1. Range(X,Y) 生成的序列是 包括前面X,不包括后面Y
2. List[X:Y] 列表切割是 包括前面X,不包括后面Y. 且List[X:] 包括X直接切割到列表尾部，包括列表最后一个元素。
3. List[X:Y:Z] 列表从X到Y,但不包含Y，且Z为步长。

默认情况下不指定为Z=1.

如果Z为负数，则可以理解成开头处和结尾处互换，然后步长为Z的绝对值。所以字符串[::-1]为该字符串倒序。

Range(x,y,z) 从x到y,但不含y,且z为步长，默认情况下不指定为z=1

1. 为多个变量同时申明与赋值(例如a,b,c = [1,2,3],”GG”,{1:2,3:4})
2. Python申明多行字符串，以’’’申明结束

Message = ‘’’

Info …..

Location …

‘’’’

1. Python申明常量全部大写字母，但只是表示显示常量不推荐更改，但是python允许所有变量都可以更改不会报错
2. PYC文件的生成。首先我们编程用到的语言都是高级语言(C, JAVA,RUBY, PYTHON),而机器在执行代码中只认识机器语言。所以存在把高级语言翻译成机器语言的过程。这个过程可以大概分为编译型与解释型两种方法。

编译型（C例如）是在程序执行前，就把代码编译成机器码，然后机器去执行该对应机器码。再执行中无需进行任何编译。

解释型（Pyhon例如），是在程序执行中，把代码一条条翻译成机器码执行。在执行过程中再翻译成机器码。并且由于采用虚拟机技术，并不是一步直接翻译成机器码。细节是上，首先把代码编译成PyCodeObject，然后再交由Python虚拟机再翻译成机器码。并且Python为了优化时间，PycodeObject再执行完以后会以PYC文件保存，从而下一次执行文件便不需要再编译。

----Python只为引入的包所对应的.py文件进行编译，执行的主文件不会编译。

----当.py文件修改后，Python会有自己的检测重新编译。

----手动删除pyc文件对程序并不影响，再次执行Python只会为其再次编译。

1. Python3中没有Long 与Int整数类型之分，不管多大都是Int. Python2中才有Long与Int,并且Python根据数值他将自己转化。
2. Python3中列表如果有数字和字符串，不能直接排序。 Python2中可以。
3. id(obj) 查看obj变量的内存地址
4. isdigit(var)函数可以用于判断将变量是否可以转为数字，而无需int()时候去catch这个错误
5. 当我们需要一个标志位推出死循环的逻辑时可以：

While flag:

……

Flag = False

这样的话如果标志位变化则退出循环。而不需要while 1, 然后在循环内部进行break语句。

1. Enumerate(obj)返回一个元组，第一个是index,第二个元素是里面的元素。用途在于产生index以及对应的元素
2. Python 的API说明中，中括号表示可以选参数
3. Json中使用双引号，不使用单引号。数字可以不用引号。
4. Linux下ctrl + C结束python程序并且进程消失。而ctrl + Z 是挂起python程序。进程依然存在。
5. 带小数点的字符串可以直接通过float()转型。

