廣東工業大學
Guangdong University of Technology

Python基础

龚怡

2167570874@qq.com

目录

Python简介

- Python是一种解释型、面向对象、动态数据类型的高级程序设计语言
- 解释型程序语言，相对于编译型，是一边解析一边编译一边执行程序语言
- Python常被昵称为胶水语言，能够把用其他语言制作的各种模块（尤其是C/C++）很轻松地联结在一起
- Python强制用空白符（white space）作为语句缩进
- 庞大的Python库和第三方库

Python发展历史

- Python 是由 Guido van Rossum 在八十年代末和九十年代初，在荷兰国家数学和计算机科学研究所设计出来的;
- Python 本身也是由诸多其他语言发展而来的,这包括 ABC、Modula-3、C、C++、Algol-68、SmallTalk、Unix shell 和其他的脚本语言等等;
- 像 Perl 语言一样，Python 源代码同样遵循 GPL(GNU General Public License)协议;
- 现在 Python 是由一个核心开发团队在维护，Guido van Rossum 仍然占据着至关重要的作用，指导其进展;

Python特点(1)

- 易于学习：Python有相对较少的关键字，结构简单，和一个明确定义的语法，学习起来更加简单;
- 易于阅读：Python代码定义的更清晰;
- 易于维护：Python的成功在于它的源代码是易维护的;
- 一个广泛的标准库：Python的最大的优势之一是丰富的库，跨平台的，在UNIX，Windows和Macintosh兼容很好;
- 互动模式：互动模式的支持，您可以从终端输入执行代码并获得结果的语言，互动的测试和调试代码片断;
- 可移植：基于其开放源代码的特性，Python已经被移植(也就是使其工作)到许多平台;

Python特点(2)

- 可扩展：如果你需要一段运行很快的关键代码，或者是想要编写一些不愿开放的算法，你可以使用C或C++完成那部分程序，然后从你的Python程序中调用;
- 数据库：Python提供所有主要的商业数据库的接口;
- GUI编程：Python支持GUI可以创建和移植到许多系统调用;
- 可嵌入：你可以将Python嵌入到C/C++程序，让你的程序的用户获得"脚本化"的能力;

Python应用领域(1)

- 科学计算

随着NumPy、SciPy、Matplotlib、Enthought librarys等众多程序库的开发，使得Python越来越适合做科学计算、绘制高质量的2D和3D图像。

- 人工智能

MASA和Google早期大量使用Python，为Python积累了丰富的科学运算库，当AI时代来临后，目前市面上大部分的人工智能的代码都是使用Python来编写，尤其PyTorch之后，Python作为AI时代首先语言的位置基本确定。

Python应用领域(2)

- 数据分析

Python已成为数据分析和数据科学事实上的标准语言和标准平台之一，Numpy、Pandas、Scipy和Matplotlib程序库共同构成了Python数据分析的基础。

- Web开发

Python拥有很多免费数据函数库、免费Web网页模板系统、以及与Web服务器进行交互的库，可以实现web开发，搭建Web框架，目前比较有名的Python Web框架为Django

- 爬虫开发

在爬虫领域，Python几乎是霸主地位，将网络一切数据作为资源，通过自动化程序进行有针对性的数据采集以及处理

Python应用领域(3)

