

法之讲

XINGJIE  
DE MIMI

幸福

的秘密  
——法之讲庄子

科学出版社

# 目录

Introduction	1.1
python常用标准库概览	1.2
python迭代器	1.3
[python数据结构( <a href="#">python数据结构.md</a> )	1.3.1
python条件控制与循环	1.3.2
python异常	2.1
python输入与输出	2.1.1
python模块	2.1.2
python面向对象	2.2

# Introduction

除了这些选项外，**book.json**还提供更多选项,如:

```
"styles": {  
  "website": "styles/website.css",  
  "ebook": "styles/ebook.css",  
  "pdf": "styles/pdf.css",  
  "mobi": "styles/mobi.css",  
  "epub": "styles/epub.css"  
}
```

配置好**book.json**内的**plugins**选项后，要使用**gitbook install**安装插件

Gitbook默认带有5个插件:

- highlight
- search
- sharing
- font-settings
- livereload

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-06 22:12:42

[toc]

## OS

```
#os模块提供了不少与操作系统相关联的函数
>>> 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
-----
>>> import os
>>> dir(os)
<returns a list of all module functions>
>>> help(os)
<returns an extensive manual page created from the module's docstrings>

#在使用os这样的大型模块时内置的dir()和help()函数非常有用
```

## glob

```
#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']
```

## sys

```
#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
>>> import sys
>>> print(sys.argv)
['demo.py', 'one', 'two', 'three']
```

```
#错误输出重定向和程序终止
>>> sys.stderr.write('Warning, log file not found starting a new one\n')
Warning, log file not found starting a new one
#即使在stdout被重定向时,stderr也可以用于显示警告和错误信息
```

## re

```
#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'\b[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,因为它们非常简单
>>> 'tea for too'.replace('too', 'two')
'tea for two'
```

## math

```
#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
```

## zlib

```
>>> 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
```

## smtpplib

```
#用于发送电子邮件的smtpplib
>>> 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 smtpplib
>>> server = smtpplib.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()
```

## datetime

```
#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
>>> # 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
```

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-05 18:47:09

## 变量赋值

python中的变量不需要声明,变量的赋值即是变量声明和定义的过程

```
#-*- coding:UTF-8 -*-
#规定编码为utf-8编码,以显示中文
a = b = c = 1;
#fgg
"""
#三引号用于多行注释
js中也可以这样多变量赋值var a,b,c=1,
或var a=b=c=1,都是可以的,
但不能var a,b,c=1,2,3
"""
a, b, c = 1, 2.7, 'john';
#这与matlab中的[y1,y2,...] = deal(x1,x2,...)语句类似
#在python中,' ','"'都可用于表示字符串
```

## python数据类型

虽然python是一门弱类型语言,在赋值定义时变量是不考虑类型的,但同js等其它弱类型语言一样,它的数据在内存中存储也是有数据类型的区别的

**python3**中共有六个标准的数据类型:

- Number 数值
- String 字符串
- List 列表
- Tuple 元组
- Sets 集合
- Dictionary 字典

- 数值 Number

python支持四种不同的数值类型:

- int 有符号整型
- float 浮动型
- bool 布尔型
- complex 复数

```
a, b, c, d = 10, 5.5, True, 4+3i
print(type(a), type(b), type(c), type(d))
#与python2.x不同,python3.x中的print的参数需要加上括号
#注意:bool值是True和False

#输出:
#<class 'int'> <class 'float'> <class 'bool'> <class 'complex'>
```

```
#数值运算
5 + 4 #加法,9
4.3 - 2 #减法,2.3
3 * 7 #乘法,21
2 / 4 #除法得浮点数,0.5
2 // 4 #除法得整数,0
17 % 3 #取余,2
2 ** 5 #乘方(2^5),32
7.0 / 2 #不同类型的数混合运算时,会自动将整数转换为浮点数
-----
#命令行交互模式下,最后一次运算结果会被保存在内置变量 _ 中
>>>price = 100.50
>>>sum = price ** 2
10100.25
```

```
>>>10899.75 + _
11000
```

- 字符串 String

字符串是由数字,字母和下划线组成的一串字符.