带小数点的字符串不可以直接通过int()转型。

不带小数点的字符串可以直接通过int()转型。

1. Python中的int, string, dict, list,set 实际上是对应与Python中的int, str, dict, list, set的类的实例。所以每一个实例中的方法都定义保存在class中，而不需要每个实例都将方法保存到内存中。
2. Python中的dict(), list(), set()方法可以用于创建字典，列表以及集合。他们将调用对应类中的\_\_init\_\_方法，并且返回对应的数据类型。而如果我们以example = {}, example = [x,x,x..], example = {x,x,x,x,} 这种方式创建的时候，实际上Python还是调用dict(), list(), set()方法去创建，即是\_\_init\_\_方法创建。
3. Python的Set集合是无序不重复的列表。可以用于执行交集，差集，并集等集合操作。
4. Python必须先定义方法，然后才可以调用方法。这是由于Python的执行顺序从上到下决定的。而JAVA, C#是整体预加载然后执行的，所以不需要先定义再运行。
5. 定义函数的参数叫做形式参数（[formal parameter](http://www.baidu.com/link?url=QbY23OT87imXDzEt4t7GoTp4ffbn1xSxpGLWKoDMtp-X8f4TOmGh3A6y0Xbs-qhJase6_5z4Y-nwxBcL_hKtobA9QJr5ucXV_pF6lT5GKNF6R81Am-eWHN86PjXJ_LRN)），调用函数时传递的参数叫做实参（actual parameter）
6. Python解释器自动带有垃圾回收机制，定时的把没有引用的内存空间回收。
7. 虽然数字，字符串，字典，列表在Python中都是对应类的实例，在传递参数的时候，Python会对传递参数的不同而分别采取传递值或者传递引用的方式。

def Test(a):

a.append(“G”)

test\_list = [1,2,3]

Test(test\_list) # 执行完后test\_list值为[1,2,3,”G”]，Python将list的引用传入的函数当中。Dict也是这样的情况。

def Test(a):

a = 3

b = 4

Test(b) # 执行完后b值为4，Python将数字，字符串传入函数中，完全拷贝一份传入。

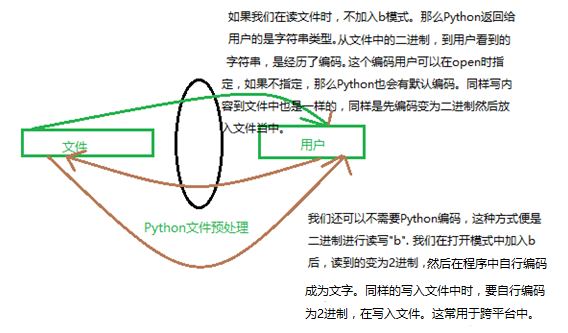
1. Python定义函数时，开头使用”””与”””进行块注释，说明程序的用途，传入的参数意义，以及返回的值
2. Python的lamda表达式，lamda表达式的意义在与定义一个含有简单逻辑的函数，并且返回。

返回函数名称 = lambda 需要传入参数（可以多个）: 操作 (默认函数将返回该值，并且只能有一句话)

Example:

F1 = lambda a: a+100 # 调用时F1(100)返回200

1. 在utf-8中，一个汉字占有3个字节，每个字节转为8位的二进制。
2. Python读取文件时对文件内容的编码预处理



Note: Python的二进制表达是bytes, bytes是用16进制表达的2进制串，为了表达简洁。

从bytes转为字符串, str(bytesObj, encoding = “Encoding”)

从字符串转为bytes, bytes(str Obj, encoding = “Encoding”)

1. 对文件句柄进行循环(即是for line in File\_Handle)则会将文件内容在每次循环中逐次读出。
2. Pycharm中Ctrl + 鼠标单击进行函数定义体

Pycharm中 Atl + Enter忽略这个PEP8的格式警告

1. Python函数如果不手动指定任何东西，那么Python函数默认返回None.
2. 将数组中的元素连接起来并且输出，便捷的方法””.join(list)
3. len()内置函数，在Python3中是默认以字符作为长度。在Python2中默认以字节作为长度。
4. Python字符串的strip()函数，会去除开头结尾处的回车还行以及空格。但是中间的不去除。
5. 字符串转为列表,直接通过list转换不行，需要先eval再list转换。即list(eval(str))

Example:

test = “[1,2,3,4]”

test = list(eval(test)) # test = [1,2,3,4]

1. Python序列化：将数据类型（包括基本数据类型，复杂数据类型）变为字符串。

Python反序列化：将字符串读入程序中，形成数据类型。

Python主要的序列化与反序列化为JSON以及 PICKLE.

