打印

对象 = 属性 + 方法

Python面向对象编程

- 面向对象编程：定义学生类（Class）
  - 类是一系列具有相同属性和方法的对象的抽象
  - 对象是类的实例

表示父类，Python中object类是所有类的父类

类的初始化方法，创建对象时自动调用
self在类内部表示对象自身
name, score是方法的普通参数

```
class Student(object):
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('%s: %s' % (self.name, self.score))
```

```
bart = Student('Bart Simpson', 59)
lisa = Student('Lisa Simpson', 87)
bart.print_score()
lisa.print_score()
```

类的普通方法，通过对象实例调用

Python面向对象编程

- __init__方法：
  - 在创建对象时自动调用的方法
  - 用于初始化对象的属性
  - 第一个参数总是self，在类内部表示对象自身
  - 其他参数 name, score 为普通参数
  - 调用（创建对象）时只需提供普通参数，不需要提供 self

```
class Student(object):
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('%s: %s' % (self.name, self.score))
```

self 在类内部表示对象自身

Python面向对象编程

- print_score方法：
  - 类的普通方法同样总是需要定义self参数
  - 与普通函数并无其他不同
  - 通过对象实例来调用类方法

```
class Student(object):
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('%s: %s' % (self.name, self.score))
```

```
bart = Student('Bart Simpson', 59)
lisa = Student('Lisa Simpson', 87)
bart.print_score()
lisa.print_score()
```

Python特性与技巧

- 类的特殊方法（魔术方法）
- 特殊方法都以两个下划线开始和结束
- __init__就是一个魔法方法，创建对象时自动调用，用于初始化对象
- 同样还有：__del__，在销毁对象时自动调用

```
from os.path import join

class FileObject:
    '''给文件对象进行包装从而确认在删除时文件流关闭'''

    def __init__(self, filepath='~', filename='sample.txt'):
        #读写模式打开一个文件
        self.file = open(join(filepath, filename), 'r+')

    def __del__(self):
        self.file.close()
        del self.file
```

类的魔法方法

- 算术运算符的魔法方法：允许对象进行四则运算
  - __add__(self, other)、__sub__(self, other)
  - __mul__(self, other)、__div__(self, other)
- 大小比较的魔法方法：在对象间定义比较方法
  - __eq__(self,other)：使用==操作符（equal）
  - __ne__(self,other)：使用!=操作符（not equal）
  - __lt__(self,other)：使用<操作符（less than）
  - __gt__(self,other)：使用>操作符（greater than）
  - __le__(self,other)：使用<=操作符（less or equal）
  - __ge__(self,other)：使用>=操作符（greater or equal）
- 实现了lt等比较方法之后，还可以使用sorted()函数排序对象列表

类的魔法方法

- 其他魔法方法：
- 字符串函数 __str__(self)
  - 定义了将对象转换成字符串的默认方法
  - 允许将对象当成字符串一样使用，例如直接print
- 长度函数 __len__(self)
  - 返回该对象（通常，作为容器所包含的元素序列）的长度
  - 允许将该对象作为 len 函数的参数

类的魔法方法

- 像容器一样使用对象：
  - __iter__(self)与__next__(self)：使对象成为可迭代对象，用于for循环遍历
  - __getitem__(self, n)：通过数字索引获取元素
  - __setitem__(self, n)：通过数字索引设置元素
  - __getattr__(self, n)：通过key来获取value
  - __getattr__(self, n)：通过key来设置value

Python推导式

- 再来看学生成绩的例子：

```
students_scores = [
    {"name": "张三", "math": 85, "english": 90, "chinese": 88},
    {"name": "李四", "math": 78, "english": 82, "chinese": 76},
    {"name": "王五", "math": 92, "english": 88, "chinese": 90},
    {"name": "赵六", "math": 67, "english": 75, "chinese": 70},
    {"name": "孙七", "math": 80, "english": 85, "chinese": 82}
]
```

- 需求：找出其中数学成绩80分以上的学生
  - 写段程序遍历比较？
    - for student in student_scores:

      if student[‘math’] > 80: …….
  - 推导式：

    [student[‘name’] for student in students_scores if student[‘math’] > 80]

Python推导式

- 研究这一推导式：

  [student[‘name’] for student in students_scores if student[‘math’] > 80]