\*注:没有单独的字符类型,长度为1的字符串即为一个字符

python的字符串数组有两者取值顺序:

- 从左往右索引默认从0(第一个字符)开始,最大长度是字符串长度len少1
- 从右往左索引默认从-1(最后一个字符)开始,最大长度为字符串长度

```
#####转义字符#####
str = 'Yes, he doesn\'t'
print(str,type(str),len(s)) #输出:Yes, he doesn't 14
print('C:Users\name')
#输出:C:\Users
#ame
print(r'C:\Users\name') #输出:C:\Users\name
#在字符串前加上r表示输出原始字符串,不进行转义
```

```
#####字符串输出#####
str = 'hello world!'
print(str) #输出完整字符串,等价于print(str[:])或print(str[:i]+str[i:])
print(str[0]) #输出第一个字符
print(str[2:5]) #输出第三个到第五个之间的字符(不包括2,包括3,5)
#冒号(切片)运算符:, 'abcd'中的'cd'可表示为[1]...,用切片表示为[1:]
print(str[2:]) #输出第二个字符以后字符串(不包括2)
print(str[-10:-6])
print(str * 2) #输出字符串两次
print(str + "TEST") #这里的+可以理解为js中的字符连接号+
print(str[2:100]) #返回2以后的子串
print(str[2:1]) #返回一个空字符串''
print(str[-100:]) #返回整个字符串
str = 'abc'+'123'
#两个紧邻的字面字符串会被自动串联成一个字符串
print(str)
```

```
#输出:
Hello World!
H
llo
llo World!
love
Hello World!Hello World!
Hello World!TEST
abc123
```

```
#####定义一个跨越多行的字符串#####
str1 = 'I am \na boy' #在字符串中嵌套'\n'
str2 = '''I am
a boy''' #用三引号的断行垂直连接作用
str3 = 'I am \ #用'\ '的断行水平拼接作用
a boy'
print(str1)
print(str2)
print(str3)
```



```
#输出:
I am
a boy
I am
a boy
I am a boy
```

与C等其他语言不同,python中的字符串是常量,即不可更改, `str[0]='f'` 是错误的(同 C++中的 *string* 类型一样,是全局内存区的常量)

- 列表List

python中的列表数据类型可以看做是matlab中的元胞数组(里面可以存储不同类型的值),只不过增加了许多新的特性

```
__ coding:UTF-8 __
#!/usr/bin/python

list = [ 'abcd', 786 , 2.23, 'john']
series = list + [True, 'steve'] #列表串联,+为列表连接运算符

print(list) # 输出完整列表
print(list[0]) # 输出列表的第一个元素,列表索引
print(list[1:3]) # 输出第二个至第三个的元素,列表切片
print(list[2:]) # 输出从第三个开始至列表末尾的所有元素
print(list) * 2 # *指重复操作
print(series)
series[0] = 'abc' #更改列表中的元素
print(series)
series.append('123') #用列表对象的append方法向列表末尾添加新项
newList = [series, ['abc', 123]] #嵌套列表
series[:] = [] #清除列表

#输出:
['abcd', 786, 2.23, 'john']
abcd
[786, 2.23]
[2.23, 'john']
['abcd', 786, 2.23, 'john', 'abcd', 786, 2.23, 'john']
['abcd', 786, 2.23, 'john', True, 'steve']
['abc', 786, 2.23, 'john', True, 'steve']
```

与python字符串不同的是,列表中的元素是可以改变的

- 元组 Tuple

Tuple和列表List基本等同,但它不能二次赋值,即它相当于只读的列表List

```
#coding:UTF-8
#!/usr/bin/python

tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
tuple[0] = 'abc' #修改元组元素的操作是非法的

print(tuple[0], tuple[2:5], type(tuple), len(tuple))
tuple = () #空元祖,删除元组

tup1 = (20, ) #只有一个元素的元组需要在后面加上一个逗号
tup2, tup3 = (1, 2, 3),(4, 5, 6)
print(tup1, tup2 + tup3) #元组也支持用+连接,即元组串联
#print(str1, str2, ...)会输出'str1str2...',即print的多个参数间可以自动连接成一个字符串
del tup1 #删除元组(值赋为空只能算是清空变量,只有用del才能从符号表中删除这个变量,使之重新变成未定义)
#可变元组(元组项中包含列表的元组)
itup = ('a', 'b', ['X', 'Y'])
L = itup[2]
L[0] = 'A'
```

