# Python Cheat Sheet-4. 基本语句\_条件语句（if、elif、else），循环语句(for、 while)和列表推导式（comprehension）

?> PCS-4（    ）：4.1 缩进（indentation）和代码（语句）块（code block）；4.2 条件语句；4.3 循环语句\_for loops 模式；4.4 循环语句\_while 模式；4.5 列表推导式（comprehension）

知识点

描述

代码段

运算结果

备注

**4.1** 缩进（indentation）和代码（语句）块（code block）

代码语句的书写上，Python与C、C++等语言最大的不同是Python强制缩进，这样的好处是任何人书写的Python代码都具有统一的“格式”，无需另行规定基本的代码书写规范，方便代码传播和复用；同时，缩进的方式使得代码易读，不用特意去寻找语句块结束的标志，通过段落就可以轻易判断。

可以把一行代码看作一个基本单元，即为一个可执行的语句。语句通常是从上至下逐行执行，当定义函数和类之后，调用函数或方法则可以跳转执行语句，但跳转后仍是从上至下执行语句，结束调用方法后从之前调用的语句部分从新向下执行；或者，遇到条件语句，需要根据条件判断将要执行哪部分语句；或者，遇到循环语句和递归算法，将从循环位置反复执行同一语句或语句块，直至遇到结束条件；代码也可以不在同一个文件中，通过调用其它文件中的代码，语句的执行顺序也会跳转。

代码块是一段可以作为一个单元执行的一行或多行语句（程序文本），例如一个条件或循环的代码段，一个函数、一个类等；一个代码块也可以包含其它代码块，或调用执行其它代码块，因此对于代码块的定义相对比较宽泛，可以根据是否执行了一个任务来确定，无关任务的大小。

**4.2** 条件语句

条件语句的基本语法如下：

if test1:  
 statement1  
elif test2:  
 statement2  
elif test3:  
 statement3  
...  
else:  
 statements

同一条件代码块，if通常只用一次，elif可以执行0次或多次，else为不满足上述所有条件后，执行的语句，也可以不调用，但最好通过else表明其它情况如何处理。

下面应用了《漫画统计学》[1]“美味拉面畅销前50”上刊载的拉面馆的拉面价格数据，并将其存储在了ranmen\_price\_lst列表中。这里给了一个input()内置函数来在外部交互输入指令，这里的指令就是条件语句中的test部分，如果输入指令满足if或elif后的要求，则对应执行该语句缩进后的代码。如果都不满足则执行else后的语句，提示”Please enter the correct command:(“。

数据分析时很少用到input()函数来外部输入参数值，而通常使用交互图表，一般选择，[tkinter](https://docs.python.org/3/library/tkinter.html) GUI（Graphical User Inteface）①工具包，[Plotly](https://plotly.com/python/#controls)②自定义控件，[Pygame](https://www.pygame.org/news)游戏编程模块③，[gradio](https://gradio.app/)④以Web界面演示机器学习模型等既有成熟完善的库来处理。

import numpy as np  
ranmen\_price\_lst=[700,850,600,650,980,750,500,890,880,700,890,720,680,650,790,670,680,900,880,720,850,700,780,850,750,  
 80,590,650,580,750,800,550,750,700,600,800,800,880,790,790,780,600,690,680,650,890,930,650,777,700]  
  
command=input("Enter your command('mean,std,max,min,median'):")  
  
if command=='mean':  
 print(np.mean(ranmen\_price\_lst))  
elif command=='std':  
 print(np.std(ranmen\_price\_lst))  
elif command=='max':  
 print(np.max(ranmen\_price\_lst))  
elif command=='min':  
 print(np.min(ranmen\_price\_lst))   
elif command=='median':  
 print(np.median(ranmen\_price\_lst))  
else:  
 print("Please enter the correct command:(")

Enter your command(‘mean,std,max,min,median’):

* 嵌套条件语句（Nested if statements）

一个条件下可以再嵌套多个条件，例如下述代码外层条件语句是判断变量price\_x值是否属于列表ranmen\_price\_lst，如果属于则打印该价格，并执行嵌套条件语句块，判断该值是否大于或者小于等于平均价格；回到外层条件，如果price\_x值不属于列表ranmen\_price\_lst，则寻找最近值，这里使用了一个lambda匿名函数计算绝对值的功能，并将其作为min(iterable, \*[, default=obj, key=func])函数的key参数值，即比较的是匿名函数所定义返回值（差值的绝对值）的最小值，并返回对应绝对值最小的价格列表中的值。打印该值，同时执行嵌套条件，与if下嵌套条件一样来判断大于或者小于等于价格均值。

import numpy as np  
ranmen\_price\_lst=[700,850,600,650,980,750,500,890,880,700,890,720,680,650,790,670,680,900,880,720,850,700,780,850,750,  
 80,590,650,580,750,800,550,750,700,600,800,800,880,790,790,780,600,690,680,650,890,930,650,777,700]  
  
price\_x=200.68  
  
abs\_difference\_func=lambda value:abs(value-price\_x)  
if price\_x in ranmen\_price\_lst:  
 print(price\_x)  
 if price\_x>np.mean(ranmen\_price\_lst):  
 print('The price is higher than the average price.')  
 else:  
 print('The price is lower than the average price.')   
else:  
 print('%.3f is no in ranmen\_price\_lst.'%price\_x)  
 closest\_value=min(ranmen\_price\_lst,key=abs\_difference\_func)  
 print('the nearest value to %s is %s.'%(price\_x,closest\_value))  
 # 定义了与if中同样的功能代码块，不过将price\_X替换为closest\_value  
 price\_mean=np.mean(ranmen\_price\_lst)  
 if closest\_value>price\_mean:  
 print('The price is higher than the average price.')  
 else:  
 print('The price is lower than the average price %s.'%price\_mean)

200.680 is no in ranmen\_price\_lst.  
the nearest value to 200.68 is 80.  
The price is lower than the average price 729.34.