- 云计算开发

Python是从事云计算工作需要掌握的一门编程语言，目前很火的云计算框架OpenStack就是由Python开发的。

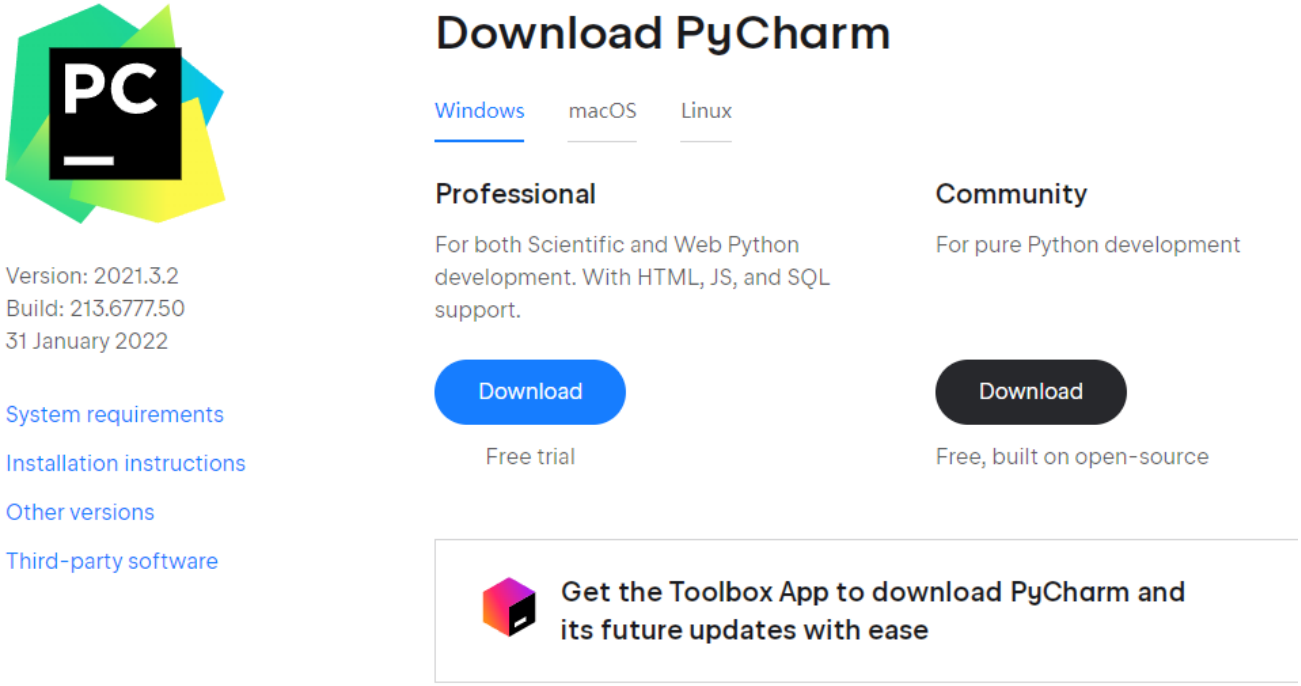
- 自动化运维

Python是一门综合性的语言，能满足绝大部分自动化运维需求，前端和后端都可以做。

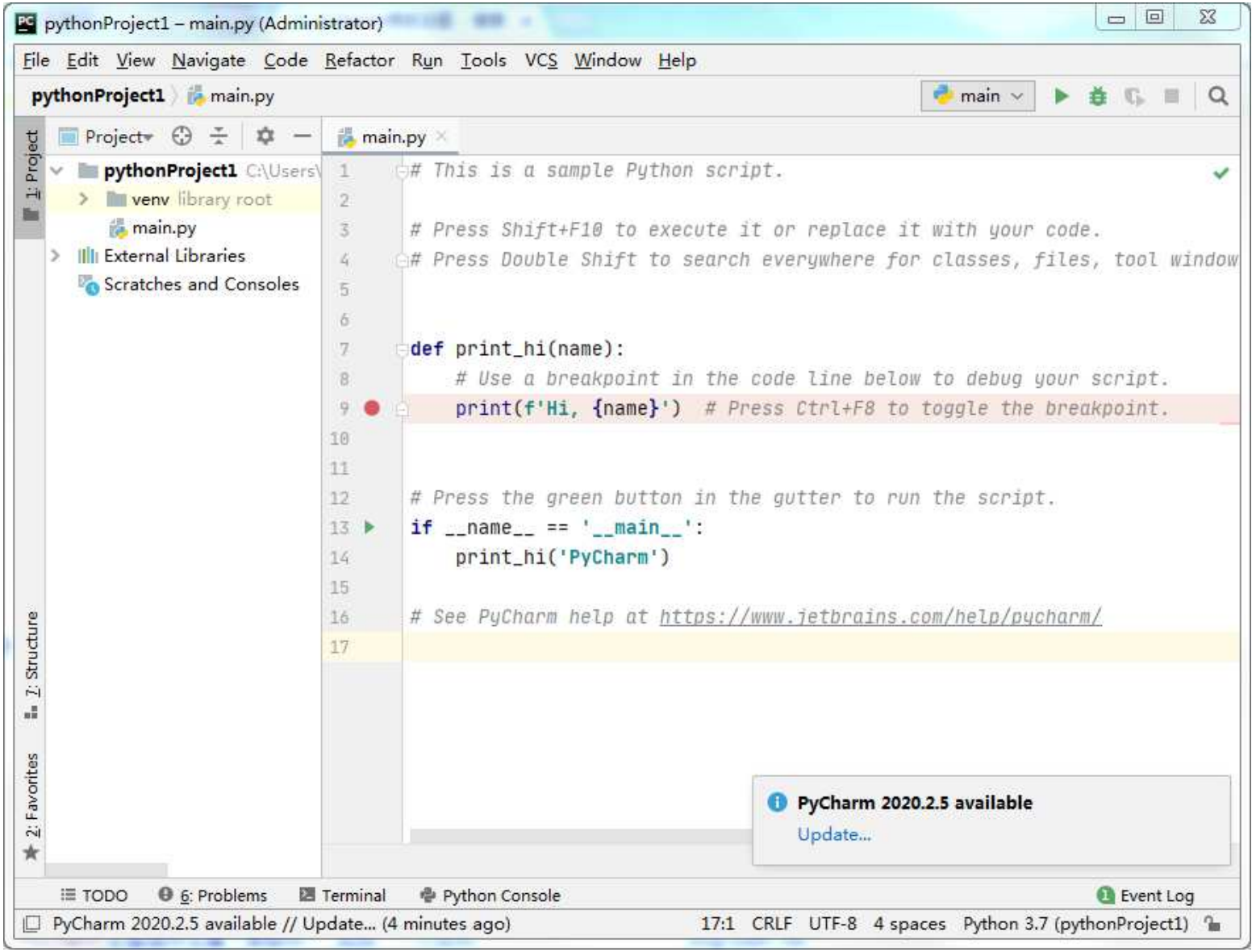
Python安装

本课程使用IDE开发环境为PyCharm，选择社区版下载和安装

https://www.jetbrains.com/pycharm/download/#section=windows



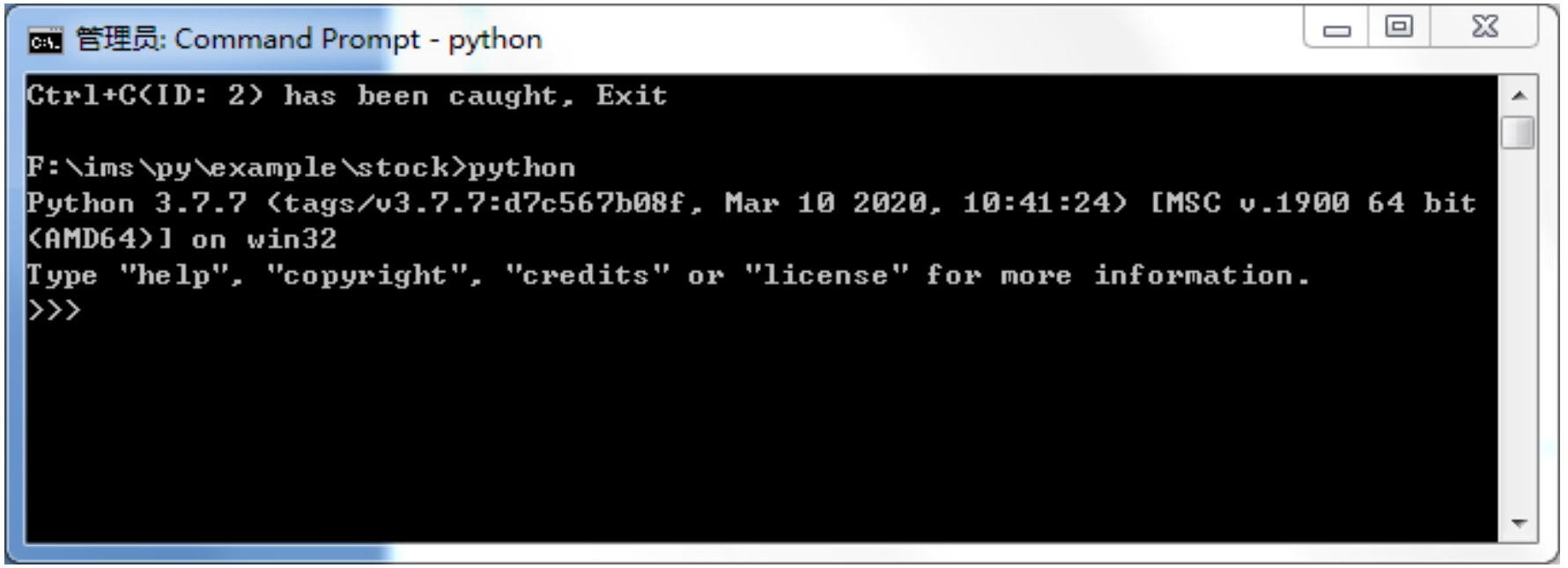
PyCharm主界面


pythonProject1 – main.py (Administrator)
File Edit View Navigate Code Refactor Run Tools VCS Window Help
pythonProject1 main.py
main
Project
main.py
pythonProject1 C:\Users\
venv library root
main.py
External Libraries
Scratches and Consoles
1: Project
7: Structure
2: Favorites
# This is a sample Python script.
# Press Shift+F10 to execute it or replace it with your code.
# Press Double Shift to search everywhere for classes, files, tool window
def print_hi(name):
# Use a breakpoint in the code line below to debug your script.
print(f'Hi, {name}') # Press Ctrl+F8 to toggle the breakpoint.
# Press the green button in the gutter to run the script.
if __name__ == '__main__':
print_hi('PyCharm')
# See PyCharm help at https://www.jetbrains.com/help/pycharm/
PyCharm 2020.2.5 available
Update...
TODO 6: Problems Terminal Python Console Event Log
PyCharm 2020.2.5 available // Update... (4 minutes ago)
17:1 CRLF UTF-8 4 spaces Python 3.7 (pythonProject1)

Python基础语法-交互式编程

交互式编程不需要创建脚本文件，是通过 Python 解释器的交互模式进来编写代码。

只需要在命令行中输入 Python 命令即可启动交互式编程，提示窗口如下：



Python基础语法-脚本式编程

- 通过脚本参数调用解释器开始执行脚本，直到脚本执行完毕。当脚本执行完成后，解释器不再有效。
- 让我们写一个简单的 Python 脚本程序。所有 Python 文件将以 .py 为扩展名。将以下的源代码拷贝至 test.py 文件中。
- 运行脚本 c:\>python test.py
- 运行语句 c:\>python -c "a = 10;b=20;print(a+b)"

Python的print输出

- Python2.x使用print “abc”, python 3.x 使用print()函数
- Python2.x 中使用 Python3.x 的 print 函数
- 如果 Python2.x 版本想使用使用 Python3.x 的 print 函数，可以导入 __future__ 包，该包禁用 Python2.x 的 print 语句，采用 Python3.x 的 print 函数

Python输出中文乱码

Python 文件中如果未指定编码，在执行过程会出现报错：

```
#!/usr/bin/python
print ("你好，世界")
```

以上程序执行输出结果为：

```
  File "test.py", line 2
SyntaxError: Non-ASCII character '\xe4' in file
   test.py on line 2, but no encoding declared; see
