我的 Python 学习笔记

作者：HaujetZhao

原文链接：https://ld246.com/article/1595481801029

来源网站：链滴

查看 Python 版本

python -V #大写V

更改源为清华

pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple

Linux 下 python 脚本

#!/usr/bin/python3

print("Hello, World!")

Python 环境变量

变量名	描述
PYTHONPATH	PYTHONPATH是Python搜索路径，默认我们import的模块都会从PYTHONPATH里面寻找。
PYTHONSTARTUP	Python启动后，先寻找PYTHONSTARTUP环境变量，然后执行此变量指定的文件中的代码。
PYTHONCASEOK	加入PYTHONCASEOK的环境量, 就会使python导入模块的时候不区分大小写.
PYTHONHOME	另一种模块搜索路径。它通常内于的PYTHONSTARTUP或PYTHONPATH目录中，使得两个模块库更容易切换。

Python命令行参数

选项	描述
-d	在解析时显示调试信息
-O	生成优化代码（.pyo 文件）
-S	启动时不引入查找Python路径的位置
-V	输出Python版本号
-X	从 1.6版本之后基于内建的异常（仅仅用于字串）已过时。
-c cmd	**执行 Python 脚本，并将运行结果作为 cmd 字符串。**
file	**在给定的python文件执行python脚本。**

PyCharm

PyCharm 是由 JetBrains 打造的一款 Python IDE，支持 macOS、 Windows、 Linux 系统。下载址

ipython

ipython 是一个 python 的交互式 shell，比默认的 python shell 好用得多，支持变量自动补全，自缩进，支持 bash shell 命令，内置了许多很有用的功能和函数。

此 ipython 中的 i 代表 “交互(interaction)” 。官网

安装:

```
pip install ipython
```

Python3 基础语法

编码

默认情况下，Python 3 源码文件以 **UTF-8** 编码，所有字符串都是 unicode 字符串。

标识符

- 第一个字符必须是字母下划线 。
- 其他部分由字母、数字和下划线组成。
- 大小写敏感。

在 Python 3 中，可以用中文作为变量名。

保留字

```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'exce
t', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise'
 'return', 'try', 'while', 'with', 'yield']
```

注释

单行注释以 # 开头

多行用 ''' 和 """

行与缩进

python最具特色的就是使用缩进来表示代码块，不需要使用大括号 {} 。

缩进的空格数可变，同一个代码块的语句必须包含相同的缩进空格数。

缩进不一致，执行后会出现错误

多行语句

可以使用反斜杠()来实现多行语句

```
total = item_one + \
        item_two + \
        item_three
```

在 [], {}, 或 () 中的多行语句，不需要使用反斜杠()，
如：

```
total = ['item_one', 'item_two', 'item_three',
        'item_four', 'item_five']
```

数字(Number)类型

python中数字有四种类型：整数、布尔型、浮点数和复数。

- **int** (整数), 如 1, 只有一种整数类型 int，表示为长整型，没有 python2 中的 Long。
- **bool** (布尔), 如 True。
- **float** (浮点数), 如 1.23、3E-2
- **complex** (复数), 如 1 + 2j、 1.1 + 2.2j

字符串(String)

- 单引号、双引号使用完全相同。
- 三引号('''或""")可以指定一个 **多行字符串** 。
- 转义符 '\'
- 反斜杠可以用来转义，使用 **r** 可以让反斜杠不发生转义。。 如**r"this is a line with \n"** 则**\n** 会显示，并不是换行。
- 按字面意义级联字符串，如**"this " "is " "string"** 会被自动转换为 **this is string** 。
- 字符串可以用**+** 运算符连接在一起，用 ***** 运算符重复。
- Python 中的字符串有两种索引方式，从左往右以 0 开始，从右往左以 -1 开始。
- Python中的字符串不能改变。
- Python 没有单独的字符类型，一个字符就是长度为 1 的字符串。
- 字符串的截取的语法格式如下： **变量[头下标:尾下标:步长]**

```
word = '字符串'
sentence = "这是一个句子。"
paragraph = """这是一个段落，
可以由多行组成"""
```

```
#!/usr/bin/python3

str='Runoob'

print(str)                 # 输出字符串
print(str[0:-1])           # 输出第一个到倒数第二个的所有字符
print(str[0])              # 输出字符串第一个字符
```

```
print(str[2:5])          # 输出从第三个开始到第五个的字符
print(str[2:])          # 输出从第三个开始后的所有字符
print(str * 2)          # 输出字符串两次
print(str + '你好')       # 连接字符串

print('------------------------------')

print('hello\nrunoob')      # 使用反斜杠(\)+n转义特殊字符
print(r'hello\nrunoob')     # 在字符串前面添加一个 r，表示原始字符串，不会发生转义，这里的 r 指raw，即 raw string。
```

一行，多条语句

语句之间使用分号 **;** 分割

多个语句构成代码组

像if、while、def和class这样的复合语句，首行以关键字开始，以冒号(:)结束，该行之后的一行或行代码构成代码组。

首行及后面的代码组称为一个子句(clause)。

Print 输出

print 默认输出换行，如果要实现不换行，需要在变量末尾加上 **end=""**：

```
#!/usr/bin/python3

x="a"
y="b"
print( x, end=" " )
print( y, end=" " )
print()
```

import 与 from...import

将整个模块(somemodule)导入，格式为： **import somemodule**

从某个模块中导入多个函数,格式为： **from somemodule import firstfunc, secondfunc, thirdfunc**

将某个模块中的全部函数导入，格式为： **from somemodule import ***

```
from sys import argv,path  #  导入特定的成员

print('================python from import===========================')
print('path:',path) # 因为已经导入path成员，所以此处引用时不需要加sys.path
```

Python3 基本数据类型

变量不需要声明。变量在使用前，必须赋值。

变量就是变量，它没有类型。

多个变量赋值

Python允许你同时为多个变量赋值。例如：创建一个整型对象，值为 1，从后向前赋值，三个变量被予相同的数值。

```
a = b = c = 1
```

也可以为多个对象指定多个变量。例如：

```
a, b, c = 1, 2, "runoob"
#  1 和 2 的分配给 a 和 b，  "runoob" 分配给 c。
```

标准数据类型

Python3 中有六个标准的数据类型：

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

Python3 的六个标准数据类型中：

- **不可变数据（3 个）：**Number（数字）、String（字符串）、Tuple（元组）；
- **可变数据（3 个）：**List（列表）、Dictionary（字典）、Set（集合）。

Number（数字）

type() 函数可以用来查询变量所指的对象类型

此外还可以用 isinstance 来判断：

```
>>>a = 111
>>> isinstance(a, int)
True
>>>
```

isinstance 和 type 的区别在于：

- type()不会认为子类是一种父类类型。
- isinstance()会认为子类是一种父类类型。

```
>>> class A:
```

```
...     pass
...
>>> class B(A):
...     pass
...
>>> isinstance(A(), A)
True
>>> type(A()) == A
True
>>> isinstance(B(), A)
True
>>> type(B()) == A
False
```

可以用del语句删除对象：

```
del var
del var_a, var_b
```

数值运算

```
>>>5 + 4  # 加法
9
>>> 4.3 - 2 # 减法
2.3
>>> 3 * 7  # 乘法
21
>>> 2 / 4  # 除法，得到一个浮点数
0.5
>>> 2 // 4 # 除法，得到一个整数
0
>>> 17 % 3 # 取余
2
>>> 2 ** 5 # 乘方
32
```

除法包含两个运算符：**/** 返回一个浮点数，**//** 返回一个整数。

混合计算时，Python会把整型转换成为浮点数。

复数可以用 **a + bj** ,或者 **complex(a,b)** 表示， 复数的实部 **a** 和虚部 **b** 都是 **浮点型**

数值类型实例

int	float	complex
10	0.0	3.14j
100	15.20	45.j
-786	-21.9	9.322e-36j
080	32.3e+18	.876j
-0490	-90.	-.6545+0J

-0x260	-32.54e100	3e+26J
0x69	70.2E-12	4.53e-7j

Python还支持复数，复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示， 复数实部a和虚部b都是浮点型

String（字符串）

Python中的字符串用单引号 ' 或双引号 " 括起来，同时使用反斜杠 ** 转义特殊字符。

字符串的截取的语法格式如下：

变量[头下标:尾下标]

索引值以 0 为开始值，-1 为从末尾的开始位置。

加号 + 是字符串的连接符， 星号 ***** 表示复制当前字符串

```
#!/usr/bin/python3

str = 'Runoob'

print (str)          # 输出字符串
print (str[0:-1])    # 输出第一个到倒数第二个的所有字符
print (str[0])       # 输出字符串第一个字符
print (str[2:5])     # 输出从第三个开始到第五个的字符
print (str[2:])      # 输出从第三个开始的后的所有字符
print (str * 2)      # 输出字符串两次，也可以写成 print (2 * str)
print (str + "TEST") # 连接字符串
```

不想让反斜杠发生转义，可以在字符串前面添加一个 **r**

反斜杠()可以作为续行符，表示下一行是上一行的延续。

也可以使用 """...""" 或者 '''...''' 跨越多行。

Python 字符串不能被改变。向一个索引位置赋值，比如word[0] = 'm'会导致错误。

List（列表）[]

列表是写在方括号 **[]** 之间、用逗号分隔开的元素列表。

列表截取的语法格式如下：

变量[头下标:尾下标]

