



Python快速入门

- Python基础知识
- Python字符串
- Python数据类型
- Python运算符
- Python流程控制
- Python文件操作
- Python函数
- Python常用内建函数

Python 基础知识

- Python介绍
- Python安装
- Python解释器
- 运行第一个程序
- 基本数据类型
- 算术运算符
- 变量
- 赋值操作符
- 转义字符
- 获取用户输入
- 注释
- 综合案例：实现简单的计算器



Python是一种面向对象、解释型、多用途设计语言，具有很丰富和强大的库，语法简洁，强制用空格作为语法缩进，能够完成快速项目开发，相比传统语言开发效率提高数倍。

应用领域：系统运维、网站开发、科学计算、爬虫、人工智能等

Web框架：

- Django（最流行）
- Flask（轻量级）
- Tornado（异步）

Python介绍

为什么选择Python?

- 语法简介，易于学习
- 广泛的标准库，适合快速开发
- 跨平台，基本所有操作系统都能运行
- 是DevOps开发领域应用最广泛的语言

Oct 2020	Oct 2019	Change	Programming Language	Ratings	Change
1	2	⬆	C	16.95%	+0.77%
2	1	⬇	Java	12.56%	-4.32%
3	3		Python	11.28%	+2.19%
4	4		C++	6.94%	+0.71%
5	5		C#	4.16%	+0.30%
6	6		Visual Basic	3.97%	+0.23%
7	7		JavaScript	2.14%	+0.06%
8	9	⬆	PHP	2.09%	+0.18%
9	15	⬆	R	1.99%	+0.73%
10	8	⬇	SQL	1.57%	-0.37%
11	19	⬆	Perl	1.43%	+0.40%
12	11	⬇	Groovy	1.23%	-0.16%
13	13		Ruby	1.16%	-0.16%
14	17	⬆	Go	1.16%	+0.06%
15	20	⬆	MATLAB	1.12%	+0.19%

编程语言热门程度
<https://www.tiobe.com/tiobe-index>

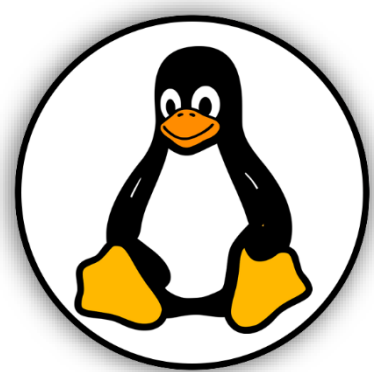
Python介绍



学习Python编程：

- Python官方文档： <https://www.python.org/doc>
- iPython：升级版的python解释器
- PyCharm：一款功能强大的Python集成开发环境
- Sublime：代码编辑器
- Jupyter notebook：在网页中编写和运行代码
- Pip：Python模块安装工具

Python安装



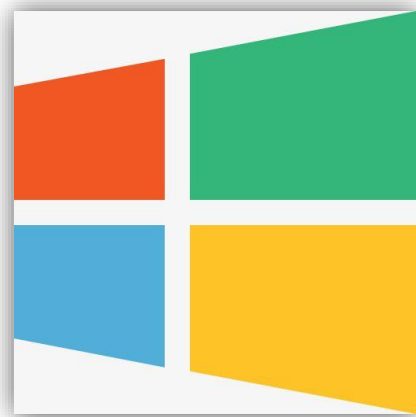
Linux安装: `yum install python36 -y`
一条命令完成安装。

或者编译安装指定版本: <https://www.python.org/downloads/source/>

```
wget https://www.python.org/ftp/python/3.8.6/Python-3.8.6.tgz
yum install zlib-devel -y
tar zxvf Python-3.8.6.tgz
cd Python-3.8.6
./configure
make && make install
```

- [Python 3.8.6 - Sept. 24, 2020](#)
 - Download [Gzipped source tarball](#)
 - Download [XZ compressed source tarball](#)

Python安装



Windows安装: <https://www.python.org/downloads/windows/>
官网下载安装程序。

- [Python 3.8.6 - Sept. 24, 2020](#)
Note that Python 3.8.6 cannot be used on Windows XP or earlier.
 - Download [Windows help file](#)
 - Download [Windows x86-64 embeddable zip file](#)
 - Download [Windows x86-64 executable installer](#)
 - Download [Windows x86-64 web-based installer](#)
 - Download [Windows x86 embeddable zip file](#)
 - Download [Windows x86 executable installer](#)
 - Download [Windows x86 web-based installer](#)

Windows设置环境变量:

右击我的电脑->属性->高级系统设置->环境变量->Path->编辑->新建->粘贴Python安装目录
(例如D:\Python3.7\Scripts)

Python安装

交互式解释器：

```
C:\Users\zhenl>python
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> _
```

```
[root@localhost ~]# python3.8
Python 3.8.6 (default, Oct 29 2020, 23:10:08)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

运行第一个程序

运行第一个程序：

```
vim hello.py
```

```
#!/usr/bin/python3.8
```

```
print("Hello World!")
```

```
python3.8 hello.py
```

```
Hello World!
```

基本数据类型

Python的基本数据类型：

- 整数（int），例如6
- 浮点数（float），例如6.6
- 字符串（str），例如"6","python"
- 布尔值（bool），例如True、False

注：使用type()内建函数查看对象类型。

算术运算符

什么是运算符？

举个简单的例子 $6+6=12$ ，其中两个6被称为操作数，+称为运算符。

运算符	描述	示例
+	加法	(6 + 6) 结果12
-	减法	(6 - 6) 结果0
*	乘法	(6 * 6) 结果36
/	除	(8 / 6) 结果1.33333333
//	整除	(8 / 6) 结果1
%	取余	(6 % 6) 结果0
**	幂	(6 ** 3) 结果46656，即6 * 6 * 6
()	小括号	小括号用来提高运算优先级，即(1+2)*3 结果为9

变量

变量：编程语言中能储存结果或能表示值的抽象概念。

用途：给一段数据赋予一个简短、易于记忆的名字，方便重用。

变量赋值：

变量名=变量值

例如：name= “aliang”

多重赋值：

name1, name2 = “aliang” ,” lizhenliang”

变量引用：

print(变量名)

变量

操作符号	描述
%s	字符串
%d	整数
%f	浮点数，可指定小数点后的精度

格式化字符串：
print("hello %s" %name)

保留2位小数点：
calc = 100 / 88
print('计算结果： %.2f' %calc)

赋值操作符

操作符	描述	示例
=	赋值，将=左侧的结果赋值给等号左侧的变量	a = 10 b = 20
+=	加法赋值	c += a 等价 c = c + a
-=	减法赋值	c -= a 等价 c = c - a
*=	乘法赋值	c *= a 等价 c = c * a
/=	除法赋值	c /= a 等价 c = c / a
//=	整除赋值	c //= a 等价 c = c // a
%=	取余赋值	c %= a 等价 c = c % a
**=	幂赋值	c **= a 等价 c = c ** a

转义符

转义字符	说明
\n	换行符，将光标位置移到下一行开头
\r	回车符，将光标位置移到本行开头
\t	水平制表符，也即 Tab 键，一般相当于四个空格
\b	退格（Backspace），将光标位置移到前一个
\\	反斜线
\'	单引号
\"	双引号
\\	在字符串行尾的续行符，即一行未完，转到下一行继续写

示例：