*string, list*和*tuple*都属于序列(sequence)

- 集合 Sets

集合是一个无序不重复元素的集

```
Student = {'Tom', 'Jim', 'Mary', 'Tom', 'Rose'}
print(Student) #在集合中重复的元素会被自动滤掉
#输出: {'Jim', 'Jack', 'Mary', 'Rose'}

print('Rose' in Student) #membership testing成员测试
#输出: True

blank = set()
#set()函数用于创建集合, 它会自动滤掉重复值
#创建空集合不能用blank = {}, 只能用set()

#python集合可以进行数学上的集合运算
a = set('abracadabra')
b = set('alacazam')
print(a)
print(b)
print(a-b) #a,b的差集
print(a | b) #a,b的并集
print(a & b) #a,b的交集
print(a ^ b) #a,b的并集与a,b的交集的差集, 即a,b中不同时存在的元素

#输出:
{'r', 'a', 'c', 'd', 'b'}
{'m', 'a', 'z', 'c', 'l'}
{'r', 'd', 'b'}
{'r', 'z', 'a', 'm', 'd', 'b', 'c', 'l'}
{'a', 'c'}
{'r', 'z', 'm', 'd', 'b', 'l'}
```

- 字典 Dictionaries

字典是一种映射类型(mapping type), 它是一组无序的键值对集(有点类似json)

```
dic = {} #创建空字典
tel = {'Jack':1557, 'Tom':1320, 'Rose':1886}

tel['Jack'] #通过key查询
del tel['Rose'] #删除一个键值对
tel['Mary'] = 4127 #添加一个键值对
list(tel.keys()) #keys()为字典对象的内置函数, 返回所有由tel中的key组成的list
sorted(tel.keys()) #将字典del按key排序
'Tom' in tel #成员测试

#用dict()从序列(String, List, Tuple)创建字典
dict([('sape', 4139), ('guido', 4127), ('jack', 4098)])
#输出{'jack': 4098, 'sape': 4139, 'guido': 4127}

通过列表推导式来构建字典
{x: x*2 for x in (2,4,6)}
#in()运算符与sql中的in类似, 表示一个可选值的集合
#输出:{2: 4, 4: 16, 6: 36}

dict(sape=4139, guido=4127, jack=4098)
#输出:{'jack': 4098, 'sape': 4139, 'guido': 4127}

tel.clear()
```

```
del tel
#都可用于删除字典
```

字典的关键字`key`必须为不可变类型(即不能是`list`类型和包含可变类型的`tuple`),且不能重复,`{'abc':123,}`, `{123:456}`, `{('hbmy'):789}`都是可行的,但`{['name']:123}`就不行

## 迭代器

迭代器是一个可以记住遍历的位置的对象.迭代则可以作为访问集合元素的一些方式

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

#使用常规的for遍历迭代器对象
for x in it:
    print(x, end='')
#输出:1 2 3 4
#print()打印的是一行值,末尾会自动增加'\n',使用end=''表示不在末尾加'\n',即表示该行未结束

#使用next()遍历
import sys
while True:
    try:
        print(next(it))
    except StopIteration:
        #next(iterobj)一直迭代取出迭代器里的下一个元素,当取到最后一个值之后再打算取值(此时已无值可取)时,会抛出一个StopIteration异常,可使用类似函数定义的方法设置异常发生时的处理方式
        sys.exit()
#输出:
1
2
3
4
```



```
#!/usr/bin/python
#自定义迭代器(斐波那契数列)
#迭代器:任何实现了__iter__和__next__()方法的对象都是迭代器,__iter__返回迭代器自身,而__next__返回迭代器中的下一个值
from itertools import islice
class Fib:
    def __init__(self):
        self.prev = 0
        self.curr = 1

    def __iter__(self):
        return self

    def __next__(self):
        value = self.curr
        self.curr += self.prev
        self.prev = value
        return value

f = Fib()
print(list(islice(f, 0, 10)))