JSON可以转换普通数据类型，包括数字，字符串，列表字典。无法转化对象等复杂数据对象。转出来的字符串可以跨平台使用。

PICKLE模块可以转换所有数据类型，但是无法跨平台使用，只能Python自己读取。

1. Python在路径中找寻引入的模块时候，当找到的时候便不再寻找，而不是继续找进行覆盖行为。
2. Python的模块实际上对应的是一个Python文件。

\_\_init\_\_.py文件主要有2个作用，第一个是将文件夹变为Python的一个包，使得可以批量的引入包内的模块，而不需要每一个都执行import语句逐一引入。第二个重要的原因是在当引入这个包时，实际上执行的是\_\_init\_\_.py文件。\_\_init\_\_.py中可以定义当\*时，具体引入此包下的什么模块（如果为空，则什么都不引入）， 可以将一些深层次的包在\_\_init\_\_.py中引入，使得引入这个包的程序可以以同一名称使用。

1. Python的Logging模块，是Python内置的日志模块。可是向日志中输出，时间，多少行引起，什么函数引起。从而不需要自己手写。但是需要配置。Logging模块配置的过程分为,logger, handler, formatter.

流程为：建立logging的一个实例，然后我们将handler注册到这个实例当中，handler表示输出到屏幕或者文件的渠道。然后建立formatter将输出的格式再加入到handler当中。当有日志信息需要输出的时候，logging实例将通过各个handler，以及handler上面的格式输出日志。

在日志中信息由4种等级, DEBUG - > INFO -> ERROR -> CRITICAL.

P.S 在handler中可以各自设置输出日志的最低等级，全局logger实例也可以设置。全局充当的作用是全局过滤，然后交给handler中的等级再过滤。（所以如果全局日志等级中设为CRITICAL,各个handler中设为INFO,也不会看到DEBUG, INFO, ERROR的信息）

40. OS模块主要函数

|  |  |
| --- | --- |
| os.path.abspath(filename) | 返回文件的绝对路径 |
| os.path.dirname(filename) | 返回路径的上一层目录路径 |
| os.path.join(path,filename) | 返回拼接完的路径 |
| os.getcwd() | 返回当前的工作目录 |
| os.listdir([path]) | 返回当前目录的子目录以及文件 |
| os.chdir(path), os.mkdir([path]), os.rmdir([path]), | 改变工作目录, 创建文件夹，删除文件夹 |
| isfile([path]),isdir([path]), exist([path]) | 判断是否为文件，是否为文件夹，路径所指是否存在 |

1. Python字符串有两种格式化方法，一种是百分号%，一种是使用format函数。

百分号语法：

print (“Output string is %s, number is %d, float is %.2f” % (variable1, variable2, variable3))

Format语法：

可以通过位置参数来使用format语法，例如：

"{0} love {1}{2}".format("I", "ice","cream")

也可以通过命名参数来使用format语法，例如：

"{a} love {b}{c}".format(a="I", b="ice",c="cream")

还可以通过位置参数和命名参数混用的方式来使用format语法，但是位置参数必须在命名参数之前：

"{0} love {b}{c}".format("I", b="ice",c="cream")

1. Python使用hashlib模块进行加密，里面含有SHA, MD5算法。过程为，先建立hashlib的md5对象，可选设置密钥，然后对传入的字符串进行加密。MD5为单向加密，从明文可以转为密文，但是从密文无法转为明文。通过MD5算法，相同的密钥，相同的明文，所得出的密文相同。所以密码比较时，直接比较密文。

import hashlib

hash\_obj = hashlib.md5(bytes(“密钥”,encoding=”utf-8”))

hash\_obj.update(bytes(“需要加密字符串”,encoding=”utf-8”))

hash\_obj.hexdigest() # 生成以16进制表示的密文

1. 有时候在 Windows下面使用Notepad++等文本编辑器编辑Python文件以后，上传到Linux服务器中执行会报错: python^M: bad interpreter: No such file or directory

