# Python Cheat Sheet-5. 函数（function）、作用域（scope）与命名空间（namespace）、参数（arguments）

?> PCS-5（    ）：5.1 定义函数；5.2 作用域（scope）和命名空间（namespace）；5.3 工厂函数（Factory Functions）；5.4 函数的输入参数（Arguments）；5.5 函数定义综合实验-自定义箱型图打印样式

知识点

描述

代码段

运算结果

备注

**5.1** 定义函数

不定义函数，按行执行代码可以完成一个任务，但是如果要用该组代码完成不止一次同样的计算任务，重复的复制代码，或者调整位于不同位置行的输入参数，则异常繁琐，并容易发生错误；如果所要处理的任务较为复杂，例如完成遗传算法或蚁群算法等复杂的任务，采取逐行执行代码的方式不是很现实。为了让代码书写更流畅、精简、易读，方便调试与减少出错率，及代码的迁移，重复调用，函数定义必不可免。

函数定义的基本语法如下：

def name(arg1,arg2,...,argN):   
 statemetns  
 return value # 可以返回值，也可以移除该行，则返回值默认为空（None）

下述定义了两个小函数，通过获取当前时间和代码运行后时间与其差值，用于较大计算量代码运行时间长度的计算。start\_time()函数没有输入参数，定义函数名为start\_time，在函数内部，调入了一个时间模块datetime，使用datetime.datetime.now()方法获取当前时间，并赋值给变量start\_time， 打印当前时间，及return start\_time返回当前时间变量值。调用执行该函数时，如果需要接收函数的返回值，则赋值给一个变量。

第2个函数duration(start\_time)有一个输入参数，为时间格式的值。函数块内部，同样计算了当前时间并赋值给变量end\_time，由(end\_time-start\_time).seconds/60方法计算时间差，并将时间格式转换为易读的分钟形式。这个函数没有提供返回值。

是在函数内，还是在函数外调用模块（module，库的部分），需要根据具体情况衡量利弊。例如，如果模块调入在文件开始，则只需要调入一次；但是如果并不是所有函数均调用该模块，而该模块可能会与其它模块（库的安装）发生冲突，或者模块自身体量庞大，而使用者只使用该模块中的其它方法时，则函数调入在特定的函数内执行会更合理。

def start\_time():  
 import datetime  
   
 start\_time=datetime.datetime.now()  
 print("start time:",start\_time)  
 return start\_time  
  
def duration(start\_time):  
 import datetime  
   
 end\_time=datetime.datetime.now()  
 print("end time:",end\_time)  
 duration=(end\_time-start\_time).seconds/60  
 print("Total time spend:%.2f minutes"%duration)

s\_t=start\_time()  
print(type(s\_t))

start time: 2022-08-07 19:19:12.115484  
<class 'datetime.datetime'>

duration(s\_t)

end time: 2022-08-07 19:19:18.036481  
Total time spend:0.08 minutes

下例为应用上述组合函数的一个场景，计算代码for i in range(10\*\*8):value=i运行的时间。

s\_t=start\_time()  
for i in range(10\*\*8):value=i  
duration(s\_t)

start time: 2022-08-07 20:25:19.170276  
end time: 2022-08-07 20:25:29.551021  
Total time spend:0.17 minutes

* 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。

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\_

如果不区分参数输入的数据类型，常常不容易直接判断参数类型是什么，因此可以增加形如def func(arg1:type,arg2:type)->type的函数注释，其中:type为参数类型，->type为返回值类型。注意，Python解释器并不会因为这些注解而提供额外的校验，没有任何的类型检查工作，这些类型注解加不加，对代码来说没有任何影响。

def times\_type(x:float,y:float)->float:  
 print('- - '\*3,'X={};y={}'.format(x,y))  
 return (x\*y)  
  
print(times\_type(5,7))  
print(times\_type([5],3))  
print(times\_type('polymorphism\_',3))

- - - - - - X=5;y=7  
35  
- - - - - - X=[5];y=3  
[5, 5, 5]  
- - - - - - X=polymorphism\_;y=3  
polymorphism\_polymorphism\_polymorphism\_

* 函数定义的诸多考量——定义描述性统计函数

这是一个略微复杂些的例子，用于对给定的一组数据做描述性统计分析。下述示例结果是书写调试，增加对函数功能、输入和输出参数说明后的最终定稿代码段。实际编写代码过程是一个反复修改，调试的过程，这不仅包括函数功能实现的内容的修改、增补或削减；也包括功能实现过程中，结构逻辑的调整，例如是否用字典的形式先计算所有的统计量，如果给定了measure的方法，这样的逻辑设计将会增加无关的计算量；再者，选用何种的方式计算这些统计量，不调用库，而自行根据公式编写计算流程，或调入哪个库计算，这可以用[math](https://docs.python.org/3/library/math.html)，[statistics](https://docs.python.org/3/library/statistics.html)，[NumPy](https://numpy.org/)，[pandas](https://pandas.pydata.org/)，[SciPy](https://scipy.org/)等①任何方法，如果需要调入多个库，往往需要综合考虑，尽量减少库的调入数量；而返回值的形式也需要认真考虑，如果需要返回值参与到其它计算中，则字符串表述形式的返回形式是不合适的，需要考虑直接返回具体的数值。对于函数的定义，在实际的数据分析时，考虑的内容会因为所要解决问题的不同而存在差异，需要具体情况具体分析。

