**1.**

**# 常用的三种输出格式**

# 1. 带格式输出,{0}是指输出的第0个元素，同理{1}为第1个元素，{2}为第2个... 可以不按顺序排列

**print( "Hello {0}， I'm {2}, I,m {1} year old".format("world", age, name) )**

# 2. 使用类型输出，指定输出类型

**print( "I am %s, today is %d year"%(name, year) )**

# f字符串，{}中为元素，是.format的简化形式

**print( f"Today is {year}")**

Python print format() 格式化内置函数

<https://www.cnblogs.com/itfat/p/7455253.html>

**2.**

\_\_name\_\_ 是当前模块名，当模块被直接运行时模块名为 \_\_main\_\_ 。这句话的意思就是，当模块被直接运行时，以下代码块将被运行，当模块是被导入时，代码块不被运行。

<https://www.zhihu.com/question/49136398>

**3.**

Python编程中raise可以实现报出错误的功能，而报错的条件可以由程序员自己去定制。在面向对象编程中，可以先预留一个方法接口不实现，在其子类中实现。如果要求其子类一定要实现，不实现的时候会导致问题，那么采用raise的方式就很好。而此时产生的问题分类是**NotImplementedError**。

<https://www.runoob.com/python/python-exceptions.html>

我们可以使用raise语句自己触发异常

**raise语法**格式如下：

raise [Exception [, args [, traceback]]]

语句中 Exception 是异常的类型（例如，NameError）参数标准异常中任一种，args 是自已提供的异常参数。

最后一个参数是可选的（在实践中很少使用），如果存在，是跟踪异常对象。

**try-finally 语句**

try:

<语句>

finally:

<语句> #退出try时总会执行

raise

**4.**

虽然Python语法允许代码块**随意缩进几个空格**，但是位于同一个代码块中的所有语句必须**保持相同的缩进**，不能一下缩进3个空格，一下缩进4个空格。

**5.**

**\*args和\*\*kwargs**



args，kwargs仅仅是一个约定俗成的形参的写法

**args类型是一个元组tuple，而kwargs则是一个字典dict，并且args只能位于kwargs的前面。**

∗ 把函数接受到的多个参数打包成了元组，赋值给了形参。

∗的作用：函数接受实参时，按顺序分配给函数形参，如果遇到带∗的形参，那么就把还未分配出去的实参以元组形式打包（pack）,分配给那个带∗的形参。

∗这次没有用在函数定义中，而是用在了函数**调用**中。作用是

把实参（元组或列表），拆分（unpack）成单个的，依次赋值给函数的形参。

打包（pack）：\*args是把多个位置参数打包成元组，\*\*kwargs是把多个关键字参数打包成字典。

拆分（unpack）：\*args是把打包了的参数拆成单个的，依次赋值给函数的形参，\*\*kwargs是把字典的键值拆成单个的，依次赋值给函数的形参。

**6.**

TODO 在 python 中作为一种 助记符 (Mnemonics)，用来解释 将要做什么 。

**7.**

实例调用\_\_class\_\_属性时会指向该实例对应的类，然后可以再去调用其它类属性，类属性由类调用

**8.**

**python的@property**是python的一种装饰器，是用来修饰方法的。

<https://zhuanlan.zhihu.com/p/64487092>

作用：

我们可以使用@property装饰器来创建只读属性，@property装饰器会将方法转换为相同名称的只读属性,可以与所定义的属性配合使用，这样可以防止属性被修改。

**9.**

Python3 assert（断言）

Python assert（断言）用于判断一个表达式，在表达式条件为 false 的时候触发异常。

断言可以在条件不满足程序运行的情况下直接返回错误，而不必等待程序运行后出现崩溃的情况，例如我们的代码只能在 Linux 系统下运行，可以先判断当前系统是否符合条件。