原因在于不同系统编码方式引起的。

解决方法是在Linux服务器上执行 sed -i 's/^M//g'  filename ，意思是去掉filename中的每一行中的^M。（注意^M 在linux 下写法 按^M 是回车换行符,输入方法是按住CTRL+v,松开v,按m，而不是直接打字形成^再M)

1. Python的virtualenv是python自带的虚拟环境，这个虚拟环境在于当使用pip的时候，安装的包都局限在这个虚拟环境当中不会影响外部的Python安装包。另外virtualenvwrapper是virtualenv辅助包，更加方便。可以统一虚拟环境存储地址，自动结束虚拟环境等。
2. Python判断变量是否为None, 是 变量 is None判断，而不是变量 == None
3. Python的异常捕获语句，try except else finally

当有异常的时候执行except语句

当没有异常的时候执行else语句

无论是否存在异常执行finally语句

🡪

Python区别大小写。逻辑真为True,逻辑假为False

🡪

Python多线程和多进程

对于操作系统来说，一个任务就是一个进程，例如打开浏览器，打开计算器，打开记事本等等。同一种任务打开多个，分别属于不同进程。对于一个任务来说，它有很多子任务，例如播放器，既要解码视频、也要解码音频，所以在进程下存在多线程。在一个进程下至少存在一个线程，可以称它为主线程。操作系统创建进程时，会单独为每一个进程分配各自的资源，进程与进程之间相互隔离。而进程内的线程，则共享了当前进程内的资源。操作系统可以调度的最小系统粒度为线程，而分配资源的最小粒度是进程。我们的多任务操作系统，在单核CPU上是在各个线程上不断切换而达到目的（并发），而在多核CPU上则能同时执行多个线程任务（并行）。

并发（concurrency）和并行（parallellism）的区别：

并行是指两个或者多个事件在同一时刻发生；而并发是指两个或多个事件在同一时间间隔发生。

并行是在不同实体上的多个事件，并发是在同一实体上的多个事件。

在Python中，我们可以使用multiprocessing模块进行多进程开发。程序执行时候，程序自身为主进程，然后 fork出多个子进程。

在Python中，我们可以使用threading进行多线程(threading)开发。

🡪

Python上已经开发出了许多和数据库交互的接口DB-API.不同的数据库Python需要不同的API去与其交互。大致的流程都是引入模块，获取与数据库的链接并且返回cursor, 通过cursor输入语句（增删改查），通过cursor方法获得返回的结果。

🡪

Python 的telnet模块在执行每一句语句之前，都必须又要read\_until. 不管上一句话能运行多快，如果没有那么下一句话python会马上直接发送，对方的server根本没那么快反应，会没法接受。

🡪

API与Python包的区别

API是一个成熟的系统提供给其他系统调用的接口，表现方式可以有两种。一种是一个URL地址。例如OpenStack的API，Jira的REST API。 一种是Python包的API,里面说明了API调用方式。

Python包是一个函数集合。一般利用Python包和API合用来达到目的。

例如：

我们想使用RESTAPI创建一个JIRA ISSUE，我们可以用两种方式。

第一种：我们使用PYTHON的REQEUST包，按照文档说明的格式，向RESTAPI发送请求，发送JSON格式数据，达到目的。

第二种：我们可以直接使用Python的JIRA包，按照Python JIRA包的文档说明达到目的。（其实Python JIRA包实际上便又封装了一层，在PYTHON JIRA包里面，其实也是引入了Requests模块，然后向RESTAPI发送。但是我们使用Python的JIRA包，变得更加方便，而不需要直接操作RESTAPI地址。

🡪

Python的telnetlib包中的timeout定义最好在每个需要等待的操作中都指定。例如telnetlib. Telnet("regsrv-sas.in.alcatel-lucent.com", port=23, timeout=10)这句话，只是让python在建立对服务器连接的时候，timeout=10. 连接上以后tn.read\_until不会承接timeout=10的属性，所以如果在tn.read\_until的时候卡住，python一样不会超市，会一直block住。所以在接下来的语句中还需要写成tn.read\_until(…,timeout=10).