   http://www.python.org/peps/pep-0263.html for
   details
```

Python输出中文乱码

Python中默认的编码格式是 ASCII 格式，在没修改编码格式时无法正确打印汉字，所以在读取中文时会报错。

解决方法为只要在文件开头加入

```
# -*- coding: UTF-8 -*-
```

或者

```
# coding=utf-8
```

指定编码格式

注意：`# coding=utf-8` 的 = 号两边不要空格。

Python代码注释

- Python单行注释

井号（#）常被用作单行注释符号，在代码中使用#时，它右边的任何数据在程序执行时都会被忽略，被当做是注释。

```
>>> print('Hello world.')  #输出Hello world.
Hello world.
```

Python代码注释

- Python多行注释

在Python中，当注释有多行时，需用多行注释符来对多行进行注释。多行注释用三个单引号'''或者三个双引号"""将注释括起来，例如：

```
'''
这是多行注释，用三个单引号
这是多行注释，用三个单引号
这是多行注释，用三个单引号
'''
print("Hello, World!")
```

Python标识符

- 标识符由字母、数字、下划线组成，但不能以数字开头；
- Python 中的标识符是区分大小写的；
- 以下划线开头的标识符是有特殊意义的。以单下划线开头（_foo）的代表不能直接访问的类属性，需通过类提供的接口进行访问，不能用"from xxx import *"而导入；
- 以双下划线开头的（__foo）代表类的私有成员；
- 以双下划线开头和结尾的（__foo__）代表python 里特殊方法专用的标识，如__init__（）代表类的构造函数。
- Python 可以同一行显示多条语句，方法是用分号 ； 分开

.

Python 保留字符

assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Python标准数据类型

- Numbers（数字）
- String（字符串）
- List（列表）
- Tuple（元组）
- Dictionary（字典）
- Set（集合）

Python的4种数字类型

- int（有符号整型）
- long（长整型，也可以代表八进制和十六进制）
- float（浮点型）
- complex（复数）

Python的布而类型

- 布尔型：True、False

Python的空值

- Python语言中有一个特殊的值None，它表示空值，它不同于逻辑值False、数值0、空字符串''，它表示的含义就是没有任何值，它与其它任何值的比较结果都是False

Python的字符串类型

Python中的字符串属于不可变序列，是用单引号(')、双引号(")、三单引号(''')或三双引号(""")等界定符括起来的字符序列。

创建字符串

只要为变量分配一个用字符串界定符括起来的字符序列即可创建一个字符串。例如：

```
var1 = 'Hello World!'
var2 = "Hello Word!"
var3 = '''Hello
Word!'''
```

Python的字符串类型

操作符	描述
+	字符串连接
*	重复输出字符串
[]	通过索引获取字符串中字符
[:]	截取字符串中的一部分
in	成员运算符--如果字符串中包含给定的字符串返回True
not in	成员运算符--如果字符串中不包含给定的字符串返回True
r/R	原始字符串--在字符串的第一个引号前加上字母r或R，字符串中的所有的字符直接按照字面的意思来使用，不再转义特殊或不能打印的字符。
%	格式化字符串

Python的字符串类型

```
>>> str1= 'Python'
>>> str2= ' good'
>>> str3=str1+str2          #字符串连接
>>> print(str3)
Python good
>>> print (str1 * 2)        # 输出字符串两次
PythonPython
>>> print(2* str1)
PythonPython
```

列表数据类型

列表是写在方括号[ ]之间、用逗号分隔开的元素列表。列表的大小是可变的，它可以根据需求增加或减少。列表中元素的类型可以不相同，列表中可以同时存在数字、字符串、元组、字典、集合等数据类型的对象，甚至可以包含列表（即嵌套）。

下面几个都是合法的列表对象：

- ['Google', 'Baidu', 1997, 2008]
- [1, 2, 3, 4, 5]
- ["a", "b", "c", "d"]
- [123, ["das", "aaa"], 234]

