**1、常用函数**

1.1 input函数

<变量> = input(<提示性文字>)

返回一个字符串。

1.2 eval函数

eval(<字符串>)

以python表达式的方式解析并执行字符串，并结果输出

1.3 abs(x)

返回x的绝对值

1.4 divmod(x, y)

输入二元组形式(x//y, x%y)

1.5 round(x[, ndigits])

对x四舍五入，保留ndigits位小数

1.6 complex(re[, im])

生成一个复数，re表示实数部分，im表示虚数部分

1.7 len(x)

返回字符串x的长度，也可以返回其他组合数据类型元素的个数

1.8 strip()

返回的是字符串的副本，并删除前导和后缀字符，例如：b = a.strip()

1.9 split()

str.split(str="", num=string.count(str))

str -- 分隔符，默认为所有的空字符，包括空格、换行(\n)、制表符(\t)等。

num -- 分割次数。默认为 -1, 即分隔所有。

1.10 vars([object])

返回对象object的属性和属性值的字典对象

**2、numpy模块**

import numpy as np

2.1 np.arange(n)

返回0~(n-1)数字组成的数组。

2.2 np.where()

返回输入数组中满足给定条件的元素的索引；数组可以使用索引来提取满足条件的元素。

2.3 np.extract()

根据某个条件从数组中抽取元素，返回满足条件的元素。

2.4 np.argmax(), np.argmin()

分别沿指定轴返回最大元素，最小元素的索引。

其中，axis = 0，表示按列索引，axis = 1，表示按行索引。

2.5 np.argsort(a, axis=-1, kind=’quicksort’, order=None)

返回矩阵a按照axis指定的维度进行排序后的下标（注：a不发生改变）

2.6 np.concatenate((a1, a2, …), axis=0)

用于沿指定轴连接相同形状的两个或多个数组，未添加新维度。

2.7 np.stack(arrays, axis)

用于沿指定轴堆叠数组序列，相当于添加一个维度。

2.8 np.split(array, indices\_or\_sections, axis)

沿指定轴将数组分割为子数组，等分或者在指定位置处分割。

**3、\*args, \*\*kwargs的用法**

def foo(\*args, \*\*kwargs):  
 print('args = ', args)  
 print('kwargs = ', kwargs)  
 print('---------------------------------------')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 foo(1, 2, 3, 4)  
 foo(a=1, b=2, c=3)  
 foo(1, 2, 3, 4, a=1, b=2, c=3)  
 foo('a', 1, None, a=1, b='2', c=3)

运行结果：

args = (1, 2, 3, 4)

kwargs = {}

---------------------------------------

args = ()

kwargs = {'a': 1, 'b': 2, 'c': 3}

---------------------------------------

args = (1, 2, 3, 4)

kwargs = {'a': 1, 'b': 2, 'c': 3}

---------------------------------------

args = ('a', 1, None)

kwargs = {'a': 1, 'b': '2', 'c': 3}

---------------------------------------

可以看到，这两个是python中的可变参数。\*args表示任何多个无名参数，它是一个tuple；\*\*kwargs表示关键字参数，它是一个dict。并且同时使用\*args和\*\*kwargs时，必须\*args参数列要在\*\*kwargs前，像foo(a=1, b='2', c=3, a', 1, None, )这样调用的话，会提示语法错误“SyntaxError: non-keyword arg after keyword arg”。

**4、文件路径操作**

# glob模块

from glob import glob

# os.path模块

from os import path as osp

# 正则表达式

import re

例子如下：

EXPER\_PATH = 'G:\\practice\opencv\\testData'

def get\_paths(exper\_name):  
 *"""  
 Return a list of paths to the outputs of the experiment.  
 """* return glob(osp.join(EXPER\_PATH, 'outputs/{}/\*.npz'.format(exper\_name)))

def main():  
 exper\_name = 'magicpoint'  
 # 获取文件夹中所有文件的路径，组成列表  
 paths = get\_paths(exper\_name)  
 print('paths:\n', paths)  
 # 获取路径列表中的各个文件名  
 print('basename\_paths:')  
 for path in paths:  
 print(osp.basename(path))  
 # 获取文件名前缀  
 print('basename\_prefix\_paths:')  
 for path in paths:  
 basename = osp.basename(path)  
 temp = re.findall(r'(.+?)\.', basename)  
 basename\_prefix = temp[0]  
 print(basename\_prefix)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 main()

运行结果：

paths:

['G:\\practice\\opencv\\testData\\outputs/magicpoint\\1.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\10.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\2.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\3.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\4.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\5.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\6.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\7.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\8.npz', 'G:\\practice\\opencv\\testData\\outputs/magicpoint\\9.npz']

basename\_paths:

1.npz

10.npz

2.npz

3.npz

4.npz

5.npz

6.npz

7.npz

8.npz

9.npz

basename\_prefix\_paths:

1

10

2

3

4

5

6

7

8

9

**5、.txt文件存取**

5.1 将数据存储到txt文件中

with open(filename.txt, 'w') as f:  
 print(data, file=f)

5.2 读取文件

（1）方式1：

data = np.loadtxt(filename.txt)

（2）方式2：

with open('odom.txt', 'r') as f:  
 lines = f.readlines() # txt中所有字符串读入data  
 for line in lines:  
 temp = line.split() # 将单个数据分隔开存好  
 data = list(map(float, temp)) # 转化为浮点数

**6、.npz文件**

6.1 npz对象

常用属性

1）files

列出对.npz文件中，对各类数据的索引。

2）f

可以作为一种可选的方式执行Npzfile实例的索引。对象.f.索引<==>对象[‘索引’]

6.2 存储方式

Np.savez(filename, attr\_name=data) # eg: attr\_name is points

6.3 读取方式

tmp = np.load(filename)

data = tmp[‘points’]