在PCS\_4中，例举的函数是将一段代码调整为函数定义，这个过程需要注意对变量名的重新定义，和重复代码段的调整。而该处的函数定义则直接定义函数，不涉及代码的转换，变量名的定义时，就已经考量到命名的一般性，以及上述所考量的内容。因此，在实际代码书写时，非必要单行时，则直接以函数形式定义，建议避免由逐行再转换为函数形式，因为转换过程会耗费不必要的精力。

def descriptive\_statistics(data,measure=None,decimals=2):  
 '''  
 计算给定数值列表的描述性统计值，包括数量、均值、标准差、方差、中位数、众数、最小值和最大值。  
   
   
 Parameters  
 ----------  
 data : list(numerical)  
 待统计的数值列表.  
 measure : str, optional  
 包括：'count', 'mean', 'std', 'variance', 'median', 'mode', 'min', 'max'. The default is None.  
 decimals : int, optional  
 小数位数. The default is 2.  
  
 Returns  
 -------  
 dict  
 如果不给定参数measure，则以字典形式返回所有值；否则返回给定measure对应值的表述字符串.  
  
 '''  
 import statistics  
   
 d\_s={  
 'count':len(data), # 样本数  
 'mean':round(statistics.mean(data),decimals), # 均值  
 'std':round(statistics.stdev(data),decimals), # 标准差  
 'variance': round(statistics.variance(data),decimals), # 方差   
 'median':statistics.median(data), # 中位数  
 'mode':statistics.mode(data), # 众数  
 'min':min(data), # 最小值  
 'max':max(data), # 最大值   
 }  
   
 if measure:  
 return '{}={}'.format(measure,d\_s[measure])  
 else:  
 return d\_s  
   
ranmen\_price\_lst=[700,850,600,650,980,750,500,890,880,  
 700,890,720,680,650,790,670,680,900,  
 880,720,850,700,780,850,750,780,590,  
 650,580,750,800,550,750,700,600,800,  
 800,880,790,790,780,600,690,680,650,  
 890,930,650,777,700]  
d\_s\_1=descriptive\_statistics(ranmen\_price\_lst)  
print(d\_s\_1)  
print('--'\*30)  
d\_s\_2=descriptive\_statistics(ranmen\_price\_lst,'std')  
print(d\_s\_2)  
d\_s\_3=descriptive\_statistics(ranmen\_price\_lst,measure='mean',decimals=1)  
print(d\_s\_3)

{'count': 50, 'mean': 743.34, 'std': 108.26, 'variance': 11720.64, 'median': 750.0, 'mode': 700, 'min': 500, 'max': 980}  
------------------------------------------------------------  
std=108.26  
mean=743.3

**5.2** 作用域（scope）和命名空间（namespace）

* 对作用域的描述

Python中变量的访问权限取决于该变量赋值的位置，这个位置所在的代码块称为该变量所属的作用域。首先需要明确作用域是一个嵌套的关系，如下图；同时需要明确这作用域的嵌套关系作用于一个文件（或模块，module），如果要使用另一个文件内定义的方法（函数），或属性（变量），则需要使用import方法调入模块。整个模块（或文件）即为内置作用域，包含Python自身内置函数所包括的各类函数或方法名称，例如示例中的print()函数，及open()，range()等，可以通过import builtins;print(dir(builtins))查看所有内容。内置作用域可以访问函数外定义的变量，但是无法访问函数内定义的变量，除非使用global关键字，将在函数内（局部作用域）定义的变量声明为全局变量，从而函数外也可以访问。闭包局部作用域是函数内作用域，只有函数内对象可以访问的变量，但是如果含有嵌套函数，则嵌套函数形成一个局部作用域，嵌套函数外的对象无法访问，并以此类推，除非使用nonlocal关键字，当将嵌套局部作用域外定义的变量在嵌套局部作用域内更新时，嵌套局部作用域外的变量值也会发生对应改变。从上述描述可以注意到，作用域是从外层作用域中剥离内层作用域的过程，外层作用域无法访问内层作用域变量，而内层作用域可以访问外层作用域变量。如果外层作用域要访问内层作用域，则需要是使用global或nonlocal关键字，其中global可以在任何内层作用域中使用，将变量声明为全局变量；但nonlocal仅作用于函数内使用，并需要在外层存在有该变量名，只是在内层操作更新该变量时，外层的变量对应更新。

通常将这4个作用域Built-in Scope、Global Scope、 Enclosed Local Scope和Nested Local Scope缩写为B、G、E、L。

* 命名空间

命名空间是为了防止项目中命名冲突的一种机制。如果代码量较大，项目内定义的变量名较多，必然容易发生重复命名的事件，而命名空间与作用域对应，即为命名空间，不同作用域之间变量名没有关联，可以用同样的名称在不同作用域中定义，但是需要注意不能与Python内置函数名称和关键字名称同。可以将命名空间对应作用域分为内置名称（Built-tin Names），全局名称（Global Names）和局部名称（Local Names）。命名空间的查找顺序为局部名称到全局名称到内置名称，如果找不到变量，则会引发NameError异常。

内层作用域要访问外层作用域的变量，最好是通过函数的输入参数调入，而不是直接使用，避免在局部代码迁移时发生变量名未找到的错误，也能够更好更清晰的组织代码结构，避免内外层命名混乱，削弱了代码易读性。