#输出:[1, 1, 2, 3, 5, 8, 13, 21, 34, 55]
```

## 生成器

生成器是一个返回迭代器的函数,只能用于迭代操作,更简单点理解生成器就是一个迭代器

```
import sys
from itertools import islice
def fibonacci(n):
    curr, prev, counter = 0, 1, 0
    while True:
        if (counter > n):
            return
        yield curr
        prev, curr = curr, curr + prev
#与一般函数不同,生成器函数返回返回值用的是yield而不是return
#在调用生成器运行的过程中,每次遇到yield时函数会暂停并保存当前所有的运行信息,返回yield的值,并在下一次执行next()方法时从当前位置继续运行
        counter++
f = fibonacci(10) # f 是一个迭代器,由生成器函数fibonacci返回yield生成
#f=fibonacci()返回的是一个生成器对象,此时函数体中的代码并不会执行,只有显示或隐式地调用next的时候才会真正执行里面的代码
while True:
    try:
        print (next(f), end=" ")
    except StopIteration:
        sys.exit()

#输出:0 1 1 2 3 5 8 13 21 34 55
```

容器是一系列元素的集合, `str`, `list`, `set`, `dict`, `file`, `sockets` 对象都可以看作是容器,容器都可以被迭代(用在 `for`, `while` 等语句中),因此他们也被称为可迭代对象

- 可迭代对象实现了 `__iter__` 方法, 该方法可返回一个迭代器对象.
- 迭代器持有一个内部状态的字段, 用于记录下次迭代返回值, 它实现了 `__next__` 和 `__iter__` 方法, 迭代器不会一次性把所有元素加载到内存, 而是需要的时候才生成返回结果
- 生成器是一种特殊的迭代器, 它的返回值不是通过 `return` 而是用 `yield`,

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook 该文件修订时间: 2018-12-06 22:27:03

## 条件控制

```
age = int(input("Age of the dog: "))
print()
#if和elif的冒号:后接的是条件满足时应执行的代码块(代码块不用大括号{}表示,而是以缩进表示,每个代码块前的缩进量应是一样的)
#python中没有switch语句
if age < 0:
    print("This can hardly be true!")
elif age == 1:
    print("about 14 human years")
elif age == 2:
    print("about 22 human years")
elif age > 2:
    human = 22 + (age - 2)*5

print("Human years: ", human)

input('press Return')
```

## 循环

- while循环

```
#语法:
while condition:
    <statements>
```

```
``` n = 100 sum = 0 counter = 1 while counter <= n: sum = sum + counter counter += 1
```

## 代码块与非代码块之间最好保持一个空行的间距

```
print("Sum of 1 until %d: %d" % (n,sum))
```

+ for循环

for in : else:

```
```python
lang = ['C', 'C++', 'Perl', 'Python']
for x in lang:
    if x == 'Python':
        print('I am learning python')
        print(x)
    else:
        print("Finally")
#当循环正常终止(不是break)时,会执行循环后附带的else语句
```

```
for i in range(5):
    print(i)
#range(n)或range(m,n),range(m,n,step)产生一个等差数组序列

a = ['Mary', 'had', 'little', 'lamb']
for i in range(len(a)):
    print(i,a[i])
#用len()遍历一个序列的索引
```

- pass

```
while Ture:
    pass
#pass相当于system('pause'),不进行任何操作
```

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-05 18:46:05

即便Python程序的语法是正确的,在运行它的时候,也有可能发生错误,运行期检测到的错误被称为异常。  
大多数的异常都不会被程序处理,都以错误信息的形式展现出来

```
except (RuntimeError, TypeError, NameError):  
    #一个except同时处理多个异常  
    pass  
#多个异常可以放在一个元组里
```

```
import sys  
  
try:  
    #一个try可以对应多个except子句  
    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:  
    #未被上述捕获的异常都会在这里被处理,即默认通用的异常处理方式  
    print("Unexpected error:", sys.exc_info()[0])  
    raise
```

异常**else**子句

```
import sys  
  
for arg in sys.argv[1:]:  
    try:  
        f = open(arg, 'r')  
    except IOError:  
        #捕获异常并处理  
        print('cannot open', arg)  
    else:  
        #当上述except所述所有异常都未发生时,执行该else语句  
        print(arg, 'has', len(f.readlines()), 'lines')  
        f.close()
```

抛出异常