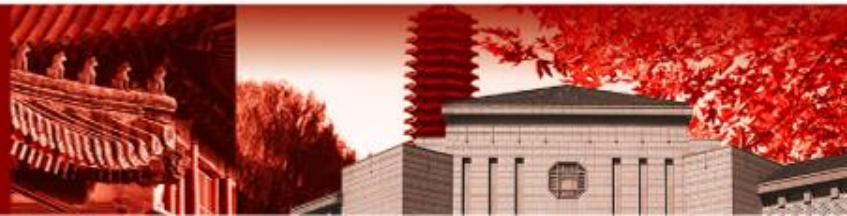
  – 推导式：基于一个容器快速创建一个新的容器

  – 循环部分：定义了待遍历的容器与循环变量

  – 求值部分：基于循环变量计算新的值，构成结果

  – 条件部分[可选]：对原容器中的变量进行筛选

- 列表推导式：[<表达式> for <变量> in <可迭代对象> if <逻辑条件>]
- 字典推导式：{<键值表达式>:<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}
- 集合推导式：{<元素表达式> for <变量> in <可迭代对象> if <逻辑条件>}
- 推导式还支持更复杂的二重循环情形



Python推导式

- OJ练习时：输入为一行数字，如何快速转换为数值？
- input：1 2 3 4 5

```
numbers = input()                    # “1 2 3 4 5”
numbers = numbers.split()            # [‘1’,‘2’,‘3’,‘4’,‘5’]
numbers = [int(x) for x in numbers]
```

- 合起来只需要一行：

```
numbers = [int(x) for x in input().split()]
```

函数的可选参数

- 可选参数：为参数提供默认值
- 举例：为小学生入学注册的函数，需要传入name, gender

```
def enroll(name, gender):
    print('name:', name)
    print('gender:', gender)
```

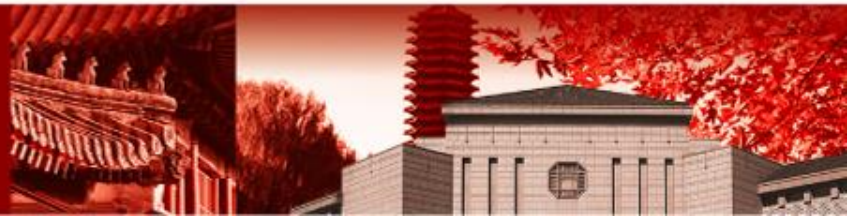
```
>>> enroll('Sarah', 'F')
name: Sarah
gender: F
```

– 如果还要传入年龄、城市等信息怎么办？这增加了函数的复杂性

– 使用可选参数：大多数学生仍然只需要传入2个参数

```
def enroll(name, gender, age=6, city='Beijing'):
    print('name:', name)
    print('gender:', gender)
    print('age:', age)
    print('city:', city)
```

```
>>> enroll('Sarah', 'F')
name: Sarah
gender: F
age: 6
city: Beijing
```

函数的可变参数

- 考虑：给定一组数字a, b, c,…，请计算 $a^2+b^2+c^2+\ldots$
  - 要定义出这个函数，我们必须确定输入的参数。
  - 由于参数个数不确定，首先想到可以将参数作为list/tuple传入
    - 但是在调用之前，必须组装这样一个list/tuple
- Python提供了更方便的机制
  - 在参数前加*号，表示可变参数
  - numbers作为可变参数，接受任意数量个参数
  - numbers将自动成为tuple类型的变量
- 如果已经有了list/tuple变量，怎么办？
  - 在调用时同样可以加*号
  - 表示将其中的所有元素作为可变参数传入

```
def calc(*numbers):
    sum = 0
    for n in numbers:
        sum = sum + n * n
    return sum
```

```
>>> calc(1, 2, 3)
14
>>> calc(1, 3, 5, 7)
84
```

```
>>> nums = [1, 2, 3]
>>> calc(*nums)
14
```

函数的关键字参数

- 可变参数允许你传入0个或任意个参数
  - 这些可变参数在函数调用时自动组装为一个tuple。
- 关键字参数允许你传入0个或任意个含参数名的参数
  - 这些关键字参数在函数内部自动组装为一个dict。

函数的关键字参数

- 函数接受关键字参数**kw（keyword argument）
- kw变量在函数中为字典类型
- 在调用时，函数接受任意个数的关键字参数
- 如果已有一个字典结构，同样可以使用**直接传入
  - 这大大增加了函数的可扩展性。想象：你正在实现用户注册的功能，除了姓名与年龄为必填项外，仍有大量的可选项
  - 传入关键字参数也有助于保持良好的代码风格
    - 可读性：每个参数的含义十分明确，即使参数顺序被打乱也不会引起错误
    - 灵活性：如果需要修改参数，使用关键字参数的代码则不需要修改

```
def person(name, age, **kw):
    print('name:', name, 'age:', age, 'other:', kw)
```

```
>>> person('Bob', 35, city='Beijing')
name: Bob age: 35 other: {'city': 'Beijing'}
>>> person('Adam', 45, gender='M', job='Engineer')
name: Adam age: 45 other: {'gender': 'M', 'job': 'Engineer'}
```