加号 + 是列表连接运算符，星号 ***** 是重复操作

```
#!/usr/bin/python3

list = [ 'abcd', 786 , 2.23, 'runoob', 70.2 ]
```

```
tinylist = [123, 'runoob']

print (list)            # 输出完整列表
print (list[0])         # 输出列表第一个元素
print (list[1:3])       # 从第二个开始输出到第三个元素
print (list[2:])        # 输出从第三个元素开始的所有元素
print (tinylist * 2)    # 输出两次列表
print (list + tinylist) # 连接列表
```

列表中的元素是可以改变的：

```
>>>a = [1, 2, 3, 4, 5, 6]
>>> a[0] = 9
>>> a[2:5] = [13, 14, 15]
>>> a
[9, 2, 13, 14, 15, 6]
>>> a[2:5] = []   # 将对应的元素值设置为 []
>>> a
[9, 2, 6]
```

索引 1 到索引 4 的位置并设置为步长为 2（间隔一个位置）来截取字符串：

图片不可用

如果第三个参数为负数表示逆向读取，以下实例用于翻转字符串：

```
def reverseWords(input):

    # 通过空格将字符串分隔符，把各个单词分隔为列表
    inputWords = input.split(" ")

    # 翻转字符串
    # 假设列表 list = [1,2,3,4],
    # list[0]=1, list[1]=2 ，而 -1 表示最后一个元素 list[-1]=4 ( 与 list[3]=4 一样)
    # inputWords[-1::-1] 有三个参数
    # 第一个参数 -1 表示最后一个元素
    # 第二个参数为空，表示移动到列表末尾
    # 第三个参数为步长，-1 表示逆向
    inputWords=inputWords[-1::-1]

    # 重新组合字符串
    output = ' '.join(inputWords)

    return output

if __name__ == "__main__":
    input = 'I like runoob'
    rw = reverseWords(input)
    print(rw)
```

Tuple（元组）()

元组（tuple）与列表类似，元组的元素不能修改。元组写在小括号 **()** 里，元素之间用逗号隔开。

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。

构造包含 0 个或 1 个元素的元组比较特殊：

```
tup1 = ()    # 空元组
tup2 = (20,) # 一个元素，需要在元素后添加逗号
```

Set（集合）{}

基本功能是进行成员关系测试和删除重复元素。

可以使用大括号 **{ }** 或者 **set()** 创建集合

创建一个空集合必须用 **set()** 而不是 **{ }**，因为 **{ }** 是用来创建一个空字典。

```
parame = {value01,value02,...}
```

```
#!/usr/bin/python3

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

print(student)   # 输出集合，重复的元素被自动去掉

# 成员测试
if 'Rose' in student :
    print('Rose 在集合中')
else :
    print('Rose 不在集合中')


# set可以进行集合运算
a = set('abracadabra')
b = set('alacazam')

print(a)

print(a - b)     # a 和 b 的差集

print(a | b)     # a 和 b 的并集

print(a & b)     # a 和 b 的交集

print(a ^ b)     # a 和 b 中不同时存在的元素
```

Dictionary（字典）{}

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)** 的集合。

键(key) 必须使用不可变类型。

在同一个字典中，**键(key)** 必须是唯一的。

```
#!/usr/bin/python3
```

```
dict = {}
dict['one'] = "1 - 菜鸟教程"
dict[2]     = "2 - 菜鸟工具"

tinydict = {'name': 'runoob','code':1, 'site': 'www.runoob.com'}


print (dict['one'])       # 输出键为 'one' 的值
print (dict[2])           # 输出键为 2 的值
print (tinydict)          # 输出完整的字典
print (tinydict.keys())   # 输出所有键
print (tinydict.values()) # 输出所有值
```

构造函数 **dict()** 可以直接从键值对序列中构建字典：

```
>>>dict([('Runoob', 1), ('Google', 2), ('Taobao', 3)])
{'Taobao': 3, 'Runoob': 1, 'Google': 2}
```

Python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转换到一个浮点数
complex(real [,imag])	建一个复数
str(x)	将对象 x 转换为字符串
repr(x)	将对象 x 转换为表达式字符串
eval(str)	用来计算在字符串中的有效Python表达,并返回一个对象
tuple(s)	将序列 s 转换为一个元组
list(s)	将序列 s 转换为一个列表
set(s)	转换为可变集合
dict(d)	创建一个字典。d 必须是一个 (key, value 元组序列。
frozenset(s)	转换为不可变集合
chr(x)	将一个整数转换为一个字符
ord(x)	将一个字符转换为它的整数值
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

Python3 解释器

Linux/Unix的系统上

```
$ PATH=$PATH:/usr/local/python3/bin/python3    # 设置环境变量
$ python3 --version
Python 3.4.0
```

在Window系统下：

```
set path=%path%;C:\python34
```

在Linux/Unix系统中，加入下面头：

```
#! /usr/bin/env python3
```

Python3 运算符

Python算术运算符

运算符	描述
+	加 - 两个对象相加
-	减 - 得到负数或是一个数减去另一个数
*	乘 - 两个数相乘或是返回一个被重复若干次的符串
/	除 - x 除以 y
%	取模 - 返回除法的余数
**	幂 - 返回x的y次幂
//	取整除 - 向下取接近商的整数

Python比较运算符

运算符	描述
==	等于 - 比较对象是否相等
!=	不等于 - 比较两个对象是否不相等
>	大于 - 返回x是否大于y
<	小于
>=	大于等于 - 返回x是否大于等于y。
<=	小于等于 - 返回x是否小于等于y。

Python赋值运算符

运算符	描述
=	简单的赋值运算符
+=	加法赋值运算符
-=	减法赋值运算符

*=	乘法赋值运算符
/=	除法赋值运算符
%=	取模赋值运算符
**=	幂赋值运算符
//=	取整除赋值运算符
:=	海象运算符，可在表达式内部为变量赋值。**Python3.8 版本新增运算符。**

Python位运算符

运算符	描述
&	按位与
|	按位或
^	按位异或
~	按位取反运算符：对数据的每个二进制位取反,把1变为0,把0变为1。**~x** 类似于 **-x-1**
<<	左移动运算符：运算数的各二进位全部左移若干位，由"<<"右边的数指定移动的位数，高位丢弃，低位补0。
>>	右移动运算符：把">>"左边的运算数的各二位全部右移若干位，">>"右边的数指定移动的位数

Python逻辑运算符

假设变量 a 为 10, b为 20

运算符	逻辑表达式	描述	例
and	x and y	布尔"与" - 如果 x 为 False，x and y 返回 False，否则它返回 y 的计算值。	(a and b) 返回 20。
or	x or y	布尔"或" - 如果 x 是 True，它返回 x 的，否则它返回 y 的计算值。	(a or b) 返回 10。
not	not x	布尔"非" - 如果 x 为 True，返回 False。如果 x 为 False，它返回 True。	not(a and b) 返回 False

运算顺序：not and or

Python成员运算符

Python还支持成员运算符，测试实例中包含了一系列的成员，包括字符串，列表或元组。

运算符	描述	实例
in	如果在指定的序列中找到值返回 True，否则返回 False。	在 y 序列中 , 如果 x 在 y 序列中返回 True。

not in 如果在指定的序列中没有找到值返回 True，否则返回 False。
不在 y 序列中 , 如果 x 不在 y 序列中返回 True。

Python身份运算符

身份运算符用于比较两个对象的存储单元

运算符	描述	实例
is	is 是判断两个标识符是不是引用自一个对象	**is y**, 类似 **id(x) == id(y)** , 如果引用的是同一个对象则返回 True，否则返回 False
is not	is not 是判断两个标识符是不是引用自不同对象	**is not y** ， 类似 **id(a) != id(b)**。如果引用的不是同一个对象则返回结果 True，否则返回 False。

注： id() 函数用于获取对象内存地址。

Python3 数字(Number)

```
var1 = 1
var2 = 10
del var1[,var2[,var3[....,varN]]]
del var
del var_a, var_b
```

Python 支持三种数值类型：

- **整型(Int)** - 正或负整数，不带小数点。Python3 整型没有限制大小
- **浮点型(float)** - 浮点型由整数部分与小数部分组成，也可以使用科学计数法表示（2.5e2 = 2.5 x 10 = 250）
- **复数((complex))** - 复数由实数部分和虚数部分构成，可以用a + bj,或者complex(a,b)表示， 复的实部a和虚部b都是浮点型。

转换：

- **int(x)** 将x转换为一个整数。
- **float(x)** 将x转换到一个浮点数。
- **complex(x)** 将x转换到一个复数，实数部分为 x，虚数部分为 0。
- **complex(x, y)** 将 x 和 y 转换到一个复数，实数部分为 x，虚数部分为 y。x 和 y 是数字表达式。

注意：**// 得到的并不一定是整数类型的数，它与分母分子的数据类型有关系。

```
>>> 7//2
3
>>> 7.0//2
3.0
>>> 7//2.0
3.0
>>>
```