```
print("姓名: %s, \"年龄: %d" %(name, age))
```

获取用户输入

input()内建函数：用于与用户交互，接收一个参数，即要向用户显示的提示或者说明，让用户知道该怎么做。

示例：

```
name = input( "请输入你的姓名：" )  
print(name)
```

获取用户输入

- 一个#号表示单行注释

```
# print(name)
```

- 三个单引号或者三个双引号表示多行注释

```
'''
```

多行注释

```
'''
```

综合案例：简单计算器实现

```
print("选择算术运算符: ")
print("""
    1、加
    2、减
    3、乘
    4、除
    """)
choice = input("请输入编号: ")
num1 = int(input("请输入第一个数字: "))
num2 = int(input("请输入第二个数字: "))

if choice == '1':
    print(num1 + num2)
elif choice == '2':
    print(num1 - num2)
elif choice == '3':
    print(num1 * num2)
elif choice == '4':
    print(num1 / num2)
else:
    print("输出格式错误!")
```

Python 字符串

- 字符串格式化输出
- 字符串拼接
- 获取字符串长度
- 字符串切片
- 字符串处理方法

字符串格式化输出

```
name = "aliang"
```

```
age = 30
```

```
# 方法1
```

```
print("我的名字是%s,今年%s岁了。" % (name, age))
```

```
# 方法2
```

```
print(f"我的名字是{name},今年{age}岁了。")
```

字符串拼接

使用 “+” 可以对多个字符串进行拼接。

示例：

```
str1 = "hello"
```

```
str2 = "world"
```

```
print(str1 + str2)
```

其他方式：

- 格式化输出
- 逗号
- `join()` 内建函数，将序列中的元素拼接成一个字符串。后面讲到

获取字符串长度

len()内建函数： 计算字符串的长度。

语法格式：len(string)

字符串切片

切片：截取字符串中的某部分内容

语法格式：string[start:end:step]

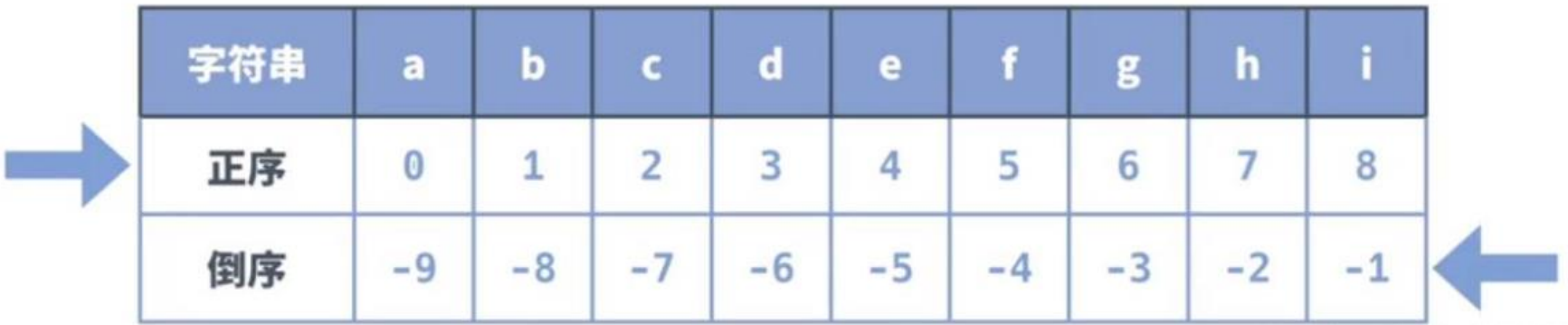
- string 要切片的字符串
- start 要切片的第一个字符的索引（包括该字符），如果不指定默认为0
- end 要切片的最后一个字符的索引（不包括该字符），如果不指定默认为字符串的长度
- step 表示切片的步长，如果不指定默认为1

示例：

截取第5个字符：s[4]

截取第1个字符到第5个字符：s[0:5]

截取最后1个字符：s[-1]



The diagram illustrates string indexing for the string 'abcdefghi'. It consists of a table with three rows and ten columns. The first row is the header with the string '字符串' and characters 'a' through 'i'. The second row, labeled '正序' (Forward), shows indices from 0 to 8. The third row, labeled '倒序' (Reverse), shows indices from -9 to -1. A blue arrow points to the first column (index 0/-9), and another blue arrow points to the last column (index 8/-1).

字符串	a	b	c	d	e	f	g	h	i
正序	0	1	2	3	4	5	6	7	8
倒序	-9	-8	-7	-6	-5	-4	-3	-2	-1

字符串处理方法

```
xxoo = "abcdef"
print("首字母大写: %s" % xxoo.capitalize())
print("字符l出现次数: %s" % xxoo.count('l'))
print("感叹号是否结尾: %s" % xxoo.endswith('!'))
print("w字符是否是开头: %s" % xxoo.startswith('w'))
print("w字符索引位置: %s" % xxoo.find('w')) # xxoo.index('W')
print("格式化字符串: Hello{0} world!".format(','))
print("是否都是小写: %s" % xxoo.islower())
print("是否都是大写: %s" % xxoo.isupper())
print("所有字母转为小写: %s" % xxoo.lower())
print("所有字母转为大写: %s" % xxoo.upper())
print("感叹号替换为句号: %s" % xxoo.replace('!', '.'))
print("以空格分隔切分成列表: %s" % xxoo.split(' '))
print("切分为一个列表: %s" % xxoo.splitlines())
print("去除两边空格: %s" % xxoo.strip())
print("大小写互换: %s" % xxoo.swapcase())
```

运行结果:

```
首字母大写: Abcdef
字符l出现次数: 0
感叹号是否结尾: False
w字符是否是开头: False
w字符索引位置: -1
格式化字符串: Hello, world!
是否都是小写: True
是否都是大写: False
所有字母转为小写: abcdef
所有字母转为大写: ABCDEF
感叹号替换为句号: abcdef
以空格分隔切分成列表: ['abcdef']
切分为一个列表: ['abcdef']
去除两边空格: abcdef
大小写互换: ABCDEF
```

Python 数据类型

- 列表
- 元组
- 集合
- 字典
- 常用数据类型转换

在Python中，组合数据类型有：列表（list）、元组（tuple）、字典（dict）、集合（set）。

组合数据类型：为了方便处理数据，把一些同类数据放到一起，是一组数据的集合。

列表

列表（List）：是一个序列的数据结构。

什么是序列？

是指它成员都是有序排列，并且可以通过索引访问一个或多个成员。

格式：名称 = [“元素1” ， “元素2” ， ...]

列表：基本操作

定义列表：computer = ["主机","显示器","鼠标","键盘"]

类型	方法	用途
查	index("元素")	查看元素索引位置
	count("元素")	统计元素出现次数
	reverse()	倒序排序元素
	sort()	进行排序
增	append("元素")	追加一个元素
	insert(index, "元素")	在指定索引位置插入一个元素
改	computer[index] = "元素"	修改指定索引的值
删	remove("元素")	删除指定元素
	pop(index=-1)	通过索引删除元素并返回索引

列表：切片

与字符串切片使用方法一样。

语法格式：list[start:end:step]

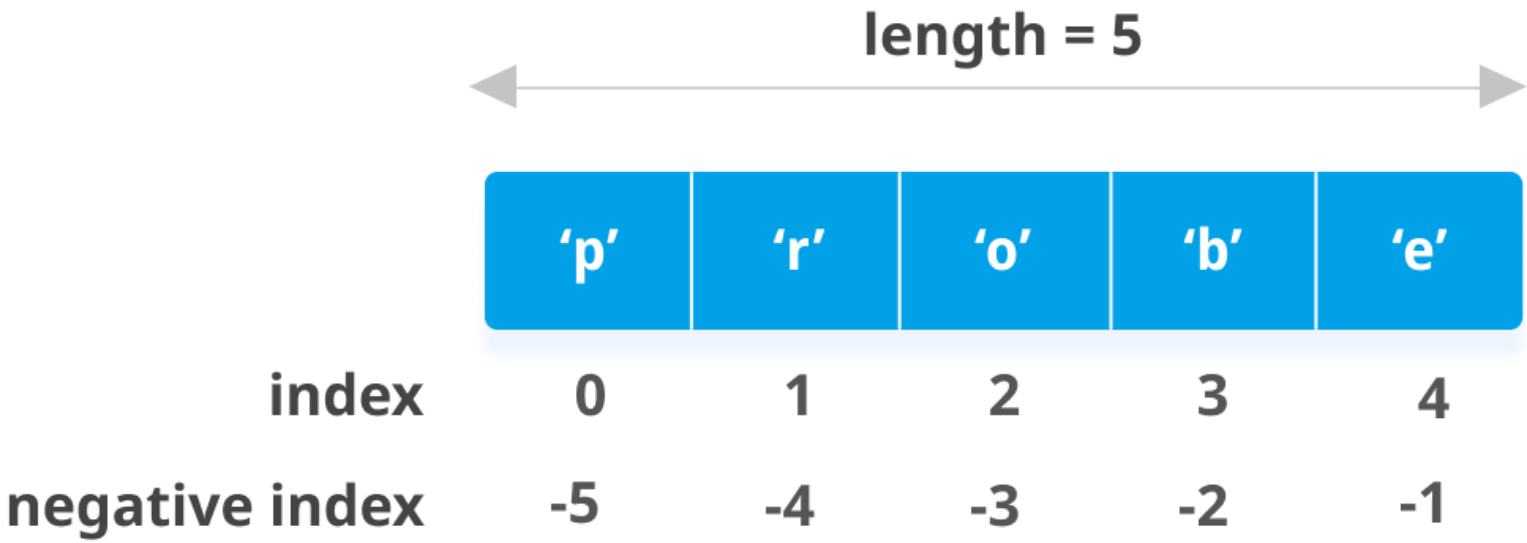
示例：

截取第1个元素：computer[0]

截取第1个字符到第5个元素：computer[0:5]

截取第1个到倒数第2个元素：computer[0:-1]

截取最后1个元素：computer[-1]



列表：清空列表

- 重新初始化列表

```
computer = []
```

- del语句删除列表

```
del computer
```

元组

元组 (Tuple)：与列表类似，也是一个序列数据结构。主要区别在于元组中的元素不能修改。

格式：名称 = ("元素1" , "元素2" , ...)

元组：基本操作

定义元组：computer = ("主机","显示器","鼠标","键盘")

类型	方法	用途
查	index("元素")	查看元素索引位置
	count("元素")	统计元素出现次数

同样支持切片，与列表使用方法一样。

集合

集合 (Set) : 是一个无序、不重复元素的序列, 主要用于元素去重和关系测试。

关系测试支持:

- 联合
- 交集
- 差集
- 对称差集

定义方法: `set()`函数或者大括号来创建集合。

注意: 想要创建空集合, 必须使用`set()`而不是`{}`。后者用于创建空字典。

集合：基本操作

定义空数组：computer = set()

定义元组：

computer = {"主机","显示器","鼠标","键盘"}

或者

computer = set(["主机","显示器","鼠标","键盘","主机"])

类型	方法	用途
查		不支持，一般用for遍历
增	add("元素")	添加元素
删	remove("元素")	删除指定元素
	discard("元素")	如果有元素则删除
	pop("元素")	删除第1个元素并返回元素

列表去重：

computer = ["主机","显示器","鼠标","键盘","显示器","鼠标"]

s = set(computer)

print(s)

集合：关系测试

关系符号	描述
-	差集
&	交集
	合集、并集
!=	不等于
==	等于

对两个列表进行关系测试：

```
a = set([1, 2, 3, 4, 5, 6])
```

```
b = set([4, 5, 6, 7, 8, 9])
```

返回a集合中元素在b集合没有的

```
print(a - b)
```

返回b集合中元素在a集合中没有的

```
print(b - a)
```

返回交集，即两个集合中一样的元素

```
print(a & b)
```

返回合集，即合并去重

```
print(a | b)
```

判断是否不相等

```
print(a != b)
```

判断是否相等

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

字典

字典 (Dict) : 是一个具有映射关系的数据结构。用于存储有一定关系的元素。

格式: `d = {'key1':value1, 'key2':value2, 'key3':value3}`

注意: 字典通过key来访问value, 因此字典中的key不允许重复。

A Key Value Pair

Key Value
“1”: “FACE Prep”

A Dictionary

Dict1 = {“1”: “FACE Prep”, “2”: “Python”}

Separator (comma)

字典：基本操作

定义字典：computer = {"主机":5000,"显示器":1000,"鼠标":60,"键盘":150}

类型	方法	用途
查	computer["key"]	获取key的值
	computer.get("key", None)	获取key的值， 如果不存在返回None
	keys()	获取所有键
	values()	获取所有值
	items()	获取所有键值
增	computer["key"] = value	添加键值， 如果键存在则覆盖
	update("key")	添加新字典
	setdefault("key", default=None)	如果键不存在， 添加键并将值设置默认值 如果键存在返回值
删	pop("key")	删除指定键
	computer.popitem()	删除最后一对键值并返回

字典：嵌套

字典里的值不但是可以写整数、字符串，也可以是其他数据类型，例如列表、元组、集合、字典，这样可满足一个键还包含其他属性。

```
computer = {"主机":{"CPU":1300,"内存":400,"硬盘":200},"显示器":1000,"鼠标":60,"键盘":["机械键盘","薄膜键盘"]}
```

操作key中字典：