```
>>> try:  
    raise NameError('HiThere')  
#raise(不是throw)唯一的一个参数指定了要被抛出的异常,它必须是一个异常类的实例或是异常类(也就是 Exception的子类)  
except NameError:  
    print('An exception flew by!')  
    raise  
#再次抛出异常  
  
An exception flew by!  
Traceback (most recent call last):  
  File "<stdin>", line 2, in ?  
NameError: HiThere
```

自定义异常

```
>>> class MyError(Exception):  
    def __init__(self, value):
```

```

        self.value = value
    def __str__(self):
        return repr(self.value)

>>> try:
    raise MyError(2*2)
except MyError as e:
    print('My exception occurred, value:', e.value)

My exception occurred, value: 4
>>> raise MyError('oops!')
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
__main__.MyError: 'oops!'
#在这个例子中,类Exception默认的 __init__()被覆盖

```

#当创建一个模块有可能抛出多种不同的异常时,一种通常的做法是为这个包建立一个基础异常类,然后基于这个基础类为不同的错误情况创建不同的子类

```

class Error(Exception):
    """Base class for exceptions in this module."""
    pass

class InputError(Error):
    """Exception raised for errors in the input.

    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """

    def __init__(self, expression, message):
        self.expression = expression
        self.message = message

class TransitionError(Error):
    """Raised when an operation attempts a state transition that's not
    allowed.

    Attributes:
        previous -- state at beginning of transition
        next -- attempted new state
        message -- explanation of why the specific transition is not allowed
    """

    def __init__(self, previous, next, message):
        self.previous = previous
        self.next = next
        self.message = message
#大多数的异常的名字都以"Error"结尾,就跟标准的异常命名一样

```

定义清理行为

```

>>> try:
    raise KeyboardInterrupt
    finally:
#无论在任何情况下都会执行finally所指的清理行为(不管try子句里面有没有发生异常,finally子句都会执行)
    print('Goodbye, world!')

Goodbye, world!

```



## KeyboardInterrupt

```
>>> def divide(x, y):
    try:
        result = x / y
    except ZeroDivisionError:
        print("division by zero!")
    else:
        print("result is", result)
    finally:
        print("executing finally clause")

>>> divide(2, 1)
result is 2.0
executing finally clause
>>> divide(2, 0)
division by zero!
executing finally clause
>>> divide("2", "1")
executing finally clause
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
  File "<stdin>", line 3, in divide
TypeError: unsupported operand type(s) for /: 'str' and 'str'
```

## 预定义的清理行为

一些对象定义了标准的清理行为,无论系统是否成功的使用了它,一旦不需要它了,那么这个标准的清理行为就会执行

```
#打开一个文件,然后把内容打印到屏幕上
for line in open("myfile.txt"):
    print(line, end="")
```

```
#就算处理过程中出问题了,文件流f也会正常关闭
with open("myfile.txt") as f:
    for line in f:
        print(line, end="")
```

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-05 18:47:22

str() 函数返回一个用户易读的表达式  
repr() 产生一个解释器易读的表达式

```
>>> s = 'Hello, world.'
>>> str(s)
'Hello, world.'
>>> repr(s)
"'Hello, world.'"
>>> str(1/7)
'0.14285714285714285'
>>> x = 10 * 3.25
>>> y = 200 * 200
>>> s = 'The value of x is ' + repr(x) + ', and y is ' + repr(y) + '...'
>>> print(s)
The value of x is 32.5, and y is 40000...
>>> # repr() 函数可以转义字符串中的特殊字符
... hello = 'hello, world\n'
>>> hellos = repr(hello)
>>> print(hellos)
'hello, world\n'
>>> # repr() 的参数可以是 Python 的任何对象
... repr((x, y, ('spam', 'eggs'))))
"(32.5, 40000, ('spam', 'eggs'))"
```

```
>>> for x in range(1, 11):
...     print(repr(x).rjust(2), repr(x*x).rjust(3), end=' ')
...     # 注意前一行 'end' 的使用
...     print(repr(x*x*x).rjust(4))
...
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