不同类型的数混合运算时会将整数转换为浮点数

在交互模式中，最后被输出的表达式结果被赋值给变量 _ 。

数学函数

函数	返回值（描述）
abs(x)	
ceil(x)	上入整数，如math.ceil(4.1) 返回 5
cmp(x, y)	**Python 3 已废弃
exp(x)	返回e的x次幂(ex)
fabs(x)	
floor(x)	下舍整数，如math.floor(4.9)返回 4
log(x)	如math.log(math.e)返回1.0,math.log(100 10)返回2.0
log10(x)	返回以10为基数的x的对数，如math.lo 10(100)返回 2.0
max(x1, x2,...)	返回给定参数的最大值，参数可为序列。
min(x1, x2,...)	返回给定参数的最小值，参数可以序列。
modf(x)	返回x的整数部分与小数部分，两部分的值符号与x相同，整数部分以浮点型表示。
pow(x, y)	x**y 运算后的值。
round(x ,n)	返回浮点数 x 的四舍五入值，如给出值，则代表舍入到小数点后的位数。**其实准确的说是保留值将保留到离上一位更近的一端。**
sqrt(x)	返回数字x的平方根。

随机数函数

函数	描述
choice(seq)	从序列的元素中随机挑选一个元素
randrange ([start,] stop [,step])	start 到 stop 之间生成一个随机数，步进为 step
random()	随机生成 0 - 1 间的随机数
seed(x)	改变随机数生成器的种子seed。
shuffle(lst)	随机排序序列的所有元素
uniform(x, y)	生成 x 到 y 之间的随机实数

三角函数

函数	描述
acos(x)	返回x的反余弦弧度值。

asin(x)	返回x的反正弦弧度值。
atan(x)	返回x的反正切弧度值。
atan2(y, x)	返回给定的 X 及 Y 坐标值的反正切。
cos(x)	返回x的弧度的余弦值。
hypot(x, y)	返回欧几里德范数 sqrt(xx + yy)。
sin(x)	返回的x弧度的正弦值。
tan(x)	返回x弧度的正切值。
degrees(x)	将弧度转换为角度,如degrees(math.pi/2)， 返回90.0
radians(x)	将角度转换为弧度

数学常量

常量	描述
pi	数学常量 pi（圆周率，一般以π来表示）
e	数学常量 e，e即自然常数（自然常数）。

Python3 字符串

Python转义字符

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(Backspace)
\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** 代表的字符，例如：**\o12** 代换行，其中 o 是字母，不是数字 0。
\xyy	十六进制数，yy代表的字符，例如：\x0a代换行

\other	其它的字符以普通格式输出

Python字符串运算符

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，遵循**左闭右开**原则，tr[0:2] 是不包含第 3 个字符的。
in	成员运算符 - 如果字符串中包含给定的字符返回 True
not in	成员运算符 - 如果字符串中不包含给定的字符返回 True
r/R	原始字符串 - 原始字符串
%	格式字符串

Python字符串格式化

```
#!/usr/bin/python3

print ("我叫 %s 今年 %d 岁!" % ('小明', 10))
```

```
我叫 小明 今年 10 岁!
```

python字符串格式化符号:

符 号	描述
%c	格式化字符及其ASCII码
%s	格式化字符串
%d	格式化整数
%u	格式化无符号整型
%o	格式化无符号八进制数
%x	格式化无符号十六进制数
%X	格式化无符号十六进制数（大写）
%f	格式化浮点数字，可指定小数点后的精度
%e	用科学计数法格式化浮点数
%E	作用同%e，用科学计数法格式化浮点数
%g	%f和%e的简写
%G	%f 和 %E 的简写
%p	用十六进制数格式化变量的地址

格式化操作符辅助指令:

符号	功能
*	定义宽度或者小数点精度
-	用做左对齐
+	在正数前面显示加号(+)
	在正数前面显示空格
#	在八进制数前面显示零('0')，在十六进制前面显示'0x'或者'0X'(取决于用的是'x'还是'X')
0	显示的数字前面填充'0'而不是默认的空格
%	'%%'输出一个单一的'%'
(var)	映射变量(字典参数)
m.n.	m 是显示的最小总宽度,n 是小数点后的位数(如果可用的话)

Python2.6 开始，新增了一种格式化字符串的函数 str.format()，它增强了字符串格式化的功能。

Python三引号

三引号允许一个字符串跨多行

```
#!/usr/bin/python3

para_str = """这是一个多行字符串的实例
多行字符串可以使用制表符
TAB ( \t )。
也可以使用换行符 [ \n ]。
"""
print (para_str)
```

```
这是一个多行字符串的实例
多行字符串可以使用制表符
TAB (    )。
也可以使用换行符 [
 ]。
```

f-string

是 python3.6 之后版本添加的，称之为字面量格式化字符串

f-string 格式化字符串以 f 开头，后面跟着字符串，字符串中的表达式用大括号 {} 包起来，它会将变或表达式计算后的值替换进去：

```
>>> name = 'Runoob'
>>> f'Hello {name}'  # 替换变量

>>> f'{1+2}'         # 使用表达式
'3'
```

```
>>> w = {'name': 'Runoob', 'url': 'www.runoob.com'}
>>> f'{w["name"]}: {w["url"]}'
'Runoob: www.runoob.com'
```

在 Python 3.8 的版本中可以使用 = 符号来拼接运算表达式与结果：

```
>>> x = 1
>>> print(f'{x+1}')   # Python 3.6
2

>>> x = 1
>>> print(f'{x+1=}')   # Python 3.8
'x+1=2'
```

Unicode 字符串

普通字符串是以8位ASCII码进行存储的，而Unicode字符串则存储为16位unicode字符串

使用的语法是在字符串前面加上前缀 **u**。

在Python3中，所有的字符串都是Unicode字符串。

Python 的字符串内建函数

序号	方法及描述
1	capitalize() 将字符串的首字符大写
2	center(width, fillchar) 返回一个指定的宽度 width 居中的字符串，fillchar 为填充的字符，默认为空格。
3	count(str, beg= 0,end=len(string)) 返回 str 在 string 里面出现的次数，如果 beg 或者 end 指定则返回指定范围内 str 出现的次数
4	bytes.decode(encoding="utf-8", errors="strict") Python3 中没有 decode 方法，但我们可以使用 bytes 对象的 decode() 方法来解码给定的 bytes 对象，这个 bytes 对象可以由 str.encode() 来编码返回。
5	encode(encoding='UTF-8',errors='strict') 以 encoding 指定的编码格式编码字符串，如果出错默认报一个ValueError 的异常，除非 errors 指定的是 ignore'或者'replace'
6	endswith(suffix, beg=0, end=len(string)) 检字符串是否以 obj 结束，如果beg 或者 end 指定则检查指定的范围内是否以 obj 结束，如果是，返回 True,否则返回 False.
7	expandtabs(tabsize=8) 把字符串 string 中的 tb 符号转为空格，tab 符号默认的空格数是 8 。
8	find(str, beg=0, end=len(string)) 检测 str 是包含在字符串中，如果指定范围 beg 和 end ，则检查是否包含在指定范围内，如果包含返回开始的引值，否则返回-1
9	index(str, beg=0, end=len(string)) 跟find()法一样，只不过如果str不在字符串中会报一个异常.
10	isalnum() 如果字符串至少有一个字符并且所字符都是字母或数字则返 回 True,否则返回 False

11 isalpha() 如果字符串至少有一个字符并且所有字符都是字母则返回 True, 否则返回 False

12 isdigit() 如果字符串只包含数字则返回 True 则返回 False..

13 islower() 如果字符串中包含至少一个区分大写的字符，并且所有这些(区分大小写的)字符都是小写，则返回 True，否则返回 False

14 isnumeric() 如果字符串中只包含数字字符，返回 True，否则返回 False

15 isspace() 如果字符串中只包含空白，则返回 True，否则返回 False.

16 istitle() 如果字符串是标题化的(见 title())则返回 True，否则返回 False

17 isupper() 如果字符串中包含至少一个区分大写的字符，并且所有这些(区分大小写的)字符都是大写，则返回 True，否则返回 False

18 join(seq) 以指定字符串作为分隔符，将 seq 所有的元素(的字符串表示)合并为一个新的字符串

19 len(string) 返回字符串长度

20 [ljust(width, fillchar]) 返回一个原字符串左对齐,并使用 fillchar 填充至长度 width 的新字符串，fillchar 默认为空格。

21 lower() 转换字符串中所有大写字符为小写.

22 lstrip() 截掉字符串左边的空格或指定字符。

23 maketrans() 创建字符映射的转换表，对于接受两个参数的最简单的调用方式，第一个参数是字符串，表示需要转换的字符，第二个参数也是字符串表示转换的目标。

24 max(str) 返回字符串 str 中最大的字母。

25 min(str) 返回字符串 str 中最小的字母。

26 [replace(old, new, max]) 把 将字符串中的 str1 替换成 str2,如果 max 指定，则替换不超过 max 次。

27 rfind(str, beg=0,end=len(string)) 类似于 find()函数，不过是从右边开始查找.

28 rindex(str, beg=0, end=len(string)) 类似于 index()，不过是从右边开始.

29 [rjust(width,, fillchar]) 返回一个原字符串右对齐,并使用fillchar(默认空格）填充至长度 width 的新字符串

30 rstrip() 删除字符串字符串末尾的空格.

