Python语言知识

**1、numpy.linspace（）函数**

numpy.linspace(start,stop,num=50,endpoint=True,retstep=False, dtype=None)

在指定的间隔内返回均匀间隔的数字。

返回num均匀分布的样本，在[start, stop]。这个区间的端点可以任意的被排除在外。

Parameters(参数):

start : scalar(标量)

The starting value of the sequence(序列的起始点).

stop : scalar

序列的结束点，除非endpoint被设置为False，在这种情况下, the sequence consists of all but the last of num + 1 evenly spaced samples(该序列包括所有除了最后的num+1上均匀分布的样本(感觉这样翻译有点坑)), 以致于stop被排除.当endpoint is False的时候注意步长的大小(下面有例子).

num : int, optional(可选)

生成的样本数，默认是50。必须是非负。

endpoint : bool, optional

如果是真，则一定包括stop，如果为False，一定不会有stop

retstep : bool, optional

If True, return (samples, step), where step is the spacing between samples.

dtype : dtype, optional

The type of the output array. If dtype is not given, infer the data type from the other input arguments(推断这个输入用例从其他的输入中).

**2、numpy.meshgrid（）函数**

[X,Y] = meshgrid(x,y) 将向量x和y定义的区域转换成矩阵X和Y，这两个矩阵可以用来表示mesh和surf的三维空间点以及两个变量的赋值。其中矩阵X的行向量是向量x的简单复制，而矩阵Y的列向量是向量y的简单复制。

用于从数组a和b产生网格。生成的网格矩阵A和B大小是相同的。它也可以是更高维的。

**3、resize函数**

OpenCV提供了resize函数来改变图像的大小，函数原型如下：

void resize(InputArray src, OutputArray dst, Size dsize, double fx=0, double fy=0, int interpolation=INTER\_LINEAR );

先解释一下各个参数的意思：

src：输入，原图像，即待改变大小的图像；

dst：输出，改变大小之后的图像，这个图像和原图像具有相同的内 容，只是大小和原图像不一样而已；

dsize：输出图像的大小。如果这个参数不为0，那么就代表将原图像缩放到这个Size(width，height)指定的大小；如果这个参数为0，那么原图像缩放之后的大小就要通过下面的公式来计算：

dsize = Size(round(fx\*src.cols), round(fy\*src.rows))

其中，fx和fy就是下面要说的两个参数，是图像width方向和height方向的缩放比例。

fx：width方向的缩放比例，如果它是0，那么它就会按照(double)dsize.width/src.cols来计算；

fy：height方向的缩放比例，如果它是0，那么它就会按照(double)dsize.height/src.rows来计算；

interpolation：这个是指定插值的方式，图像缩放之后，肯定像素要进行重新计算的，就靠这个参数来指定重新计算像素的方式，有以下几种：

A）INTER\_NEAREST - 最邻近插值

B）INTER\_LINEAR - 双线性插值，如果最后一个参数你不指定，默认使用这种方法

C）INTER\_AREA - resampling using pixel area relation. It may be a preferred method for image decimation, as it gives moire’-free results. But when the image is zoomed, it is similar to the INTER\_NEAREST method.

D）INTER\_CUBIC - 4x4像素邻域内的双立方插值

E）INTER\_LANCZOS4 - 8x8像素邻域内的Lanczos插值

**4、使用\_\_future\_\_**

Python的每个新版本都会增加一些新的功能，或者对原来的功能作一些改动。有些改动是不兼容旧版本的，也就是在当前版本运行正常的代码，到下一个版本运行就可能不正常了。

从Python 2.7到Python 3.x就有不兼容的一些改动，比如2.x里的字符串用'xxx'表示str，Unicode字符串用u'xxx'表示unicode，而在3.x中，所有字符串都被视为unicode，因此，写u'xxx'和'xxx'是完全一致的，而在2.x中以'xxx'表示的str就必须写成b'xxx'，以此表示“二进制字符串”。

要直接把代码升级到3.x是比较冒进的，因为有大量的改动需要测试。相反，可以在2.7版本中先在一部分代码中测试一些3.x的特性，如果没有问题，再移植到3.x不迟。

Python提供了\_\_future\_\_模块，把下一个新版本的特性导入到当前版本，于是我们就可以在当前版本中测试一些新版本的特性。举例说明如下：

为了适应Python 3.x的新的字符串的表示方法，在2.7版本的代码中，可以通过unicode\_literals来使用Python 3.x的新的语法：

