Chap04-内置函数和模块

Outline

- 4.1 内置函数
- 4.2 对象的删除
- 4.3 基本输入输出
- 4.4 模块的使用
- 4.5 Python代码规范
- 4.6 Python文件名

4.1 内置函数

- 内置函数不需要导入任何模块即可使用
- 执行下面的命令可以列出所有内置函数

```
>>> dir(__builtins__)
```

4.1 内置函数

函数	功能简要说明
abs(x)	返回数字x的绝对值
all(iterable)	如果对于可迭代对象中所有元素x都有bool(x)为True，则返回True。对于空的可迭代对象也返回True
any(iterable)	只要可迭代对象中存在元素x使得bool(x)为True，则返回True。对于空的可迭代对象，返回False
bin(x)	把数字x转换为二进制串
callable(object)	测试对象是否可调用。类和函数是可调用的，包含__call__()方法的类的对象也是可调用的
chr(x)	返回ASCII编码为x的字符
cmp(x, y)	比较大小，如果x<y则返回负数，如果x==y则返回0，如果x>y则返回正数。Python 3.x不再支持该函数
dir()	返回指定对象的成员列表
eval(s[, globals[, locals]])	计算字符串中表达式的值并返回
filter(function or None, sequence)	返回序列中使得函数值为True的那些元素，如果函数为None则返回那些值等价于True的元素。如果序列为元组或字符串则返回相同类型结果，其他则返回列表

4.1 内置函数

float(x)	把数字或字符串x转换为浮点数并返回
help(obj)	返回对象obj的帮助信息
hex(x)	把数字x转换为十六进制串
id(obj)	返回对象obj的标识（地址）
input([提示内容字符串])	接收键盘输入的内容，返回字符串。Python 2.x和Python 3.x对该函数的解释不完全一样，详见后面的1.4.8节
int(x[, d])	返回数字的整数部分，或把d进制的字符串x转换为十进制并返回，d默认为十进制
isinstance(object, class-or-type-or-tuple)	测试对象是否属于指定类型的实例
len(obj)	返回对象obj包含的元素个数，适用于列表、元组、集合、字典、字符串等类型的对象
list([x])、 set([x])、tuple([x])、dict([x])	把对象转换为列表、集合、元组或字典并返回，或生成空列表、空集合、空元组、空字典
map(函数,序列)	将函数映射至序列中每个元素，返回列表或map对象
max(x)、 min(x)、 sum(x)	返回序列中的最大值、最小值或数值元素之和

4.1 内置函数

open(name[, mode[, buffering]])	以指定模式打开文件并返回文件对象
ord(s)	返回1个字符s的编码
pow(x, y)	返回x的y次方，等价于x**y
range([start,] end [, step])	返回一个等差数列（Python 3.x中返回一个range对象），不包括终值
reduce(函数,序列)	将接收2个参数的函数以累积的方式从左到右依次应用至序列中每个元素，最终返回单个值作为结果
reversed(列表或元组)	返回逆序后的迭代器对象
round(x [, 小数位数])	对x进行四舍五入，若不指定小数位数，则返回整数
str(obj)	把对象obj转换为字符串
sorted(列表[, cmp[, key[reverse]]]	返回排序后的列表。Python 3.x中的sorted()方法没有cmp参数
type(obj)	返回对象obj的类型
zip(seq1 [, seq2 [...]])	返回[(seq1[0], seq2[0] ...), (...)]形式的列表

4.1 内置函数

- ord()和chr()是一对功能相反的函数，ord()用来返回单个字符的序数或Unicode码，而chr()则用来返回某序数对应的字符，str()则直接将其任意类型参数转换为字符串。

```
>>> ord('a')
97
>>> chr(65)
'A'
>>> chr(ord('A')+1)
'B'
>>> str(1)
'1'
>>> str(1234)
'1234'
>>> str([1,2,3])
'[1, 2, 3]'
>>> str((1,2,3))
'(1, 2, 3)'
>>> str({1,2,3})
'set([1, 2, 3])'
```

4.1 内置函数

- max()、min()、sum()这三个内置函数分别用于计算列表、元组或其他可迭代对象中所有元素最大值、最小值以及所有元素之和，sum()只支持数值型元素的序列或可迭代对象，max()和min()则要求序列或可迭代对象中的元素之间可比较大小。例如下面的示例代码，首先使用列表推导式生成包含10个随机数的列表，然后分别计算该列表的最大值、最小值和所有元素之和。

```
>>> import random
>>> a = [random.randint(1,100) for i in range(10)]
>>> a
[72, 26, 80, 65, 34, 86, 19, 74, 52, 40]
>>> print(max(a), min(a), sum(a))
86 19 548
```