>>> 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
```

字符串对象的 rjust() 方法, 它可以将字符串靠右, 并在左边填充空格

还有类似的方法, 如 ljust() 和 center()

这些方法并不会写任何东西, 它们仅仅返回新的字符串 另一个方法 zfill(), 它会在数字的左边填充 0.

#zfill()使用

```
>>> '12'.zfill(5)
'00012'
>>> '-3.14'.zfill(7)
'-003.14'
>>> '3.14159265359'.zfill(5)
'3.14159265359'
```

#str.format() 基本使用方法

```
>>> print('We are the {} who say "{}!".format('knights', 'Ni'))
We are the knights who say "Ni!"
```

#括号及其里面的字符 (称作格式化字段) 将会被 format() 中的参数替换  
#在括号中的数字用于指向传入对象在 format() 中的位置顺序

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```

#如果在 format() 中使用了关键字参数, 那么它们的值会指向使用该名字的参数

```
>>> print('This {food} is {adjective}.'.format(
...     food='spam', adjective='absolutely horrible'))
This spam is absolutely horrible
```

#位置及关键字参数可以任意的结合

```
>>> print('The story of {0}, {1}, and {other}'.format('Bill', 'Manfred', other='Georg'))
The story of Bill, Manfred, and Georg.
```

#'!a' (使用 ascii()), '!s' (使用 str()) 和 '!r' (使用 repr()) 可以用于在格式化某个值之前对其进行转化:

```
>>> import math
>>> print('The value of PI is approximately {}'.format(math.pi))
The value of PI is approximately 3.14159265359.
>>> print('The value of PI is approximately {!r}'.format(math.pi))
The value of PI is approximately 3.141592653589793.
```

#可选项 ':' 和格式标识符可以跟着字段名。这就允许对值进行更好的格式化

#将 Pi 保留到小数点后三位:

```
>>> import math
>>> print('The value of PI is approximately {:.3f}'.format(math.pi))
The value of PI is approximately 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, 用来打印表格

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 7678}
>>> for name, phone in table.items():
...     print('{0:10} ==> {1:10d}'.format(name, phone))
...
Jack      ==>      4098
Dcab      ==>      7678
Sjoerd    ==>      4127
```

#如果你有一个很长的格式化字符串, 而你不想将它们分开, 那么在格式化时通过变量名而非位置会是很好的事情. 最简单的就是传入一个字典, 然后使用方括号 '[]' 来访问键值:

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {0[Jack]:d}; Sjoerd: {0[Sjoerd]:d}; '
...       'Dcab: {0[Dcab]:d}'.format(table))
```

```
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

#也可以通过在 table 变量前使用 '\*\*' 来实现相同的功能

```
>>> table = {'Sjoerd': 4127, 'Jack': 4098, 'Dcab': 8637678}
>>> print('Jack: {Jack:d}; Sjoerd: {Sjoerd:d}; Dcab: {Dcab:d}'.format(**table))
Jack: 4098; Sjoerd: 4127; Dcab: 8637678
```

旧式字符串格式化

```
>>> import math
>>> print('The value of PI is approximately %5.3f.' % math.pi)
The value of PI is approximately 3.142.
```

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-05 18:47:17

```
#fibonacci.py
#fibonacci模块(自定义模块)
# Fibonacci numbers module

def fib(n):
    # write Fibonacci series up to n
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
        print()
def fib2(n):
    # return Fibonacci series up to n
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

```
>>> import fibo
#导入整个模块,通过moduleName.moduleContent访问模块内的函数和变量
>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
>>> fibo.__name__
'fibo'

#将函数句柄引用给一般变量
>>> fib = fibo.fib
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

```
#从模块内导入特定的函数/变量
#只能使用特定的函数/变量
>>> from fibo import fib, fib2
>>> fib(500)
#直接使用模块内函数,不需要用moduleName.moduleContent
1 1 2 3 5 8 13 21 34 55 89 144 233 377

#导入全部内容
>>> from fibo import *
>>> fib(500)
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

## \_\_name\_\_

一个模块被另一个程序第一次引入时,其主程序将运行,如果我们想在模块被引入时,模块中的某一程序块不执行,我们可以用\_\_name\_\_属性来使该程序块仅在该模块自身运行时执行(模块自身作为主程序执行和模块被引用到另一个程序中被执行)

```
#!/usr/bin/python3
# Filename: using_name.py

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

## dir()函数

内置的函数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

