代码内容审核建议 10 （PCS_1-5）

1. **PCS_1: Print 方法格式化字符串新方法补充，**

在此部分之后，可以添加 Python print 新方法（after py3.6）

> 如果想把变量值作为字符串的一部分打印出来，可以使用字符串格式化方法，例如 `%` 形式，或者 `'{}'.format(variable)` 方式。

新的格式化字符串方式，即在普通字符串前添加 `f` 或 `F` 前缀，其效果类似于 `str.format()`。比如

```
name = "red"
print(f"He said his name is {name}.")
# 'He said his name is red.'
```

相当于：

```
print("He said his name is {name}.".format(**locals()))
```

此外，此特性还支持嵌套字段，比如：

```
import decimal
width = 10
precision = 4
value = decimal.Decimal("12.34567")
print(f"result: {value:{width}.{precision}}")
#'result:    12.35'
```

2. 可以补充 print()多个变量的格式化方法输出，比如提示 PCS_2 中的此部分与输出格式有关

以及 **PCS_3** 中的如下部分

在数据分析时，会涉及到很多计算结果显示查看，尤其用于交流的代码。下述是正态分布（normal distribution/Gaussian distribution）的概率计算，调用了SciPy库的 `norm.sf(x,loc,scale)` ， `norm.cdf()` 和 `norm.ppf()` 的方法，计算给定值(x)，给定正态分布均值（loc）和标准差（scale），求取大于等于（ `sf` ）或小于等于(`cdf`)给定值的概率；反之，求取满足概率的值（ `ppf` ）。

```
from scipy.stats import norm

print("用.sf计算值大于或等于0.7待概率为：%s",norm.sf(0.7,0,1))
print("用.cdf计算值小于或等于0.7的概率为：%f"%norm.cdf(0.7,0,1)) #
print("可以观察到.cdf（<=0.7）概率结果+.sf(>=0.7)概率结果为：%.3f"%(norm.cdf(113,0,1)+norm.sf(113,0,1)))
print("用.ppf找到给定概率值为0.758036(约75.80%)的数值为：%e"%norm.ppf(0.758036,0,1))
```

```
用.sf计算值大于或等于0.7待概率为：%s 0.24196365222307303
用.cdf计算值小于或等于0.7的概率为：0.758036
可以观察到.cdf（<=0.7）概率结果+.sf(>=0.7)概率结果为：1.000
用.ppf找到给定概率值为0.758036(约75.80%)的数值为：6.999989e-01
```

```
print('name:%s,category:%s,score:%s'%('湘村菜馆','美食_中餐厅',4))
info_dict={'name':'湘村菜馆','category':'美食_中餐厅','score':4}
print('name:%(name)s,category:%(category)s,score:%(score)s'%info_dict)

print('_'*50)
import math
print('%+-10.3f:)'%-math.pi)
print('%+-10.3f:)'%math.pi)
print('%+-10.*f:)'%(3,math.pi))
print('%010.3f:)'%math.pi)
```

```
name:湘村菜馆,category:美食_中餐厅,score:4
name:湘村菜馆,category:美食_中餐厅,score:4
--------------------------------------------------
-3.142    :)
+3.142    :)
+3.142    :)
000003.142:)
```

```
import math
print(' {0:10}={1:5}'.format('pi',math.pi))
print(' {0:>10}={1:5}'.format('pi',math.pi))
print(' {0:<10}={1:5}'.format('pi',math.pi))
print(' {0}={1:.3f}'.format('pi',math.pi))
```

```
pi        =3.141592653589793
        pi=3.141592653589793
pi        =3.141592653589793
pi=3.142
```

补充资料：

python 格式化字符串的三种方法（%，format，f-string)

https://blog.csdn.net/zjbyough/article/details/96466658

Python 新特性补充链接

且现在 py3.9，3.10 已经发布

Python3.6、3.7、3.8 新特性-刘江的博客,

https://www.liujiangblog.com/blog/51/

Python 这几年都更新了啥 - 浮云计算 (liriansu.com)

3. **PCS_3：对于打印输出的距离，最好不要太长，影响查看**

```
import re

kml_description='<description>线路开始时间：2017-07-20 08:14:41,结束时间：2017-07-
pattern='description'
print(re.findall(pattern,kml_description)) #使用re.findall()方法以列表形式返回给定

# .
pattern='.description'
print(re.findall(pattern,kml_description))

# ? +
pattern=r'w?cadesign\.cn, w+\.cadesign\.cn' #用转义字符使用点号，而不是用作特殊字
text='cadesign.cn, www.cadesign.cn'
print(re.findall(pattern,text))  #? 号可以匹配0个 或者多个字符，因此即使不存在字符

# {m}
pattern=r'w{2}\.cadesign\.cn'
print(re.findall(pattern,text))

# [...]
pattern='[Py]*thon!'
textA='Hello Python!'
textB='Hello Pthon!'
textC='Hello ython!'
textD='Hello thon!'
print(re.findall(pattern,textA))
print(re.findall(pattern,textB))
print(re.findall(pattern,textC))
print(re.findall(pattern,textD))

# A|B
pattern='<description>|</description>'
print(re.findall(pattern,kml_description))
```

```
['description', 'description']
['<description', '/description']
['cadesign.cn, www.cadesign.cn']
['ww.cadesign.cn']
['Python!']
['Pthon!']
['ython!']
['thon!']
['<description>', '</description>']
```

