05/09/18 08:52:02 /Users/frankyoung/Documents/Python3/18 May/Nobody_gets_ready.py

```
""" formatting"""
from math import pi
import datetime
# print('{:07,.2f}'.format(pi))
lst = ['python', 'java']
dct = {'name': 'frank', 'age': 27}
dt = datetime.date.today()
str_formatting = 'the pi is {0:07,.2f}, my name is {name}, i am {age} years old,i am learing {1} and {2} ,today is {3:%B %Y %d}'.format(pi, *lst, dt, **dct)
# try:{name.upper()} ,Not Supported./AttributeError: 'str' object has no attribute 'upper()'
# "cant not use method call in ''.format(),only subscription (indexing by number or by unquoted (!) name), and attribute access is supported.but f'{name.upper()}' works.
# name = 'frank'
# print(f'{name.upper()}')
# print('{.upper()}'.format(name))
print(str_formatting)


"""object sorting"""
from operator import itemgetter, attrgetter, methodcaller

student_tuples = [('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
print(sorted(student_tuples, key=itemgetter(2)))
print(sorted(student_tuples, key=itemgetter(1, 2)))


class Student:
    def __init__(self, name, grade, age):
        self.name = name
        self.grade = grade
        self.age = age

    def __repr__(self):
        return repr((self.name, self.grade, self.age))


student_objects = [Student('john', 'A', 15), Student('jane', 'B',
```

```
    12), Student('dave', 'B', 10), ]
36
37  print(sorted(student_objects, key=attrgetter('age')))
38  print(sorted(student_objects, key=attrgetter('grade', 'age')))
39
40  messages = ['critical!!!', 'hurry!', 'standby', 'immediate!!']
41  print(sorted(messages, key=methodcaller('count', '!')))
42
43
44  """contextmanger"""
45  import os
46  import glob
47  import contextlib
48
49
50  @contextlib.contextmanager
51  def find_py(path):
52      ori_path = os.getcwd()
53      try:
54          os.chdir(path)
55          result = glob.glob('*.py')
56          yield result
57      finally:
58          os.chdir(ori_path)
59
60
61  # with find_py('/Users/frankyoung/Documents/Python3/18 March') as
    f:
62  #     for pyfile in f:
63  #         print(pyfile)
64  class Find_Py:
65      def __init__(self, path):
66          self.path = path
67          self.ori_path = os.getcwd()
68
69      def __enter__(self):
70          os.chdir(self.path)
71          result = glob.glob('*.py')
72          return result
73
74      def __exit__(self, exc_type, exc_val, traceback):
75          os.chdir(self.ori_path)
76
77
78  # with Find_Py('/Users/frankyoung/Documents/Python3/18 March') as
```

```
    f:

#     for pyf in f:
#         print(pyf)
# print(os.getcwd())


@contextlib.contextmanager
# Nick Coghlan
def suppress(*exceptions):
    try:
        yield
    except exceptions:
        pass


# decorator,scope,closure
# the LEGB rule , for accessing(say,print) and modifying(append) mutable var only.But not reasigning after referenced to a nonlocal(enclosing,Global) Var.UnboundLocalError: local variable referenced to  before assignment.
# <Thomas Ballinger - Finding closure with closures - PyCon 2016>"It turns out that Python analyzes function source code, even compiles it, when a function is defined. During this process it determines the scope of each variable. This determines the process that will be used to find the value of each variables, but does not actually look up this value yet."
# "the scope of a var is determined when function is defined , the value of a var is determined when the function is called".
# http://docs.python-guide.org/en/latest/writing/gotchas/
# Late Binding Closures
funcs = [lambda x: x * i for i in range(3)]  # by the way , i here is a local var.so i doesnt ni=ot exist in the global scope.if try to print i , 'NameError: name 'i' is not defined'
for func in funcs:
    print(func(2))
# 4,4,4
# you will get 2*2=4,2*2=4,2*2=4,because the three 'i's  are in the same scope,when the functions is called , i=2.so you get 4,4,4.
# Solution:use keyword args.'Python's default arguments are evaluated once when the function is defined, not each time the function is called '
funcs = [lambda x, i=i: x * i for i in range(3)]
for func in funcs:
```

```
107     print(func(2))
108 # 0,2,4
109 # or you can use generator expression, without keyword
    args.because generator look up the value as it goes.
110 funcs = (lambda x: x * i for i in range(3))
111 for func in funcs:
112     print(func(2))
113 # 0,2,4
114 #
115 # function attribute, functions can have attributes.
116
117 # make a counter decorator using function attr.