+ 尝试下定义函数的优势（下一PCS预热）

如果要重复比较不同值和不同列表值的关系，返回列表最近值，那么上述的代码使用起来不方便，还会很繁琐，也很难分享，不易被其它程序调用（代码复用），因此需要将这一功能代码块定义为函数形式。从下述转换为函数后的代码可以观察到几个需要注意的点：

1. 关于变量名和函数名的命名，可以发现下述的变量名并没有延续上一代码段定义的各类名称，这包括变量名、函数名，及参数名。因为该函数代码的主要功能是比较一个值和一个列表中的值的关系，给的数据不一定是拉面价格，因此函数中各个名称的定义应该尽量通用化，主要表述和反应定义函数所要解决的内容或问题；
2. 对于重复的代码段或变量，通常不会重复书写，例如上述代码中内层的两个判断与均值大小的条件语句块重复书写，因此将其定义为单独的匿名函数comparisonOF2values方便调用。也可以看到列表均值的计算np.mean(ranmen\_price\_lst)被书写了两次，可以将该计算语句放置于条件代码块之外赋值给单独变量名，之后只需要用该变量就可，避免重复较长语句的书写；
3. 函数内的打印语句文字，同变量名的定义一样应通用化。

def value2values\_comparison(x,lst):  
 import numpy as np  
   
 lst\_mean=np.mean(lst)  
 abs\_difference\_func=lambda value:abs(value-x)  
 comparisonOF2values=lambda v1,v2:print('x is higher than the average %s of the list.'%lst\_mean) if v1>v2 else print('x is lower than the average %s of the list.'%lst\_mean)   
   
 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  
   
ranmen\_price\_lst=[700,850,600,650,980,750,500,890,880,700,890,720,680,650,790,670,680,900,880,720,850,700,780,850,750,  
 80,590,650,580,750,800,550,750,700,600,800,800,880,790,790,780,600,690,680,650,890,930,650,777,700]  
price\_x=200.68   
price\_x\_closestValue=value2values\_comparison(price\_x,ranmen\_price\_lst)   
print(price\_x\_closestValue)

200.680 is not in the list.  
the nearest value to 200.68 is 80.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
x is lower than the average 729.34 of the list.  
80

price\_x\_closestValue=value2values\_comparison(890,ranmen\_price\_lst)   
print(price\_x\_closestValue)

890 in the given list.  
x is higher than the average 729.34 of the list.  
890

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 729.34 of the list.  
88

* 三元表达式（Ternary Expression）

形如variable=v1 if test else v2的语句即为三元表达式，该语句等同于：

if test:  
 variable=v1  
else:  
 variable=v2

三元表达式通常用于较简单的条件语句，因为用一行表述较之多行书写更为便捷；但是对于较长，较复杂的条件语句则建议按常规缩进书写。

v1=33.5  
v2=78.3  
max\_v1Nv2=v1 if v1>v2 else v2  
print(max\_v1Nv2)

78.3

* 用;连接简单的语句为一行

如果语句非常的简单，则可以使用;将其连接置于一行。下述示例还包括了一个简单的三元表达。

x=3.5;y=7.8;print(x if x>y else y)

7.8

* 条件语句与比较运算符、逻辑运算符和成员运算符

条件语句通常会用逻辑运算符连接多个比较运算符或其他条件，实现条件判断的目的。

a,b,c=23,57,68  
lst=[23,77,96]  
  
if a<b and c>b:  
 print('a is less than c.')  
if a<b or b>c:  
 print('b is not sure greater than c.')  
   
if a not in lst:  
 print('a not in list')  
else:  
 print('a in list')  
   
if a in lst:  
 print('a in list')  
else:  
 print('a not in list')

a is less than c.  
b is not sure greater than c.  
a in list  
a in list

很多变量可以直接用于条件之后，简化条件书写，例如下述是否为空列表的判断，一个直接使用变量，一个则计算列表的长度来判断是否为0，从而证实是否为空列表。

if 1:  
 print('return true.')  
if 0:  
 print('This statement will not be executed.')  
else:  
 print('It is 0.')  
   
if True:  
 print('This statement is executed!')  
  
if '':  
 print('Empty string...')  
else:  
 print('This test is an empty string.')   
  
empty\_lst=[]  
if not empty\_lst:  
 print('This is an empty list!!!')  
if len(empty\_lst)==0:  
 print('This is an empty list!!!')  
   
lst=[3,4,5]  
if lst:  
 print('This is not an empty list!!!')

return true.  
It is 0.  
This statement is executed!  
This test is an empty string.  
This is an empty list!!!  
This is an empty list!!!  
This is not an empty list!!!

