Python 速查表中文版

本手册是 Python cheat sheet 的中文翻译版。原作者：Arianne Colton and Sean Chen(data.scientist.info@gmail.com)

惯例

- Python 对大小写敏感；
- Python 的索引从 0 开始（所有编程语言均如此）；
- Python 使用空白符（制表符或空格）来缩进代码，而不是使用花括号。

获取帮助

- 获取主页帮助：`help()`
- 获取函数帮助：`help(str.replace)`
- 获取模块帮助：`help(re)`

模块（亦称库）

模块只是一个简单地以 `.py` 为后缀的文件。

- 列出模块内容：`dir(module1)`
- 导入模块：`import module`
- 调用模块中的函数：`module1.func1()`

注：**`import`** 语句会创建一个新的名字空间，并且在该名字空间内执行 **`.py`** 文件中的所有语句。如果你想把模块内容导入到当前名字空间，请使用 **`from module1 import *`** 语句。

数值类类型

查看变量的数据类型：`type(variable)`

六种经常使用的数据类型

1. **int/long**：过大的 `int` 类型会被自动转化为 `long` 类型。
2. **float**：64 位，Python 中没有 `double` 类型。
3. **bool**：真或假。
4. **str**：在 Python 2 中默认以 ASCII 编码，而在 Python 3 中默认以 Unicode 编码；
   - 字符串可置于单/双/三引号中；
   - 字符串是字符的序列，因此可以像处理其他序列一样处理字符串；
   - 特殊字符可通过 `\` 或者前缀 `r` 实现：

```
str1 = r'this\f?ff'
```

   - 字符串可通过多种方式格式化：

```
template = '%.2f %s haha $%d' ;
str1 = template % (4.88, 'hola', 2)
```

5. **NoneType(None)**：Python `null` 值（只有 None 对象的一个实例中存在）。
   - `None` 不是一个保留关键字，而是 **NoneType** 的一个唯一实例。

- `None` 通常是可选函数参数的默认值：

```
def func1(a, b, c = None)
```

- `None` 的常见用法：

```
if variable is None :
```

6. **datatime**：Python 内建的 datetime 模块提供了 `datetime`、`data` 以及 `time` 类型。
   - `datetime` 组合了存储于 `date` 和 `time` 中的信息。

```
#从字符串中创建 datetime
dt1 = datetime.strptime('20091031', '%Y%m%d')
#获取 date 对象
dt1.date()
#获取 time 对象
dt1.time()
#将 datetime 格式化为字符串
dt1.strftime('%m/%d/%Y%H:%M')
#更改字段值
dt2 = dt1.replace(minute = 0, second = 30)
#做差, diff 是一个 datetime.timedelta 对象
diff = dt1 - dt2
```

注：**Python** 中的绝大多数对象都是可变的，只有字符串和元组例外。

数据结构

注：所有的 **non-Get** 函数调用，比如下面例子中的 `list1.sort()` 都是原地操作，即不会创建新的对象，除非特别声明。

元组

元组是 Python 中任何类型的对象的一个一维、固定长度、不可变的序列。

```
#创建元组
tup1 = 4, 5, 6
# or
tup1 = (6, 7, 8)
#创建嵌套元组
tup1 = (4, 5, 6), (7, 8)
#将序列或迭代器转化为元组
tuple([1, 0, 2])
#连接元组
tup1 + tup2
#解包元组
a, b, c = tup1
```

元组应用：

```
#交换两个变量的值
a, b = b, a
```

列表

列表是 Python 中任何类型的对象的一个一维、非固定长度、可变（比如内容可以被修改）的序列。

```
#创建列表
list1 = [1, 'a', 3]
#or
list1 = list(tup1)
#连接列表
list1 + list2
#or
list1.extend(list2)
#追加到列表的末尾
list1.append('b')
#插入指定位置
list1.insert(PosIndex, 'a')
#反向插入，即弹出给定位置的值/删除
ValueAtIdx = list1.pop(PosIndex)
#移除列表中的第一个值, a 必须是列表中第一个值
list1.remove('a')
#检查成员资格
3 in list1 => True or False
#对列表进行排序
list1.sort()
#按特定方式排序
list1.sort(key = len) # 按长度排序
```

- 使用 + 连接列表会有比较大的开支，因为这个过程中会创建一个新的列表，然后复制对象。因此，使用 `extend()` 是更明智的选择；
- `insert` 和 `append` 相比会有更大的开支（时间/空间）；
- 在列表中检查是否包含一个值会比在字典和集合中慢很多，因为前者需要进行线性扫描，而后者是基于哈希表的，所以只需要花费常数时间。