31 split(str="", num=string.count(str)) num=string.count(str)) 以 str 为分隔符截取字符串，如果 num 有指定值，则仅截取 num+1 个子字符串

32 [splitlines(keepends])
照行('\r', '\r\n', \n')分隔，返回一个包含各行作为元素的列表，如果参数 keepends 为 False，不包换行符，如果为 True，则保留换行符。

33 startswith(substr, beg=0,end=len(string))
查字符串是否是以指定子字符串 substr 开头，是则返回 True，否则返回 False。如果beg 和 end 指值，则在指定范围内检查。

34 [strip(chars]) 在字符串上执行 lstri
()和 rstrip()

35 swapcase() 将字符串中大写转换为小写，小
转换为大写

36 title() 返回"标题化"的字符串,就是说所有单词
是以大写开始，其余字母均为小写(见 istitle())

37 translate(table, deletechars="") 根据 str 给
的表(包含 256 个字符)转换 string 的字符, 要过滤掉的字符放到 deletechars 参数中

38 upper() 转换字符串中的小写字母为大写

39 zfill (width) 返回长度为 width 的字符串，原
符串右对齐，前面填充0

40 isdecimal() 检查字符串是否只包含十进制字
，如果是返回 true，否则返回 false。

Python3 列表

更新列表

```
#!/usr/bin/python3

list = ['Google', 'Runoob', 1997, 2000]

list[2] = 2001
```

删除列表元素

```
#!/usr/bin/python3

list = ['Google', 'Runoob', 1997, 2000]

del list[2]
```

Python列表脚本操作符

+ 号用于组合列表，* 号用于重复列表

嵌套列表

```
x[0][1]
```

Python列表函数&方法

Python包含以下函数:

序号	函数
1	len(list) 列表元素个数

2	max(list) 返回列表元素最大值
3	min(list) 返回列表元素最小值
4	list(seq) 将元组转换为列表

Python包含以下方法:

序号	方法
1	list.append(obj) 在列表末尾添加新的对象
2	list.count(obj) 统计某个元素在列表中出现的数
3	list.extend(seq) 在列表末尾一次性追加另一个列中的多个值（用新列表扩展原来的列表）
4	list.index(obj) 从列表中找出某个值第一个匹配的索引位置
5	list.insert(index, obj) 将对象插入列表
6	[list.pop(index=-1]) 移除表中的一个元素（默认最后一个元素），并且返回该元素的值
7	list.remove(obj) 移除列表中某个值的第一个匹项
8	list.reverse() 反向列表中元素
9	list.sort(key=None, reverse=False) 对原列进行排序
10	list.clear() 清空列表
11	list.copy() 复制列表

Python3 元组

元组中只包含一个元素时，需要在元素后面添加逗号，否则括号会被当作运算符使用

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

Python3 字典

```
dict = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}
print (dict['Alice'])
del dict['Name'] # 删除键 'Name'
dict.clear()     # 清空字典
del dict         # 删除字典
```

Python字典包含了以下内置函数：

序号	函数及描述
1	len(dict) 计算字典元素个数，即键的总数。
2	str(dict) 输出字典，以可打印的字符串表示。
3	type(variable) 返回输入的变量类型，如果变

是字典就返回字典类型。

Python字典包含了以下内置方法：

序号	函数及描述
1	radiansdict.clear() 删除字典内所有元素
2	radiansdict.copy() 返回一个字典的浅复制
3	radiansdict.fromkeys() 创建一个新字典，以列seq中元素做字典的键，val为字典所有键对应的初始值
4	radiansdict.get(key, default=None) 返回指键的值，如果值不在字典中返回default值
5	key in dict 如果键在字典dict里返回true，否返回false
6	radiansdict.items() 以列表返回可遍历的(键, 值)元组数组
7	radiansdict.keys() 返回一个迭代器，可以使用 list() 来转换为列表
8	radiansdict.setdefault(key, default=None) get()类似, 但如果键不存在于字典中，将会添加键并将值设为default
9	radiansdict.update(dict2) 把字典dict2的键/对更新到dict里
10	radiansdict.values() 返回一个迭代器，可以用 list() 来转换为列表
11	[pop(key,default]) 删除字给定键 key 所对应的值，返回值为被删除的值。key值必须给出。 否则，返回default值。
12	popitem() 随机返回并删除字典中的最后一对和值。

Python3 集合

集合（set）是一个无序的不重复元素序列。

可以使用大括号 **{ }** 或者 **set()** 函数创建集合，创建一个空集合必须用 **set()**

{ } 是用来创建一个空字典。

```
>>>basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket)                      # 这里演示的是去重功能
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket                 # 快速判断元素是否在集合内
True
>>> 'crabgrass' in basket
False

>>> # 下面展示两个集合间的运算.
...
>>> a = set('abracadabra')
>>> b = set('alacazam')
```

```
>>> a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b                              # 集合a中包含而集合b中不包含的元素
{'r', 'd', 'b'}
>>> a | b                              # 集合a或b中包含的所有元素
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b                              # 集合a和b中都包含了的元素
{'a', 'c'}
>>> a ^ b                              # 不同时包含于a和b的元素
{'r', 'd', 'b', 'm', 'z', 'l'}
```

类似列表推导式，同样集合支持集合推导式(Set comprehension):

```
>>>a = {x for x in 'abracadabra' if x not in 'abc'}
>>> a
{'r', 'd'}
```

集合的基本操作

1、添加元素

s.add(x)

还有一个方法，也可以添加元素，且参数可以是列表，元组，字典等，语法格式如下：

s.update(x)

x 可以有多个，用逗号分开。

2、移除元素

s.remove(x)

此外还有一个方法也是移除集合中的元素，且如果元素不存在，不会发生错误。格式如下所示：

s.discard(x)

随机删除集合中的一个元素：

s.pop()

set 集合的 pop 方法会对集合进行无序的排列，然后将这个无序排列集合的左面第一个元素进行删除。

3、计算集合元素个数

len(s)

4、清空集合

s.clear()

5、判断元素是否在集合中存在

```
x in s
```

集合内置方法完整列表

方法	描述
add()	为集合添加元素
clear()	移除集合中的所有元素
copy()	拷贝一个集合
difference()	返回多个集合的差集
difference_update()	移除集合中的元素，该元素指定的集合也存在。
discard()	删除集合中指定的元素
intersection()	返回集合的交集
intersection_update()	返回集合的交集。
isdisjoint()	判断两个集合是否包含相同的元素，果没有返回 True，否则返回 False。
issubset()	判断指定集合是否为该方法参数集合子集。
issuperset()	判断该方法的参数集合是否为指定合的子集
pop()	随机移除元素
remove()	移除指定元素
symmetric_difference()	返回两个集合中不重复元素集合。
symmetric_difference_update()	移除当前集合在另外一个指定集合相同的元素，并将另外一个指定集合中不同的元素插入到当前集合中。
union()	返回两个集合的并集
update()	给集合添加元素

Python3 条件控制

if 语句

```
if condition_1:
    statement_block_1
elif condition_2:
    statement_block_2
else:
    statement_block_3
```

每个条件后面要使用冒号 **:**

在Python中没有switch – case语句

Python3 循环语句

while 循环

```
while 判断条件(condition)：
    执行语句(statements)……
```

同样需要注意冒号和缩进。另外，在 Python 中没有 do..while 循环。

while 循环使用 else 语句

```
while <expr>:
    <statement(s)>
else:
    <additional_statement(s)>
```

简单语句组

如果你的while循环体中只有一条语句，你可以将该语句与while写在同一行中

```
while (flag): print ('欢迎访问菜鸟教程!')
```

for 语句

```
for <variable> in <sequence>:
    <statements>
else:
    <statements>
```

break 语句用于跳出当前循环体

```
#!/usr/bin/python3

sites = ["Baidu", "Google","Runoob","Taobao"]
for site in sites:
    if site == "Runoob":
        print("菜鸟教程!")
        break
    print("循环数据 " + site)
else:
    print("没有循环数据!")
print("完成循环!")
```

range()函数

它会生成数列遍历：

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

```
...
0
1
2
3
4

>>>for i in range(5,9) :
    print(i)


5
6
7
8
>>>
```

也可以指定不同的增量：

```
>>>for i in range(-10, -100, -30) :
    print(i)


-10
-40
-70
>>>
```

可以结合 range() 和 len() 遍历一个序列的索引：

```
>>>a = ['Google', 'Baidu', 'Runoob', 'Taobao', 'QQ']
>>> for i in range(len(a)):
...     print(i, a[i])
...
0 Google
1 Baidu
2 Runoob
3 Taobao
4 QQ
>>>

>>>list(range(5))
[0, 1, 2, 3, 4]
>>>
```

while 中使用 continue：

```
n = 5
while n > 0:
    n -= 1
    if n == 2:
        continue
    print(n)
print('循环结束。')
```

更多实例

```
#!/usr/bin/python3

for letter in 'Runoob':     # 第一个实例
   if letter == 'b':
      break
   print ('当前字母为 :', letter)

var = 10                    # 第二个实例
while var > 0:
   print ('当期变量值为 :', var)
   var = var -1
   if var == 5:
      break

print ("Good bye!")
```

循环语句可以有 else 子句，但循环被 break 终止时不执行。

pass 语句

Python 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，如下实例

```
>>>while True:
...     pass  # 等待键盘中断 (Ctrl+C)
```

Python3 迭代器与生成器

迭代器

迭代是Python最强大的功能之一，是一个可以记住遍历的位置的对象。

迭代器有两个基本的方法：**iter()** 和 **next()**。

```
>>> list=[1,2,3,4]
>>> it = iter(list)    # 创建迭代器对象
>>> print (next(it))   # 输出迭代器的下一个元素
1
>>> print (next(it))
2
>>>
```

```
#!/usr/bin/python3

list=[1,2,3,4]
it = iter(list)    # 创建迭代器对象
for x in it:
    print (x, end=" ")
```

```
#!/usr/bin/python3
```

```
import sys         # 引入 sys 模块

list=[1,2,3,4]
it = iter(list)    # 创建迭代器对象

while True:
    try:
        print (next(it))
    except StopIteration:
        sys.exit()
```