列表数据类型

（1）列表创建、删除

```
>>> list1 = list()    #创建空列表
>>> list2 = list ('chemistry')
>>> list2
['c', 'h', 'e', 'm', 'i', 's', 't', 'r', 'y']
```

也可以使用“=”直接将一个列表赋值给变量来创建一个列表对象：

```
>>> lista= []
>>> listb = [ 'good', 123 , 2.2, 'best', 70.2 ]
```

列表数据类型

（2）列表截取（也称分片、切片）

- 列表中的元素可以使用下标操作符list[index]访问列表中下标为index的元素。
- 列表下表是从0开始的，下标的范围从0到len(list)-1，len(list)获取列表list的长度。
- list[index]可以像变量一样使用，被称为下标变量。
- Python允许使用负数作为下标来引用相对于列表末端的位置，将列表长度和负数下标相加就可以得到实际的位置。例如：

```
>>> list1=[1,2,3,4,5]
>>> list1[-1]
```

列表数据类型

（2）列表截取（也称分片、切片）

- 列表截取（也称分片、切片）操作使用语法list[start:end]返回列表的一个片段。这个片段是下标从start到end-1的元素构成的一个子列表。列表被截取后返回一个包含指定元素的新列表。

```
>>> list1 = [ 'good', 123 , 2.2, 'best', 70.2 ]
>>> print (list1[1:3])      # 输出第二个至第三个元素
[123, 2.2]
```

列表数据类型

（3）改变列表

- 列表元素改变：

```
>>> x = [1,1,3,4]
>>> x[1] = 2          #将列表中第二个1改为2
```

- 列表元素分段改变：

```
>>>name = list('Perl')
>>>name[1:] = list('ython')
>>>name
['P', 'y', 't', 'h', 'o', 'n']
```

列表数据类型

（3）改变列表

- 在列表中插入序列：

```
>>> number=[1,6]
>>> number[1:1]=[2,3,4,5]
>>> number
[1, 2, 3, 4, 5, 6]
```

- 在列表中删除一段元素：

```
>>> names = ['one', 'two', 'three', 'four', 'five', 'six']
>>> del names[1]    #删除names的第二个元素
>>> names[1:4]=[]   #删除names的第二至第四个元素
>>> names
['one', 'six']
```

列表数据类型

（4）列表是一种序列类型

在Python中字符串、列表和以及后面要讲的元组都是序列类型。

- 所谓序列，即成员有序排列，并且可以通过偏移量访问到它的一个或者几个成员。
- 序列中的每个元素都被分配一个数字——它的位置，也称为索引，第一个索引是0，第二个索引是1，依此类推。
- 序列都可以进行的操作包括索引、切片、加、乘、检查成员。
- Python已经内置确定序列的长度以及确定最大和最小的元素的方法。

列表数据类型

(4) 列表是一种序列类型，序列的常用操作：

操作	描述
x in s	如果元素x在序列s中则返回True
x not in s	如果元素x不在序列s中则返回True
s1+s2	连接两个序列s1和s2，得到一个新序列
s*n, n*s	序列s复制n次得到一个新序列
s[i]	得到序列s的第i个元素
s[i:j]	得到序列s从下标i到j-1的片段
len(s)	返回序列s包含的元素个数
max(s)	返回序列s的最大元素
min(s)	返回序列s的最小元素
sum(x)	返回序列s中所有元素之和
<、<=、>、>=、==、!=	比较两个序列

列表对象常函数

方 法	描述
list.append(x)	在列表list末尾添加新的对象x
list.count(x)	返回x在列表list中出现的次数
list.extend(seq)	在列表list末尾一次性追加seq序列中的所有元素
list.index(x)	返回列表list中第一个值为x的元素的下标，若不存在抛出异常
list.insert(index, x)	在列表list中index位置处添加元素x
list.pop([index])	删除并返回列表指定位置的元素，默认为最后一个元素
list.remove(x)	移除列表list中x的第一个匹配项
list.reverse()	反向列表list中的元素
list. sort(key=None, reverse=None)	对列表list进行排序，key参数的值为一个函数，此函数只有一个参数且返回一个值，此函数将在每个元素比较前被调用， reserve表示是否逆序
list.clear()	删除列表list中的所有元素，但保留列表对象
list.copy()	用于复制列表，返回复制后的新列表

元组数据类型

元组（tuple）与列表类似，但元组的元素不能修改。元组元素写在小括号( )里，元素之间用逗号隔开，元组中的元素类型可以不相同。元组可以被索引且下标索引从0开始，-1为从末尾开始，也可以进行截取。元组对象举例：(1, 2, 3, 4, 5)、("a", "b", "c")

（1）访问元组

使用下标索引来访问元组中的值，如下实例：

```
>>> tuple1 = ( 'hello', 18 , 2.23, 'world', 2+4j)
>>> print(tuple1[0])                    # 输出元组的第一个元素
```

元组数据类型

（2）修改元组

元组属于不可变序列，一旦创建，元组中的元素是不允许修改的，也无法增加或删除元素。因此，元组没有提供append()、extend()、insert()、remove()、pop()方法，也不支持对元组元素进行del操作，但能用del命令删除整个元组。

- 元组中的元素值是不允许修改的，但我们可以对元组进行连接组合，得到一个新元组：

```
>>> tuple1 = ( 'hello', 18 , 2.23, 'world', 2+4j)
>>> tuple2 = ( 'best', 16)
>>> tuple3 = tuple1 + tuple2                    # 连接元组
>>> print(tuple3)
('hello', 18, 2.23, 'world', (2+4j), 'best', 16)
```

元组数据类型

（2）修改元组

虽然tuple的元素不可改变，但它可以包含可变的对象，比如list列表，可改变元组中可变对象的值。

```
>>> tuple4 = ('a', 'b', ['A', 'B'])
>>> tuple4[2][0] = 'X'
>>> tuple4[2][1] = 'Y'
>>> tuple4[2][2:]= 'Z'
>>> tuple4
('a', 'b', ['X', 'Y', 'Z'])
```

字典数据类型

- 字典是一种映射类型，字典用"{ }"标识，它是一个无序的“键(key) : 值(value)”对集合。“键(key)”必须使用不可变类型，如整型、浮点型、复数型、布尔型、字符串、元组等，但不能使用诸如列表、字典、集合或其它可变类型作为字典的键。在同一个字典中，“键(key)”必须是唯一的，但“值(value)”是可以重复的。
- 列表是有序的对象集合，字典是无序的对象集合。两者之间的区别在于：字典当中的元素是通过键来存取的，而不是通过偏移存取。