内建的 `bisect` 模块

- 对一个排序好的列表进行二分查找或插入；
- `bisect.bisect` 找到元素在列表中的位置，`bisect.insort` 将元素插入到相应位置。用法：

```
import bisect
list1 = list(range(10))
#找到 5 在 list1 中的位置，从 1 开始，因此 position = index + 1
bisect.bisect(list1, 5)
#将 3.5 插入 list1 中合适位置
bisect.insort(list1, 3.5)
```

注：`bisect` 模块中的函数并不会去检查列表是否排序好，因为这会花费很多时间。所以，对未排序好的列表使用这些函数也不会报错，但可能会返回不正确的结果。

针对序列类型的切片

序列类型包括 `str`、`array`、`tuple`、`list` 等。

用法：

```
list1[start:stop]
#如果使用 step
list1(start:stop:step)
```

注：切片结果包含 **`start`** 索引，但不包含 **`stop`** 索引；**`start/stop`** 索引可以省略，如果省略，则默认为序列从开始到结束，如 **`list1 == list1[:]`** 。

`step` 的应用：

```
#取出奇数位置的元素
list1[::2]
#反转字符串
str1[::-1]
```

字典（哈希映射）

```
#创建字典
dict1 = {'key1': 'value1', 2: [3,2]}
#从序列创建字典
dict(zip(KeyList, ValueList))
#获取/设置/插入元素
dict1['key1']
dict1['key1'] = 'NewValue'
#get 提供默认值
dict1.get('key1', DefaultValue)
#检查键是否存在
'key1' in dict1
#获取键列表
dict1.keys()
#获取值列表
dict1.values()
#更新值
dict1.update(dict2)#dict1 的值被 dict2 替换
```

- 如果键不存在，则会出现 `KeyError Exception` 。
- 当键不存在时，如果 `get()` 不提供默认值则会返回 `None` 。
- 以相同的顺序返回键列表和值列表，但顺序不是特定的，又称极大可能非排序。

有效字典键类型

- 键必须是不可变的，比如标量类型(`int`、`float`、`string`)或者元组（元组中的所有对象也必须是不可变的）。
- 这儿涉及的技术术语是 `hashability`。可以用函数 `hash()` 来检查一个对象是否是可哈希的，比如 `hash('This is a string')` 会返回一个哈希值，而 `hash([1,2])` 则会报错（不可哈希）。

集合

- 一个集合是一些无序且唯一的元素的聚集；
- 你可以把它看成只有键的字典；

```
#创建集合
set([3, 6, 3])
#or
{3, 6, 3}
#子集测试
set1.issubset(set2)
#超集测试
set1.issuperset(set2)
#测试两个集合中的元素是否完全相同
set1 == set2
```

集合操作

- 并（又称或）：`set1 | set2`
- 交（又称与）：`set1 & set2`
- 差：`set1 - set2`
- 对称差（又称异或）：`set1 ^ set2`

函数

Python 的函数参数传递是通过引用传递。

- 基本形式

```
def func1(posArg1, keywordArg1 = 1, ..)
```

注

- 关键字参数必须跟在位置参数的后面；
- 默认情况下，Python 不会“延迟求值”，表达式的值会立刻求出来。

函数调用机制

- 所有函数均位于模块内部作用域。见“模块”部分。
- 在调用函数时，参数被打包成一个元组和一个字典，函数接收一个元组 `args` 和一个字典 `kwargs`，然后在函数内部解包。

“函数是对象”的常见用法：

```
def func1(ops = [str.strip, user_define_func , ..], ..):
  for function in ops:
    value = function(value)
```

返回值

- 如果函数末尾没有 `return` 语句，则不会返回任何东西。
- 如果有多个返回值则通过一个元组来实现。

```
return (value1, value2)
value1, value2 = func1(..)
```

匿名函数（又称 LAMBDA 函数）

- 什么是匿名函数？

匿名函数是一个只包含一条语句的简单函数。

```
lambda x : x * 2
#def func1(x) : return x * 2
```

- 匿名函数的应用：'curring'，又称利用已存在函数的部分参数来派生新的函数。

```
ma60 = lambda x : pd.rolling_mean(x, 60)
```

一些有用的函数（针对数据结构）

- `enumerate()` 返回一个序列`(i, value)`元组，`i` 是当前 `item` 的索引。

```
for i, value in enumerate(collection):
```