生成器

使用了 yield 的函数被称为生成器，生成器是一个返回迭代器的函数

```
#!/usr/bin/python3

import sys

def fibonacci(n): # 生成器函数 - 斐波那契
    a, b, counter = 0, 1, 0
    while True:
        if (counter > n):
            return
        yield a
        a, b = b, a + b
        counter += 1
f = fibonacci(10) # f 是一个迭代器，由生成器返回生成

while True:
    try:
        print (next(f), end=" ")
    except StopIteration:
        sys.exit()
```

Python3 函数

参数

- 必需参数
- 关键字参数

允许参数的顺序与声明时不一致

```
#!/usr/bin/python3

#可写函数说明
def printinfo( name, age ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)
   return
```

```
#调用printinfo函数
printinfo( age=50, name="runoob" )
```

- 默认参数

如果没有传递参数，则会使用默认参数

```
#!/usr/bin/python3

#可写函数说明
def printinfo( name, age = 35 ):
   "打印任何传入的字符串"
   print ("名字: ", name)
   print ("年龄: ", age)
   return

#调用printinfo函数
printinfo( age=50, name="runoob" )
print ("------------------------")
printinfo( name="runoob" )
```

- 不定长参数

```
def functionname([formal_args,] *var_args_tuple ):
   "函数_文档字符串"
   function_suite
   return [expression]
```

加了星号 ***** 的参数会以元组(tuple)的形式导入，存放所有未命名的变量参数。

```
#!/usr/bin/python3

# 可写函数说明
def printinfo( arg1, *vartuple ):
   "打印任何传入的参数"
   print ("输出: ")
   print (arg1)
   print (vartuple)

# 调用printinfo 函数
printinfo( 70, 60, 50 )
```

加两个星号 ** 会以字典形式导入

```
#!/usr/bin/python3

# 可写函数说明
def printinfo( arg1, **vardict ):
   "打印任何传入的参数"
   print ("输出: ")
   print (arg1)
   print (vardict)

# 调用printinfo 函数
printinfo(1, a=2,b=3)
```

参数中星号 ***** 可以单独出现， ***** 后的参数必须用关键字传入

```
>>> def f(a,b,*,c):
...     return a+b+c
...
>>> f(1,2,c=3) # 正常
6
```

匿名函数

python 使用 lambda 来创建匿名函数

lambda [arg1 [,arg2,.....argn]]:expression

```
#!/usr/bin/python3

# 可写函数说明
sum = lambda arg1, arg2: arg1 + arg2

# 调用sum函数
print ("相加后的值为 : ", sum( 10, 20 ))
print ("相加后的值为 : ", sum( 20, 20 ))
```

强制位置参数

Python3.8 新增了一个函数形参语法 / 用来指明函数形参必须使用指定位置参数，不能使用关键字参的形式。

在以下的例子中，形参 a 和 b 必须使用指定位置参数，c 或 d 可以是位置形参或关键字形参，而 e 或 要求为关键字形参

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)

f(10, 20, 30, d=40, e=50, f=60)
```

Python3 数据结构

列表

方法	描述
list.append(x)	相当于 a[len(a):] = [x]。
list.extend(L)	相当于 a[len(a):] = L。
list.insert(i, x)	在指定位置插入一个元素。i 是索引
list.remove(x)	删除列表中值为 x 的第一个元素
如果没有这样的元素，就会返回错误。	
list.pop([i])	从列表的指定位置移除元素，并将其
回。如果没有指定索引，a.pop()返回最后一个元素。	
list.clear()	移除列表中的所有项，等于del a[:]。
list.index(x)	返回列表中第一个值为 x 的元素的

引。如果没有匹配的元素就会返回一个错误。

list.count(x)	返回 x 在列表中出现的次数。
list.sort()	对列表中的元素进行排序。
list.reverse()	倒排列表中的元素。
list.copy()	返回列表的浅复制，等于a[:]。

将列表当做堆栈使用

用 append() 方法加到堆栈顶。用 pop() 方法可以把一个元素从堆栈顶释放

将列表当作队列使用

速度不快

```
>>> from collections import deque
>>> queue = deque(["Eric", "John", "Michael"])
>>> queue.append("Terry")           # Terry arrives
>>> queue.append("Graham")          # Graham arrives
>>> queue.popleft()                 # The first to arrive now leaves
'Eric'
>>> queue.popleft()                 # The second to arrive now leaves
'John'
>>> queue                           # Remaining queue in order of arrival
deque(['Michael', 'Terry', 'Graham'])
```

列表推导式

```
>>> vec = [2, 4, 6]
>>> [3*x for x in vec]
[6, 12, 18]

>>> [[x, x**2] for x in vec]
[[2, 4], [4, 16], [6, 36]]

>>> vec1 = [2, 4, 6]
>>> vec2 = [4, 3, -9]
>>> [x*y for x in vec1 for y in vec2]
[8, 6, -18, 16, 12, -36, 24, 18, -54]
>>> [x+y for x in vec1 for y in vec2]
[6, 5, -7, 8, 7, -5, 10, 9, -3]
>>> [vec1[i]*vec2[i] for i in range(len(vec1))]
[8, 12, -54]

>>> [str(round(355/113, i)) for i in range(1, 6)]
['3.1', '3.14', '3.142', '3.1416', '3.14159']
```

嵌套列表解析

3X4的矩阵列表：

```
>>> matrix = [
...     [1, 2, 3, 4],
...     [5, 6, 7, 8],
...     [9, 10, 11, 12],
... ]
```

将3X4的矩阵列表转换为4X3列表：

```
>>> [[row[i] for row in matrix] for i in range(4)]
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

也可以使用以下方法来实现：

```
>>> transposed = []
>>> for i in range(4):
...     transposed.append([row[i] for row in matrix])
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

另外一种实现方法：

```
>>> transposed = []
>>> for i in range(4):
...     # the following 3 lines implement the nested listcomp
...     transposed_row = []
...     for row in matrix:
...         transposed_row.append(row[i])
...     transposed.append(transposed_row)
...
>>> transposed
[[1, 5, 9], [2, 6, 10], [3, 7, 11], [4, 8, 12]]
```

元组和序列

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> # Tuples may be nested:
... u = t, (1, 2, 3, 4, 5)
>>> u
((12345, 54321, 'hello!'), (1, 2, 3, 4, 5))
```

集合

集合是一个无序不重复元素的集。集合也支持推导式。

dict() 直接从键值对元组列表中构建字典。

```
>>> dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

字典推导：

```
>>> {x: x**2 for x in (2, 4, 6)}
{2: 4, 4: 16, 6: 36}

>>> dict(sape=4139, guido=4127, jack=4098)
{'sape': 4139, 'jack': 4098, 'guido': 4127}
```

遍历技巧

在字典中遍历时，用 items() 方法同时解读出 **关键字** 和 **对应的值**：

```
>>> knights = {'gallahad': 'the pure', 'robin': 'the brave'}
>>> for k, v in knights.items():
...     print(k, v)
...
gallahad the pure
robin the brave
```

索引位置和对应值可以使用 enumerate() 函数同时得到：

```
>>> for i, v in enumerate(['tic', 'tac', 'toe']):
...     print(i, v)
...
0 tic
1 tac
2 toe
```

同时遍历两个或更多的序列，可以使用 zip() 组合：

```
>>> questions = ['name', 'quest', 'favorite color']
>>> answers = ['lancelot', 'the holy grail', 'blue']
>>> for q, a in zip(questions, answers):
...     print('What is your {0}?  It is {1}.'.format(q, a))
...
What is your name?  It is lancelot.
What is your quest?  It is the holy grail.
What is your favorite color?  It is blue.
```

要反向遍历一个序列，首先指定这个序列，然后调用 reversed() 函数：

```
>>> for i in reversed(range(1, 10, 2)):
...     print(i)
...
9
7
5
3
1
```

要按顺序遍历一个序列，使用 sorted() 函数返回一个已排序的序列，并不修改原值：

```
>>> basket = ['apple', 'orange', 'apple', 'pear', 'orange', 'banana']
>>> for f in sorted(set(basket)):
```

```
...     print(f)
...
apple
banana
orange
pear
```

Python3 模块

```
#!/usr/bin/python3
# 文件名: using_sys.py

import sys

print('命令行参数如下:')
for i in sys.argv:
   print(i)

print('\n\nPython 路径为：', sys.path, '\n')
```

- sys.argv 是一个包含命令行参数的列表。
- sys.path 包含了一个 Python 解释器自动查找所需模块的路径的列表。

import 语句

```
import module1[, module2[,... moduleN]
```

sys.path 输出是一个列表，空串''代表当前目录

from … import * 语句

把一个模块的所有内容全都导入到当前的命名空间

重新载入

```
import importlib
importlib.reload(moduel)
```

__name__属性

```
if __name__ == '__main__':
   print('程序自身在运行')
else:
   print('我来自另一模块')
```

dir() 函数

dir() 可以找到模块内定义的所有名称。以一个字符串列表的形式返回

包

```
import sound.effects.echo

from sound.effects import echo
```

从一个包中导入*

```
__all__ = ["echo", "surround", "reverse"]
```

Python3 输入和输出

文件对象的 write() 方法

标准输出文件可以用 sys.stdout 引用

str.format()

str()： 函数返回一个用户易读的表达形式。

repr(): 产生一个解释器易读的表达形式。

rjust() 字符串靠右, 并在左边填充空格 ljust() 和 center()