- 如果需要计算该列表中的所有元素的平均值，可以直接使用下面的方法：

```
>>> a = [72, 26, 80, 65, 34, 86, 19, 74, 52, 40]
>>> sum(a)*1.0/len(a) #Python 2.7.11
54.8
>>> sum(a)/len(a) #Python 3.5.1
54.8
```

4.1 内置函数

- 使用dir()函数可以查看指定模块中包含的所有成员或者指定对象类型所支持的操作，而help()函数则返回指定模块或函数的说明文档。

4.1 数学库函数

函数	返回值（描述）
abs(x)	返回数字的绝对值，如abs(-10) 返回 10
ceil(x)	返回数字的上入整数，如math.ceil(4.1) 返回 5
cmp(x, y)	如果 x < y 返回 -1, 如果 x == y 返回 0, 如果 x > y 返回 1
exp(x)	返回e的x次幂(ex),如math.exp(1) 返回2.718281828459045
fabs(x)	返回数字的绝对值，如math.fabs(-10) 返回10.0
floor(x)	返回数字的下舍整数，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.log10(100)返回 2.0
max(x1, x2,...)	返回给定参数的最大值，参数可以为序列。
min(x1, x2,...)	返回给定参数的最小值，参数可以为序列。
modf(x)	返回x的整数部分与小数部分，两部分的数值符号与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n值，则代表舍入到小数点后的位数。
sqrt(x)	返回数字x的平方根，数字可以为负数，返回类型为实数，如math.sqrt(4)返回 2+0j

4.2 对象的删除

- 在Python中具有自动内存管理功能，Python解释器会跟踪所有的值，一旦发现某个值不再有任何变量指向，将会自动删除该值。尽管如此，自动内存管理或者垃圾回收机制并不能保证及时释放内存。显式释放自己申请的资源是程序员的好习惯之一，也是程序员素养的重要体现之一。
- 在Python中，可以使用del命令来显式删除对象并解除与值之间的指向关系。删除对象时，如果其指向的值还有别的变量指向则不删除该值，如果删除对象后该值不再有其他变量指向，则删除该值。

4.2 对象的删除

```
>>> x = [1,2,3,4,5,6]
>>> y = 3
>>> z = y
>>> print(y)
3
>>> del y #删除对象
>>> print(y)
NameError: name 'y' is not defined
>>> print(z)
3
>>> del z
>>> print(z)
NameError: name 'z' is not defined
>>> del x[1] #删除列表中指定元素
>>> print(x)
[1, 3, 4, 5, 6]
>>> del x #删除整个列表
>>> print(x)
NameError: name 'x' is not defined
```

4.2 对象的删除

- del命令无法删除元组或字符串中的指定元素，而只可以删除整个元组或字符串，因为这两者均属于不可变序列。

```
>>> x = (1,2,3)
>>> del x[1]
Traceback (most recent call last):
  File "<pyshell#62>", line 1, in <module>
    del x[1]
TypeError: 'tuple' object doesn't support item deletion
>>> del x
>>> print(x)
Traceback (most recent call last):
  File "<pyshell#64>", line 1, in <module>
    print(x)
NameError: name 'x' is not defined
```

4.3 基本输入输出

用Python进行程序设计，输入是通过input( )函数来实现的，imput( )的一般格式为：

x=input('提示：')

该函数返回输入的对象。可输入数字、字符串和其它任意类型对象。

4.3 基本输入输出

- 尽管形式一样，Python 2.x和Python 3.x对该函数的解释略有不同。在Python 2.x中，该函数返回结果的类型由输入值时所使用的界定符来决定，例如下面的Python 2.7.11代码：

```
>>> x = input("Please input:")
Please input:3 #没有界定符，整数
>>> print type(x)
<type 'int'>
>>> x = input("Please input:")
Please input:'3' #单引号，字符串
>>> print type(x)
<type 'str'>
>>> x = input("Please input:")
Please input:[1,2,3] #方括号，列表
>>> print type(x)
<type 'list'>
```

4.3 基本输入输出

- 在Python 2.x中，还有另外一个内置函数raw_input()也可以用来接收用户输入的值。与input()函数不同的是，raw_input()函数返回结果的类型一律为字符串，而不论用户使用什么界定符。例如：

```
>>> x = raw_input("Please input:")
Please input:[1,2,3]
>>> print type(x)
<type 'str'>
```

4.3 基本输入输出

- 在Python 3.x中，不存在raw_input()函数，只提供了input()函数用来接收用户的键盘输入。在Python 3.x中，不论用户输入数据时使用什么界定符，input()函数的返回结果都是字符串，需要将其转换为相应的类型再处理，相当于Python 2.x中的raw_input()函数。例如下面的Python 3.5.1代码：

```
>>> x = input('Please input:')
Please input:3
>>> print(type(x))
<class 'str'>
>>> x = input('Please input:')
Please input:'1'
>>> print(type(x))
<class 'str'>
>>> x = input('Please input:')
Please input:[1,2,3]
>>> print(type(x))
<class 'str'>
>>> x = raw_input('Please input:')
NameError: name 'raw_input' is not defined
```