**4.3** 循环语句\_for loops 模式

for循环的基本语法为：

for target in object:  
 statements  
else: # 可选部分  
 statements # 如果for循环没有被终断（break）

object 为序列或者任何可迭代的对象，例如strings、lists、truples、dict和其它内置可迭代（iterable）对象，如zip()，map()等返回的可迭代对象。

* 循环列表与enumerate()

在解释循环语句时，使用了[The Cityscapes Dataset](https://www.cityscapes-dataset.com/)⑤的标签数据（Cityscapes数据集集中于城市街道场景的语义解释（semantic understanding），如图像语义分割、对象检测等深度学习模型的训练，这非常适用于对城市空间内容的分析）。为了方便数据的处理，将标签数据存储为namedtuple数据格式（结构），namedtuple是由Python内置库[collections](https://docs.python.org/3/library/collections.html)⑥提供，一般翻译为具名元组。Python内置数据结构tuple（元组）, 不能像表格抬头（例如pandas的DataFrame数据结构）一样为数据指定字段名（列名），因此不能够很好的管理数据，这包括对于数据的存储更新和提取，因此collections.namedtuple类型的数据结构就解决了这个问题。namedtuple(typename, field\_names, \*, rename=False, defaults=None, module=None)，定义namedtuple的输入参数中typename为元组的名称，filed\_names为元组中元素的名称，rename为如果元素名称含有Python的关键字，则必须配置该参数为rename=True。使用namedtuple首先定义一个namedtuple对象，例如示例中的Label对象，然后应用该对象定义不同的namedtuple变量存储数据，例如label\_building和label\_caravan。可以通过类属性值（object.attribute）的途径读取字段值，及更新字段值。

[collections](https://docs.python.org/3/library/collections.html)库提供有专门的容器数据类型（container datatype），即数据结构，为dict、 list、 set 和tuple提供了可替代数据存储管理方式。

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)

<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')  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
building  
11  
construction  
(70, 70, 70)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
caravan  
29  
vehicle  
(0, 0, 90)  
schooner  
(30, 30, 60)  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
{'name': 'caravan', 'id': 29, 'trainID': 255, 'category': 'schooner', 'categoryID': 7, 'hasInstances': True, 'igoreInEval': True, 'color': (30, 30, 60)}

cityscapes的标签数据以namedtuple列表形式存储，列表中的每一个值就为一个namedtuple对象，具有相同的字段名称。通过namedtuple读取值的方法，并配合列表推导式很容易提取各个字段名为单独的列表，或两个到多个字段名提取为字典的模式，建立不同字段之间的映射。为了清晰的观察数据，在输入数据时，有意识的将其各列对其，每一列就为一个具有名称的元素，例如name字段列对齐方便观察名称。注意，这里修改了color字段的值，使用了[ANSI Escape Sequences/Codes，ANSI code](https://en.wikipedia.org/wiki/ANSI_escape_code)⑦，可翻译为ANSI转义序列/代码，ANSI code用于控制光标位置、颜色和字体样式，也包括视频文本终端或终端仿真器。某些字节序列（大多数以 ASCII 转义字符和括号字符开头）被嵌入到文本中。 终端将这些序列解释为命令，而不是逐字显示的文本。

在Python解释器中显示字体的颜色，包括16色模式（8个字体颜色和8个背景颜色）和256色模式。16色字符串格式化的模式示例为print('\033[2;31;43m CHEESY \033[0;0m')，或print('\x1b[2;31;43m CHEESY \x1b[0;0m')，其中\033[0;0m')是重置终端打印颜色为默认，防止继续打印设置的颜色，各字符含义如图：

256色打印方式例如print("\033[48;5;236m\033[38;5;231mStack \033[38;5;208mAbuse\033[0;0m")， 各字符含义如图：

参考[How to Print Colored Text in Python](https://stackabuse.com/how-to-print-colored-text-in-python/)⑧；[American National Standards Institute , ANSI](https://www.ansi.org/)

Label=namedtuple('label',['name','id','trainId','category','catId','hasInstances','igoreInEval','color'])  
  