**LEGB Rule**

G\_1=3.5  
G\_2=6.0  
  
def outer():  
 L\_E\_1=5.3  
   
 global L\_E\_2  
 L\_E\_2=7.9  
   
 L\_E\_N\_3=78  
   
 def inner():  
 L\_N\_1=6.7   
   
 global L\_N\_2  
 L\_N\_2=5.5  
   
 nonlocal L\_E\_N\_3  
 L\_E\_N\_3+=1  
   
 print('Nested Scope:\nG\_1={};\nG\_2={};\nL\_E\_1={};\nL\_E\_2={};\nL\_N\_1={}\nL\_N\_2={};\nL\_N\_3={}'.format(G\_1,G\_2,L\_E\_1,L\_E\_2,L\_N\_1,L\_N\_2,L\_E\_N\_3))  
   
 print("\_"\*50)  
 print('L\_E\_N\_3={} in Enclosd Scope.'.format( L\_E\_N\_3))  
 inner()  
   
outer()  
  
print('--'\*30)  
print('Global Scope:\nG\_1={};\nG\_2={};\nL\_E\_2={};\nL\_N\_2={}'.format(G\_1,G\_2,L\_E\_2,L\_N\_2))

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
L\_E\_N\_3=78 in Enclosd Scope.  
Nested Scope:  
G\_1=3.5;  
G\_2=6.0;  
L\_E\_1=5.3;  
L\_E\_2=7.9;  
L\_N\_1=6.7  
L\_N\_2=5.5;  
L\_N\_3=79  
------------------------------------------------------------  
Global Scope:  
G\_1=3.5;  
G\_2=6.0;  
L\_E\_2=7.9;  
L\_N\_2=5.5

外层作用域无法访问内层作用域中定义的变量

print(L\_E\_1)

---------------------------------------------------------------------------  
  
NameError Traceback (most recent call last)  
  
Input In [30], in <cell line: 1>()  
----> 1 print(L\_E\_1)  
  
  
NameError: name 'L\_E\_1' is not defined

import builtins  
print(dir(builtins))

['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning', 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError', 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning', 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError', 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError', 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError', 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError', 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError', 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError', 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError', 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError', 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True', 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning', 'ValueError', 'Warning', 'WindowsError', 'ZeroDivisionError', '\_\_IPYTHON\_\_', '\_\_build\_class\_\_', '\_\_debug\_\_', '\_\_doc\_\_', '\_\_import\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'abs', 'all', 'any', 'ascii', 'bin', 'bool', 'breakpoint', 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'display', 'divmod', 'enumerate', 'eval', 'exec', 'execfile', 'filter', 'float', 'format', 'frozenset', 'get\_ipython', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance', 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max', 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print', 'property', 'range', 'repr', 'reversed', 'round', 'runfile', 'set', 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type', 'vars', 'zip']

* 局部作用域对全局作用域变量的更新

在配置参数时，参数值往往需要配置为不同值观察比较计算结果，例如如果参数值为全局变量，可以定义下述示例函数来更新该变量，使得代码易读、不容易发生混淆，尤其避免不容易查找到的错误出现。

g\_var=5927  
  
def setGvar(new\_Gvar):  
 global g\_var  
 g\_var=new\_Gvar  
  
setGvar(9527)   
print(g\_var)

9527

**5.3** 工厂函数（Factory Functions）

工厂函数类似于类方法（Class，称为工厂方法）的本质，可以实例化外层函数，再显式的调用嵌套函数，只是只能返回一个嵌套函数，而不能并行多个内层函数。整个过程为当调用外层函数，并将其赋值给一个变量ds，即实例化，该变量称为实例化对象，此时运行到嵌套函数时只是完成对嵌套函数的定义，并不执行该函数；当执行实例化对象时ds(ranmen\_price\_lst)，将会完成对内层函数的调用。

def descriptive\_statistics\_factory(decimals=2):  
 def std(data):  
 import statistics  
 return round(statistics.stdev(data),decimals)  
 return std  
   
ranmen\_price\_lst=[700,850,600,650,980,750,500,890,880,  
 700,890,720,680,650,790,670,680,900,  
 880,720,850,700,780,850,750,780,590,  
 650,580,750,800,550,750,700,600,800,  
 800,880,790,790,780,600,690,680,650,  
 890,930,650,777,700]  
ds\_a=descriptive\_statistics\_factory(5)  
print(ds\_a)  
ds\_a(ranmen\_price\_lst)

<function descriptive\_statistics\_factory.<locals>.std at 0x0000026CD4E8DAF0>  
  
108.26189

可以实例化多个对象，例如下述实例化为输入参数decimals为3的实例对象ds\_b，并多次调用该实例化对象，计算不同列表值的标准差。

ds\_b=descriptive\_statistics\_factory(3)  
print(ds\_b(ranmen\_price\_lst))  
  
course\_grade\_lst=[90,81,73,97,85]  
print(ds\_b(course\_grade\_lst))

108.262  
9.066

**5.4** 函数的输入参数（Arguments）

* mutable（可变）和immutable（不可变）数据结构作为输入参数

数据结构含有mutable（可变）和immutable（不可变）两种类型，对应到函数的输入参数则为不可变参数（immutable arguments）和可变参数（mutable arguments）。对于不可变参数，诸如整数（int），字符串（string），是按值传递（by value），虽然通过引用（reference）而非复制（copy）来传递参数值，但不可变对象无法原地更改，因此效果同复制；对于可变参数，诸如列表和字典，则是通过指针传递（by pointer），类似于C语言的指针传递方式，可变参数可以就地更改，因此函数内部传入可变参数，改变引用的可变参数，对应的全局变量也会发生变化。为防止改变全局变量，通常通过复制的方法copy()避免此类更改。