P.S Perl中在构建telnet对象的时候，设置的timeout属性会被后面的操作所继承。

🡪

Python中如果用with去打开文件，然后在with的上下文中进行json.dump()去写入文件的内容。那需要注意，with在上下文结束以后，还要给写入的文件加入标记结束。所以不能在with的上下文中，进行cp,因为那时候在with的上下文中，要生成的文件还没有完全生成，cp过去的文件是半成品。

🡪

Python中使用paramiko作为SSH的通信模块。这个模块可以使用SSH去和服务器进行交互，执行命令以及获取返回值。

对服务器执行命令以及获取返回的内容为：stdin,stdout,stderr = ssh.exec\_command(command,timeout=xx).

获取返回的内容为: stdin.read(),stdout.read(), stderr.read()

🡪

在python里，string object和unicode object是两种不同的类型。  
  
string object是由characters组成的sequence，而unicode object是Unicode code units组成的sequence

🡪

Python文件在引入模块的时候需要注意可能发生的循环引用问题，意思是A需要B，而B又需要A。

解决的办法有：

将引入改为单方向，进行引入整合。

将引入放入某个方法当中，当引入语句放在方法当中时，则只有执行该方法才会执行该引入语句。

🡪

Python的\_\_init\_\_.py的功能

🡪

当建立了一个\_\_init\_\_.py文件时候, Python就会认为这个目的时Python的一个包。

🡪

并且我们在引入这个包的时候，在\_\_init\_\_.py中的代码会自动运行，所以如果我们在\_\_init\_\_.py中加入一些import代码等，那么这些代码便会被执行。

🡪

在\_\_init\_\_.py中指定\_\_all\_\_这个特殊的变量，变会在以后from xx import \*引入所有的时候，引入指定\_\_all\_\_中指定的变量。

🡪

使用Python解析一个文档的时候，如果需要的匹配十分复杂，我们可以不直接使用正则表达式，而是通过文件句柄，将文件的内容都入，然后逐句进行正则分析。

🡪

Pycharm是由JetBrain公司开发的用于编码Python的开发集成环境，除Pycharm以外，JetBrain公司还提供其他JAVA (IntelliJ), PHP, 数据库开发等多种继承模式。需要购买证书作为使用权限。

Pycharm的offline是指当前没有连上任何的版本控制，所以无法进行提交同步代码。但是可以进行编辑等。

🡪

PEP8是一个编程规范，内容是一些关于如何让你的程序更具可读性的建议。

例如包括了缩进。4个空格的缩进（编辑器都可以完成此功能），不使用Tap，更不能混合使用Tap和空格。

每行最大长度79，换行可以使用反斜杠，最好使用圆括号。换行点要在操作符的后边敲回车。

类和top-level函数定义之间空两行；类中的方法定义之间空一行；函数内逻辑无关段落之间空一行；其他地方尽量不要再空行。

🡪

我们用python的json模块读文件的时候，文件中定义的数据类型将影响读入的数据类型。

例如在文件中定义：

{“A”:10} 或者 {“A”: 10} （空格不影响）

那么json.load() 以后A所对应的是数字10而不是字符串10.如果是{“A”:”10”},那么json.load()以后A所对应的值是字符串‘10’而不是数字10.