**10.**

**python 类中变量方法命名时有关下划线的规则**

**1.单下划线：**按照Python中的习惯单下划线一般不应该被外界访问，尽管事实上是可以访问的。

**2.名字前面有双下划线：**Python类中以这种命名的变量或方法不能被外界访问。事实上在Python会把双下划线命名的变量改成另外的名字，因此在外部访问时会报找不到名称的错误

**3.名字前后都有双下划线**：这类名称是Python内置的变量名，本身和我们定义的名称没有什么区别，只是python为了加以区分。如构造函数\_\_init\_\_ 。这类变量或方法是可以访问的。

<https://blog.csdn.net/weixin_44339148/article/details/95216762>

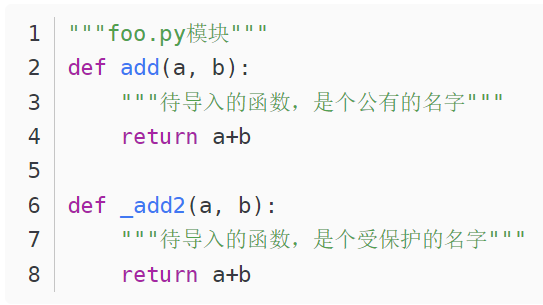
**Python带\_的变量或函数命名，带下划线的方法**

单前置下划线的名字:

《流畅的python》作者称之为“受保护的”名字，有两种主要的用法：

1、作为类名或函数名时，

会阻止其他python脚本通过【from module import \*】语句导入该名字，即该名字不会被星号匹配；



<https://blog.csdn.net/qq_35290785/article/details/93476722>

**11.**

**Python 实例方法、类方法、静态方法的区别与作用**

<https://www.cnblogs.com/wcwnina/p/8644892.html>

Python中至少有三种比较常见的方法类型，即实例方法，类方法、静态方法。它们是如何定义的呢？如何调用的呢？它们又有何区别和作用呢？且看下文。

首先，这三种方法都定义在类中。下面我先简单说一下怎么定义和调用的。（PS：实例对象的权限最大。）

实例方法

定义：第一个参数必须是实例对象，该参数名一般约定为“self”，通过它来传递实例的属性和方法（也可以传类的属性和方法）；

调用：只能由实例对象调用。

类方法

定义：使用装饰器@classmethod。第一个参数必须是当前类对象，该参数名一般约定为“cls”，通过它来传递类的属性和方法（不能传实例的属性和方法）；

调用：类和实例对象都可以调用。

静态方法

定义：使用装饰器@staticmethod。参数随意，没有“self”和“cls”参数，但是方法体中不能使用类或实例的任何属性和方法；

调用：类和实例对象都可以调用。

12.

**Python装饰器**

<https://www.zhihu.com/question/26930016>

**13.**

**占位符**

placeholder，占位符，在tensorflow中类似于函数参数，运行时必须传入值。函数说明：tf.placeholder(dtype, shape=None, name=None)

**14.**

**Python lambda介绍**

[**https://www.cnblogs.com/evening/archive/2012/03/29/2423554.html**](https://www.cnblogs.com/evening/archive/2012/03/29/2423554.html)

**15.**

**global语句是适用于当前整个代码块的声明。它是全局变量的标识符。如果某名字在局部名字空间中没有定义, 就自动使用相应的全局名字. 没有global是不可能手动指定一个名字是全局的.在 global 中出现的名字不能在global 之前的代码中使用.在 global 中出现的名字不能作为形参, 不能作为循环的控制对象, 不能在类定义, 函数定义, import语句中出现.**

**与nonlocal关键字的区别：**

**global语句用以知名某个特定的变量为全局作用域，并重新绑定它。nonlocal语句用以指明某个特定的变量为封闭作用域，并重新绑定它。**

**Python中定义函数时，若想在函数内部对函数外的变量进行操作，就需要在函数内部声明其为global**

**基本格式是：**

**关键字global，后跟一个或多个变量名**

**16.**

**python路径拼接os.path.join()函数的用法**

os.path.join()函数：连接两个或更多的路径名组件

1.如果各组件名首字母不包含’/’，则函数会自动加上

2.如果有一个组件是一个绝对路径，则在它之前的所有组件均会被舍弃

3.如果最后一个组件为空，则生成的路径以一个’/’分隔符结尾

<https://www.cnblogs.com/an-ning0920/p/10037790.html>

**os之遍历文件夹**

[**https://www.cnblogs.com/jim-fan/articles/13053252.html**](https://www.cnblogs.com/jim-fan/articles/13053252.html)

[**https://www.jianshu.com/p/582910d13501**](https://www.jianshu.com/p/582910d13501)