下述定义的3个函数，第一个直接使用全局变量；第二个传入参数，直接引用全局变量；第3个传入参数，复制引用的全局变量。可以发现前2个定义的函数都更新了全局变量，但第3个因为复制而没有更新全局变量。

biology\_score\_dict={"Mason":59,"Reece":73,'A':47,'B':38,'C':63,'D':56,'E':75,'F':53,'G':80,'H':50,'I':41,'J':62,'K':44,'L':26,'M':91,'N':35,'O':53,'P':68}  
name,new\_score='Reece',100  
print(biology\_score\_dict)  
print("\_"\*50)  
  
def biology\_score\_update\_A(name,new\_score):  
 biology\_score\_dict[name]=new\_score   
   
score\_update\_A(name,new\_score)   
print(biology\_score\_dict)  
  
print("--"\*30)  
def biology\_score\_update\_B(score\_dict,name,new\_score):  
 score\_dict[name]=new\_score   
 return score\_dict  
biology\_score\_updated\_B=biology\_score\_update\_B(biology\_score\_dict,'Mason',100)  
print(biology\_score\_dict,'\n',biology\_score\_updated\_B)  
  
print("--"\*30)  
def biology\_score\_update\_C(score\_dict,name,new\_score):  
 import copy  
 score\_dict\_copy=copy.copy(score\_dict)  
 score\_dict\_copy[name]=new\_score   
 return score\_dict\_copy  
biology\_score\_updated\_C=biology\_score\_update\_C(biology\_score\_dict,'A',100)  
print(biology\_score\_dict,'\n',biology\_score\_updated\_C)

{'Mason': 59, 'Reece': 73, 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
{'Mason': 59, 'Reece': 100, 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}  
------------------------------------------------------------  
{'Mason': 100, 'Reece': 100, 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}   
 {'Mason': 100, 'Reece': 100, 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}  
------------------------------------------------------------  
{'Mason': 100, 'Reece': 100, 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}   
 {'Mason': 100, 'Reece': 100, 'A': 100, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}

对于复制需要注意，包含浅复制copy.copy()和深复制copy.deepcopy()，如下述案例，对于嵌套字典或列表为参数值传递，修改嵌套部分的值时，对于浅复制，全局变量值仍会发生改变；而深复制，则可以避免嵌套字典或列表对全局对应变量的更改。

import copy  
test\_score\_dic={"English":{"Mason":90,"Reece":81,'A':73,'B':97,'C':85,'D':60,'E':74,'F':64,'G':72,'H':67,'I':87,'J':78,'K':85,'L':96,'M':77,'N':100,'O':92,'P':86},  
 "Chinese":{"Mason":71,"Reece":90,'A':79,'B':70,'C':67,'D':66,'E':60,'F':83,'G':57,'H':85,'I':93,'J':89,'K':78,'L':74,'M':65,'N':78,'O':53,'P':80},  
 "history":{"Mason":73,"Reece":61,'A':74,'B':47,'C':49,'D':87,'E':69,'F':65,'G':36,'H':7,'I':53,'J':100,'K':57,'L':45,'M':56,'N':34,'O':37,'P':70},  
 "biology":{"Mason":59,"Reece":73,'A':47,'B':38,'C':63,'D':56,'E':75,'F':53,'G':80,'H':50,'I':41,'J':62,'K':44,'L':26,'M':91,'N':35,'O':53,'P':68},  
 }  
print(test\_score\_dic)  
print("\_"\*50)  
  
def test\_score\_update\_A(score\_dict,subject,name,new\_score):  
 import copy  
 score\_dict\_copy=copy.copy(score\_dict)  
 score\_dict\_copy[subject][name]=new\_score   
 return score\_dict\_copy  
  
score\_dict,subject,name,new\_score=test\_score\_dic,'biology','Reece','100'  
test\_score\_updated\_A=test\_score\_update\_A(score\_dict,subject,name,new\_score)  
print(test\_score\_dic,'\n',test\_score\_updated\_A)  
  
print("--"\*30)  
def test\_score\_update\_B(score\_dict,subject,name,new\_score):  
 import copy  
 score\_dict\_copy=copy.deepcopy(score\_dict)  
 score\_dict\_copy[subject][name]=new\_score   
 return score\_dict\_copy  
  
score\_dict,subject,name,new\_score=test\_score\_dic,'biology','Mason','100'  
test\_score\_updated\_B=test\_score\_update\_B(score\_dict,subject,name,new\_score)  
print(test\_score\_dic,'\n',test\_score\_updated\_B)