字典数据类型

（1）创建字典

使用赋值运算符将使用{ }括起来的“键:值”对赋值给一个变量即可创建一个字典变量。

```
>>> dict1 = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
>>> dict1['Jack'] ='1234'                    #为字典添加元素
>>> print(dict1)                             # 输出完整的字典
{'Alice': '2341', 'Beth': '9102', 'Cecil': '3258', 'Jack': '1234'}
>>> type(dict1)
<class 'dict'>      #显示dict1的类型为dict
```

字典数据类型

访问字典里的值

通过“字典变量[key]”的方法返回键key对应的值value：

```
>>> dict1={'Alice': '2341', 'Beth': '9102', 'Cecil': '3258', 'Jack': '1234'}
>>> print (dict1['Beth'])                    # 输出键为'Beth'的值
9102
>>> print (dict1.values())                   # 输出所有值
dict_values(['2341', '9102', '3258', '1234'])
```

字典数据类型

（3）字典元素添加、修改与删除

向字典添加新元素的方法是增加新的键/值对：

```
>>> school={'class1': 60, 'class2': 56, 'class3': 68, 'class4': 48}
>>> school['class5']=70   # 添加新的元素
>>> school['class1']=62       # 更新 class1的值
>>> school
{'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 70}
```

由上可知，当以指定“键”为下标为字典元素赋值时，有两种含义：①若该“键”不存在，则表示为字典添加一个新元素，即一个“键:值”对；②若该“键”存在，则表示修改该“键”所对应的“值”。

字典数据类型

（3）字典元素添加、修改与删除

使用字典对象的update()方法可以将另一个字典的元素一次性全部添加到当前字典对象中，如果两个字典中存在相同的“键”，则只保留另一个字典中的键值对，如下所示：

```
>>> school1={'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 70}
>>> school2={ 'class5': 78,'class6': 38}
>>> school1.update(school2)
>>> school1     #'class5'所对应的值取school2中'class5'所对应的值
78
{'class1': 62, 'class2': 56, 'class3': 68, 'class4': 48, 'class5': 78, 'class6': 38}
```

字典对常函数

方法	描述
dict1.copy()	返回一个字典的浅复制，即复制时只会复制父对象，而不会复制对象的内部的子对象，复制后对原dict的内部的子对象进行操作时，浅复制dict会受操作影响而变化。
dict1.fromkeys(seq[, value]))	创建一个新字典，以序列seq中元素做字典的键，value为字典所有键对应的初始值
dict1.get(key, default=None)	返回指定键key的值，如果值不在字典中返回默认值
dict1.items()	以列表形式返回可遍历的(键, 值) 元组数组
dict1.keys()	以列表返回一个字典所有的键
dict1.update(dict2)	把字典dict2的键/值对更新到dict1里
dict1.values()	以列表返回字典中的所有值

集合数据类型

集合是无序可变序列，使用一对大括号作为界定符，元素之间使用逗号分隔，集合中的元素互不相同。集合的基本功能是进行成员关系测试和删除重复元素。集合不能有可变元素(如列表、集合或字典)

集合数据类型

（1）创建集合

使用赋值操作“=”直接将一个集合赋值给变量来创建一个集合对象：

```
>>> student = {'Tom', 'Jim', 'Mary', 'Tom', 'Jack', 'Rose'}
```

也可以使用set()函数将列表、元组等其它可迭代对象转换为集合，如果原来的数据中存在重复元素，则在转换为集合的时候只保留一个。

```
>>> set1 = set('cheeseshop')
>>> set1
{'s', 'o', 'p', 'c', 'e', 'h'}
```

集合数据类型

（2）增加集合元素

可以添加或删除集合中的元素。可以使用集合对象的add()方法添加单个元素，使用update()方法添加多个元素，update() 可以使用元组、列表、字符串或其他集合作为参数。

```
>>> set3 = {'a', 'b'}
>>> set3.add('c')                    # 添加一个元素
>>> set3
{'b', 'a', 'c'}
>>> set3.update(['d', 'e', 'f'])          # 添加多个元素
>>> set3
{'a', 'f', 'b', 'd', 'c', 'e'}
```

集合数据类型

（3）删除集合中的元素

- 可以使用集合对象的discard()和remove()方法删除集合中特定的元素。区别：如果集合中不存在指定的元素，使用discard()，集合保持不变；但在这种情况下，使用remove()会引发KeyError。
- 集合对象的pop()方法用于随机删除并返回集合中的一个元素，如果集合为空则抛出异常。
- 集合对象的clear()方法用于删除集合的所有元素。

```
>>> set4 = {1, 2, 3, 4}
>>> set4.discard(4)
>>> set4
{1, 2, 3}
```

集合数据类型

（4）集合运算

Python集合支持交集、并集、差集、对称差集等运算，如下所示：

```
>>> A={1,2,3,4,6,7,8}
>>> B={0,3,4,5}
```

使用“&”操作符执行交集操作，也可使用集合对象的方法intersection()完成，如下所示：

```
>>> A&B          #求集合A和B的交集
{3, 4}
>>> A.intersection(B)
{3, 4}
```

集合数据类型

集合运算

使用操作符“|”执行并集操作，也可使用集合对象的方法union()完成，如下所示：

```
>>> A | B
{0, 1, 2, 3, 4, 5, 6, 7, 8}
>>> A.union(B)
{0, 1, 2, 3, 4, 5, 6, 7, 8}
```

使用操作符“-”执行差集操作，也可使用集合对象的方法difference()完成，如下所示：

```
>>> A - B
{1, 2, 6, 7, 8}
>>> A.difference(B)
```

集合数据类型

集合运算

对称差：集合A与集合B的对称差集是由只属于其中一个集合，而不属于另一个集合的元素组成的集合，使用“^”操作符执行对称差集操作，也可使用集合对象的方法symmetric_difference()完成，如下所示：

```
>>> A ^ B
{0, 1, 2, 5, 6, 7, 8}
>>> A.symmetric_difference(B)
{0, 1, 2, 5, 6, 7, 8}
```

集合数据类型

集合运算

子集：由某个集合中一部分元素所组成的集合，使用操作符“<”判断“<”左边的集合是否是“<”右边的集合的子集，也可使用集合对象的方法issubset()完成，如下所示：

```
>>> A={1,2,3,4,6,7,8}
>>> C={1,3,4}
>>> C < A     #C集合是A集合的子集，返回True
True
>>> C.issubset(A)
True
```