4. PCS_4 中，函数定义，参数名未修改到位

Ranmen_price_lst,price_x 应该修改为 lst 和 x

```
def value2values_comparison(x,lst):
    import numpy as np

    lst_mean=np.mean(ranmen_price_lst)
    abs_difference_func=lambda value:abs(value-price_x)
    comparisonOF2values=lambda v1,v2:print('x is higher than the average %s of the list.'%lst_mean) if v1>v2 else print('x

    if x in lst:
        print("%s in the given list."%x)
        comparisonOF2values(x,lst_mean)
        return x
    else:
        print('%.3f is not in the list.'%x)
        closest_value=min(lst,key=abs_difference_func)
        print('the nearest value to %s is %s.'%(x,closest_value))
        print("_"*50)
        comparisonOF2values(closest_value,lst_mean)
        return closest_value
```

后面的代码测试中，出现一个打印错误，99 应该是 88

```
price_x_closestValue=value2values_comparison(78,[3,4,5,733,66,22,99,88,11])
print(price_x_closestValue)
```

```
78.000 is not in the list.
the nearest value to 78 is 99.
__________________________________________________
x is lower than the average 729.34 of the list.
99
```

将上述变量名更正后，可以得到正确结果

```
>>> price_x_closestValue=value2values_comparison(78,[3,4,5,733,66,22,99,88,11])
... print(price_x_closestValue)
78.000 is not in the list.
the nearest value to 78 is 88.
--------------------------------------------------
x is lower than the average 114.55555555555556 of the list.
88
```

5. 变量名错误

```python
from collections import namedtuple

Label=namedtuple('label',['name','id','trainID','category','categoryID','hasInstances','igoreInEval','color'])
print(Label)
print("_"*50)
label_building=Label( 'building',11,2,'construction',2,False,False, ( 70, 70, 70))
print(label)
print(label_building._fields)

print("_"*50)
print(label_building.name)
print(label_building.id)
print(label_building.category)
print(label_building.color)

print("_"*50)
caravan_lst=['caravan', 29,255,'vehicle',7,True,True, (  0,  0, 90)]
label_caravan=Label._make(caravan_lst)
print(label_caravan.name)
print(label_caravan.id)
print(label_caravan.category)
print(label_caravan.color)

label_caravan=label_caravan._replace(category='schooner',color=(30,30,60)) #替换属性值
print(label_caravan.category)
print(label_caravan.color)

print("_"*50)
caravan_dict=label_caravan._asdict() #将nametuple转换为dict
print(caravan_dict)
```

Python

```
<class '__main__.label'>
__________________________________________________
label(name='building', id=11, trainID=2, category='construction', categoryID=2, hasInstances=False, igoreInEval=False, color=(70, 70, 70))
('name', 'id', 'trainID', 'category', 'categoryID', 'hasInstances', 'igoreInEval', 'color')
__________________________________________________
```

第二个红框中的“label”应该改为“label_building”才能出现最下面的打印结果

6. PCS_5 中，“多态性（polymorphism）-数据类型类”，现在 python 也可以对参数类型进行设置规定，

- 多态性（polymorphism）-数据类型类

一些运算不仅对于数值起作用，同样对其它类型的数据起作用，例如下述案例中定义了一个乘积函数 `times(x,y)`，含两个输入参数，返回二者之积。python中并不会在赋值变量或输入参数定义时，定义变量或输入参数的数据类型，python会自动判断数据类型，并根据提供的运算返回计算结果，这样的处理方式可以减轻程序员思考的负担，也使得语言精简并富有弹性。

```python
def times(x,y):
    print('- - -'*3,'X={};y={}'.format(x,y))
    return (x*y)

print(times(5,7))
print(times([5],3))
print(times('polymorphism_',3))
```

```
- - - - - -  X=5;y=7
35
- - - - - -  X=[5];y=3
[5, 5, 5]
- - - - - -  X=polymorphism_;y=3
polymorphism_polymorphism_polymorphism_
```

可在此部分添加，“python 新特性，函数注解和参数数据类型注解”的介绍

参考资料:

Python 3 新特性：类型注解 - 知乎

https://zhuanlan.zhihu.com/p/37239021

typing —— 类型注解支持 — Python 3.10.6 文档

https://docs.python.org/zh-cn/3/library/typing.html

python 限定方法参数类型、返回值类型、变量类型等 - python 学习者 0 - 博客园
https://www.cnblogs.com/xxpythonxx/p/12198876.html

讨论和建议

目前来看，我们的代码基本上是基于 python3.5 的基本特性的，可以在前言介绍环境搭建的时候说明软件版本，或者，我们后面进行下软件版本升级，