{'English': {'Mason': 90, 'Reece': 81, 'A': 73, 'B': 97, 'C': 85, 'D': 60, 'E': 74, 'F': 64, 'G': 72, 'H': 67, 'I': 87, 'J': 78, 'K': 85, 'L': 96, 'M': 77, 'N': 100, 'O': 92, 'P': 86}, 'Chinese': {'Mason': 71, 'Reece': 90, 'A': 79, 'B': 70, 'C': 67, 'D': 66, 'E': 60, 'F': 83, 'G': 57, 'H': 85, 'I': 93, 'J': 89, 'K': 78, 'L': 74, 'M': 65, 'N': 78, 'O': 53, 'P': 80}, 'history': {'Mason': 73, 'Reece': 61, 'A': 74, 'B': 47, 'C': 49, 'D': 87, 'E': 69, 'F': 65, 'G': 36, 'H': 7, 'I': 53, 'J': 100, 'K': 57, 'L': 45, 'M': 56, 'N': 34, 'O': 37, 'P': 70}, 'biology': {'Mason': 59, 'Reece': 73, 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}}  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
{'English': {'Mason': 90, 'Reece': 81, 'A': 73, 'B': 97, 'C': 85, 'D': 60, 'E': 74, 'F': 64, 'G': 72, 'H': 67, 'I': 87, 'J': 78, 'K': 85, 'L': 96, 'M': 77, 'N': 100, 'O': 92, 'P': 86}, 'Chinese': {'Mason': 71, 'Reece': 90, 'A': 79, 'B': 70, 'C': 67, 'D': 66, 'E': 60, 'F': 83, 'G': 57, 'H': 85, 'I': 93, 'J': 89, 'K': 78, 'L': 74, 'M': 65, 'N': 78, 'O': 53, 'P': 80}, 'history': {'Mason': 73, 'Reece': 61, 'A': 74, 'B': 47, 'C': 49, 'D': 87, 'E': 69, 'F': 65, 'G': 36, 'H': 7, 'I': 53, 'J': 100, 'K': 57, 'L': 45, 'M': 56, 'N': 34, 'O': 37, 'P': 70}, 'biology': {'Mason': 59, 'Reece': '100', 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}}   
 {'English': {'Mason': 90, 'Reece': 81, 'A': 73, 'B': 97, 'C': 85, 'D': 60, 'E': 74, 'F': 64, 'G': 72, 'H': 67, 'I': 87, 'J': 78, 'K': 85, 'L': 96, 'M': 77, 'N': 100, 'O': 92, 'P': 86}, 'Chinese': {'Mason': 71, 'Reece': 90, 'A': 79, 'B': 70, 'C': 67, 'D': 66, 'E': 60, 'F': 83, 'G': 57, 'H': 85, 'I': 93, 'J': 89, 'K': 78, 'L': 74, 'M': 65, 'N': 78, 'O': 53, 'P': 80}, 'history': {'Mason': 73, 'Reece': 61, 'A': 74, 'B': 47, 'C': 49, 'D': 87, 'E': 69, 'F': 65, 'G': 36, 'H': 7, 'I': 53, 'J': 100, 'K': 57, 'L': 45, 'M': 56, 'N': 34, 'O': 37, 'P': 70}, 'biology': {'Mason': 59, 'Reece': '100', 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}}  
------------------------------------------------------------  
{'English': {'Mason': 90, 'Reece': 81, 'A': 73, 'B': 97, 'C': 85, 'D': 60, 'E': 74, 'F': 64, 'G': 72, 'H': 67, 'I': 87, 'J': 78, 'K': 85, 'L': 96, 'M': 77, 'N': 100, 'O': 92, 'P': 86}, 'Chinese': {'Mason': 71, 'Reece': 90, 'A': 79, 'B': 70, 'C': 67, 'D': 66, 'E': 60, 'F': 83, 'G': 57, 'H': 85, 'I': 93, 'J': 89, 'K': 78, 'L': 74, 'M': 65, 'N': 78, 'O': 53, 'P': 80}, 'history': {'Mason': 73, 'Reece': 61, 'A': 74, 'B': 47, 'C': 49, 'D': 87, 'E': 69, 'F': 65, 'G': 36, 'H': 7, 'I': 53, 'J': 100, 'K': 57, 'L': 45, 'M': 56, 'N': 34, 'O': 37, 'P': 70}, 'biology': {'Mason': 59, 'Reece': '100', 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}}   
 {'English': {'Mason': 90, 'Reece': 81, 'A': 73, 'B': 97, 'C': 85, 'D': 60, 'E': 74, 'F': 64, 'G': 72, 'H': 67, 'I': 87, 'J': 78, 'K': 85, 'L': 96, 'M': 77, 'N': 100, 'O': 92, 'P': 86}, 'Chinese': {'Mason': 71, 'Reece': 90, 'A': 79, 'B': 70, 'C': 67, 'D': 66, 'E': 60, 'F': 83, 'G': 57, 'H': 85, 'I': 93, 'J': 89, 'K': 78, 'L': 74, 'M': 65, 'N': 78, 'O': 53, 'P': 80}, 'history': {'Mason': 73, 'Reece': 61, 'A': 74, 'B': 47, 'C': 49, 'D': 87, 'E': 69, 'F': 65, 'G': 36, 'H': 7, 'I': 53, 'J': 100, 'K': 57, 'L': 45, 'M': 56, 'N': 34, 'O': 37, 'P': 70}, 'biology': {'Mason': '100', 'Reece': '100', 'A': 47, 'B': 38, 'C': 63, 'D': 56, 'E': 75, 'F': 53, 'G': 80, 'H': 50, 'I': 41, 'J': 62, 'K': 44, 'L': 26, 'M': 91, 'N': 35, 'O': 53, 'P': 68}}