Python 运算符

- 算术运算符
- 比较运算符
- 赋值运算符
- 逻辑运算符
- 位运算符
- 成员运算符
- 身份运算符
- 运算符优先级

算术运算符和算术表达式

1. 算术运算符

运算符	含义	优先级	结合性
+	加法	这些运算符的优先级相同，但比下面的运算符优先级低	左结合
-	减法		
*	乘法		
/	除法	这些运算符的优先级相同，但比上面的运算符优先级高	
//	取整除		
**	幂运算		
%	取模		

算术运算符和算术表达式

Python中除法有两种：/和//，在Python3.x分别表示除法和整除运算。

```
>>> 3/5
0.6
```

```
>>> 3//5
0
```

```
>>> -3.0//5
-1.0
```

```
>>> 3.0//-5
-1.0
```

算术运算符和算术表达式

**运算符实现乘方运算，其优先级高于*和/。

例如：

```
>>> 2**3
8
-------------------------------------------------------------

>>> 2**3.5
11.313708498984761
-------------------------------------------------------------

>>> 4*3**2
36
```

算术运算符和算术表达式

运算符的多重含义：

```
>>> 3*5          #整数相乘运算
15
-----------------------------------------------------------
>>> 'a'*10       #字符串重复运算
'aaaaaaaaaa'
-----------------------------------------------------------
```

算术运算符和算术表达式

%运算符表示对整数和浮点数的取模运算。

由于受浮点数精确度的影响，计算结果会有误差。

```
>>> 5%-3
-1
-----------------------------------------------------------
>>> 10.5%2.1                    #浮点数取模运算
2.0999999999999996        #结果有误差
```

算术运算符和算术表达式

2. 算术表达式

例如：

3+a*b/5-2.3+' b'

3. 数据转换

在Python中，同一个表达式允许不同类型的数据参加运算，这就要求在运算之前，先将这些不同类型的数据转换成同一类型，然后再进行运算。

【例2.4】自动类型转换。

```
>>> 10/4*4
10.0
>>> type(10/4*4)
<class 'float'>
```

算术运算符和算术表达式

当自动类型转换达不到转换需求时，可以使用类型转换函数，将数据从一种类型强制（或称为显式）转换成另一种类型。

函数	功能描述
int(x)	将x转换为整数
float(x)	将x转换为浮点数
complex(x)	将x转换为复数，其中实部为x，虚部为0
complex(x,y)	将x、y转换为复数，其中实部为x，虚部为y
str(x)	将x转换为字符串
chr(x)	将一个整数转换为一个字符，整数为字符的ASCII编码
ord(x)	将一个字符转换为它的ASCII编码的整数值
hex(x)	将一个整数转换为一个十六进制字符串
oct(x)	将一个整数转换为一个八进制字符串
eval(x)	将字符串str当做有效表达式求值，并返回计算结果

算术运算符和算术表达式

1. 赋值运算符

赋值运算符用“=”表示。

2. 赋值表达式

变量 = 表达式

等号的左边必须是变量，右边是表达式。

```
>>>y=2
>>>x=(y+2)/3
>>>x
1.3333333333333333
```

算术运算符和算术表达式

注意： Python的赋值和一般的高级语言的赋值有很大的不同，它是引用赋值。看下面的代码1：例如：

```
>>> a = 5
>>> b = 8
>>> a = b
```

执行a=5和b=8之后a指向的是5，b指向的是8，当执行a = b的时候，b把自己指向的地址（也就是8的内存地址）赋给了a，那么最后的结果就是a和b同时指向了8。

3. 多变量赋值

（1）链式赋值

在Python中，可通过链式赋值将同一个值赋给多个变量的，一般形式为：

```
>>>x=y=5
>>>x
5
```

算术运算符和算术表达式

（2）多变量并行赋值

变量1, 变量2, …, 变量n=表达式1, 表达式2, …, 表达式n

变量个数要与表达式的个数一致，其过程为：首先计算表达式右边n个表达式的值，然后同时将表达式的值赋给左边的n个变量。例如：

```
>>>x, y, z=2, 5, 8
>>>x
2
------------------------------------------------------------

>>>y
5
------------------------------------------------------------

>>>z
8
```

算术运算符和算术表达式

例如：

```
>>>x=20
>>>x, x=3, x*3
>>>x
60
```

采取并行赋值，可以使用一条语句就可以交换两个变量的值：x, y=y, x。

算术运算符和算术表达式

4. 复合的赋值运算符

赋值运算符“＝”与7种算术运算符（+、-、*、/、//、**、%）和5种位运算符（>>、<<、&、^、|）结合构成12种复合的赋值运算符。结合方向为自右至左。

例如：

a+=3	等价于	a=a+3
a*=a+3	等价于	a=a*(a+3)
a%=3	等价于	a=a%3

注意：“a*=a+3”与“a=a*a+3”是不等价的，“a*=a+3”等价于“a=a*(a+3）”，这里括号是必需的。

关系运算符和关系表达式

1. 关系运算符

运算符	含义	优先级	结合性
>	大于	这些运算符的优先级相同，但比下面的运算符优先级低	左结合
>=	大于等于		
<	小于		
<=	小于等于	这些运算符的优先级相同，但比上面的运算符优先级高	
==	等于		
!=	不等于		
<>	不等于		

关系运算符和关系表达式

关系运算符的优先级：{ >、>=、<、<=}→{==、!= 、<> }

在Python中，真用“True”表示，假用“False”表示。

```
>>> x, y, z=2, 3, 5
>>> x>y
False
>>> x<y<z
True
```

关系运算符和关系表达式

注意： 浮点数的相等。在计算机中，浮点数是实数的近似值。执行一系列浮点数的运算后，可能会发生四舍五入的情况。

例如：

```
>>> x=123456
>>> y=-111111
>>> z=1.2345678
>>> a=(x+y)+z
>>> b=x+(y+z)
>>> a
12346.2345678
>>> b
12346.234567799998
```

逻辑运算符和逻辑表达式

1. 逻辑运算符

运算符	含义	优先级	结合性
not	逻辑非	高 ↑ ↓ 低	右结合
and	逻辑与		左结合
Or	逻辑或		

逻辑运算符和逻辑表达式

2. 逻辑表达式及短路运算

（1）对于与运算a and b

如果a为真，继续计算b，b将决定最终整个表达式的真值，所以，结果为b的值。

如果a为假，无需计算b，就可以得知整个表达式的真值为假，所以，结果为a的值。**例如：**

```
>>>True and 0
0
```

```
>>> False and 12
False
```