labels = [  
 # name id trainId category catId hasInstances ignoreInEval color  
 Label( 'unlabeled' , 0 , 255 , 'void' , 0 , False , True , (0, 30, 47) ),  
 Label( 'ego\_vehicle' , 1 , 255 , 'void' , 0 , False , True , (0, 31, 46) ),  
 Label( 'rectification\_border' , 2 , 255 , 'void' , 0 , False , True , (0, 32, 45) ),  
 Label( 'out\_of\_roi' , 3 , 255 , 'void' , 0 , False , True , (0, 33, 44) ),  
 Label( 'static' , 4 , 255 , 'void' , 0 , False , True , (0, 34, 43) ),  
 Label( 'dynamic' , 5 , 255 , 'void' , 0 , False , True , (1, 35, 42) ),  
 Label( 'ground' , 6 , 255 , 'void' , 0 , False , True , (1, 36, 41) ),  
 Label( 'road' , 7 , 0 , 'flat' , 1 , False , False , (1, 37, 40) ),  
 Label( 'sidewalk' , 8 , 1 , 'flat' , 1 , False , False , (1, 30, 41) ),  
 Label( 'parking' , 9 , 255 , 'flat' , 1 , False , True , (1, 31, 42) ),  
 Label( 'rail\_track' , 10 , 255 , 'flat' , 1 , False , True , (2, 32, 43) ),  
 Label( 'building' , 11 , 2 , 'construction' , 2 , False , False , (2, 33, 44) ),  
 Label( 'wall' , 12 , 3 , 'construction' , 2 , False , False , (2, 34, 45) ),  
 Label( 'fence' , 13 , 4 , 'construction' , 2 , False , False , (2, 35, 46) ),  
 Label( 'guard\_rail' , 14 , 255 , 'construction' , 2 , False , True , (2, 36, 47) ),  
 Label( 'bridge' , 15 , 255 , 'construction' , 2 , False , True , (3, 37, 40) ),  
 Label( 'tunnel' , 16 , 255 , 'construction' , 2 , False , True , (3, 30, 42) ),  
 Label( 'pole' , 17 , 5 , 'object' , 3 , False , False , (3, 31, 43) ),  
 Label( 'polegroup' , 18 , 255 , 'object' , 3 , False , True , (3, 32, 44) ),  
 Label( 'traffic\_light' , 19 , 6 , 'object' , 3 , False , False , (3, 33, 45) ),  
 Label( 'traffic\_sign' , 20 , 7 , 'object' , 3 , False , False , (4, 34, 46) ),  
 Label( 'vegetation' , 21 , 8 , 'nature' , 4 , False , False , (4, 35, 47) ),  
 Label( 'terrain' , 22 , 9 , 'nature' , 4 , False , False , (4, 36, 40) ),  
 Label( 'sky' , 23 , 10 , 'sky' , 5 , False , False , (4, 37, 41) ),  
 Label( 'person' , 24 , 11 , 'human' , 6 , True , False , (4, 30, 43) ),  
 Label( 'rider' , 25 , 12 , 'human' , 6 , True , False , (5, 31, 44) ),  
 Label( 'car' , 26 , 13 , 'vehicle' , 7 , True , False , (5, 32, 45) ),  
 Label( 'truck' , 27 , 14 , 'vehicle' , 7 , True , False , (5, 33, 46) ),  
 Label( 'bus' , 28 , 15 , 'vehicle' , 7 , True , False , (5, 34, 47) ),  
 Label( 'caravan' , 29 , 255 , 'vehicle' , 7 , True , True , (5, 35, 42) ),  
 Label( 'trailer' , 30 , 255 , 'vehicle' , 7 , True , True , (5, 36, 41) ),  
 Label( 'train' , 31 , 16 , 'vehicle' , 7 , True , False , (0, 37, 40) ),  
 Label( 'motorcycle' , 32 , 17 , 'vehicle' , 7 , True , False , (1, 30, 44) ),  
 Label( 'bicycle' , 33 , 18 , 'vehicle' , 7 , True , False , (2, 32, 45) ),  
 Label( 'license\_plate' , -1 , -1 , 'vehicle' , 7 , False , True , (3, 33, 47) ),  
]  
  
print(labels[:3])

[label(name='unlabeled', id=0, trainId=255, category='void', catId=0, hasInstances=False, igoreInEval=True, color=(0, 30, 47)), label(name='ego\_vehicle', id=1, trainId=255, category='void', catId=0, hasInstances=False, igoreInEval=True, color=(0, 31, 46)), label(name='rectification\_border', id=2, trainId=255, category='void', catId=0, hasInstances=False, igoreInEval=True, color=(0, 32, 45))]

print('\x1b[2;31;43m CHEESY \x1b[0;0m')  
print("\033[48;5;236m\033[38;5;231mStack \033[38;5;208mAbuse\033[0;0m")

color\_lst为提取的ANSI code格式颜色数据列表，每一元组值对应text styles（字体类型，包括normal/0, bold/1, light/2, italicized/3, underlined/4, blink/5），foreground（Text）color（字体颜色，包括black/30、 red/31、 green/32、 yellow/33、 blue/34、 purple/35、 cyan/36、 white/37计8个颜色），及background color（字体的背景色，颜色同字体色，但是编号为40-47）。每次循环配置打印颜色值，并以颜色值为打印的字符串。

需要注意对于循环语句，通常包括多个值，甚至千万个待循环值，因此在书写代码时需要增加终止循环的代码if i==5:break，当变量i每次循环自增1到5时，调用break终止语句，跳出循环，待调试一次或几次循环无误后，再循环所有的值，避免等待运算时间，尤其需要花费10分钟以上，甚至多到几个小时或几天才能运算完的代码段。

下述示例代码保留了调试代码，除变量i和终止条件语句行外，调试时，要不断用print()函数查看变量值，确定变量值是否正确，及确认变量值结构，从而知晓后续代码行应用该变量的方式，或者通过后续要求的数据结构来处理数据为后续所用结构的类型（通常是用后者的方式判断和书写代码）。例如示例中通过print(color)来查看通过color=';'.join([str(i) for i in c])语句编写满足ANSI code要求的颜色格式，例如0;30;47，注意这里的数字为使用str()转换数字为字符串，满足使用%s格式化符号的要求；也可以不转换为字符串，而是使用%d的方式直接格式化。

color\_lst=[label.color for label in labels]  
print(color\_lst)  
  