* 参数匹配语法（Argument Matching Syntax）

函数的参数匹配包括两个位置，一个是定义函数时的传入参数；再者为调用时的传入参数。常规模式为位置参数，按照顺序从左到右对应参数；调用时可以给定关键字参数，不受位置参数顺序的影响，但是需要将关键字参数放置于位置参数之后。如果是在定义函数时，给定关键字参数，则为为该参数指定默认值，当调用时，不传递该参数值，则以提供的默认值替代；收集参数（Varargs collecting）包括只有一个星号\*的元组形式收集模式，和包括有两个星号\*\*的字典形式收集模式。

不同的匹配语法可以根据需要自由组合。但排序通常为，一般模式在前，再跟元组收集，再跟字典收集。如果位置不对，会引发异常提示，根据提示修改位置，直至满足要求。

| 语法（Syntax） | 位置（Location） | 解释（Interpretation） |
| --- | --- | --- |
| func(value) | 调用（Caller） | 常规参数（位置参数）：按位顺序匹配（从左至右） |
| func(name=value) | 调用（Caller） | 关键字参数：按名称匹配 |
| func(\*iterable) | 调用（Caller） | 将可迭代对象作为单独的位置参数传入：按位顺序匹配 |
| func(\*\*dict) | 调用（Caller） | 将字典键值对作为关键字参数传入：按键名匹配 |
| def func(name) | 函数（Function） | 常规参数（位置参数）：按位置或名称匹配任何传递值 |
| def func(name=value) | 函数（Function） | 配置函数默认参数值，如果没有在调用中传递值（配置参数默认值 default value） |
| def func(\*name) | 函数（Function） | 以元组形式匹配并收集剩余的位置参数：收集参数（Varargs collecting）-位置参数 |
| def func(\*\*name) | 函数（Function） | 以字典的形式匹配并收集剩余的关键字参数：收集参数（Varargs collecting）-关键字参数 |
| def func(\*other, name) | 函数（Function） | 只能在调用中通过关键字传递的参数 |
| def func(\*, name=value) | 函数（Function） | 只能在调用中通过关键字传递的参数 |

x=2  
y=3  
z=5  
xyz\_lst=[2,3,5]  
xyz\_dict={'X':2,'Y':3,'Z':5}  
  
# 常规参数  
def xyz\_normal(X,Y,Z):  
 print(X,Y,Z)  
xyz\_normal(x,y,z)  
xyz\_normal(X=x,Y=y,Z=z)   
xyz\_normal(x,y,Z=z)   
xyz\_normal(\*xyz\_lst)  
xyz\_normal(\*\*xyz\_dict)  
  
print("--"\*30)  
# 配置参数默认值  
def xyz\_default(X,Y=7,Z=9):  
 print(X,Y,Z)  
xyz\_default(x,y,z)  
xyz\_default(x,y)  
xyz\_default(x)  
  
print("--"\*30)  
# 收集参数-位置参数  
def xyz\_collect\_positional(\*args):  
 print(args)  
xyz\_collect\_positional(x,y,z)  
xyz\_collect\_positional(xyz\_lst)  
  
print("--"\*30)  
# 收集参数-位置参数-变化组合  
def xyz\_collect\_positional\_alter(X,\*args,Z):  
 a,b,c=args  
 print(X,args,Z)  
 print(a,b,c)  
xyz\_collect\_positional\_alter(x,12,13,15,Z=z)   
xyz\_collect\_positional\_alter(11,12,13,15,Z=z)  
  
print("--"\*30)  
# 收集参数-关键字参数  
def xyz\_collect\_keyword(\*\*args):  
 print(args)  
xyz\_collect\_keyword(x=2,y=3,z=5)  
xyz\_collect\_keyword(\*\*xyz\_dict)  
  
print("--"\*30)  
# 只能由关键字传递参数  
def xyz\_keyword(X,\*,Y,Z):  
 print(X,Y,Z)  
xyz\_keyword(x,Y=3,Z=5)  
  
print("--"\*30)  
# 组合匹配  
def xyz\_normal\_collect(X\_n,Y\_d=97,\*pargs,\*\*kargs):  
 print(X\_n,Y\_d,pargs,kargs)  
xyz\_normal\_collect(x,y,\*xyz\_lst,\*\*xyz\_dict)

2 3 5  
2 3 5  
2 3 5  
2 3 5  
2 3 5  
------------------------------------------------------------  
2 3 5  
2 3 9  
2 7 9  
------------------------------------------------------------  
(2, 3, 5)  
([2, 3, 5],)  
------------------------------------------------------------  
2 (12, 13, 15) 5  
12 13 15  
11 (12, 13, 15) 5  
12 13 15  
------------------------------------------------------------  
{'x': 2, 'y': 3, 'z': 5}  
{'X': 2, 'Y': 3, 'Z': 5}  
------------------------------------------------------------  
2 3 5  
------------------------------------------------------------  
2 3 (2, 3, 5) {'X': 2, 'Y': 3, 'Z': 5}