```
computer["主机"]["硬盘"]
```

```
computer["主机"]["硬盘"] = "300 "
```

操作key中列表：

```
computer["键盘"]
```

```
computer["键盘"].append("其他")
```

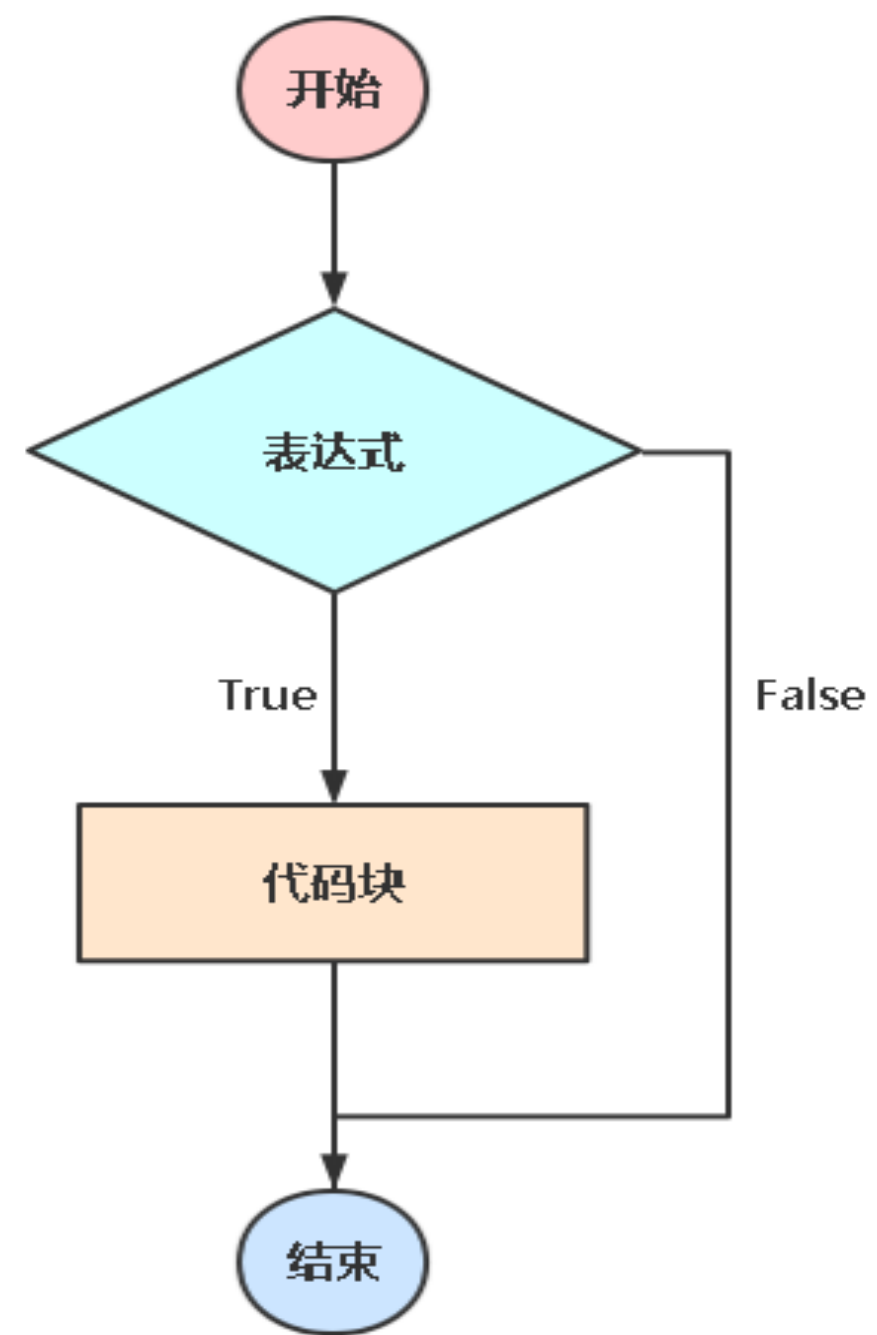
常见数据类型转换

函数	描述
int(x[,base])	将x转换为一个整数
float(x)	将x转换为一个浮点数
str(x)	将对象x转换为字符串
tuple(seq)	将序列seq转换为一个元组
list(seq)	将序列seq转换为一个列表

Python 操作符

- 比较操作符
- 逻辑操作符
- 成员操作符
- 身份操作符

操作符有什么用?



操作符：一个特定的符号，用它与其他数据类型连接起来组成一个表达式。常用于条件判断，根据表达式返回True/False采取动作。

比较操作符

比较操作符： 比较两边值

操作符	描述	示例
==	相等，两边值是否相等	(6 == 6) 结果 True
!=	不相等，两边值是否不相等	(6 != 6) 结果False
>	大于，左边值是否大于右边值	(8 > 6) 结果True
<	小于，左边值是否小于右边值	(8 < 6) 结果False
>=	大于等于，左边值是否大于等于右边值	(6 >= 6) 结果True
<=	小于等于，左边值是否小于等于右边值	(6 <= 6) 结果True

逻辑操作符

逻辑操作符：判断条件逻辑

操作符	逻辑表达式	描述
and	x and y	与，所有的条件都 True结果才为True； 只要有一个为 False， 结果就为False
or	x or y	或，所有的条件只要有一个是True结果就为True
not	not x	非， 结果取反

成员操作符

成员操作符： 判断某个元素是否在数据类型里

操作符	描述	示例
in	如果在指定的序列中找到值返回True， 否则返回False	computer = ["主机","显示器","鼠标","键盘"] ("主机" in computer) 结果True ("音响" in computer) 结果False
not in	如果在指定的序列中没有找到值返回True， 否则返回False	print("主机" not in computer) 结果False print("音响" not in computer) 结果True

身份操作符

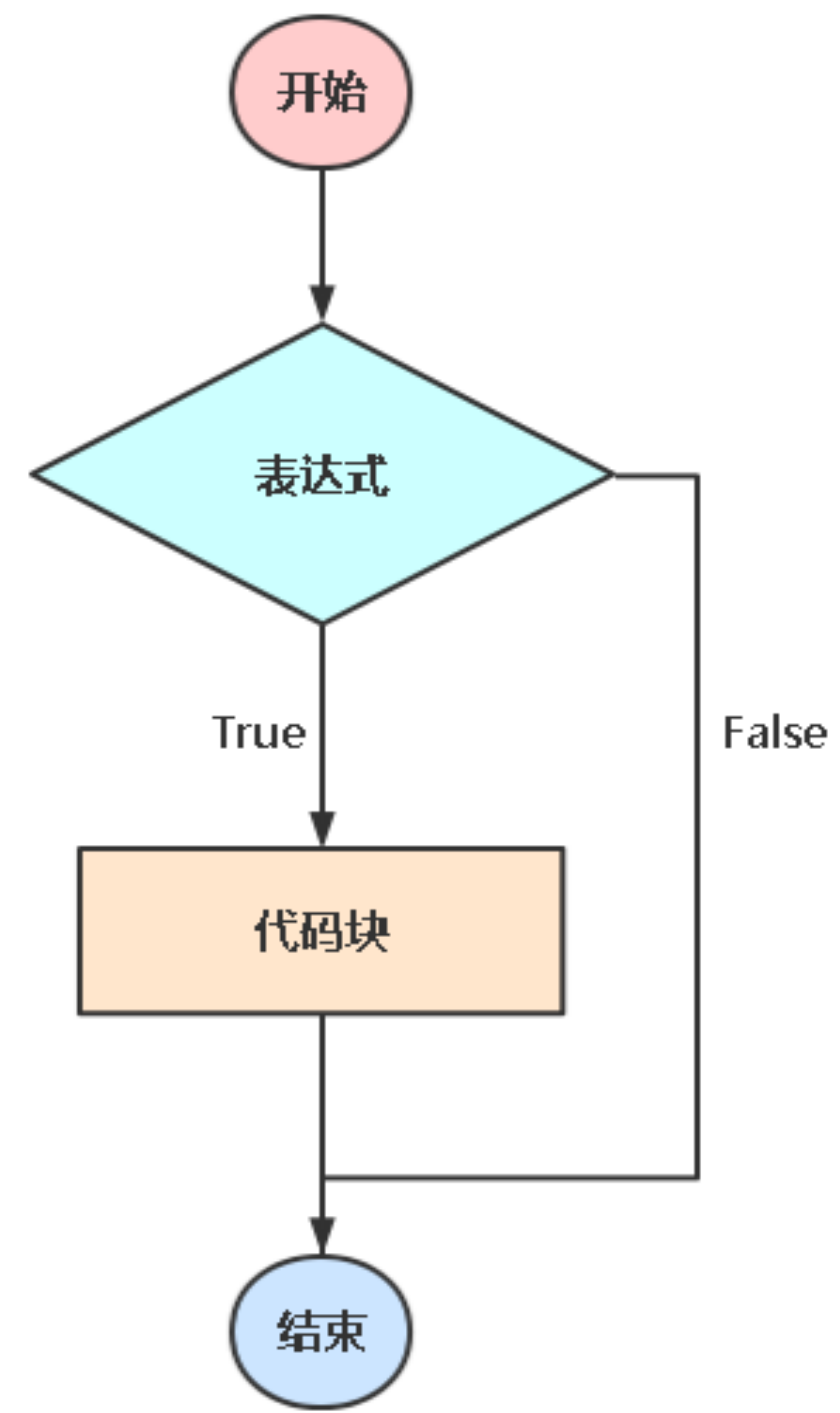
身份操作符：判断两个对象是否相等

操作符	描述
is	判断两个标识符是否引用一个对象
is not	判断两个标识符是否引用不同对象

Python 流程控制

- 条件判断
- 循环语句
- pass语句

条件判断



语法:

if <表达式>:

 <代码块>

elif <表达式>:

 <代码块>

else:

 <代码块>

条件判断：单分支

示例：判断是否成年