🡪

Python作用域的产生

就作用域而言，Python与C有着很大的区别，在Python中并不是所有的语句块中都会产生作用域。只有当变量在Module(模块)、Class(类)、def(函数)中定义的时候，才会有作用域的概念。在if-elif-else、for-else、while、try-except\try-finally等关键字的语句块中并不会产成作用域。

使用def从而产生作用域：

#!/usr/bin/env python

def func():

variable = 100

print variable

print variable

代码的输出为：

NameError: name 'variable' is not defined

使用if-else语句不产生作用域：

if True:

variable = 100

print (variable)

print ("\*\*\*\*\*\*")

print (variable)

代码的输出为：

100

\*\*\*\*\*\*

100

19.

Python的续行，如果Python中一行太长，我们根据PEP8的原则，可以将一行分成多行来写。主要有两种写法

--第一种使用 “\” 进行换行输入：

例如2+\

3

Python将输出5.

--第二种使用括号（），方括号 [] 来续行。

例如

test = [1,

2,3,4,

5]

print test [3]

Python输出4

又如：

with open(  
 '/samregression/generatedToolData/Vresource/Galvatron/' + regression\_server + '/vResourceTrack\_lock\_' + regression\_server + '.json',  
 'w+') as outfile:

为合法的Python语句

🡪

当我们使用Python的循环去删除文件夹的时候，比较安全的做法是将删除文件夹的语句放在判断语句内，加强删除时的条件，防止误删除。

例如：

For each\_dir in dir\_list: For each dir in dir\_list:

Os.system(“rm -rf “ + each\_dir) --🡪 if “Should\_Deleted” in each\_dir:

Os.system(“rm -rf “ + each\_dir)

🡪

Python的字典是无序的，在于

（1）键值的哈希碰撞，hash(key1) == hash(key2)时，向字典里连续添加的这个两个键的顺序是不可以控制的，也是无法做到连续的，后来的键会按算法调整到其它位置。  
（2）字典空间扩容，当键的数量超过字典默认开的空间时，字典会做空间扩容，扩容后的键顺和创建顺序就会发生变化，不受人为控制。

使字典有序化的输出最好的方法是，当要输出字典的时候，先取出函数,keys(),values(),items()取出内容，然后通过排序来达到输出有序化字典的目的。

另外我们可以通过Python的有序字典[OrderedDict](http://www.cnblogs.com/mjoy/tag/python%E5%88%9B%E5%BB%BA%E6%9C%89%E5%BA%8F%E5%AD%97%E5%85%B8OrderedDict/)来让字典变得有序。它背后的实现方式是将插入的key放入一个列表中，当读取的时候从这个列表中都key,达到字典可以按key的插入顺序进行输出。

🡪

re.split（“正则表达式”，内容，分割次数）， 作用是根据正则表达式进行分割，并且正则表示式作为分隔符. 如果正则表达式不被括号包围，将不会放入结果当中。如果被括号包围，则加入结果当中。

例如：

m = re.split('\d+' , '123abc321cba') 🡪 不在括号当中，匹配的结果将保留

print m

print "\n"

#保留匹配项

m = re.split('(\d+)' , '123abc321cba') 🡪 在括号当中，匹配的结果将保留

print m

🡪

Python 使用open打开文件进行写操作的时候，将使用缓冲机制，缓存分三种：

全缓冲 : open函数的buffering设置大于1的整数n,n为缓冲区大小，linux默认为page的大小4096 满了n 个字节才会写入磁盘 。

例如：

f=open(“demo.txt”,’w’,buffering=1)

行缓冲 : open 函数的buffering设置为1, 碰到换行就会将缓冲区的写入磁盘。

例如：

f=open(“demo.txt”,’w’,buffering=1)

无缓冲 : open 函数的buffering设置为0 有输入就写入磁盘。

例如：

f=open(“demo.txt”,’w,’,buffering=0)

🡪

[python中\_\_name\_\_的使用](https://www.cnblogs.com/1204guo/p/7966461.html)