118 from functools import wraps
119
120
121 def counter(my_func):
122     @wraps(my_func)
123     def inner(*args, **kwargs):
124         inner.count += 1
125         return my_func(*args, **kwargs)
126     inner.count = 0
127     return inner
128
129
130 @counter
131 def i_tell_you_what():
132     return 'i tell you what'
133
134
135 i_tell_you_what()
136 i_tell_you_what()
137 print(f'{i_tell_you_what.__name__} run {i_tell_you_what.count}
    times')
138
139 # make a cache with a default return dictionary as an arg
140
141
142 def cache_with_default(dct=None):
143     if dct is None:
144         dct = {}
145
146     def cache(my_func):
147         @wraps(my_func)
148         def wrapper(*args):
```

```
149             if args in dct:
150                 return dct[args]
151             result = my_func(*args)
152             dct[args] = result
153             return result
154         return wrapper
155     return cache
156 
157 
158 @cache_with_default({(1,): 100})  # be careful, pass 1 as a tuple (1,) or it wont work.becaues the args will be (1,)
159 def times_two(x):
160     return x + x
161 
162 
163 print(times_two(1))  # get 100 inside of 2 , because it was looked up in the dct
164 
165 
166 # Brett Slatkin - How to Be More Effective with Functions - PyCon 2015 - YouTube
167 # keword_only_args:forced to be clear.
168 def bobby(*, propane=True, charcoal=False):
169     if propane:
170         print('there you go')
171     else:
172         print('the hell you say')
173 
174 
175 try:
176     bobby(True, False)
177 except Exception as e:
178     print(e)  # bobby() takes 0 positional arguments but 2 were given
179 bobby(propane=True, charcoal=False)  # there you go
180 #same idea , if you can ,put return value into a namedtuple ,to be clear.--> 'Raymond Hettinger's Transforming Code into Beautiful, Idiomatic Python'
181 
182 
183 
184 # what is generator?
185 # iter(foo) is iter(foo)
186 # base on the talk-->
```

```
187 # if iter(foo) is iter(foo):
188 #     now,then =itertools.tee(foo,2)
189 # customize iteration : "Brett Slatkin - How to Be More Effective
    with Functions - PyCon 2015 - YouTube" + "Loop like a native_
    while, for, iterators, generators" ---->by using class __iter__
    method
190 # http://nvie.com/posts/iterators-vs-generators/    :
    iter(iterable)-->iteration
191 # how to detect a generator
192 # how does generator function(yield) run?
193 lst = [1, 2, 3]
194 a = iter(lst)
195 b = iter(lst)
196 print(a is b)  # False
197 print(a == b)  # false
198 print([a] == [b])  # false
199 print(list(a) == list(b))  # true
200 c = iter(a)
201 print(a is c)  # True
202 print(a == c)  # True
203 print([a] == [c])  # false
204
205 lst = [1, 2, 3]
206 a = iter(lst)
207 c = iter(a)
208 print(list(a) == list(c))  # False  ([1,2,3] ==[])
209 # print(list(a))
210 # print(list(c))
211 # so if iter(foo) is iter(foo), foo is a generator; is iter(foo)
    is not iter(foo), foo is a container. 'iter over a iterator
    returns itself.'
212 # that is 'is' how about '=', how about [a],[b],[c] and
    list(a),list(d),list(c),see above.(this how i see it)basiclly,
    python doesn't look inside a iterator see what value it carry(and
    it shouldn't),so if 2 iterator object with different address, it
    is not equal(you can see as not 'is' ,so not '='). same thing
    with [],but list() is different. list() will really loop up the
    value.
213
214
215 # how yield works
216
217 import contextlib
218
219
```

```
220 def HYW():
221     print('hello')
222     yield
223     print('world')
224 
225 
226 a = HYW()  # Nothing happend, ! hello was not printed.
227 next(a)  # ---> now hello was printed. so when you call next,
    generator will run till it hits a yield
228 
229 with contextlib.suppress(StopIteration):
230     next(a)  # -----> world was printed, and then it hits the
    StopIteration
231 
232 
233 #  Ned Batchelder - Facts and Myths about Python names and values
    - PyCon 2015
234 a = [1, 2, 3]
235 b = a
236 a += [4, 5]  # what happened here unline is "a.extend([4,5]) and
    a =a "
237 print(b)  # -->[1, 2, 3, 4, 5]
238 
239 a = [1, 2, 3]
240 b = a
241 a = a + [4, 5]
242 print(b)  # -->[1, 2, 3]
243 
244 a = [1, 2, 3]
245 b = a
246 a.extend([4, 5])
247 print(b)  # -->[1, 2, 3, 4, 5]
248 
249 a = [1, 2, 3]
250 b = a
251 a = a.extend([4, 5])
252 print(a)  # None
253 print(b)  # [1, 2, 3, 4, 5]
254 
255 a = [1, 2, 3]  # try to make a =[10,20,30]
256 for x in a:
257     x = x * 10
258 print(a)  # [1, 2, 3] failed:) beacuse a[0]still is 1 . the right
    way ,a=[x*10 for x in a]
```

```
259
260
261 # "Fact: Python passes function arguments by assigning to
    them."means when you call a function, you assign the parameter to
    the "value" of the arg.