```
>>> for x in range(1, 11):
...     print('{0:2d} {1:3d} {2:4d}'.format(x, x*x, x*x*x))
...
 1   1    1
 2   4    8
 3   9   27
 4  16   64
 5  25  125
 6  36  216
 7  49  343
 8  64  512
 9  81  729
10 100 1000
```

str.format() 的基本使用如下:

```
>>> print('{0} 和 {1}'.format('Google', 'Runoob'))
Google 和 Runoob

>>> print('站点列表 {0}, {1}, 和 {other}。'.format('Google', 'Runoob', other='Taobao'))
站点列表 Google, Runoob, 和 Taobao。
```

!a (使用 **ascii()**), **!s** (使用 **str()**) 和 **!r** (使用 **repr()**)：

```
>>> import math
>>> print('常量 PI 的值近似为： {}。'.format(math.pi))
常量 PI 的值近似为： 3.141592653589793。
>>> print('常量 PI 的值近似为： {!r}。'.format(math.pi))
常量 PI 的值近似为： 3.141592653589793。
```

: 和格式标识符进行格式化：

```
>>> import math
>>> print('常量 PI 的值近似为 {0:.3f}。'.format(math.pi))
常量 PI 的值近似为 3.142。
```

在 **:** 后传入一个整数, 可以保证该域至少有这么多的宽度。 用于美化表格时很有用。

```
>>> table = {'Google': 1, 'Runoob': 2, 'Taobao': 3}
>>> for name, number in table.items():
...     print('{0:10} ==> {1:10d}'.format(name, number))
...
Google     ==>          1
Runoob     ==>          2
Taobao     ==>          3
```

然后使用方括号 **[]** 来访问键值 :

```
>>> table = {'Google': 1, 'Runoob': 2, 'Taobao': 3}
>>> print('Runoob: {0[Runoob]:d}; Google: {0[Google]:d}; Taobao: {0[Taobao]:d}'.format(table)

Runoob: 2; Google: 1; Taobao: 3
```

也可以通过在 table 变量前使用 ** 来实现相同的功能:

```
>>> table = {'Google': 1, 'Runoob': 2, 'Taobao': 3}
>>> print('Runoob: {Runoob:d}; Google: {Google:d}; Taobao: {Taobao:d}'.format(**table))
Runoob: 2; Google: 1; Taobao: 3
```

旧式 字符串格式化

```
>>> import math
>>> print('常量 PI 的值近似为：%5.3f。' % math.pi)
常量 PI 的值近似为：3.142。
```

因为 str.format() 是比较新的函数， 大多数的 Python 代码仍然使用 % 操作符。但是因为这种旧式格式化最终会从该语言中移除, 应该更多的使用 str.format().

读取键盘输入

```
#!/usr/bin/python3

str = input("请输入：");
print ("你输入的内容是: ", str)
```

读和写文件

```
open(filename, mode)
```

模式	描述
r	以只读方式打开文件。文件的指针将会放在文的开头。这是默认模式。
rb	以二进制格式打开一个文件用于只读。

r+	读写打开。
rb+	二进制读写打开。
w	写入。已存在则打开，不存在则创建。
wb	
w+	读写打开
wb+	
a	追加打开
ab	
a+	追加读写打开
ab+	

```
#!/usr/bin/python3

# 打开一个文件
f = open("/tmp/foo.txt", "w")

f.write( "Python 是一个非常好的语言。\n是的，的确非常好!!\n" )

# 关闭打开的文件
f.close()
```

文件对象的方法

f.read()

f.read(size)

f.readline()

f.readlines()

还可以迭代一个文件对象：

```
#!/usr/bin/python3

# 打开一个文件
f = open("/tmp/foo.txt", "r")

for line in f:
    print(line, end='')

# 关闭打开的文件
f.close()
```

f.write()

返回写入的字符数

f.tell()

返回文件对象当前所处的位置, 它是从文件开头开始算起的字节数

f.seek()

改变文件当前的位置，f.seek(offset, from_what)

- seek(x,0)： 从起始位置即文件首行首字符开始移动 x 个字符
- seek(x,1)： 表示从当前位置往后移动x个字符
- seek(-x,2)：表示从文件的结尾往前移动x个字符

f.close()

当处理一个文件对象时, 使用 with 关键字是非常好的方式。在结束后, 它会帮你正确的关闭文件。

```
>>> with open('/tmp/foo.txt', 'r') as f:
...     read_data = f.read()
>>> f.closed
True
```

pickle 模块

```
pickle.dump(obj, file, [,protocol])
x = pickle.load(file)
```

注解：从 file 中读取一个字符串，并将它重构为原来的python对象。

```
#!/usr/bin/python3
import pickle

# 使用pickle模块将数据对象保存到文件
data1 = {'a': [1, 2.0, 3, 4+6j],
         'b': ('string', u'Unicode string'),
         'c': None}

selfref_list = [1, 2, 3]
selfref_list.append(selfref_list)

output = open('data.pkl', 'wb')

# Pickle dictionary using protocol 0.
pickle.dump(data1, output)

# Pickle the list using the highest protocol available.
pickle.dump(selfref_list, output, -1)

output.close()

#!/usr/bin/python3
import pprint, pickle
```

```
#使用pickle模块从文件中重构python对象
pkl_file = open('data.pkl', 'rb')

data1 = pickle.load(pkl_file)
pprint.pprint(data1)

data2 = pickle.load(pkl_file)
pprint.pprint(data2)

pkl_file.close()
```

Python3 File(文件) 方法

使用 open() 方法一定要保证关闭文件对象，即调用 close() 方法。

```
open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

- file: 必需，文件路径（相对或者绝对路径）。
- mode: 可选，文件打开模式
- buffering: 设置缓冲
- encoding: 一般使用utf8
- errors: 报错级别
- newline: 区分换行符
- closefd: 传入的file参数类型
- opener:

序号	方法及描述
1	file.close()关闭文件。关闭后文件不能再进行写操作。
2	file.flush()刷新文件内部缓冲，直接把内部缓冲的数据立刻写入文件, 而不是被动的等待输出缓冲区写入。
3	file.fileno()返回一个整型的文件描述符(file descriptor FD 整型), 可以用在如os模块的read方法等一些底层操作上。
4	file.isatty()如果文件连接到一个终端设备返回 True，否则返回 False。
5	file.next()**Python 3 中的 File 对象不支持 next() 方法。**返回文件下一行。
6	[file.read(size]) 从文件读取指定的字数，如果未给定或为负则读取所有。
7	[file.readline(size]) 读取整行，包括 "\n" 字符。
8	[file.readlines(sizeint]) 读取所行并返回列表，若给定sizeint>0，返回总和大约为sizeint字节的行, 实际读取值可能比 sizeint 较大, 为需要填充缓冲区。

9 [file.seek(offset, whence])
动文件读取指针到指定位置

10 file.tell()返回文件当前位置。

11 [file.truncate(size]) 从文件的首行首
符开始截断，截断文件为 size 个字符，无 size 表示从当前位置截断；截断之后后面的所有字符被删
，其中 Widnows 系统下的换行代表2个字符大小。

12 file.write(str)将字符串写入文件，返回的是写
的字符长度。

13 file.writelines(sequence)向文件写入一个序
字符串列表，如果需要换行则要自己加入每行的换行符。

Python3 OS 文件/目录方法

序号	方法及描述
1	os.access(path, mode) 检验权限模式
2	os.chdir(path) 改变当前工作目录
3	os.chflags(path, flags) 设置路径的标记为数字
记。
4	os.chmod(path, mode) 更改权限
5	os.chown(path, uid, gid) 更改文件所有者
6	os.chroot(path) 改变当前进程的根目录
7	os.close(fd) 关闭文件描述符 fd
8	os.closerange(fd_low, fd_high) 关闭所有文件
述符，从 fd_low (包含) 到 fd_high (不包含), 错误会忽略
9	os.dup(fd) 复制文件描述符 fd
10	os.dup2(fd, fd2) 将一个文件描述符 fd 复制
另一个 fd2
11	os.fchdir(fd) 通过文件描述符改变当前工作目录
12	os.fchmod(fd, mode) 改变一个文件的访问
限，该文件由参数fd指定，参数mode是Unix下的文件访问权限。
13	os.fchown(fd, uid, gid) 修改一个文件的所有
，这个函数修改一个文件的用户ID和用户组ID，该文件由文件描述符fd指定。
14	os.fdatasync(fd) 强制将文件写入磁盘，该文
由文件描述符fd指定，但是不强制更新文件的状态信息。
15	[os.fdopen(fd, mode[, bufsize]
) 通过文件描述符 fd 创建一个文件对象，并返回这个文件对象
16	os.fpathconf(fd, name) 返回一个打开的文件
系统配置信息。name为检索的系统配置的值，它也许是一个定义系统值的字符串，这些名字在很多
准中指定（POSIX.1, Unix 95, Unix 98, 和其它）。
17	os.fstat(fd) 返回文件描述符fd的状态，像stat(
。
18	os.fstatvfs(fd) 返回包含文件描述符fd的文件
文件系统的信息，Python 3.3 相等于 statvfs()。