\_\_name\_\_是一个变量。前后加了双下划线是因为是因为这是系统定义的名字。普通变量不要使用此方式命名变量。  
Python有很多模块，而这些模块是可以独立运行的！这点不像C++和C的头文件。  
而且需要注意的是import的时候是要执行所import的模块的。  
\_\_name\_\_就是标识模块的名字的一个系统变量。这里分两种情况：假如当前模块是主模块（也就是调用其他模块的模块），那么此模块名字就是\_\_main\_\_，通过if判断这样就可以执行 “\_\_main\_\_:” 后面的主函数内容；假如此模块是被import的，则此模块名字为文件名字（不加后面的.py），通过if判断这样就会跳过 “\_\_mian\_\_:” 后面的内容。

经常会出现的情况是，如果我们不加判断条件，那么在引入包的时候，如果在函数内有一些定时任务绑定等，就会被绑定多次，每一个被引入都会被绑定。

🡪

Python内置函数locals () 记录的所有局部变量，并且以字典形式存储。所以要判断某个变量是否申明，使用xx in locals (). keys (). 并且locals()所返回的字典是只读字典，我们无法手动的直接对其更新键，或者键值。

Python内置函数globals () 记录的所有全局变量

🡪

Python字符串的count () 内置函数，可以知道某个字符出现在该字符串中几次

例如 “ABCC”.count(“C”) # 结果为2

🡪

Linux下Python的安装方式主要有3两种。

第一种为采用系统自带包管理器安装例如CentOS, RHEL中的yum.如果安装Python2.7.9以及Python3.4以后的版本pip自动包含。这是最简单的方式。

yum install -y python3

P.s默认python模块包安装到/usr/local/lib/Pythonx.x.x/site-packages中

第二种是编译手动安装，即经历make, make install步骤，在编译安装中指定输出Python的位置以及安装的包。如果安装Python2.7.9以及Python3.4以后的版本pip自动包含。

需要注意的是编译安装时候和环境有关系。例如是否安装yum的opendev-ssl包，sqlite-dev包等会将决定编译安装的Python是否原生支持SSL连接以及是原生含有sqlite数据库。

P. s采用编译安装的时候，编译出的Python是否含有某个包(例如sqlite) 有时候有很大关系。例如sqlite包。Sqlite包已经在pypi中取消，代表我们无法从以后的pip中进行安装。所以如果选择编译安装python， 而在编译安装的时候缺少环境，那么编译出来的python便无法对sqlite进行支持。解决的办法需要在服务器环境中先安装环境，然后对python进行重新编译安装。

第三种是使用一些集成Python计算平台。例如Anaconda。Anaconda除了会安装Python以外，还会自动安装一些常用计算包。比Python自带模块包要多许多。

🡪

Python的三元表达式与其他语言的三元表达式格式不太一样。三元表达式的意义在于对简单条件语句的简写，即是简单的if else条件判断语句写在同一行。

Python格式为: <expression1> if <condition> else <expression2>

Example : name = “True” if 5==3 else “False”

Example : print("True") if 5==5 else print("False")

Note: 为真为假时都只能执行一个语句。

Python没有三元运算符。不过有类似的写法

例如：

print("Yes") if 1 >2 else print("No") # 输出No, 等效于 1>2? print(“YES”):print(“No”)

🡪

从Python3.5开始，Python增加了函数的元数据支持。函数的元数据可以用于描述函数的类型，以后返回类型说明。让开发者可以获知如果使用该函数，同时为函数添加元数据以后IDE可以更好的识别参数。这些元数据保存在函数名.\_\_annotations\_\_字典中.

对于函数中的参数使用:类型 添加元数据， 对于返回值在函数末尾使用->类型 表明返回的数据类型。

例如：

def twoSum(nums: List[int], target: int) -> List[int]:

…

表明函数第一个参数nums应该时list of int,函数的第二个参数应该是int.函数最后应该返回list of int.

除了使用函数元数据描述函数以外，还可以使用三重引号添加注释，在注释中添加对函数的描述（接收参数，用途，返回结果等）。这些描述会存在help(方法名)当中。

🡪