# i=0 #调试用  
for c in color\_lst:   
 color=';'.join([str(i) for i in c])  
 #print(color) # 调试用  
 s='\x1b[%sm %s \x1b[0m' % (color,color)  
 print(s)  
 #if i==5:break # 调试用  
 #i+=1 # 调试用

[(0, 30, 47), (0, 31, 46), (0, 32, 45), (0, 33, 44), (0, 34, 43), (1, 35, 42), (1, 36, 41), (1, 37, 40), (1, 30, 41), (1, 31, 42), (2, 32, 43), (2, 33, 44), (2, 34, 45), (2, 35, 46), (2, 36, 47), (3, 37, 40), (3, 30, 42), (3, 31, 43), (3, 32, 44), (3, 33, 45), (4, 34, 46), (4, 35, 47), (4, 36, 40), (4, 37, 41), (4, 30, 43), (5, 31, 44), (5, 32, 45), (5, 33, 46), (5, 34, 47), (5, 35, 42), (5, 36, 41), (0, 37, 40), (1, 30, 44), (2, 32, 45), (3, 33, 47)]

enumerate(iterable, start=0)同时成对返回计数值和列表值，为一个枚举对象（return an enumerate object）。对列表执行enumerate()之后，在使用循环语句时可以将成对的计数值和元素值分别赋予给两个变量，如idx和c。如果并不在for循环中直接序列解包，赋予了一个变量，如i，则其如(0, (0, 30, 47))，仍然需要索引方式或序列解包方式（idx,c=i）提取值。

在打印字符时，如果不换行，可以增加参数end=''来避免起新行。

for idx,c in enumerate(color\_lst):   
 color=';'.join([str(i) for i in c])  
 s='\x1b[%sm %s \x1b[0m' % (color,idx)  
 print(s,end='')  
   
print('\n',"\_"\*50,'\n')   
for i in enumerate(color\_lst):   
 # print(i)  
 idx,c=i  
 color=';'.join([str(i) for i in c])  
 s='\x1b[%sm %s \x1b[0m' % (color,idx)  
 print(s,end='')

* 循环字典——键值对形式

以键值对形式循环字典是经常使用到的一种方式，可以很方便的同时提取键名和元素值，并在每一次循环中同时处理键名和元素值，再成对输出。例如建立了一个空字典name2NewColor\_dict，在每次成对循环原有字典值时，修改了颜色值（+1），并按键名和新颜色值成对存储在新建的字典中。

对于新键字典也在终端打印具有色彩的字符串，因为ANSI code格式颜色有值域，如果超出范围则可以看到对应部分不会发生颜色变化。

这里在调试代码时，直接使用了break语句，没有结合条件语句，因此该循环在调试时只执行一次循环。

name2color\_dict={label.name:label.color for label in labels}  
print(name2color\_dict)  
  
name2NewColor\_dict={}  
for name,color in name2color\_dict.items():  
 c=';'.join([str(i) for i in color])  
 s='\x1b[%sm %s \x1b[0m' % (c,name)  
 print(s,end='')   
   
 name2NewColor\_dict[name]=(i+1 for i in color)  
 #break  
print('\n',"\_"\*50,'\n')   
for name,color in name2NewColor\_dict.items():  
 c=';'.join([str(i) for i in color])  
 s='\x1b[%sm %s \x1b[0m' % (c,name)  
 print(s,end='')

{'unlabeled': (0, 30, 47), 'ego\_vehicle': (0, 31, 46), 'rectification\_border': (0, 32, 45), 'out\_of\_roi': (0, 33, 44), 'static': (0, 34, 43), 'dynamic': (1, 35, 42), 'ground': (1, 36, 41), 'road': (1, 37, 40), 'sidewalk': (1, 30, 41), 'parking': (1, 31, 42), 'rail\_track': (2, 32, 43), 'building': (2, 33, 44), 'wall': (2, 34, 45), 'fence': (2, 35, 46), 'guard\_rail': (2, 36, 47), 'bridge': (3, 37, 40), 'tunnel': (3, 30, 42), 'pole': (3, 31, 43), 'polegroup': (3, 32, 44), 'traffic\_light': (3, 33, 45), 'traffic\_sign': (4, 34, 46), 'vegetation': (4, 35, 47), 'terrain': (4, 36, 40), 'sky': (4, 37, 41), 'person': (4, 30, 43), 'rider': (5, 31, 44), 'car': (5, 32, 45), 'truck': (5, 33, 46), 'bus': (5, 34, 47), 'caravan': (5, 35, 42), 'trailer': (5, 36, 41), 'train': (0, 37, 40), 'motorcycle': (1, 30, 44), 'bicycle': (2, 32, 45), 'license\_plate': (3, 33, 47)}