**5.5** 函数定义综合实验-自定义箱型图打印样式

数据分析，需要图表辅助观察数据变化关系或数据之间的差异，这使得难以理解的数据，在统计图表下变得易读，易于理解。[matplotlib](https://matplotlib.org/)②图表库，提供了丰富的图表形式，如果不作为最终论文发表或报告，默认的参数配置或者提供的案例代码足可以用于数据分析，但如果要发表研究内容，佐证研究结果，对图表的样式则提出了较高的一些要求。清晰表达图表，并尽量美观，会让读者更容易尝试去理解你的研究。下述定义的函数boxplot\_custom(data\_dict,\*\*args)，依托matplotlib库实现自定箱型图样式。代码书写过程是先确定输入数据参数data\_dict的数据类型，这里使用了字典的数据格式（多数图表库通常支持使用[pandas](https://pandas.pydata.org/)库的DataFrame格式数据），并给定了一个简单的数据样例test\_score\_dic；因为要调整默认的图表样式，因此函数内建立了一个字典paras用于初始化需要配置的样式参数，并以关键字参数\*\*args的方式更新字典，这样可以让使用者在不输入样式参数时，快速的打印一个默认样式箱型图，快速的查看数据关系，而不必要一开始就配置每一个参数，过于繁琐而放弃使用；在确定数据结果无误后，如果希望用于正式的论文图表，则再进一步根据需要有选择性的配置参数。

对于[matplotlib](https://matplotlib.org/)图表库的样式配置直接搜索或从官方文档说明中获取，无需记忆各个参数名。根据需要配置完所需的样式参数，确认代码逻辑设计合理，调试无误后，在函数开头书写函数功能和参数说明。说明的文件格式是直接由scipy解释器生成。

从下述箱型图中很容易发现，英语成绩整体都较高，其次为中文成绩，并且二者的成绩相对比较集中，即每一得分对排名的影响很大；而历史和生物整体得分相对低，并且成绩分散，即每一得分对排名影响相对较弱，同时可以观察到历史有成绩很高的少数得分，也有一个最低的异常值，小于分数刻度线20，经核验，该得分为7。从箱型图中可以观察出很多数据的关系，而各类图表对于不同研究内容都是很重要的分析工具，这尤其体现在数据分析领域。

即使完成了一个函数定义的所有内容，但是往往在后续调用时会出现这样那样的问题，需要不断调整代码，这是正常的代码编写过程。即使一开始认为完全无误，无需调试，也可能会出现意想不到的异常，因此很必要保持不断调整代码的心态。

def boxplot\_custom(data\_dict,\*\*args):  
 '''  
 根据matplotlib库的箱型图打印方法，自定义箱型图可调整的打印样式。   
  
 Parameters  
 ----------  
 data\_dict : dict(list,numerical)  
 字典结构形式的数据，键为横坐分类数据，值为数值列表.  
 \*\*args : keyword arguments  
 可调整的箱型图样式参数包括['figsize', 'fontsize', 'frameOn', 'xlabel', 'ylabel', 'labelsize', 'tick\_length', 'tick\_width', 'tick\_color', 'tick\_direction', 'notch', 'sym', 'whisker\_linestyle', 'whisker\_linewidth', 'median\_linewidth', 'median\_capstyle'].  
  
 Returns  
 -------  
 paras : dict  
 样式更新后的参数值.  
  
 '''  
 import matplotlib.pyplot as plt  
   
 # 计算值提取  
 data\_keys=list(data\_dict.keys())  
 data\_values=list(data\_dict.values())   
   
 # 配置与更新参数  
 paras={'figsize':(10,10),  
 'fontsize':15,  
 'frameOn':['top','right','bottom','left'],  
 'xlabel':None,  
 'ylabel':None,  
 'labelsize':15,  
 'tick\_length':7,  
 'tick\_width':3,  
 'tick\_color':'b',  
 'tick\_direction':'in',  
 'notch':0,  
 'sym':'b+',  
 'whisker\_linestyle':None,  
 'whisker\_linewidth':None,  
 'median\_linewidth':None,  
 'median\_capstyle':'butt'}  
   
 print(paras)  
 paras.update(args)  
 print(paras)  
   
 # 根据参数调整打印图表样式  
 plt.rcParams.update({'font.size': paras['fontsize']})  
 frameOff=set(['top','right','bottom','left'])-set(paras['frameOn'])  
   
   
 # 图表打印  
 fig, ax=plt.subplots(figsize=paras['figsize'])  
 ax.boxplot(data\_values,  
 notch=paras['notch'],  
 sym=paras['sym'],  
 whiskerprops=dict(linestyle=paras['whisker\_linestyle'],linewidth=paras['whisker\_linewidth']),  
 medianprops={"linewidth": paras['median\_linewidth'],"solid\_capstyle": paras['median\_capstyle']})  
   
 ax.set\_xticklabels(data\_keys) # 配置X轴刻度标签  
 for f in frameOff:  
 ax.spines[f].set\_visible(False) # 配置边框是否显示  
   
 # 配置X和Y轴标签  
 ax.set\_xlabel(paras['xlabel'])  
 ax.set\_ylabel(paras['ylabel'])  
   
 # 配置X和Y轴标签字体大小  
 ax.xaxis.label.set\_size(paras['labelsize'])  
 ax.yaxis.label.set\_size(paras['labelsize'])  
   
 # 配置轴刻度样式  
 ax.tick\_params(length=paras['tick\_length'],  
 width=paras['tick\_width'],  
 color=paras['tick\_color'],  
 direction=paras['tick\_direction'])  
  