```
age = int(input("请输入你的年龄: "))
```

```
if age > 18:
```

```
    print("恭喜，你已经成年！")
```

```
else:
```

```
    print("抱歉，你还未成年！")
```

简写，也成三目表达式： "恭喜，你已经成年！ " if age > 18 else "抱歉，你还未成年！ "

条件判断：多分支

示例：根据人的年龄段划分

```
age = int(input("请输入你的年龄: "))
```

```
if age < 7 :
```

```
    print("儿童")
```

```
elif age >= 7 and age < 17:
```

```
    print("少年")
```

```
elif age >= 18 and age < 40:
```

```
    print("青年")
```

```
elif age >= 41 and age < 48:
```

```
    print("壮年")
```

```
else:
```

```
    print("老年")
```

循环

在了解编程中的“循环”之前，先试想下这个场景：
在阳台种花，准备种4颗种子，开始逐个挖坑，放一颗种子。



每一颗种子操作都是相同的，如果我们用一步将6颗种子重复种下的行为表示出来呢？

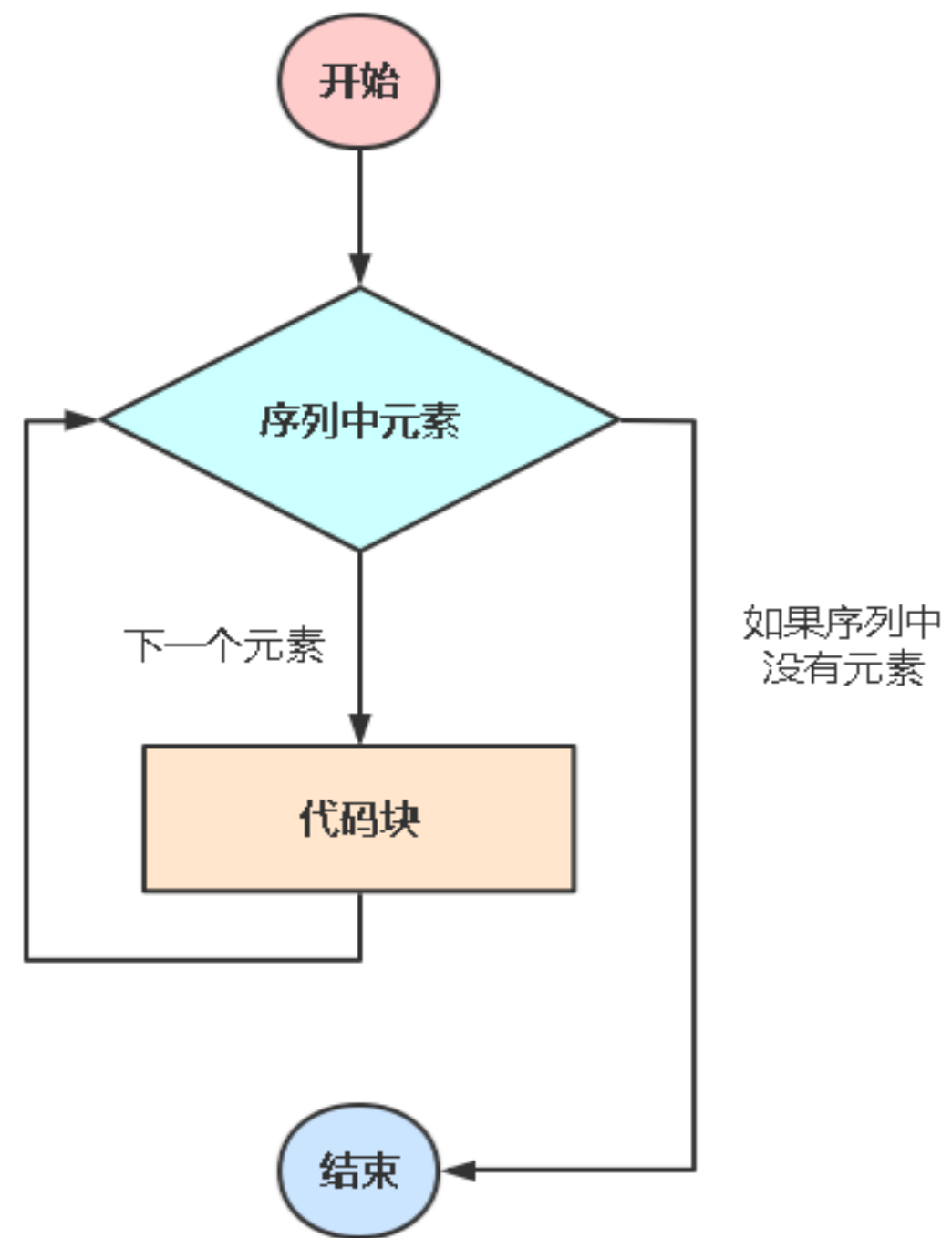
```
for n in range(1,5):  
    print("开始种花第%s次" %n)
```

循环

循环的作用在于将一段代码重复执行多次。

Python中实现循环常用有两个语句：for、while

for语句



for语句：一般用于遍历数据类型的元素进行处理，例如字符串、列表。

语法：