应用：创建一个序列中值与其在序列中的位置的字典映射（假设每一个值都是唯一的）。

- `sort()`可以从任意序列中返回一个排序好的序列。

```
sorted([2, 1, 3]) => [1, 2, 3]
```

应用：

```
sorted(set('abc bcd')) => [' ',
'a', 'b', 'c', 'd']
# 返回一个字符串排序后无重复的字母序列
```

- `zip()`函数可以把许多列表、元组或其他序列的元素配对起来创建一系列的元组。

```
zip(seq1, seq2) => [('seq1_1', 'seq2_1'), (..), ..]
```

1. `zip()`可以接收任意数量的序列作为参数，但是产生的元素的数目取决于最短的序列。

应用：多个序列同时迭代：

```
for i, (a, b) in enumerate(zip(seq1, seq2)):
```

2. `unzip`：另一种思考方式是把一些行转化为一些列：

```
seq1, seq2 = zip(zipOutput)
```

- `reversed()` 将一个序列的元素以逆序迭代。

```
list(reversed(range(10)))
```

`reversed()` 会返回一个迭代器，**`list()`** 使之成为一个列表。

控制流

- 用于 `if-else` 条件中的操作符：

```
#检查两个变量是否是相同的对象
var1 is var2
#检查两个变量是否是不同的对象
var1 is not var2
#检查两个变量的值是否相等
var1 == var2
```

注：**Python** 中使用 **`and`**、**`or`**、**`not`** 来组合条件，而不是使用 **`&&`**、**`||`**、**`!`**。

- `for`循环的常见用法：

```
#可迭代对象（list、tuple）或迭代器
for element in iterator:
#如果元素是可以解包的序列
for a, b, c in iterator:
```

- `pass`：无操作语句，在不需要进行任何操作的块中使用。
- 三元表达式，又称简洁的 `if-else`，基本形式：

```
value = true-expr if condition else false-expr
```

- Python 中没有 `switch/case` 语句，请使用 `if/elif`。

面向对象编程

- 对象（**object**）是 Python 中所有类型的根。
- 万物（数字、字符串、函数、类、模块等）皆为对象，每个对象均有一个类型（type）。对象变量是一个指向变量在内存中位置的指针。
- 所有对象均为引用计数。

```
sys.getrefcount(5) => x
a = 5, b = a
#上式会在等号的右边创建一个对象的引用，因此 a 和 b 均指向 5
sys.getrefcount(5)
=> x + 2
del(a); sys.getrefcount(5) => x + 1
```

- 类的基本形式：

```
class MyObject(object):
  # 'self' 等价于 Java/C++ 中的 'this'
  def __init__(self, name):
    self.name = name
  def memberFunc1(self, arg1):
      ..
  @staticmethod
  def classFunc2(arg1):
    ..
obj1 = MyObject('name1')
obj1.memberFunc1('a')
MyObject.classFunc2('b')
```

- 有用的交互式工具：

```
dir(variable1) #列出对象的所有可用方法
```

常见字符串操作

```
#通过分隔符连接列表/元组
', '.join([ 'v1', 'v2', 'v3']) => 'v1, v2, v3'

#格式化字符串
string1 = 'My name is {0}  {name}'
newString1 = string1.format('Sean', name =  'Chen')

#分裂字符串
sep = '-';
stringList1 =  string1.split(sep)

#获取子串
start = 1;
string1[start:8]

#补 '0' 向右对齐字符串
month = '5';
month.zfill(2) => '05'
month = '12';
month.zfill(2) => '12'
month.zfill(3) => '012'
```

异常处理

- 基本形式：

```
try:
  ..
except ValueError as e:
  print e
except (TypeError, AnotherError):
  ..
except:
  ..
finally:
  .. # 清理，比如 close db;
```

- 手动引发异常：

```
raise AssertionError  # 断言失败
raise SystemExit
# 请求程序退出
raise RuntimeError('错误信息 :..')
```

对列表、字典和元组的深入理解

语法糖（syntactic sugar）会使代码变得更加易读易写。

对列表的理解

将一些元素通过一个简短的语句传入一个过滤器进行过滤和转化，然后可以组成一个新的列表。

```
#基本形式
[expr for val in collection if condition]
#ShortCut
result = []
for val in collection:
  if condition:
    result.append(expr)
```

可以省略过滤条件，只留下表达式。

对字典的理解

基本形式：

```
{key-expr : value-expr for value in collection if condition}
```

对集合的理解

基本形式：和列表一样，只是应该使用 `()` 而不是 `[]` 。

嵌套列表

基本形式：

```
[expr for val in collection for innerVal in val if condition]
```