 plt.show()   
 return paras  
   
test\_score\_dic={"English":{"Mason":90,"Reece":81,'A':73,'B':97,'C':85,'D':60,'E':74,'F':64,'G':72,'H':67,'I':87,'J':78,'K':85,'L':96,'M':77,'N':100,'O':92,'P':86},  
 "Chinese":{"Mason":71,"Reece":90,'A':79,'B':70,'C':67,'D':66,'E':60,'F':83,'G':57,'H':85,'I':93,'J':89,'K':78,'L':74,'M':65,'N':78,'O':53,'P':80},  
 "history":{"Mason":73,"Reece":61,'A':74,'B':47,'C':49,'D':87,'E':69,'F':65,'G':36,'H':7,'I':53,'J':100,'K':57,'L':45,'M':56,'N':34,'O':37,'P':70},  
 "biology":{"Mason":59,"Reece":73,'A':47,'B':38,'C':63,'D':56,'E':75,'F':53,'G':80,'H':50,'I':41,'J':62,'K':44,'L':26,'M':91,'N':35,'O':53,'P':68},  
 }  
test\_score\_lst\_dic={subject:list(v\_subject.values()) for subject,v\_subject in test\_score\_dic.items()}  
print(test\_score\_lst\_dic)  
  
print("--"\*30)  
\_=boxplot\_custom(test\_score\_lst\_dic,  
 figsize=(15,10),  
 fontsize=23,  
 frameOn=['bottom','left'],  
 xlabel='subject',  
 ylabel='score',  
 labelsize='30',  
 tick\_color='r',  
 notch=1,  
 sym='rs',  
 whisker\_linestyle='--',  
 whisker\_linewidth=5,  
 median\_linewidth=5  
 )

{'English': [90, 81, 73, 97, 85, 60, 74, 64, 72, 67, 87, 78, 85, 96, 77, 100, 92, 86], 'Chinese': [71, 90, 79, 70, 67, 66, 60, 83, 57, 85, 93, 89, 78, 74, 65, 78, 53, 80], 'history': [73, 61, 74, 47, 49, 87, 69, 65, 36, 7, 53, 100, 57, 45, 56, 34, 37, 70], 'biology': [59, 73, 47, 38, 63, 56, 75, 53, 80, 50, 41, 62, 44, 26, 91, 35, 53, 68]}  
------------------------------------------------------------  
{'figsize': (10, 10), 'fontsize': 15, 'frameOn': ['top', 'right', 'bottom', 'left'], 'xlabel': None, 'ylabel': None, 'labelsize': 15, 'tick\_length': 7, 'tick\_width': 3, 'tick\_color': 'b', 'tick\_direction': 'in', 'notch': 0, 'sym': 'b+', 'whisker\_linestyle': None, 'whisker\_linewidth': None, 'median\_linewidth': None, 'median\_capstyle': 'butt'}  
{'figsize': (15, 10), 'fontsize': 23, 'frameOn': ['bottom', 'left'], 'xlabel': 'subject', 'ylabel': 'score', 'labelsize': '30', 'tick\_length': 7, 'tick\_width': 3, 'tick\_color': 'r', 'tick\_direction': 'in', 'notch': 1, 'sym': 'rs', 'whisker\_linestyle': '--', 'whisker\_linewidth': 5, 'median\_linewidth': 5, 'median\_capstyle': 'butt'}

自动生成了新的一组数据random\_val\_dict，调用该函数执行箱型图打印，输入参数仅随意配置了边框显示与中位数横线的线型宽度。

import numpy as np  
random\_val\_dict={'sample\_A':np.random.randint(low=1, high=100, size=1000),'sample\_B':np.random.randint(low=1, high=1000, size=1000)  
}  
boxplot\_custom(random\_val\_dict,frameOn=['bottom','left'],median\_linewidth=5)

{'figsize': (10, 10), 'fontsize': 15, 'frameOn': ['top', 'right', 'bottom', 'left'], 'xlabel': None, 'ylabel': None, 'labelsize': 15, 'tick\_length': 7, 'tick\_width': 3, 'tick\_color': 'b', 'tick\_direction': 'in', 'notch': 0, 'sym': 'b+', 'whisker\_linestyle': None, 'whisker\_linewidth': None, 'median\_linewidth': None, 'median\_capstyle': 'butt'}  
{'figsize': (10, 10), 'fontsize': 15, 'frameOn': ['bottom', 'left'], 'xlabel': None, 'ylabel': None, 'labelsize': 15, 'tick\_length': 7, 'tick\_width': 3, 'tick\_color': 'b', 'tick\_direction': 'in', 'notch': 0, 'sym': 'b+', 'whisker\_linestyle': None, 'whisker\_linewidth': None, 'median\_linewidth': 5, 'median\_capstyle': 'butt'}

{'figsize': (10, 10),  
 'fontsize': 15,  
 'frameOn': ['bottom', 'left'],  
 'xlabel': None,  
 'ylabel': None,  
 'labelsize': 15,  
 'tick\_length': 7,  
 'tick\_width': 3,  
 'tick\_color': 'b',  
 'tick\_direction': 'in',  
 'notch': 0,  
 'sym': 'b+',  
 'whisker\_linestyle': None,  
 'whisker\_linewidth': None,  
 'median\_linewidth': 5,  
 'median\_capstyle': 'butt'}

注释（Notes）：

① math，（<https://docs.python.org/3/library/math.html>）；statistics，（<https://docs.python.org/3/library/statistics.html>）；NumPy，（<https://numpy.org/>）；pandas，（<https://pandas.pydata.org/>）；SciPy（<https://scipy.org/>）。

② matplotlib，是一个综合库，用于在Python中创建静态、动画和交互式数据可视化（<https://matplotlib.org/>）。

PC5-ipynb download