```
for <变量> in <序列>:  
    <代码块>
```

for语句

示例1：遍历字符串

```
s = "123456"
```

```
for i in s:
```

```
    print(i)
```

示例2：遍历列表

```
computer = ["主机","显示器","鼠标","键盘"]
```

```
for i in computer:
```

```
    print(i,len(i))
```

示例3：遍历字典

```
computer = {"主机":5000,"显示器":1000,"鼠标":60,"键盘":150}
```

```
for i in computer.items():
```

```
    print(i)
```

```
    print("名称: %s\t价格: %s" % (i[0],i[1]))
```

示例4：嵌套循环

```
s1 = "123456"
```

```
s2 = "456789"
```

```
for i in s1:
```

```
    for x in s2:
```

```
        if i == x:
```

```
            print(i)
```

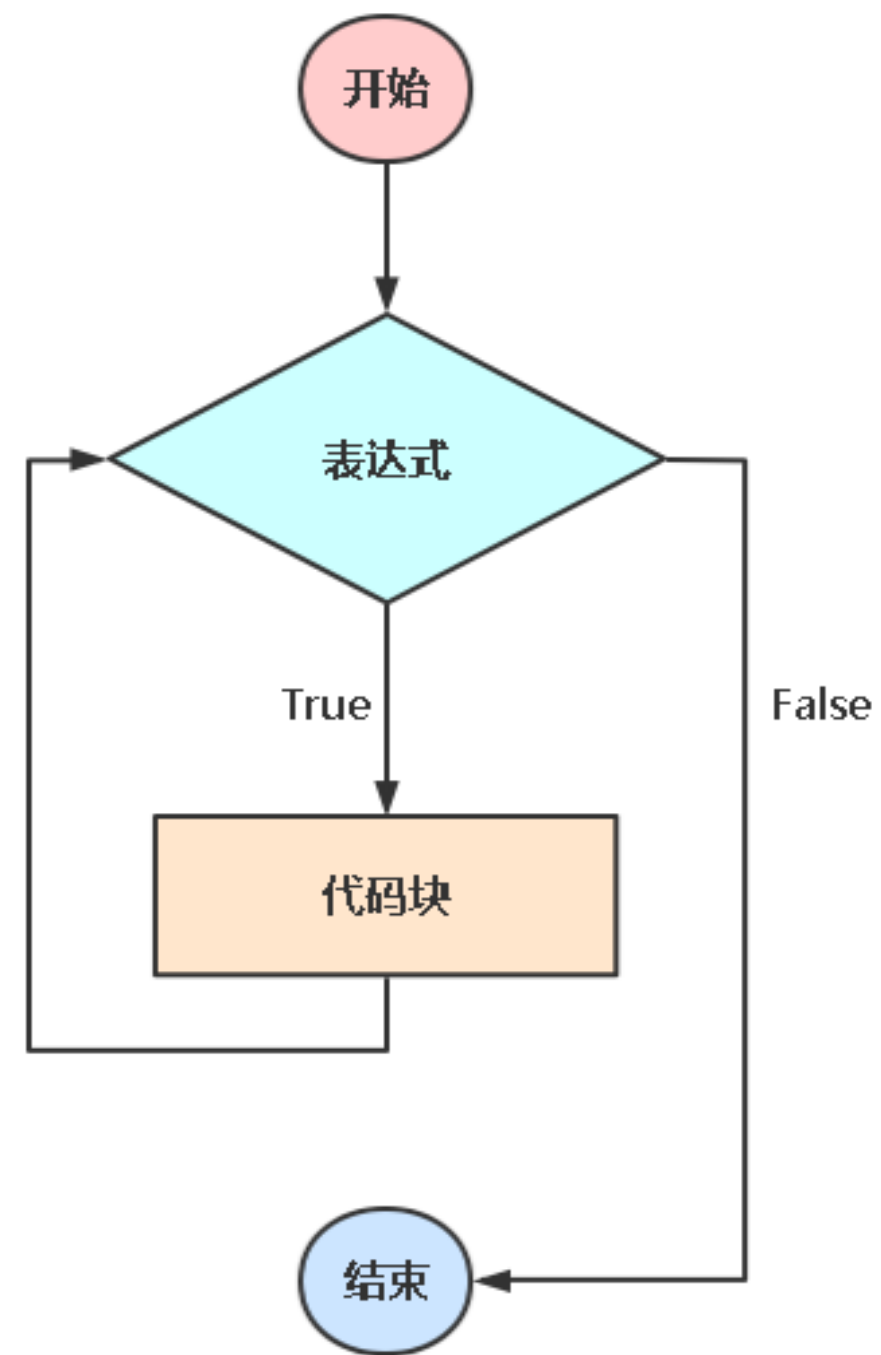
for语句

range()内建函数：动态生成数字序列，例如range(1,6)，结果类似列表[1,2,3,4,5,6]

示例：生成0-4序列

```
for i in range(5):  
    print(i)
```

while语句



while语句：在某条件下循环执行一段代码，即重复相同的任务。

语法：

while <表达式>:

<代码块>

while语句

示例1：当条件满足时停止循环

```
count = 0
```

```
while count < 5:
```

```
    print(count)
```

```
    count += 1
```

示例2：死循环

```
count = 0
```

```
while True:
```

```
    print(count)
```

```
    count += 1
```

continue与break语句

- continue 当满足条件时，跳出本次循环
- break 当满足条件时，跳出所有循环

注：只有在for、while循环语句中才有效。

示例1：continue

```
for n in range(1,6):  
    if n == 3:  
        continue  
    else:  
        print(n)
```

示例2：break

```
for n in range(1,6):  
    if n == 3:  
        break  
    else:  
        print(n)
```

综合案例：用户登录，三次错误机会

示例：