19 os.fsync(fd) 强制将文件描述符为fd的文件写硬盘。

20 os.ftruncate(fd, length) 裁剪文件描述符fd应的文件, 所以它最大不能超过文件大小。

21 os.getcwd() 返回当前工作目录

22 os.getcwdu() 返回一个当前工作目录的Unicode对象

23 os.isatty(fd) 如果文件描述符fd是打开的，同与tty(-like)设备相连，则返回true, 否则False。

24 os.lchflags(path, flags) 设置路径的标记为数标记，类似 chflags()，但是没有软链接

25 os.lchmod(path, mode) 修改连接文件权限

26 os.lchown(path, uid, gid) 更改文件所有者，似 chown，但是不追踪链接。

27 os.link(src, dst) 创建硬链接，名为参数 dst，向参数 src

28 os.listdir(path) 返回path指定的文件夹包含文件或文件夹的名字的列表。

29 os.lseek(fd, pos, how) 设置文件描述符 fd当位置为pos, how方式修改: SEEK_SET 或者 0 设置从文件开始的计算的pos; SEEK_CUR或者 1 则从当位置计算; os.SEEK_END或者2则从文件尾部开始. 在unix，Windows中有效

30 os.lstat(path) 像stat(),但是没有软链接

31 os.major(device) 从原始的设备号中提取设备ajor号码 (使用stat中的st_dev或者st_rdev field)。

32 os.makedev(major, minor) 以major和mino设备号组成一个原始设备号

33 [os.makedirs(path, mode)] 归文件夹创建函数。像mkdir(), 但创建的所有intermediate-level文件夹需要包含子文件夹。

34 os.minor(device) 从原始的设备号中提取设备inor号码 (使用stat中的st_dev或者st_rdev field)。

35 [os.mkdir(path, mode)] 以字mode的mode创建一个名为path的文件夹.默认的 mode 是 0777 (八进制)。

36 [os.mkfifo(path, mode)] 创命名管道，mode 为数字，默认为 0666 (八进制)

37 [os.mknod(filename, mode=0600, device]) 创建一个名为filename文件系统节点（文件，设备特别文件或者命名pipe）。

38 [os.open(file, flags, mode]) 开一个文件，并且设置需要的打开选项，mode参数是可选的

39 os.openpty() 打开一个新的伪终端对。返回 pt和 tty的文件描述符。

40 os.pathconf(path, name) 返回相关文件的系配置信息。

41 os.pipe() 创建一个管道. 返回一对文件描述符(rw) 分别为读和写

42 [os.popen(command, mode[, bufsize]
) 从一个 command 打开一个管道

43 os.read(fd, n) 从文件描述符 fd 中读取最多 n
个字节，返回包含读取字节的字符串，文件描述符 fd对应文件已达到结尾, 返回一个空字符串。

44 os.readlink(path) 返回软链接所指向的文件

45 os.remove(path) 删除路径为path的文件。
果path 是一个文件夹，将抛出OSError; 查看下面的rmdir()删除一个 directory。

46 os.removedirs(path) 递归删除目录。

47 os.rename(src, dst) 重命名文件或目录，从 sr
到 dst

48 os.renames(old, new) 递归地对目录进行更
，也可以对文件进行更名。

49 os.rmdir(path) 删除path指定的空目录，如
目录非空，则抛出一个OSError异常。

50 os.stat(path) 获取path指定的路径的信息，
能等同于C API中的stat()系统调用。

51 [os.stat_float_times(newvalue])
决定stat_result是否以float对象显示时间戳

52 os.statvfs(path) 获取指定路径的文件系统统
信息

53 os.symlink(src, dst) 创建一个软链接

54 os.tcgetpgrp(fd) 返回与终端fd （一个由os.
pen()返回的打开的文件描述符）关联的进程组

55 os.tcsetpgrp(fd, pg) 设置与终端fd（一个由os
open()返回的打开的文件描述符）关联的进程组为pg。

56 os.tempnam([dir[, prefix]]) **Python3 中已
除。**返回唯一的路径名用于创建临时文件。

57 os.tmpfile() **Python3 中已删除。**返回一
打开的模式为(w+b)的文件对象 .这文件对象没有文件夹入口，没有文件描述符，将会自动删除。

58 os.tmpnam() **Python3 中已删除。**为创
一个临时文件返回一个唯一的路径

59 os.ttyname(fd) 返回一个字符串，它表示与
件描述符fd 关联的终端设备。如果fd 没有与终端设备关联，则引发一个异常。

60 os.unlink(path) 删除文件路径

61 os.utime(path, times) 返回指定的path文件
访问和修改的时间。

62 [os.walk(top, topdown=True[, onerror=No
e[, followlinks=False]]]) 输出在文件夹中的文件名通过
树中游走，向上或者向下。

63 os.write(fd, str) 写入字符串到文件描述符 fd中
返回实际写入的字符串长度

64 os.path 模块 获取文件的属性信息。

Python3 错误和异常

```
while True:
    try:
        x = int(input("请输入一个数字: "))
        break
    except ValueError:
        print("您输入的不是数字，请再次尝试输入！")

except (RuntimeError, TypeError, NameError):
    pass

import sys

try:
    f = open('myfile.txt')
    s = f.readline()
    i = int(s.strip())
except OSError as err:
    print("OS error: {0}".format(err))
except ValueError:
    print("Could not convert data to an integer.")
except:
    print("Unexpected error:", sys.exc_info()[0])
    raise
```

try/except...else

```
for arg in sys.argv[1:]:
    try:
        f = open(arg, 'r')
    except IOError:
        print('cannot open', arg)
    else:
        print(arg, 'has', len(f.readlines()), 'lines')
        f.close()
```

try-finally 语句

finally 语句无论异常是否发生都会执行

```
try:
    runoob()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
finally:
    print('这句话，无论异常是否发生都会执行。')
```

抛出异常

Python 使用 raise 语句抛出一个指定的异常。

```
raise [Exception [, args [, traceback]]]
```

```
x = 10
if x > 5:
    raise Exception('x 不能大于 5。x 的值为: {}'.format(x))
```

预定义的清理行为

with 语句就可以保证诸如文件之类的对象在使用完之后一定会正确的执行他的清理方法:

```
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

以上这段代码执行完毕后，就算在处理过程中出问题了，文件 f 总是会关闭。

Python3 assert（断言）

assert（断言）用于判断一个表达式，在表达式条件为 false 的时候触发异常。

Python3 面向对象

类对象

```
class ClassName:
    <statement-1>
    .
    .
    .
    <statement-N>
```

属性引用和实例化

```
# 实例化类
x = MyClass()
```

类有一个名为 **init**() 的特殊方法（**构造方法**），该方法在类实例化时会自动调用

```
def __init__(self):
    self.data = []
```

init() 方法可以有参数，参数通过 **init**() 传递到类的实例化操作上。例如:

```
#!/usr/bin/python3

class Complex:
    def __init__(self, realpart, imagpart):
        self.r = realpart
```

```
        self.i = imagpart
x = Complex(3.0, -4.5)
print(x.r, x.i)   # 输出结果：3.0 -4.5
```

self代表类的实例，而非类

类的方法必须有一个额外的**第一个参数名称**, 按照惯例它的名称是 self。

```
class Test:
    def prt(self):
        print(self)
        print(self.__class__)

t = Test()
t.prt()
```

self 不是 python 关键字，我们把他换成 runoob 也是可以正常执行的:

```
class Test:
    def prt(runoob):
        print(runoob)
        print(runoob.__class__)

t = Test()
t.prt()
```

以上实例执行结果为：

```
<__main__.Test instance at 0x100771878>
__main__.Test
```

类的方法

类方法必须包含参数 self, 且为第一个参数

继承

```
#!/usr/bin/python3

#类定义
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性,私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法
    def __init__(self,n,a,w):
        self.name = n
        self.age = a
        self.__weight = w
    def speak(self):
        print("%s 说: 我 %d 岁。" %(self.name,self.age))
```

```
#单继承示例
class student(people):
    grade = ''
    def __init__(self,n,a,w,g):
        #调用父类的构函
        people.__init__(self,n,a,w)
        self.grade = g
    #覆写父类的方法
    def speak(self):
        print("%s 说: 我 %d 岁了，我在读 %d 年级"%(self.name,self.age,self.grade))


s = student('ken',10,60,3)
s.speak()
```

执行以上程序输出结果为：

```
ken 说: 我 10 岁了，我在读 3 年级
```

类属性与方法

类的私有属性

__private_attrs：两个下划线开头，声明该属性为私有。在类内部的方法中使用时 **self.__private_attrs。**

类的私有方法

__private_method：两个下划线开头，声明该方法为私有方法。**self.__private_methods。**

类的专有方法：

- **init :** 构造函数，在生成对象时调用
- **del :** 析构函数，释放对象时使用
- **repr :** 打印，转换
- **setitem :** 按照索引赋值
- **getitem:** 按照索引获取值
- **len:** 获得长度
- **cmp:** 比较运算
- **call:** 函数调用
- **add:** 加运算
- **sub:** 减运算
- **mul:** 乘运算
- **truediv:** 除运算

- **mod:** 求余运算
- **pow:** 乘方

运算符重载

```
#!/usr/bin/python3

class Vector:
   def __init__(self, a, b):
      self.a = a
      self.b = b

   def __str__(self):
      return 'Vector (%d, %d)' % (self.a, self.b)

   def __add__(self,other):
      return Vector(self.a + other.a, self.b + other.b)

v1 = Vector(2,10)
v2 = Vector(5,-2)
print (v1 + v2)
```

以上代码执行结果如下所示:

```
Vector(7,8)
```

Python3 命名空间和作用域

命名空间(Namespace)是从名称到对象的映射，大部分的命名空间都是通过 Python 字典来实现的。

内置作用域是通过一个名为 builtin 的标准模块来实现的，必须导入这个文件才能够使用它。