**6、math模块**

6.1 math的数学常数

1）math.pi：圆周率

2）math.e：自然对数

3）math.inf：正无穷

4）math.nan：非浮点数标记

6.2 math的数值表示函数

1）math.fabs(x)：返回x的绝对值

2）math.fmod(x, y)：返回x%y

3）math.fsum([x, y, …])：浮点数精确求和

4）math.ceil(x)：向上取整

5）math.floor(x)：向下取整

6）math.factorial(x)：返回x的阶乘

7）math.gcd(x, y)：返回x和y的最大公约数

8）math.frepx(x), x = m \* 2e ：返回(m, e)

9）math.ldexp(x, i)：返回x \* 2^i的运算值，math.frepx(x)的反运算

10）math.modf：返回x的小数和整数部分

11）math.trunc：返回x的整数部分

12）math.copysign(x, y)：用y的正负号替换x的正负号

13）math.isclose(a, b)：比较a和b的相似性，返回True或False

14）math.isfinite(x)：若x为无穷大，则返回True，否则返回False

15）math.isinf(x)：若x为正数或负数无穷大，则返回True，否则返回False

16）math.isnan(x)：若x为NaN，则返回True，否则返回False

6.3 math的幂对数函数

1）math. pow (x, y)：返回x的y次幂

2）math.exp(x)：返回e的x次幂

3）math.expml(x)：返回e的x次幂减1

4）math.sqrt(x)：返回x的平方根

5）math.log(x[,base])：返回x的对数值

6）math.log1p(x)：返回1+x的自然对数

7）math.log2(x, y)：返回x的2对数值

8）math.log10(x)：返回x的2对数值

6.4 math的三角运算函数

1）math.degree(x)：角度x的弧度值转化为角度值

2）math.radians(x)：角度x的角度值转化为弧度值

3）math.hypot(x, y)：返回(x, y)坐标到原点(0, 0)的距离

4）math.sin(x)：sin x

5）math.cos(x)：cos x

6）math.tan(x)：tan x

7）math.asin(x, y)：arcsin x

8）math.acos(x)：arccos x

9）math.atan(x)：arctan x

10）math.atan2(x, y)：arctan y/x

11）math.sinh：sinh x

12）math.cosh(x, y)：cosh x

13）math.tanh (a, b)：tanh x

14）math.asinh(x)：arcsinh x

15）math.acosh(x)：arccosh x

16）math.atanh(x)：arctanh x

**7、字符串类型的格式化**

<模板字符串>.format(<逗号分隔的参数>)

其中，模板字符串由一系列槽组成，用来控制修改字符串中嵌入值出现的位置。

槽用大括号{}表示，槽的内部样式是{<参数序号>: <格式控制标记>}。格式控制标记包括填充、对齐、，、精度和类型6个字段，用来控制参数显示时的格式。

其中，精度表示两个含义，由小数点开头，对于浮点数，精度表示小数部分输出的有效位数，对于字符串，精度表示输出的最大长度。类型表示输出整数和浮点数类型的格式规则，对于整数类型，o输出整数的八进制方式，x输出整数的小写十六进制方式，X输出整数的大写十六进制方式；对于 浮点数，e/E输出科学计数法的形式，f输出浮点数的标准浮点形式，%输出浮点数的百分形式。

**8、lambda函数（匿名函数）**

<函数名> = lambda<参数列表> : <表达式>

Lambda函数与正常的函数一样，等价于下面形式：

def <函数名>(<参数列表>):

return <表达式>

如下例所示：

>>> f = lambda x, y : x + y

>>> type(f)

<class ‘function’>

>>> f(10, 12)

22

**9、文件的写入和读取**

**9.1 JSON方式**

import json

（1）文件写入

json.dumps: 将 Python 对象编码成 JSON 字符串

hp = hp = json.dumps(vars(hparams))

with open(filename, 'w') as fout:

fout.write(hp)

（2）文件的读取

json.loads: 将已编码的 JSON 字符串解码为 Python 对象。

d = open(os.path.join(path, "hparams"), 'r').read()

flag2val = json.loads(d)

for f, v in flag2val.items():

parser.f = v

**11、with语句**

with语句适用于对资源进行访问的场合，确保不管使用过程中是否发生异常都会执行必要的“清理”操作，释放资源。比如文件使用后自动关闭，线程中锁的自动获取和释放等。

下面以文件操作为例进行说明：

try:

f = open('xxx')

except:

print('fail to open')

exit(-1)

try:

do something

except:

do something

finally:

f.close()

上面的代码与下面的代码等价：

with open('xxx') as file:

data = file.read()

do something

显然with语句能够减少冗长，还可以自动处理上下文环境产生的异常。

**12、codecs：自然语言编码转换**

import codecs

# 用codecs提供的open方法来指定打开的文件的语言编码，它会在读取的时候自动转换为内部unicode

with codecs.open(file, ‘r’, encode\_type)

13、python函数参数前面单星号\*与双星号\*\*的区别

（1）单星号（\*）：\*args，有两个用法

用法一：将参数以元组的形式导入。例如：

>>>def foo(param1, \*param2):

print(param1)

print(param2)

>>>foo(1, 2, 3, 4, 5)

1

(2, 3, 4, 5)

用法二：解压参数列表。例如：

>>>def foo(bar, lee):

print(bar, lee)

>>>l = [1, 2]

>>>foo(\*l)

1 2

（2）双星号（\*\*）：\*\*args

将参数以字典的形式导入。例如：

>>>def foo(param1, \*param2):

print(param1)

print(param1)

>>>foo(1, 2, 3, 4, 5)

1

{‘a’: 2, ‘b’: 3}