```
>>> a = [1, 2, 3, 4, 5]
>>> import fibo
>>> fib = fibo.fib
>>> dir(fibo) ##得到一个当前模块fibo中定义的属性列表
['__name__', 'fib', 'fib2']
>>> dir()
#如果没有给定参数,那么dir() 函数会罗列出当前主程序中所定义的所有名称
['__builtins__', '__name__', 'a', 'fib', 'fibo', 'sys']
>>> a = 5 # 建立一个新的变量 'a'
>>> dir()
['__builtins__', '__doc__', '__name__', 'a', 'sys']
>>>
>>> del a # 删除变量名a
>>>
>>> dir()
['__builtins__', '__doc__', '__name__', 'sys']
>>>
```

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-05 18:46:37

## 类定义

```
#!/usr/bin/python3

#类定义
class people:
    #定义基本属性
    name = ''
    age = 0
    #定义私有属性(有下划线__),私有属性在类外部无法直接进行访问
    __weight = 0
    #定义构造方法(__init__())
    def __init__(self,n,a,w):
#与一般函数定义不同,类方法必须包含参数self,且为第一个参数
        self.name = n
        self.age = a
        self.__weight = w
    def speak(self):
        print("%s 说: 我 %d 岁。" %(self.name,self.age))

# 实例化类var = className()
p = people('W3Cschoo1',10,30)
p.speak()
#输出:W3Cschoo1 说: 我 10 岁。
```

## 继承

```
#单继承示例
#DerivedClassName(BaseClassNameList)
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()
#输出:ken 说:我10岁了,我在读 3 年级
```

```
#另一个类,多重继承之前的准备
class speaker():
    topic = ''
    name = ''
    def __init__(self,n,t):
        self.name = n
        self.topic = t
    def speak(self):
        print("我叫 %s, 我是一个演说家, 我演讲的主题是 %s"%(self.name,self.topic))

#多继承
class sample(speaker,student):
    a = ''
    def __init__(self,n,a,w,g,t):
```

```

        student.__init__(self,n,a,w,g)
        speaker.__init__(self,n,t)

test = sample("Tim",25,80,4,"Python")
test.speak()    #方法名同，默认调用的是在括号中排前地父类的方法
#输出:我叫Tim,我是一个演说家,我演讲的主题是 Python

```

## 方法重写

```

#!/usr/bin/python3

class Parent:        # 定义父类
    def myMethod(self):
        print ('调用父类方法')

class Child(Parent): #定义子类
    def myMethod(self):
        #在子类中重写从父类中继承的方法
        print ('调用子类方法')

c = Child() # 子类实例
c.myMethod() #子类调用重写方法

```

## 类的属性与方法

### 类的私有属性

`__private_attrs`: 两个下划线开头，声明该属性为私有，不能在类地外部被使用或直接访问。在类内部的方法中使用时`self.__private_attrs`

### 类的方法

在类地内部,使用`def`关键字可以为类定义一个方法,与一般函数定义不同,类方法必须包含参数`self`,且为第一个参数

### 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，不能在类地外部调用。在类的内部调用 `self.__private_methods`

```

#!/usr/bin/python3

class JustCounter:
    __secretCount = 0 # 私有变量
    publicCount = 0   # 公开变量

    def count(self):
        self.__secretCount += 1
        self.publicCount += 1
        print (self.__secretCount)

counter = JustCounter()
counter.count()
counter.count()
print (counter.publicCount)
print (counter.__secretCount) # 报错，实例不能访问私有变量

#输出:
1
2
2
Traceback (most recent call last):
  File "test.py", line 16, in <module>
    print (counter.__secretCount) #这里会报错,实例不能访问私有变量
AttributeError: 'JustCounter' object has no attribute '__secretCount'

```

### 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
- **cmp**:比较运算
- **call**:函数调用
- **add**: 加运算
- **sub**: 减运算
- **mul**: 乘运算
- **div**: 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)

#输出:Vector(7,8)
```

Copyright © baibainote.pro 2018 all right reserved, powered by Gitbook该文件修订时间: 2018-12-05 18:47:15