* zip()和map()

zip(\*iterables)将多个列表（序列）返回为成对的值，for Loops可以逐个成对循环。下述示例使用了ANSI code格式颜色256模式，用catId作为背景颜色，未配置字体颜色，同时字符串之间增加了一个空格，断开名称。

name\_lst=[label.name for label in labels]  
catId\_lst=[label.catId for label in labels]  
  
for name,catId in zip(name\_lst,catId\_lst):  
 s='\033[48;5;%dm%s\033[0;0m '%(catId,name)  
 print(s,end='')  
 #break

map(func, \*iterables)输入参数func，自定义为lambda（ˈlamdə）函数，并给了两个输入参数x和n，返回一个元组，其中一个值保持不变（即name名称），另一个值加1（即用作颜色值的catId加1）。用for Loops可以逐个循环给定列表中的值经过map()中func参数函数的计算的返回值。

xAdd33\_Wname=lambda x,n:(n,x+33)  
for name,catId\_new in map(xAdd33\_Wname,catId\_lst,name\_lst):  
 # print( name,color)  
 s='\033[48;5;%dm%s\033[0;0m '%(catId\_new,name)  
 print(s,end='')   
 # break

* 循环range(len())

在组织列表中数值之间的运算模式时，经常通过列表的不同索引组合规律相加，相乘，或者任何更为复杂的计算来获取新的符合某一规律的列表值。例如regularAdd\_1为逐个计算列表中相邻两个值之和（即循环时为当前索引对应值和其后索引对应值之和）；regularAdd\_2为循环时，计算当前索引之前所有值之和；而regularAdd\_3则结合slicing，实现间隔相加的结果。

for idx in range(len(name\_lst)):  
 print('%d-%s;'%(idx,name\_lst[idx]),end='')  
  
import random  
lst=[random.randint(10,30) for i in range(10)]  
print('\n',"\_"\*50)  
print(lst)  
  
regularAdd\_1=[]  
regularAdd\_2=[]  
for i in range(len(lst)-1):  
 regularAdd\_1.append(lst[i]+lst[i+1])  
 regularAdd\_2.append(sum([lst[i] for i in range(i+1)]))  
  
regularAdd\_3=[]  
for i in range(1,len(lst)-1,2):  
 print(i)  
 regularAdd\_3.append(lst[i]+lst[i+2])  
  
print(regularAdd\_1,'\n',regularAdd\_2,'\n',regularAdd\_3)

0-unlabeled;1-ego\_vehicle;2-rectification\_border;3-out\_of\_roi;4-static;5-dynamic;6-ground;7-road;8-sidewalk;9-parking;10-rail\_track;11-building;12-wall;13-fence;14-guard\_rail;15-bridge;16-tunnel;17-pole;18-polegroup;19-traffic\_light;20-traffic\_sign;21-vegetation;22-terrain;23-sky;24-person;25-rider;26-car;27-truck;28-bus;29-caravan;30-trailer;31-train;32-motorcycle;33-bicycle;34-license\_plate;  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
[28, 24, 16, 19, 29, 20, 18, 11, 26, 21]  
1  
3  
5  
7  
[52, 40, 35, 48, 49, 38, 29, 37, 47]   
 [28, 52, 68, 87, 116, 136, 154, 165, 191]   
 [43, 39, 31, 32]

* 嵌套循环（nested for loops）

如果要获取嵌套列表、嵌套字典或这任何包含多个嵌套关系的数据结构，或者需要多个序列值运算处理，通常需要用嵌套循环逐层的拆解或嵌套循环不同序列运算。下述示例对同一个列表执行嵌套循环，并相加。为了方便观察数据关系，使用列表推导式切分为嵌套列表，并循逐行环打印，可以看到构建了一个列表中每一个元素与该列表全部值相加的矩阵，这可用于计算多个点列表，两两点之间距离的成本矩阵（起点-目的地 (OD) 成本矩阵），用于城市交通等分析。

lst=[random.randint(10,30) for i in range(10)]  
print(lst)  
  
regularAdd\_4=[]  
for i in range(len(lst)):  
 for j in range(len(lst)):  
 regularAdd\_4.append(lst[i]+lst[j])  
n=len(lst)  
  
regularAdd\_4\_chunks=[regularAdd\_4[i:i+n] for i in range(0,len(regularAdd\_4),n)]   
  
for sub\_lst in regularAdd\_4\_chunks:  
 print(sub\_lst)

[12, 18, 28, 24, 20, 14, 27, 25, 18, 23]  
[24, 30, 40, 36, 32, 26, 39, 37, 30, 35]  
[30, 36, 46, 42, 38, 32, 45, 43, 36, 41]  
[40, 46, 56, 52, 48, 42, 55, 53, 46, 51]  
[36, 42, 52, 48, 44, 38, 51, 49, 42, 47]  
[32, 38, 48, 44, 40, 34, 47, 45, 38, 43]  
[26, 32, 42, 38, 34, 28, 41, 39, 32, 37]  
[39, 45, 55, 51, 47, 41, 54, 52, 45, 50]  
[37, 43, 53, 49, 45, 39, 52, 50, 43, 48]  
[30, 36, 46, 42, 38, 32, 45, 43, 36, 41]  
[35, 41, 51, 47, 43, 37, 50, 48, 41, 46]

* for Loops中的星号\*（asterisk）

python中\*和\*\*，除了作为运算符或者字符串中的特殊字符外，通常作为前缀运算符（prefix operators），即在变量之前使用\*和\*\*运算符。\*和\*\*运算符具有丰富的用法。下述应用到循环中的示例是用\*来收集多个元素值。

for a,\*b,c in [(1,2,3,4,5),(5,6,7,8)]:  
 print(a,b,c)