```
>>> import builtins
>>> dir(builtins)
```

global 和 nonlocal关键字

内部作用域 修改 **外部作用域** 的变量：

```
#!/usr/bin/python3

num = 1
def fun1():
    global num  # 需要使用 global 关键字声明
    print(num)
    num = 123
    print(num)
fun1()
print(num)
```

修改嵌套作用域中的变量：

```
#!/usr/bin/python3

def outer():
    num = 10
    def inner():
        nonlocal num   # nonlocal关键字声明
        num = 100
        print(num)
    inner()
    print(num)
outer()
```

Python3 标准库概览

操作系统接口

```
>>> import os
>>> os.getcwd()      # 返回当前的工作目录
'C:\\Python34'
>>> os.chdir('/server/accesslogs')   # 修改当前的工作目录
>>> os.system('mkdir today')   # 执行系统命令 mkdir
0
```

建议使用 "import os" 风格而非 "from os import *"

```
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's docstrings>
```

针对日常的文件和目录管理任务，:mod:shutil 模块提供了一个易于使用的高级接口:

```
>>> import shutil
>>> shutil.copyfile('data.db', 'archive.db')
>>> shutil.move('/build/executables', 'installdir')
```

文件通配符

glob模块提供了一个函数，用于从 **目录通配符搜索** 中生成 **文件列表**：

```
>>> import glob
>>> glob.glob('*.py')
['primes.py', 'random.py', 'quote.py']
```

命令行参数

通用工具脚本经常调用命令行参数。这些命令行参数以链表形式存储于 sys 模块的 argv 变量。例如命令行中执行 "python demo.py one two three" 后：

```
>>> import sys
>>> print(sys.argv)
```

```
['demo.py', 'one', 'two', 'three']
```

错误输出重定向和程序终止

sys 还有 stdin，stdout 和 stderr 属性，即使在 stdout 被重定向时，后者也可以用于显示警告和错信息。

```
>>> sys.stderr.write('Warning, log file not found starting a new one\n')
Warning, log file not found starting a new one
```

大多脚本的定向终止都使用 "sys.exit()"。

字符串正则匹配

re模块为高级字符串处理提供了正则表达式工具。

```
>>> import re
>>> re.findall(r'\bf[a-z]*', 'which foot or hand fell fastest')
['foot', 'fell', 'fastest']
>>> re.sub(r'(\b[a-z]+) \1', r'\1', 'cat in the the hat')
'cat in the hat'
```

如果只需要简单的功能，应该首先考虑字符串方法

```
>>> 'tea for too'.replace('too', 'two')
'tea for two'
```

数学

math模块为浮点运算提供了对底层C函数库的访问:

```
>>> import math
>>> math.cos(math.pi / 4)
0.70710678118654757
>>> math.log(1024, 2)
10.0
```

random提供了生成随机数的工具。

```
>>> import random
>>> random.choice(['apple', 'pear', 'banana'])
'apple'
>>> random.sample(range(100), 10)   # sampling without replacement
[30, 83, 16, 4, 8, 81, 41, 50, 18, 33]
>>> random.random()    # random float
0.17970987693706186
>>> random.randrange(6)    # random integer chosen from range(6)
4
```

访问 互联网

有几个模块用于访问互联网以及处理网络通信协议。其中最简单的两个是用于处理从 urls 接收的数据

urllib.request 以及用于发送电子邮件的 smtplib:

```
>>> from urllib.request import urlopen
>>> for line in urlopen('http://tycho.usno.navy.mil/cgi-bin/timer.pl'):
...     line = line.decode('utf-8')  # Decoding the binary data to text.
...     if 'EST' in line or 'EDT' in line:  # look for Eastern Time
...         print(line)

<BR>Nov. 25, 09:43:32 PM EST

>>> import smtplib
>>> server = smtplib.SMTP('localhost')
>>> server.sendmail('soothsayer@example.org', 'jcaesar@example.org',
... """To: jcaesar@example.org
... From: soothsayer@example.org
...
... Beware the Ides of March.
... """)
>>> server.quit()
```

注意第二个例子需要本地有一个在运行的邮件服务器。

日期和时间

datetime模块为日期和时间处理同时提供了简单和复杂的方法。

支持日期和时间算法的同时，实现的重点放在更有效的处理和格式化输出。

该模块还支持时区处理:

```
>>> # dates are easily constructed and formatted
>>> from datetime import date
>>> now = date.today()
>>> now
datetime.date(2003, 12, 2)
>>> now.strftime("%m-%d-%y. %d %b %Y is a %A on the %d day of %B.")
'12-02-03. 02 Dec 2003 is a Tuesday on the 02 day of December.'

>>> # dates support calendar arithmetic
>>> birthday = date(1964, 7, 31)
>>> age = now - birthday
>>> age.days
14368
```

数据压缩

以下模块直接支持通用的数据打包和压缩格式：zlib，gzip，bz2，zipfile，以及 tarfile

```
>>> import zlib
>>> s = b'witch which has which witches wrist watch'
>>> len(s)
41
>>> t = zlib.compress(s)
```

```
>>> len(t)
37
>>> zlib.decompress(t)
b'witch which has which witches wrist watch'
>>> zlib.crc32(s)
226805979
```

性能度量

Python 提供了一个度量工具

```
>>> from timeit import Timer
>>> Timer('t=a; a=b; b=t', 'a=1; b=2').timeit()
0.57535828626024577
>>> Timer('a,b = b,a', 'a=1; b=2').timeit()
0.54962537085770791
```

相对于 timeit 的细粒度，:mod:profile 和 pstats 模块提供了针对更大代码块的时间度量工具。

测试模块

开发高质量软件的方法之一是为每一个函数开发测试代码，并且在开发过程中经常进行测试

doctest模块提供了一个工具，扫描模块并根据程序中内嵌的文档字符串执行测试。

测试构造如同简单的将它的输出结果剪切并粘贴到文档字符串中。

通过用户提供的例子，它强化了文档，允许 doctest 模块确认代码的结果是否与文档一致:

```
def average(values):
    """Computes the arithmetic mean of a list of numbers.

    >>> print(average([20, 30, 70]))
    40.0
    """
    return sum(values) / len(values)

import doctest
doctest.testmod()   # 自动验证嵌入测试
```

unittest模块不像 doctest模块那么容易使用，不过它可以在一个独立的文件里提供一个更全面的测集:

```
import unittest

class TestStatisticalFunctions(unittest.TestCase):

    def test_average(self):
        self.assertEqual(average([20, 30, 70]), 40.0)
        self.assertEqual(round(average([1, 5, 7]), 1), 4.3)
        self.assertRaises(ZeroDivisionError, average, [])
        self.assertRaises(TypeError, average, 20, 30, 70)
```

```
unittest.main() # Calling from the command line invokes all tests
```

Python3 实例

以下实例在 Python3.4.3 版本下测试通过：

- Python Hello World 实例
- Python 数字求和
- Python 平方根
- Python 二次方程
- Python 计算三角形的面积
- Python 计算圆的面积
- Python 随机数生成
- Python 摄氏温度转华氏温度
- Python 交换变量
- Python if 语句
- Python 判断字符串是否为数字
- Python 判断奇数偶数
- Python 判断闰年
- Python 获取最大值函数
- Python 质数判断
- Python 输出指定范围内的素数
- Python 阶乘实例
- Python 九九乘法表
- Python 斐波那契数列
- Python 阿姆斯特朗数
- Python 十进制转二进制、八进制、十六进制
- Python ASCII码与字符相互转换
- Python 最大公约数算法
- Python 最小公倍数算法
- Python 简单计算器实现
- Python 生成日历
- Python 使用递归斐波那契数列
- Python 文件 IO
- Python 字符串判断
- Python 字符串大小写转换
- Python 计算每个月天数
- Python 获取昨天日期
- Python list 常用操作

- Python 约瑟夫生者死者小游戏
- Python 五人分鱼
- Python 实现秒表功能
- Python 计算 n 个自然数的立方和
- Python 计算数组元素之和
- Python 数组翻转指定个数的元素
- Python 将列表中的头尾两个元素对调
- Python 将列表中的指定位置的两个元素对调
- Python 翻转列表
- Python 判断元素是否在列表中存在
- Python 清空列表
- Python 复制列表
- Python 计算元素在列表中出现的次数
- Python 计算列表元素之和
- Python 计算列表元素之积
- Python 查找列表中最小元素
- Python 查找列表中最大元素
- Python 移除字符串中的指定位置字符
- Python 判断字符串是否存在子字符串
- Python 判断字符串长度
- Python 使用正则表达式提取字符串中的 URL
- Python 将字符串作为代码执行
- Python 字符串翻转
- Python 对字符串切片及翻转
- Python 按键(key)或值(value)对字典进行排序
- Python 计算字典值之和
- Python 移除字典点键值(key/value)对
- Python 合并字典
- Python 将字符串的时间转换为时间戳
- Python 获取几天前的时间
- Python 将时间戳转换为指定格式日期
- Python 打印自己设计的字体
- Python 二分查找
- Python 线性查找
- Python 插入排序
- Python 快速排序
- Python 选择排序

- Python 冒泡排序
- Python 归并排序
- Python 堆排序
- Python 计数排序
- Python 希尔排序
- Python 拓扑排序
-

库参考

numpy

scipy

scipy.io

能读取 wav 文件