逻辑运算符和逻辑表达式

（2）对于或运算a or b

如果a为真，无需计算b，就可得知整个表达式的真值为真，所以结果为a的值。

如果a为假，继续计算b，b将决定整个表达式最终的值，所以结果为b的值。

例如：

```
>>> True or 0
True
>>> False or 12
12
>>> False or 12 or 0
12
```

成员运算符和成员表达式

成员运算符用于判断一个元素是否在某一个序列中，或者判断一个字符是否属于这个字符串等，运算结果仍是逻辑值。

运算符	含义	优先级	结合性
in	存在	相同	左结合
not in	不存在		

```
>>> 'a' in 'abcd'
True
>>> 'ac' in 'abcd'
False
```

成员运算符和成员表达式

not in 运算符用于在指定的序列中查找某个值是否不存在，不存在返回True，存在返回False。

```
>>> 'a' not in 'bcd'
True
>>> 3 not in [1,2,3,4]
False
```

同一性运算符和同一性表达式

同一性运算符用于测试两个变量是否指向同一个对象，其运算结果是逻辑值。

运算符	含义	优先级	结合性
is	相同	相同	左结合
is not	不相同		

is检查用来运算的两个变量是否引用同一对象，如果相同返回True，不相同则返回False。

is not检查用来运算的两个变量是否不是引用同一对象，如果不是同一个对象返回True，否则返回False。

同一性运算符和同一性表达式

```
>>> x=y=2.5
>>> z=2.5
>>> x is y
True
>>> x is z
False
>>> x is not z
True
```

运算符的优先级和结合性

优先级	运算符	结合性
高 ↑	()	从左至右
	**	
	*、/、%、//	
	+、-	
	<、<=、>、>=	
	==、!=、<>	
	is、not is	
	in、not in	
低	not	从右至左
	and	从左至右
	or	

变量

变量的值在程序运行过程中发生变化。

变量的命名规则：

变量名由字母、数字、特殊符号组成。

变量名的第1个字母必须是字母或下划线“_”。

变量名是区分大小写的。

使用id(变量名)求变量的内存地址，如：

```
>>> id(a)
1500529280
>>> f = a
>>> id(f)
1500529280
```

变量的作用域-局部变量

```
def func():
    x = 1                    #局部变量
    x = x + 5
    print("函数内x= %d"%x)
x = 10                              #全局变量
print("调用函数前 x=%d"%x)
func()
print("调用函数后 x=%d"%x)
```

变量的作用域-全局变量

在变量名前加global修饰该变量，则表示引用的是全局变量。

```
def func():
    global x
    x = x + 5
    print("函数内x= %d"%x)
x = 10
print("调用函数前 x=%d"%x)
func()
print("调用函数后 x=%d"%x)
```

变量的作用域

当全局变量在主程序中没有定义，在函数内我们可以用“global 变量”直接定义一个全局变量

```
def func():
    global x
    x = 10              #引前需要对x先赋值
    x = x + 5
    print("函数内x= %d"%x)

#print("调用函数前 x=%d"%x)      #未定义变量x，直接引用，会出错的
func()
print("调用函数后 x=%d"%x)
```

分支结构

单分支结构

```
ss = input("请输入你的成绩(0-100):")
si= int(ss)
if si < 60:
    print("不及格")
if si >= 90:
    print("优")
```

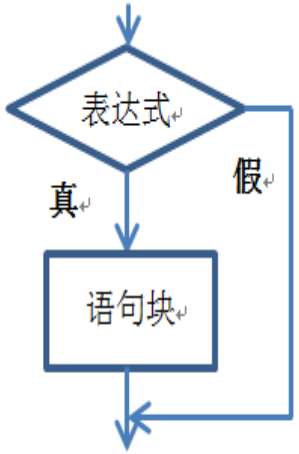
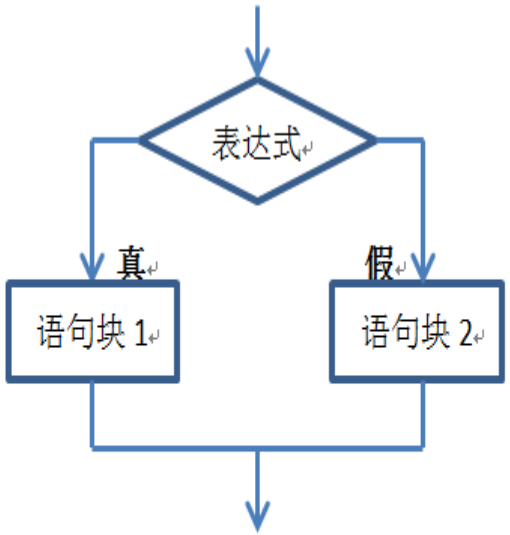


图 3.1 单分支结构

双分支结构

```
ss=input("请输入分数(0-100):")
si=int(ss) if si>=60:
    print("及格")
else:
    print("不及格")
```



3.2 双分支结构

多分支结构

用if…elif…语句实现多分支结构

```
ss = input("请输入你的成绩 (0-100) : ")
si=int(ss)
if si<60:
    print("不及格")
elif si <70:
    print("良")
elif si<=100:
    print("优")
```

循环结构

while循环

while的语法结构为：

【格式一】

```
while 条件表达式:
        循环体
```

【格式二】

```
while 条件表达式:
        循环体
else:
        语句体
```

次数不确定的while循环

```
sum=0
numstr = input("请输入一个整数(‘#’结束):")
while numstr != "#":
    numint = int(numstr)
    sum = sum + numint
    numstr = input("请输入整数")
print("sum = ", sum)
```

次数确定的While循环

```
sum = 0
i=1
while i <= 100:
    sum = sum+i
    i=i+1
print("1+2+…+100=",sum)
```

for循环

基于序列的for循环

```
sum = 0
lista = [1,6,34,26,56,2,9,86,23]
for i in lista:
    sum = sum +i
print("sum=",sum)
```

基于字典的for循环

```
dicta= {'体育':78,'英语':86,'操作系统':93,'网络安全':63,'网络编程':74}
sum=0
avr=0
for key in dicta.keys():
    sum = sum + dicta[key]
avr = sum / len(dicta)
print("总成绩: ",sum)
print("平均成绩: ",avr)
```

基于for迭代访问集合

```
set_a = {1,3,34,31,67,98,-12,0,65}
sum = 0
for i in set_a:
    sum = sum + i
print("sum=",sum)
```