1 [2, 3, 4] 5  
5 [6, 7] 8

**4.4** 循环语句\_while 模式

基于语法为：

while test:  
 statements  
else:  
 statements

或

while test:  
 statements  
 if test:break  
 if test: continue  
else:  
 statements

while的关键是处理循环停止的条件，一种是在while之后给出停止条件，例如while x<=10:，只要不满足条件就会停止循环，这时给出的条件变量通常在while代码块中参与执行相关的运算并会因为变化而不满足给出的条件后跳出循环，例如x+=1；或者在while代码块内给出条件语句，配合使用break停止循环，类似于for循环中break。

x=1  
while x<=10:  
 print(x,';',end='')  
 x+=1   
   
print('\n',"\_"\*50)  
x=1  
while True:  
 print(x,';',end='')  
 x+=1  
 if x>10:break  
   
print('\n',"\_"\*50)  
x=1  
for i in range(1000):  
 print(x,';',end='')  
 x+=1  
 if x>10:break

1 ;2 ;3 ;4 ;5 ;6 ;7 ;8 ;9 ;10 ;  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
1 ;2 ;3 ;4 ;5 ;6 ;7 ;8 ;9 ;10 ;  
 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
1 ;2 ;3 ;4 ;5 ;6 ;7 ;8 ;9 ;10 ;

将while True/break用于input()函数。

while True:  
 value=input('To calculate the square root. Enter a number:(enter "stop" to stop the operation)')  
 if value=='stop':break  
 import math  
 print("The square root of \033[1;31;40m%s\033[0m is \033[1;31;40m%.2f\033[0m."%(value, math.sqrt(float(value))))

可以给定不同的要求，使用多个elif的方式分别计算。不过对于这种在终端交互的方式通常是使用既有的交互图表库。

while True:  
 import math  
 command=input('sqrt, power,sum, or stop:')  
 if command=='sqrt':  
 number=input('Enter 1 number:')  
 print('square root=%.2f'%math.sqrt(float(number)))  
 elif command=='power':  
 number=input('Enter 1 number:')  
 print('power=%.2f'%math.pow(float(number),2))  
 elif command=='sum':  
 numbers=input('Enter 2 numbers, separated by commas:')  
 print('sum=%.2f'%sum([float(v) for v in numbers.split(",")]))  
 elif command=='stop':break

sqrt, power,sum, or stop: sqrt  
Enter 1 number: 5  
  
  
square root=2.24  
  
  
sqrt, power,sum, or stop: power  
Enter 1 number: 5  
  
  
power=25.00  
  
  
sqrt, power,sum, or stop: sum  
Enter 2 numbers, separated by commas: 5,6  
  
  
sum=11.00  
  
  
sqrt, power,sum, or stop: stop

将while True/break用于文件读取。注意在书写代码时是将poi\_lst.append(POI.\_make(line.split(",")))语句写于if not line:break中断语句之后，先判断是否已到行末尾（读取的是否为空行），否则会提示运行错误，因为空行不能执行line.split(",")运算。

xian\_poi\_fn='./data/xian\_poi.csv' # 存储有西安POI, point of interesting兴趣点数据  
f=open(xian\_poi\_fn,'r', encoding="utf-8") #  
  
from collections import namedtuple  
POI=namedtuple('POI',['idx','unknown','name','lat','lon','category','score','price','x','y'])  
poi\_lst=[]  
while True:  
 line=f.readline()   
 if not line:break  
 poi\_lst.append(POI.\_make(line.split(",")))  
 # break #调试用  
f.close()  
print('The length of the poi list is %d.'%len(poi\_lst))  
print("\_"\*50)  
for i in poi\_lst[:3]:print(i,'\n')

The length of the poi list is 13732.  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
POI(idx='1', unknown='0101000020897F000008D599CB26E312418530CECF16EB4C41', name='美香源', lat='34.23709808337344', lon='108.93100212046282', category='美食;中餐厅', score='0', price='', x='309449.6988290106', y='3790381.623479905\n')   
  
POI(idx='2', unknown='0101000020897F0000B038F4CF9BE31241389AD606BFEC4C41', name='雷记澄城水盆羊肉(红樱路店)', lat='34.244750060429915', lon='108.93113243785623', category='美食;中餐厅', score='3.9', price='20.0', x='309478.95308006834', y='3791230.053424146\n')   
  
POI(idx='3', unknown='0101000020897F00005C24B17F97E3124160F86A81E5EC4C41', name='段府农家菠菜面(红缨路店)', lat='34.245443456204875', lon='108.93110375582032', category='美食;中餐厅', score='4', price='', x='309477.8746991807', y='3791307.011076972\n')

**4.5** 列表推导式（comprehension）

print print('Hello World!')代码。

print('Hello World!')

Hello World!