```
count = 0
```

```
while 1:
```

```
    if count < 3:
```

```
        name = input("请输入你的用户名: ").strip() # .strip()去除首尾空格
```

```
        if len(name) == 0:
```

```
            print("输入不能为空!")
```

```
            continue
```

```
        elif name == "aliang":
```

```
            print("登录成功.")
```

```
            break
```

```
        else:
```

```
            print("用户名错误，请重新输入!")
```

```
            count += 1
```

```
    else:
```

```
        print("超出错误次数, 退出!")
```

```
        break
```

Python 文件操作

- `open()`函数
- 文件对象操作
- `with`语句

| open()函数

要想读取文件（如txt、csv等），第一步要用open()内建函数打开文件，它会返回一个文件对象，这个对象拥有read()、write()、close()等方法。

语法：open(file, mode='r', encoding=None)

- file：打开的文件路径
- mode（可选）：打开文件的模式，如只读、追加、写入等
 - r：只读
 - w：只写
 - a：在原有内容的基础上追加内容（末尾）
 - w+：读写

如果需要以字节（二进制）形式读取文件，只需要在mode值追加 'b' 即可，例如wb

文件对象操作

```
f = open('test.txt')
```

方法	描述
f.read([size])	读取size字节，当未指定或给负值时，读取剩余所有的字节，作为字符串返回
f.readline([size])	从文件中读取下一行，作为字符串返回。如果指定size则返回size字节
f.readlines([size])	读取size字节，当未指定或给负值时，读取剩余所有的字节，作为列表返回
f.write(str) f.flush	写字符串到文件 刷新缓冲区到磁盘
f.seek(offset[, whence=0])	在文件中移动指针，从whence（0代表文件起始位置，默认。1代表当前位置。2代表文件末尾）偏移offset个字节
f.tell()	当前文件中的位置（指针）
f.close()	关闭文件

文件对象操作

示例：遍历打印每一行

```
f = open('computer.txt')
```

```
for line in f:
```

```
    print(line.strip( '\n' )) # 去掉换行符
```

with语句

with语句：不管在处理文件过程中是否发生异常，都能保证 with 语句执行完毕后已经关闭了打开的文件句柄。

示例：

```
with open("computer.txt",encoding="utf8") as f:  
    data = f.read()  
    print(data)
```


Python 函数

- 函数定义与调用
- 函数参数
- 匿名函数 (Lambda)
- 作用域
- 闭包
- 函数装饰器

函数定义与调用

函数：是指一段可以直接被另一段程序或代码引用的程序或代码。

在编写代码时，常将一些常用的功能模块编写成函数，放在函数库中供公共使用，可减少重复编写程序段和简化代码结构。

语法：

```
def 函数名称(参数1, 参数2, ...):  
    <代码块>  
    return <表达式>
```

示例：

```
def hello():  
    print("Hello World!")  
hello() # 调用函数
```

函数参数：接收参数

示例：求和函数

```
def f(a, b):  
    return a + b
```

```
print(f(1,2))    # 按参数位置赋值
```

```
print(f(b=2,a=1)) # 按对应关系赋值
```

示例：生成序列列表

```
def seq(n):  
    result = []  
    x = 0  
    while x < n:  
        result.append(x)  
        x += 1  
    return result  
print(seq(9))
```

函数参数：参数默认值

参数默认值：预先给参数定义默认值，如果调用函数时没指定该值，则用默认值。

示例：

```
def f(a, b=2):  
    return a + b
```

```
print(f(1))
```

```
print(f(1,3))
```

函数参数：接收任意数量参数

前面学习的是固定参数数量，当不知道有多少个参数时可以在参数前面加*与**，表示可接收任意多个参数。

- *name 接收任意多个参数，并放到一个元组中。
- **name 接收一个键值，并存储为字典。

示例：传入多个参数，计算总和

```
def func(*seq):  
    x = 0  
    for n in seq:  
        x += n  
    return x  
print(func(1,2,3))
```

示例：传入多个键值，进行处理

```
def func(**computer):  
    for k,v in computer.items():  
        print("名称:%s\t价格:%s" % (k,v))  
  
func(主机=5000,显示器=1000,鼠标=60,键盘=150)
```

匿名函数

匿名函数：没有名字的函数，使用lambda关键字定义，一般仅用于单个表达式。

示例：求和函数

```
s = lambda a, b: a+b
```

```
print(s(1,2))
```

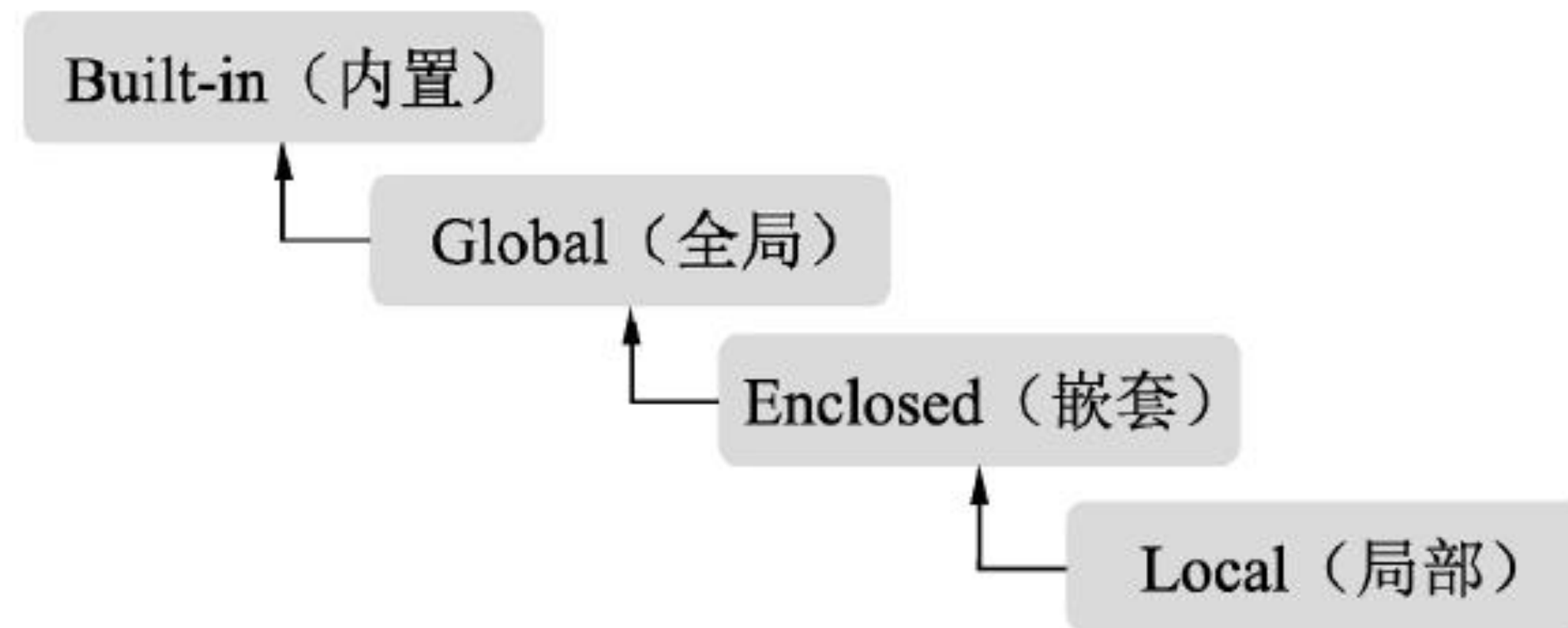
等价于

```
def func(a, b):
```

```
    return a+b
```

作用域

作用域： 限制一个变量或一段代码可用范围。好处是提高城乡逻辑的局部性，减少名字冲突。



- Local (局部作用域)：在函数中定义的变量。def关键字定义的语句块中，即函数中定义的变量。
- Enclosed (嵌套作用域)：一般出现在函数中嵌套一个函数时，在外部函数中的作用域称为嵌套作用域（闭包常见）。
- Global (全局作用域)：文件顶层定义的变量。
- Built-in (内置作用域)：系统内解释器定义的变量，例如关键字。

闭包

闭包：你可以简单粗暴地理解为**闭包就是一个定义在函数内部的函数**，闭包使得变量即使脱离了该函数的作用域范围也依然能被访问到。像上面所说的嵌套函数也是闭包的一种形式（将内部嵌套定义的函数作为返回值）。

示例：