基于range()函数的计数循环

```
sum = 0
for i in range(2,101,2):
    sum = sum + i
print("sum=",sum)
```

循环嵌套

```
for i in range(1,10):
    for j in range(1,i+1):
        print("%d×%d=%d"%(i,j,i*j),end =' ')
    print()
```

break和continue语句

执行break语句会退出循环。

```
numa= randint(1,100)
numstr = input("请猜一猜，这个数是多少(1—100)?")
while True:
    numint = int(numstr)
    if numint > numa:
        print("猜大了")
    elif numint < numa:
        print("猜小了")
    else:
        break
    numstr = input("再猜一次吧：")
print("你猜对了")
```

continue语句

continue语句在while和for循环中起到提前结束本次循环的作用，并忽略continue后面的语句，然后回到循环的顶端，继续执行下一次循环。

```
sum = 0
i=0
while i<100:
    i=i+1
    if i%2==0:
        continue
    print("i=%d"%i)
    sum=sum+i
print("1—100之间奇数的和为%d"%sum)
```

pass语句

pass 是空语句，是为了保持程序结构的完整性。

pass 不做任何事情，一般用做占位语句。

```
for ch in 'Python':
    if ch == 'h':
        pass # 什么也不做，类似于C中的";"
    else:
        print('当前字母 :', ch)
```

Python函数

函数的定义与调用

自定义函数的语法如下：

def **函数名**(**参数**)：

　　函数体

没有参数的函数

```
#coding:utf-8
import turtle as t
def drawcircle():
    t.pencolor("red")
    t.fillcolor("red")
    t.begin_fill()
    t.circle(100)
    t.end_fill()
drawcircle()
```

函数的定义与调用

有参数的函数

```
import turtle as t
def drawstar(r,pencolor,fillcolor):
    t.pencolor(pencolor)
    t.fillcolor(fillcolor)
    t.begin_fill()
    t.circle(r)
    t.end_fill()
drawstar(50,'red','blue')
```

函数的隐含返回值

函数体中 return 语句有指定返回值时返回的就是其值。

函数体中没有 return 语句时，函数运行结束会隐含返回一个 None 作为返回值，类型是 NoneType，与 return 、return None 等效，都是返回 None。

函数的隐含返回值

```
def showplus(x):
    print(x)

num = showplus(6)
print(num)
print(type(num))
```

输出结果：

```
6
None
<class 'NoneType'>
```

有返回值的函数

```
def max(a,b):
    if a>b:
        return a
    else:
        return b
x = 5
y = 7
print("%d,%d中较大的一个是 %d"%(x,y,max(x,y)))
```

多个返回值的函数

如果程序需要有多个返回值，则既可将多个值包装成列表之后返回，也可直接返回多个值。如果 Python 函数直接返回多个值，Python 会自动将多个返回值封装成元组。

```
def sum_and_avg(a,b):
    sum = a + b
    return sum, sum / 2 #相当于 return [sum, sum/2]
```

调用

```
sum, avg = sum_and_avg(2,6)
```

参数的默认值

有默认值的形参放在形参列表的后边，没有默认值的形参放在形参列表的前面

```
import turtle as t
def drawcircle(r,pencolor="black",fillcolor="black"):
    t.pencolor(pencolor)
    t.fillcolor(fillcolor)
    t.begin_fill()
    t.circle(r)
    t.end_fill()
drawcircle(100)
drawcircle(50,"red","yellow")
```

函数的可变长参数

当参数是可变长度时，只需在参数前加“*”就可以了

```
def sum(*v):                              # v为可变长形参，前面加有*
    print("参数长度为%d"%len(v))   # len(v)求v的长度
    sum = 0
    for i in v:          #对v中的每个元素枚举，然后求和
        sum = sum + i
    return sum

print(sum(1,2,3,4,5)) #给sum函数传递5个值
print(sum(1,2))                    #给sum函数传递2个值
```

可变长参数将作字典

标识符，表示可变长参数将被当作一个字典**

```
def sum(**v):                    # 函数会把传过来的可变长参数当作字典
    counts = len(v)                    # 求字典的元素个数
    print("共有 %d 门课程"%(counts))
    sum = 0
    for subj in v.keys():
        #枚举字典元素
        print("%s : %d "%(subj, v[subj]))          # 显示字典的键和值
        sum = sum+v[subj]                          # 对字典的值求和
    return sum

scores = sum(Chinese=95,English=78,数学=56)        # 传参数
print("总分为 %d"%scores)
```

lambda匿名函数

- lambda 参数列表:表达式
- 参数列表是传递参数给lambda函数的参数，可以是一个参数，也可以是多个参数。根据参数，表达式进行某种运算，表达式的值就是lambda函数的值。

```
>>> f = lambda x:x**2
>>> f(3)
9
>>> f = lambda x,y:x*y
>>> print(f(1,4))
4
```

使用lambda生成列表

```
>>> lista = [lambda x:x*2, lambda x:x**2, lambda x:x**3]
>>> print(lista[0](3),lista[1](3),lista[2](3))
6 9 27
```

使用lambda生成字典：

```
>>> dicta = {"key1":lambda x:x*2, "key2":lambda x:x**2,"key3":lambda x:x**3}
>>> dicta['key1'](2)
4
>>> dicta["key2"](3)
9
>>> dicta["key3"](2)
8
```

Python 模块

模块(Module)，是一个 Python 文件，以 .py 结尾，包含了 Python 对象定义和Python语句。

模块让你能够有逻辑地组织你的 Python 代码段。

把相关的代码分配到一个模块里能让你的代码更好用，更易懂。

模块能定义函数，类和变量，模块里也能包含可执行的代码。

Python内置函数

abs()	divmod()	input()
open()	staticmethod()	all()
enumerate()	int()	ord()
str()	any()	eval()
isinstance()	pow()	sum()
basestring()	execfile()	issubclass()
print()	super()	bin()
file()	iter()	property()
tuple()	bool()	filter()
len()	range()	type()
bytearray()	float()	list()
raw_input()	unichr()	callable()

参考：https://www.runoob.com/python/python-built-in-functions.html

Python内置函数

format()
unicode()
long()
classmethod()
repr()
globals()
zip()
memoryview()
complex()
set()
next()
hex()
dir()
locals()
chr()
reload()
getattr()
xrange()
max()
compile()
round()
hash()
delattr()
setattr()
object()
id()
oct()
reduce()
frozenset()
vars()
map()
cmp()
reverse()
hasattr()
__import__()
min()
help()
dict()
slice()
sorted()