# Python笔记v1.0

|  |  |  |
| --- | --- | --- |
| 版本号 | 修改日期 | 修改内容 |
| 1.0 | 2018-01-28 | 增加Python基础知识1-5小节 |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 一、Python基础知识

### 1.变量

**理解**：存储了数据的内存空间，变量类型由数据类型决定；

**举例**：num = 10; str = ’string’; lis = [1,2,3];

局部变量：定义在函数内部或程序特定部分的变量；

全局变量：定义在程序顶部，全程序可调用的变量，global x强制声明为全局变量；

**备注**：其中变量名只是对数据所在空间的索引，由字母、下划线、数字组成；

**深入**：[垃圾回收机制](#_1.垃圾回收机制)；[内存泄漏](#_2.内存泄漏)；

### 2.运算符

理解：对数据进行操作的标识；

举例：x + y; x & y; x << 2;

优先级：or < and < not < in/not in < is/is not < 比较 < | < < & < <</>> < +/- < \*, /, % < 正负号 < 位反 < \*\*

**备注**：is/is not和 in/not in用于判断；

### 3.数据类型

1. 数字

* int

理解：范围限制在 -2,147,483,648 到 2,147,483,647 内的整数

举例：10; -99; 3480;

* long

理解：范围无限制的整数

举例：10L; -99L; 3480L;

* float

理解：带小数点的数

举例：1.25; -3.9; -978.7;

* complex

理解：复数（虚数）

举例：4+0.5j; -3j; 4j;

1. 序列

* string

理解：用 ’ ’/ “ ”/ “”” “”” 包含在内的一连串字符；

举例：’hello’; “world”; “””I am fine”””;

* list

理解：可包含多种数据类型内容有序可变的容器；

举例：[1,2,3]; [‘a’, ’b’, ’c’]; [‘a’,12,[1,2,3]];

列表表达式：[ generate\_element\_expr for x in list if expr ]

生成器表达式：( generate\_element\_expr for x in list if expr )

深入：[迭代对象、迭代器、生成器、生成器函数](#_3._迭代对象、迭代器、生成器、生成器函数)；

* tuple

理解：可包含多种数据类型内容有序不可变的容器；

举例：(1,); (‘a’ , ’b’); (1,2,’a’,[‘x’,’y’]);

* 操作

1. 切片

理解：选取序列其中一部分，[startIndex : destIndex+1 : step]；

举例：str = ‘abcde’

str[:] 🡪 ‘abcde’ #

str[::2] 🡪 ‘ace’

str[-5:-2:-1] 🡪 ‘dcb’

**备注**：索引从左到右：0,1,2,3…；从右到左：-1,-2,-3,-4…；

step为正时从左到右取序列元素，step为负时从右到左取序列元素；

1. 内置函数

理解：常用的序列处理函数

举例：+ \* in len() max() min() cmp() filter() zip() map() reduce()

**备注**：is 是否是同一个对象

== 值是否相等

cmp(x,y) x>y 🡪 1; x<y 🡪 -1; x==y 🡪 0;

1. 字典

理解：可包含多种数据类型的“键：值”对的内容无序可变（键值不可变）的容器；

举例：{ ‘name’:’John’, ‘age’:30, ‘tel’:138 }

深入：[list、tuple、dict实现](#_3.list、tuple、dict实现)；

### 4.结构

1. if else

理解：判断分支结构；

举例：if expr:

Statements

elif:

Statements

else:

Statements

**备注**：用缩进代替语句分组；

1. for

理解：次数既定的循环结构；

举例：for x in range(10):

print x

else:

print ‘okay’

**备注**：正常执行完循环后执行else，可将for和else看作一个整体，break将跳出这个整体；

1. while

理解：次数不定的循环结构

举例：while expr:

Statements

else:

Statements

**备注**：expr为假时执行else

1. switch

理解：用字典实现的类if else 多分支判断结构；

举例：dict={ “+”:add, “-”:sub, “\*”:mul, “/”:div } #add/sub/mul/div为函数名

dict.get(operator)(x,y) #dict[“+”]=add

### 5.函数

理解：实现特定功能的程序块；

举例：def fun ( par1, par2 ) # par1=??? 定义默认形参，默认形参置于最右边，

Print “ %s --- %s ” % ( par1, par2 )

#回车结束函数，默认return None

**备注**：fun( \*tuple ) 声明传递tuple（list直接传递），形参按tuple元素对应一一取值；

fun( \*\*dict ) 声明传递dict，形参和字典中的key一一对应，取key对应的value；

fun( x, \*tuple, \*\*dict ) 可吸收多余参数；

* lambda函数

理解：对结构简洁的函数用一句代码总结；

举例：lambda par1, par2 : par1+par2 #冒号‘：’后为返回值

* 内置函数

理解：python已经定义好的函数，可直接调用；

举例：help() callable() isinstance() range()

## 二、Python实践运用

## 三、Python深入了解

### 1.垃圾回收机制

python采用的是引用计数机制为主，标记-清除和分代收集两种机制为辅的垃圾回收策略（引用计数不能处理含有循环引用的数据结构）。

* 引用计数

创建一个对象时，Python总是在对象的C结构体里保存一个整数，称为引用数。期初，Python将这个值设置为1。每当对象的引用数减为0，Python立即将其释放，把内存还给操作系统。

* 标记-清除

标记-清除机制，顾名思义，首先标记对象（垃圾检测），然后清除垃圾（垃圾回收）。首先初始所有对象标记为白色，并确定根节点对象，标记它们为黑色。将根节点引用的对象标记为灰色，检查完灰色对象引用的对象后，将灰色标记为黑色。重复直到不存在灰色节点为止，最后白色结点都是需要清除的对象。

* 分代收集

Python使用一种不同的链表来持续追踪活跃的对象，Python的内部C代码将其称为零代(Generation Zero)。每次当你创建一个对象或其他什么值的时候，Python会将其加入零代链表，零代链表是一个完全内部的Python运行时（？？？）。随后，Python会循环遍历零代链表上的每个对象，检查列表中每个互相引用的对象，根据规则减掉其引用计数。当对象的引用计数变为零时，收集器可以释放它们并回收内存空间了，剩下的活跃的对象则被移动到一个新的链表：一代链表。通过不同的GC阈值设置，Python可以在不同的时间间隔处理这些对象。Python处理零代最为频繁，其次是一代然后才是二代。

**参考**：

[1] Python垃圾回收机制——完美讲解！<https://www.cnblogs.com/pinganzi/p/6646742.html>

[2] python垃圾回收机 <https://www.cnblogs.com/mingaixin/archive/2013/01/31/2886680.html>

### 2.内存泄露

如果没有禁用垃圾回收，那么Python中的内存泄露有两种情况：要么是对象被生命周期更长的对象所引用，比如global作用域对象；要么是循环引用中的对象定义了\_\_del\_\_函数。当一个对象理论上（或者逻辑上）不再被使用了，但事实上没有被释放，那么就存在内存泄露；当一个对象事实上已经不可达，即不能通过任何变量找到这个对象，但这个对象没有立即被释放，那么则可能存在循环引用。

参考：

[1] 使用 GC、Objgraph 干掉 Python 内存泄露与循环引用！<http://python.jobbole.com/88827/>

### 3.list、tuple、dict实现

List、tuple的实质类似于指针数组，每个元素分别指向实际所存储的数据内存空间，因此可包含多种数据类型。Dict是通过哈希表实现的，也可视作彼此关联的两个数组，通过哈希函数处理key值后可寻址到value。

参考：

[1] Python中list的实现<https://www.jianshu.com/p/J4U6rR>

[2] 深入Python字典的内部实现<http://python.jobbole.com/85040/>

### 4.迭代对象、迭代器、生成器、生成器函数