262 def a_func(num):
263     num = num + 2
264
265
266 num = 2
267 num = a_func(num)
268 print(num)  # - - > None , in that function , local num was asign
    to the value of global num ,which is 2,and  local var num assign
    to 4(2+2) , now we return the func. and global num assign to
    nothing :None. local num 4 was no accessable in the global scope.
269
270 # ITERATION
271 # a trick  zip(*[iter(s)]*n)
272 lst = range(10)
273 print(iter(lst) is iter(lst))  # False
274 print(list(zip(*[iter(lst)] * 3)))  # [(0, 1, 2), (3, 4, 5), (6,
    7, 8)]
275 # https://stackoverflow.com/questions/2233204/how-does-zipitersn-
    work-in-python
276 # zip(*lst) is funny
277 # [a]*n=[a,a,a,a,a....,a],same object a. so in this case
    [iter(lst)]*3 is != [iter(lst),iter(lst),iter(lst)],becasue three
    iter(lst) are 3 different objects.if you have to:a=iter(lst),then
    [iter(lst)]*3 =[a,a,a],By the way, range is not
    iterator.iter(lst) is Not iter(lst),but map is a iterator.see
    below:
278 # https://stackoverflow.com/questions/16425166/accumulate-items-
    in-a-list-of-tuples
279 # try to make lst = [(0, 0), (2, 3), (4, 3), (5, 1)] into new_lst
    = [(0, 0), (2, 3), (6, 6), (11, 7)]
280 lst = [(0, 0), (2, 3), (4, 3), (5, 1)]
281
282 import itertools
283 new_lst = zip(*lst)  # zip_object contains ((0,2,4,5),(0,3,3,1))
284 new_lst = map(itertools.accumulate, new_lst)  # map_object
    contains ((0,2,6,11),(0,3,6,7))
285 # print(iter(new_lst) is iter(new_lst)) #True ,so map is a
    iterator
286 new_lst = list(zip(*new_lst))
287 print(new_lst)  # [(0, 0), (2, 3), (6, 6), (11, 7)]
```

```
288 # so all in one line:
    list(zip(*map(itertools.accumulate,zip(*lst))))
289
290 # itertools
291 # islice doesn't  consume the original iterator until next is
    called. most(all) itertools are like that.
292
293 # from itertools doc
294 from collections import deque
295
296
297 def consume(iterator, n=None):
298     "Advance the iterator n-steps ahead. If n is None, consume
    entirely."
299     if n is None:
300         deque(iterable, maxlen=0)
301     else:
302         # itertools.islice(iterator,n,n) # THAT IS A NONO!!!
    islice doesn't  consume the original iterator  until Next is
    called!!!!!
303         next(itertools.islice(iterator, n, n), None)  # YES
304
305
306 def tail(n, iterable):
307     "Return an iterator over the last n items"
308     # tail(3, 'ABCDEFG') --> E F G
309     return iter(collections.deque(iterable, maxlen=n))
310
311
312 # cycle+compress, wanted a serial condition ,say one False and 20
    True, forever
313 iterable = range(45)
314 result = itertools.compress(iterable,
    itertools.cycle(range(21)))  # 1-20,22-41,43,44
315
316 # itertools.repeat take container, not iterator. won't work.use
    repeat(tuple(iterator)).while cycle takes iterators.
317
318
319 # @accumulate usage: turn [1,2,3] in to int 123. or reduce
320 lst = [1, 2, 3]
321 result = itertools.accumulate(lst, lambda a, b: 10 * a + b)
322 print(list(result))  # [1, 12, 123]
323 # or use reduce , it is actually better
```

```
324 from functools import reduce
325 result = reduce(lambda a, b: 10 * a + b, lst)
326 print(result)  # 123
327
328 # takewhile,dropwhile,iter(callable func, sentinel(break)
    value);they works for <,>,=; to read a file by 32 characters --
    >iter(partial(f.read,32),'') see 'Transforming Code into
    Beautiful, Idiomatic Python'
329 # get all the fib nums  < 40,000
330 # [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610,
    987, 1597, 2584, 4181, 6765, 10946, 17711, 28657]
331
332
333 def fib():
334     a, b = 0, 1
335     while True:
336         yield a
337         a, b = b, b + a
338
339
340 result = itertools.takewhile(lambda x: x < 40000, fib())  #
    right,or you can use generator expression+break func()
341
342
343 def breakfunc():
344     'for generator seeing StopIteration will automaticlly break
    loop'
345     raise StopIteration
346
347
348 result_2 = (x if x < 40000 else breakfunc() for x in fib())
349 print(list(result) == list(result_2))  # this is right ,too.