**17.**

**列表解析 List Comprehensions**

<http://blog.sciencenet.cn/blog-3031432-1060628.html>

<https://www.cnblogs.com/RayMin/p/5781725.html>

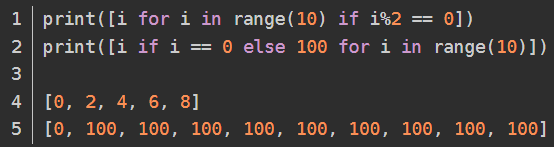
<https://www.runoob.com/note/42490>

**Python列表解析配合if else**

列表解析总共有两种形式：

1. [i for i in range(k) if condition]：此时if起条件判断作用，满足条件的，将被返回成为最终生成的列表的一员。

2. [i if condition else exp for exp]：此时if...else被用来赋值，满足条件的i以及else被用来生成最终的列表。



<https://blog.csdn.net/zk_j1994/article/details/72809260>

**18.**

**parse\_args（argsparse）：python和命令行之间的交互**

<https://www.cnblogs.com/still-smile/p/11636958.html>

19.

**python--获取当前文件绝对路径的几种方式对比**

<https://www.cnblogs.com/xiaohuboke/p/13611234.html>

**20.**

\_\_getitem\_\_方法

Python的魔法方法\_\_getitem\_\_ 可以让对象实现迭代功能，这样就可以使用for...in... 来迭代该对象了

\_\_getitem\_\_(self,key):

把类中的属性定义为序列，可以使用\_\_getitem\_\_()函数输出序列属性中的某个元素，这个方法返回与指定键想关联的值。对序列来说，键应该是0~n-1的整数，其中n为序列的长度。对映射来说，键可以是任何类型。

如果在类中定义了\_\_getitem\_\_()方法，那么它的实例对象（假设为P）就可以以P[key]形式取值，当实例对象做P[key]运算时，就会调用类中的\_\_getitem\_\_()方法。当对类的属性进行下标的操作时，首先会被\_\_getitem\_\_() 拦截，从而执行在\_\_getitem\_\_()方法中设定的操作，如赋值，修改内容，删除内容等。

<https://www.cnblogs.com/Aiyuqianer/p/14091062.html>

**21.**

Joblib Parallel delayed

<https://blog.csdn.net/weixin_42280274/article/details/107316807>

<https://zhuanlan.zhihu.com/p/65025655>

22.

**set()** 函数创建一个无序不重复元素集，可进行关系测试，删除重复数据，还可以计算交集、差集、并集等。

https://www.run oob.com/python/python-func-set.html

<https://www.cnblogs.com/sophe/p/11534146.html>

**23.**

**python并行库\_数据库并行读取和写入**

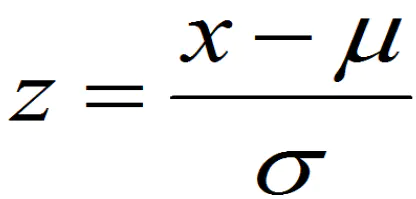
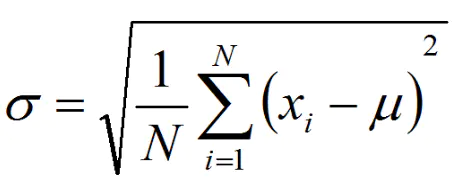
**如何实现对数据库的并行运算来节省代码运行时间**

**几十万条数据要是一条条读写，然后在本机上操作，耗时太久，可行性极低**

<https://blog.csdn.net/weixin_39649478/article/details/111062854>

**24.**

**z-score标准化**

 **其中：**

**25.**

**python中的del用法**

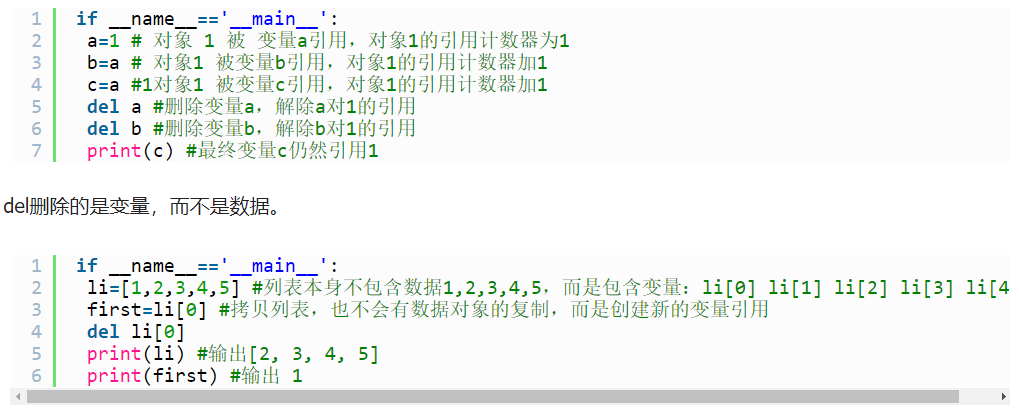
del是python关键字，就像def、and、or一样。它不是字典、列表的方法，但是可以用来删除字典、列表的元素。