```
def outer():  
    x = 1  
    def inner():  
        print(x) # 1  
    return inner # 不加括号，表示返回函数体  
sf = outer()  
sf() # 调用，等同于outer()
```

在一个外部函数内定义了一个函数，内部函数里引用外部函数的变量，并且外部函数的返回值是内函数的引用，这样就构成了一个闭包，并且满足了闭包的三个条件：

- 两层以上嵌套关系
- 内部函数调用外部函数定义的变量
- 外部函数返回内部函数体对象,而不是函数体结果（加括号）

函数装饰器

函数装饰器作用：装饰器本质上是一个函数，它可以让其他函数在不需要做任何代码变动的前提下增加额外功能，装饰器的返回值也是一个函数对象。

应用场景：记录日志、性能测试、权限效验等

函数装饰器：无参数

示例：装饰器使用

```
def hello():  
    print("我是原函数")  
  
def decorator(func):  
    def f():  
        print("原函数开始执行了")  
        func()  
        print("原函数执行结束了")  
    return f # 返回函数体  
  
dec = decorator(hello) # 装饰器传入函数  
dec() # 调用内部函数
```

函数装饰器：无参数

Python提供一个更简洁引用装饰器的方法：**语法糖 “@”**

示例：

```
@decorator
```

```
def hello():
```

```
    print("我是原函数")
```

```
hello()
```

函数装饰器：带参数

示例：

```
def decorator(func):  
    def f(msg):  
        print("原函数开始执行了")  
        func(msg)  
        print("原函数执行结束了")  
    return f
```

@decorator

```
def hello(msg):  
    print(msg)
```

```
hello("我是原函数")
```

Python 常用内建函数

- 高阶函数
- 排序函数
- 拼接字符串
- 最小值、最大值、求和函数
- 多个可迭代对象聚合
- 获取当前所有变量

内建函数

Python解释器内置了很多函数，你可以直接使用它们。
在前面学习过的内建函数有：print()、len()、open()、range()

内建函数	描述
map()	根据提供的函数处理序列中的元素，处理完后返回一个迭代器对象
filter()	用于过滤序列，过滤掉不符合条件的元素，处理完后返回一个迭代器对象
sorted	对所有可迭代对象进行排序操作
reversed()	返回一个反向的可迭代对象
join()	将序列中的元素以指定的字符连接，生成一个新的字符串
min()	返回可迭代对象中最小的元素
max()	返回可迭代对象中最大的元素
sum()	对可迭代对象求和
zip()	对多个可迭代对象创建一个聚合，返回一个元组的迭代器。
locals()	字典格式返回当前范围的局部变量
globals()	字典格式返回当前范围的全局变量

高阶函数

内建高阶函数：map()、filter()

高阶函数至少满足两个任意的一个条件：

- 能接收一个或多个函数作为输入
- 输出一个函数

高阶函数：map()

map() 函数：根据提供的函数处理序列中的元素，处理完后返回一个迭代器对象。

语法：map(function, iterable, ...)

示例：

```
num = range(1,11)
```

```
def handle(n):
```

```
    return n * 2
```

```
result = map(handle, num)
```

```
print(list(result))
```

或者使用匿名函数：

```
result = map(lambda n:n * 2, num)
```

```
print(list(result))
```


高阶函数：filter()

filter()函数：用于过滤序列，过滤掉不符合条件的元素，处理完后返回一个迭代器对象。

语法：filter(function, iterable)

示例：

```
num = range(1,11)
```

```
def handle(n):
```

```
    if n % 2 == 0:
```

```
        return n
```

```
result = filter(handle, num)
```

```
print(list(result))
```

或者使用匿名函数：

```
result = filter(lambda n:n % 2 == 0, num)
```

```
print(list(result))
```

排序函数

sorted()函数：对所有可迭代的对象进行排序操作。

语法：sorted(iterable, *, key=None, reverse=False)

- key：指定带有单个参数的函数，用于从iterable的每个元素取出比较的键，默认为None（直接比较元素）
- reverse 排序规则，True降序， False升序（默认）

示例1：对列表排序

```
n = [2, 3, 4, 1, 5]
```

```
s = ["b","c","a"]
```

```
print(sorted(n))
```

```
print(sorted(s))
```

示例2：对字典中的值排序

```
dict = {'a':86, 'b':23, 'c':45}
```

```
result = sorted(dict.items(), key=lambda x:x[1])
```

```
print(result)
```

反转函数

reversed()函数：返回一个反转的迭代器。

语法：reversed(seq)

示例：列表反向

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

```
print(list(reversed(n)))
```

拼接函数

join()函数：将序列中的元素以指定的字符连接，生成一个新的字符串。

示例1：拼接字符串

```
s = "lizhenliang"
```

```
r = '.'.join(s)
```

```
print(r)
```

示例2：拼接序列

```
computer = ["主机","显示器","鼠标","键盘"]
```

```
r = ','.join(computer)
```

```
print(r)
```

最小值、最大值、求和函数

- `min()` 返回可迭代对象中最小的元素
- `max()` 返回可迭代对象中最大的元素
- `sum()` 对可迭代对象求和

多迭代对象聚合

zip()函数：对多个可迭代对象创建一个聚合，返回一个元组的迭代器。

示例：

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

```
y = [4, 5, 6]
```

```
zipped = zip(x, y)
```

```
print(list(zipped))
```

获取当前所有变量

- `globals()`: 字典格式返回当前范围的全局变量
- `locals()`: 字典格式返回当前范围的局部变量

示例:

```
a = 1
```

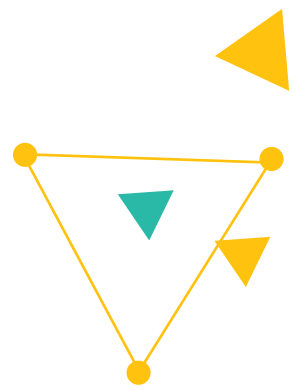
```
def f():
```

```
    b = 2
```

```
    print("局部变量: %s" % locals())
```

```
print("全局变量: %s" % globals())
```

```
f()
```



谢谢