4.3 基本输入输出

- Python 2.x和Python 3.x的输出方法也不完全一致。在Python 2.x中，使用print语句进行输出，而Python 3.x中使用print()函数进行输出。

4.3 基本输入输出

- 默认情况下，Python将结果输出到IDLE或者标准控制台，在输出时也可以进行重定向，例如可以把结果输出到指定文件。在Python 2.7.11中使用下面的方法进行输出重定向：

```
>>> fp = open(r'C:\mytest.txt', 'a+')
>>> print >>fp, "Hello,world"
>>> fp.close()
```

- 而在Python 3.5.1中则需要使用下面的方法进行重定向：

```
>>> fp = open(r'D:\mytest.txt', 'a+')
>>> print('Hello,world!', file = fp)
>>> fp.close()
```

4.3 基本输入输出

- 另外一个重要的不同是，对于Python 2.x而言，在print语句之后加上逗号“,”则表示输出内容之后不换行，例如：

```
>>> for i in range(10):
print i,
0 1 2 3 4 5 6 7 8 9
```

- 在Python 3.x中，为了实现上述功能则需要使用下面的方法：

```
>>> for i in range(10,20):
print(i, end=' ')
10 11 12 13 14 15 16 17 18 19
```

4.4 模块的使用

- Python默认安装仅包含部分基本或核心模块，但用户可以安装大量的扩展模块，pip是管理模块的重要工具。
- 在Python启动时，仅加载了很少的一部分模块，在需要时由程序员显式地加载（可能需要先安装）其他模块。
- 减小运行的压力，仅加载真正需要的模块和功能，且具有很强的可扩展性。
- 可以使用sys.modules.items()显示所有预加载模块的相关信息。

4.4 模块的使用

- import 模块名

```
>>>import math
>>>math.sin(0.5)              #求0.5的正弦
>>>import random
>>>x=random.random( )    #获得[0,1) 内的随机小数
>>>y=random.random( )
>>>n=random.randint(1,100) #获得[1,100]上的随机整数
```

- 可以使用dir函数查看任意模块中所有的对象列表，如果调用不带参数的dir()函数，则返回当前脚本的所有名字列表。
- 可以使用help函数查看任意模块或函数的使用帮助。

4.4 模块的使用

- from 模块名 import 对象名[as 别名] #可以减少查询次数，提高执行速度
- from math import *　　#谨慎使用

```
>>> from math import sin
>>> sin(3)
0.1411200080598672
>>> from math import sin as f #别名
>>> f(3)
0.141120008059867
```

4.4 模块的使用

- 在2.x中可以使用reload函数重新导入一个模块，在3.x中，需要使用imp模块的reload函数
- Python首先在当前目录中查找需要导入的模块文件，如果没有找到则从sys模块的path变量所指定的目录中查找。可以使用sys模块的path变量查看python导入模块时搜索模块的路径，也可以向其中append自定义的目录以扩展搜索路径。
- 在导入模块时，会优先导入相应的pyc文件，如果相应的pyc文件与py文件时间不相符，则导入py文件并重新编译该模块。

4.5 Python代码规范

（1）缩进

- 类定义、函数定义、选择结构、循环结构，行尾的冒号表示缩进的开始
- python程序是依靠代码块的缩进来体现代码之间的逻辑关系的，缩进结束就表示一个代码块结束了。
- 同一个级别的代码块的缩进量必须相同。
- 一般而言，以4个空格为基本缩进单位，可以通过下面的方法进行代码块的缩进和反缩进：