python中的del用法比较特殊，新手学习往往产生误解，弄清del的用法，可以帮助深入理解python的内存方面的问题。

python的del不同于C的free和C++的delete。

由于python都是引用，而python有GC机制，所以，del语句作用在变量上，而不是数据对象上。

del是python关键字，就像def、and、or一样。它不是字典、列表的方法，但是可以用来删除字典、列表的元素。



26.

**PyTorch 中的 ModuleList 和 Sequential: 区别和使用场景**

<https://zhuanlan.zhihu.com/p/64990232>

**27.**

**警惕！！！Python之在函数中使用列表作为默认参数**

当函数第一次执行时，参数lst的默认值[]被创建。这个默认值只会被创建一次。

<https://www.cnblogs.com/jclian91/p/10325849.html>

**28.**

**python模块导入顺序**

最好按照以下顺序:

1、Python 标准库模块

2、Python 第三方模块

3、应用程序自定义模块

**29.**

**PyTorch中的contiguous**

is\_contiguous直观的解释: Tensor底层一维数组元素的存储顺序与Tensor按行优先一维展开的元素顺序是否一致。

transpose、permute 后使用 contiguous 方法则会重新开辟一块内存空间保证数据是在逻辑顺序和内存中是一致的，连续内存布局减少了CPU对对内存的请求次数（访问内存比访问寄存器慢100倍），相当于空间换时间。

<https://samuel92.blog.csdn.net/article/details/90417308>

**30.**

**python中的[1:]、[::-1]、X[:,m:n]和X[1,:]**

<https://blog.csdn.net/gaofengyan/article/details/90697743>

**31.**

**PyTorch笔记之 scatter() 函数**

<https://www.cnblogs.com/dogecheng/p/11938009.html>

32.

**Numpy tolist() 用法**

<https://blog.csdn.net/csdn_pfyang/article/details/104115804>

**33.**

**using Class() or self.\_\_class\_\_() to create a new object**

<https://stackoverflow.com/questions/20623925/class-vs-self-class-when-creating-object>

**34.**

**python 中的self和cls的区别, cls含义及使用方法**

self是类（Class）实例化对象，cls就是类（或子类）本身，取决于调用的是那个类。

@staticmethod 属于静态方法装饰器，@classmethod属于类方法装饰器。

<https://www.jb51.net/article/189135.htm>

python中cls代表的是类的本身，相对应的self则是类的一个实例对象。

cls可以在静态方法中使用，并通过cls()方法来实例化一个对象

<https://www.cnblogs.com/wayne-tou/p/11896706.html>

<https://www.cnblogs.com/king-lps/p/12597680.html>

**35. python 参数传递[打包与解包问题]**

<https://blog.csdn.net/minghu9/article/details/51339918>

1.

传递参数时，可以传递基于参数名的字典f(\*\*param\_dict)

也可以传递基于位置的列表f(\*param\_list)

2.获取参数列表

对于函数，可以直接在开头param\_dict=locals()获取参数字典

对于方法（类、实例），获取的参数列表还额外含有cls、self项需要移除

可以用如下函数无副作用地得到纯洁的参数字典

36. model.eval() 和 with torch.no\_grad()

如果不在意显存大小和计算时间的话，仅仅使用model.eval()已足够得到正确的validation的结果；而with torch.zero\_grad()则是更进一步加速和节省gpu空间（因为不用计算和存储gradient），从而可以更快计算，也可以跑更大的batch来测试。

<https://zhuanlan.zhihu.com/p/191623720>

<https://blog.csdn.net/songyu0120/article/details/103884586>

<https://discuss.pytorch.org/t/model-eval-vs-with-torch-no-grad/19615/60>

<https://stackoverflow.com/questions/55627780/evaluating-pytorch-models-with-torch-no-grad-vs-model-eval>

38.

pytorch查，看模型运行在cpu还是gpu

next(model.parameters()).device

<https://discuss.pytorch.org/t/which-device-is-model-tensor-stored-on/4908/13>

<https://discuss.pytorch.org/t/how-to-check-if-model-is-on-cuda/180>