因为使用for loops会占据多行，而只要不是复杂的循环，使用列表推导式是优先选择，不仅会简化代码行，同时书写起来相对要便捷的多。其语法为newlist=[expression for item in iterable if condition == True]，或者newlist=[expression1 if condition == True else expression2 for item in iterable]

lst=[9,8,7,6,5,4,3]  
for i in range(len(lst)):  
 lst[i]+=10  
print(lst)  
  
lst=[9,8,7,6,5,4,3]  
lst\_A=[i+10 for i in lst]  
print(lst\_A)

[19, 18, 17, 16, 15, 14, 13]  
[19, 18, 17, 16, 15, 14, 13]

列表推导式也可以转换具有条件语句的for loops。

lst=[9,8,7,6,5,4,3]  
for i in range(len(lst)):  
 if i%2==0:lst[i]+=10  
 else:lst[i]+=100  
print(lst)  
  
lst=[9,8,7,6,5,4,3]  
lst\_B=[lst[i]+10 if i%2==0 else lst[i]+100 for i in range(len(lst))]  
print(lst\_B)

[19, 108, 17, 106, 15, 104, 13]  
[19, 108, 17, 106, 15, 104, 13]

将嵌套循环转换为列表推导式计算。

monogram\_lst=[]  
for i in 'abcd':  
 for j in 'hijk':  
 monogram\_lst.append('%s-%s'%(i,j))   
print(monogram\_lst)   
  
monogram\_lst\_A=['%s-%s'%(i,j) for i in 'abcd' for j in 'hijk']  
print(monogram\_lst\_A)

['a-h', 'a-i', 'a-j', 'a-k', 'b-h', 'b-i', 'b-j', 'b-k', 'c-h', 'c-i', 'c-j', 'c-k', 'd-h', 'd-i', 'd-j', 'd-k']  
['a-h', 'a-i', 'a-j', 'a-k', 'b-h', 'b-i', 'b-j', 'b-k', 'c-h', 'c-i', 'c-j', 'c-k', 'd-h', 'd-i', 'd-j', 'd-k']

列表推导式用于dict数据结构。将hasInstances字段的布尔值转换为0或1。

name2hasInstances\_dict={label.name:label.hasInstances for label in labels}  
print(name2hasInstances\_dict)  
print({k:int(v) for k,v in name2hasInstances\_dict.items()})

{'unlabeled': False, 'ego\_vehicle': False, 'rectification\_border': False, 'out\_of\_roi': False, 'static': False, 'dynamic': False, 'ground': False, 'road': False, 'sidewalk': False, 'parking': False, 'rail\_track': False, 'building': False, 'wall': False, 'fence': False, 'guard\_rail': False, 'bridge': False, 'tunnel': False, 'pole': False, 'polegroup': False, 'traffic\_light': False, 'traffic\_sign': False, 'vegetation': False, 'terrain': False, 'sky': False, 'person': True, 'rider': True, 'car': True, 'truck': True, 'bus': True, 'caravan': True, 'trailer': True, 'train': True, 'motorcycle': True, 'bicycle': True, 'license\_plate': False}  
{'unlabeled': 0, 'ego\_vehicle': 0, 'rectification\_border': 0, 'out\_of\_roi': 0, 'static': 0, 'dynamic': 0, 'ground': 0, 'road': 0, 'sidewalk': 0, 'parking': 0, 'rail\_track': 0, 'building': 0, 'wall': 0, 'fence': 0, 'guard\_rail': 0, 'bridge': 0, 'tunnel': 0, 'pole': 0, 'polegroup': 0, 'traffic\_light': 0, 'traffic\_sign': 0, 'vegetation': 0, 'terrain': 0, 'sky': 0, 'person': 1, 'rider': 1, 'car': 1, 'truck': 1, 'bus': 1, 'caravan': 1, 'trailer': 1, 'train': 1, 'motorcycle': 1, 'bicycle': 1, 'license\_plate': 0}

注释（Notes）：

① tkinter，是Tcl/Tk GUI工具包的标准Python接口（<https://docs.python.org/3/library/tkinter.html>）。

② Plotly，是一个通过数据应用实现数据驱动决策的实践者，构建有支持多种编程语言的图表库（<https://plotly.com/python/#controls>）。

③ Pygame，是一套跨平台的Python模块，用于编写视频游戏。它包括计算机图形和声音库，旨在与Python编程语言一起使用。（<https://www.pygame.org>）。

④ gradio，是演示机器学习模型的最快方式，提供一个友好的网络界面，任何人都可以在任何地方使用该模型（<https://gradio.app/>）。

⑤ The Cityscapes Dataset，大规模数据集，包含了50个不同城市的街道场景中记录的各种立体（stereo）视频序列，除了20,000个弱注释帧的大集合外，还有5,000帧的高质量像素级注释（<https://www.cityscapes-dataset.com/>）。

⑥ collections，这个模块实现了专门的容器数据类型，提供了Python的通用内置容器（数据结构），dict、list、set和tuple的替代格式（<https://docs.python.org/3/library/collections.html>）。

⑦ ANSI Escape Sequences/Codes，ANSI code，是一种带内信号（in-band signaling）标准，用于控制视频文本终端和终端仿真器上的光标位置、颜色、字体风格和其他选项。某些字节序列，大多数以ASCII转义字符和括号字符开始，被嵌入文本中。终端将这些序列解释为命令，而不是逐字显示的文本。（<https://en.wikipedia.org/wiki/ANSI_escape_code>）。

⑧ How to Print Colored Text in Python，（<https://stackabuse.com/how-to-print-colored-text-in-python/>）。

参考文献（References）:

[1] (日)高桥 信著,陈刚译.株式会社TREND-PRO漫画制作.漫画统计学[M].科学出版社.北京,2019.8.

PC4-ipynb download