Fortmat➔Indent Region/Dedent Region

4.5 Python代码规范

（2）注释

一个好的、可读性强的程序一般包含30%以上的注释。常用的注释方式主要有两种：

- 以#开始，表示本行#之后的内容为注释
- 包含在一对三引号'''...'''或"""..."""之间且不属于任何语句的内容将被解释器认为是注释

在IDLE开发环境中，可以通过下面的操作快速注释/解除注释大段内容：

- Format➔Comment Out Region/Uncomment Region

4.5 Python代码规范

（3）每个import只导入一个模块

（4）如果一行语句太长，可以在行尾加上\来换行分成多行，但是更建议使用括号来包含多行内容。

（5）必要的空格与空行

- 运算符两侧、函数参数之间、逗号两侧建议使用空格分开。
- 不同功能的代码块之间、不同的函数定义之间建议增加一个空行以增加可读性。

（6）适当使用异常处理结构进行容错，后面将详细讲解。

（7）软件应具有较强的可测试性，测试与开发齐头并进。

4.6 Python文件名

- .py：Python源文件，由Python解释器负责解释执行。
- .pyw：Python源文件，常用于图形界面程序文件。
- .pyc：Python字节码文件，无法使用文本编辑器直接查看该类型文件内容，可用于隐藏Python源代码和提高运行速度。对于Python模块，第一次被导入时将被编译成字节码的形式，并在以后再次导入时优先使用“.pyc”文件，以提高模块的加载和运行速度。对于非模块文件，直接执行时并不生成“.pyc”文件，但可以使用py_compile模块的compile()函数进行编译以提高加载和运行速度。另外，Python还提供了compileall模块，其中包含compile_dir()、compile_file()和compile_path()等方法，用来支持批量Python源程序文件的编译。
- .pyo：优化的Python字节码文件，同样无法使用文本编辑器直接查看其内容。可以使用“python -O -m py_compile file.py”或“python -OO -m py_compile file.py”进行优化编译。
- .pyd：一般是由其他语言编写并编译的二进制文件，常用于实现某些软件工具的Python编程接口插件或Python动态链接库。

4.6 Python文件名

- 每个Python脚本在运行时都有一个“__name__”属性。如果脚本作为模块被导入，则其“__name__”属性的值被自动设置为模块名；如果脚本独立运行，则其“__name__”属性值被自动设置为“__main__”。例如，假设文件nametest.py中只包含下面一行代码：

```
print(__name__)
```

- 在IDLE中直接运行该程序时，或者在命令行提示符环境中运行该程序文件时，运行结果如下：

```
__main__
```

- 而将该文件作为模块导入时得到如下执行结果：

```
>>> import nametest
nametest
```

4.6 Python文件名

- 利用“__name__”属性即可控制Python程序的运行方式。例如，编写一个包含大量可被其他程序利用的函数的模块，而不希望该模块可以直接运行，则可以在程序文件中添加以下代码：

```
if __name__ == '__main__':
    print('Please use me as a module.')
```

- 这样一来，程序直接执行时将会得到提示“Please use me as a module.”，而使用import语句将其作为模块导入后可以使用其中的类、方法、常量或其他成员。

4.6 Python文件名

- 包可以看做处于同一目录中的模块。
- 在包的每个目录中都必须包含一个__init__.py文件，该文件可以是一个空文件，仅用于表示该目录是一个包。
- __init__.py文件的主要用途是设置__all__变量以及所包含的包初始化所需的代码。其中__all__变量中定义的对象可以在使用from …import *时全部正确导入。

例子 Python快速入门

- 问题1：用户输入一个三位自然数，计算并输出其佰位、十位和个位上的数字。

```
x = input('请输入一个三位数：')
x = int(x)
a = x//100
b = x//10%10
c = x%10
print(a, b, c)
```

例子 Python快速入门

- 问题2：已知三角形的两边长及其夹角，求第三边长。

```
import math
x = input('输入两边长及夹角（度）：')
a, b, theta = map(float, x.split())
c = math.sqrt(a**2 + b**2 - 2*a*b*math.cos(theta*math.pi/180))
print('c=', c)
```

例子 Python快速入门

- 问题3：任意输入三个英文单词，按字典顺序输出。

```
s = input('x,y,z=')
x, y, z = s.split(',')
if x > y:
    x, y = y, x
if x > z:
    x, z = z, x
if y > z:
    y, z = z, y
print(x, y, z)
```

- 或直接写为：

```
s = input('x,y,z=')
x, y, z = s.split(',')
x, y, z = sorted([x, y, z])
print(x, y, z)
```

例子 Python快速入门

例4：Python程序框架生成器。

```
import os
import sys
import datetime
head = '# '+'-'*20+'\n'+\
      '# Function description:\n'+\
      '# '+'-'*20+'\n'+\
      '# Author: Dong Fuguo\n'+\
      '# QQ: 306467355\n'+\
      '# Email: dongfuguo2005@126.com\n'+\
      '#'+'-'*20+'\n'
desFile = sys.argv[1]
if os.path.exists(desFile) or not desFile.endswith('.py'):
    print('%s already exist or is not a Python code file.!'%desFile)
    sys.exit()
fp = open(desFile, 'w')
today = str(datetime.date.today().year)+'-'+str(datetime.date.today().month)+\
       '-'+str(datetime.date.today().day)
fp.write('# -*- coding:utf-8 -*-\n')
fp.write('# Filename: '+desFile+'\n')
fp.write(head)
fp.write('# Date: '+today+'\n')
fp.write('# '+'-'*20+'\n')
fp.close()
```

Appendix The Zen of Python

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!