350
351
352 # groupby+defaultdict
353 # groupby :Itertools.groupby: 2 things need to point out, they
    are 1“the iterable needs to already be sorted on the same key
    function”. 2 “the source is shared, when the groupby() object is
    advanced, the previous group is no longer visible.” _doc
354 # it returns a tuple (key,A:iterator of the items that match the
    key)since this iterator shares the data of the groupby return
    value.when we iter over the return tuple, we need to capture the
    returned  A value right away.

355 # the standard  way will be to use “for key ,items in
```

```
    return_value: print key , list(items)". so the problem I had
    before is I used the list()
356 """Must get the value right away!
357 for key , items in groupby:
358     use for loop store the items value into container.like :list
    or dictionay.most common way is list(items), you can use more
    complex as well.
359     see """
360 # https://stackoverflow.com/questions/3749512/python-group-by
361 input = [('11013331', 'KAT'), ('9085267', 'NOT'), ('5238761',
    'ETH'), ('5349618', 'ETH'), ('11788544', 'NOT'), ('962142',
    'ETH'), ('7795297', 'ETH'), ('7341464', 'ETH'), ('9843236',
    'KAT'), ('5594916', 'ETH'), ('1550003', 'ETH')]
362 # result = [
363 #           {
364 #             type:'KAT',
365 #             items: ['11013331', '9843236']
366 #           },
367 #           {
368 #             type:'NOT',
369 #             items: ['9085267', '11788544']
370 #           },
371 #           {
372 #             type:'ETH',
373 #             items: ['5238761', '962142', '7795297', '7341464',
    '5594916', '1550003']
374 #           }
375 #         ]
376 from operator import itemgetter
377 input = sorted(input, key=itemgetter(1))
378 result = itertools.groupby(input, key=itemgetter(1))
379 # for key, items in result:
380 #     print(f'{key}--->{list(items)}')
381
382 # TH--->[('5238761', 'ETH'), ('5349618', 'ETH'), ('962142',
    'ETH'), ('7795297', 'ETH'), ('7341464', 'ETH'), ('5594916',
    'ETH'), ('1550003', 'ETH')]
383 # KAT--->[('11013331', 'KAT'), ('9843236', 'KAT')]
384 # NOT--->[('9085267', 'NOT'), ('11788544', 'NOT')]
385 result = [{'type': key, 'items': [x for x, y in items]} for key,
    items in result]
386 import json
387 result = json.dumps(result, indent=2)
388 print(result) #yes
```

```
389
390 # now same thing again, with defaultdict
391 input = [('11013331', 'KAT'), ('9085267', 'NOT'), ('5238761',
    'ETH'), ('5349618', 'ETH'), ('11788544', 'NOT'), ('962142',
    'ETH'), ('7795297', 'ETH'), ('7341464', 'ETH'), ('9843236',
    'KAT'), ('5594916', 'ETH'), ('1550003', 'ETH')]
392 from collections import defaultdict
393 result=defaultdict(list)
394 for num,key in input:
395     result[key].append(num)
396 # print(result)
397 # defaultdict(<class 'list'>, {'KAT': ['11013331', '9843236'],
    'NOT': ['9085267', '11788544'], 'ETH': ['5238761', '5349618',
    '962142', '7795297', '7341464', '5594916', '1550003']})
398 result=[{'type': key, 'items':items} for key,items in
    result.items()]
399 result = json.dumps(result, indent=2)
400 print(result)#works too.
401
402 # another funny thing about groupby
403 # from [1,1,1,1,1,3,3,4,2,2,1,1,1,3,3] get[{1: 5}, {3: 2}, {4:
    1}, {2: 2}, {1: 3}, {3: 2}]
404 lst=[1,1,1,1,1,3,3,4,2,2,1,1,1,3,3] #without sorting
405 result=itertools.groupby(lst)
406 # for key,items in result:
407 #     print(key,'--->',list(items))
408 # 1 ---> [1, 1, 1, 1, 1]
409 # 3 ---> [3, 3]
410 # 4 ---> [4]
411 # 2 ---> [2, 2]
412 # 1 ---> [1, 1, 1]
413 # 3 ---> [3, 3]
414
415 result=[{key:len(list(items))}for key ,items in result]
416
417 print(result) #[{1: 5}, {3: 2}, {4: 1}, {2: 2}, {1: 3}, {3: 2}]
418
419 from collections import Counter
420 lst=[1,1,1,1,1,3,3,4,2,2,1,1,1,3,3]
421 result=Counter(lst)
422 print(result) #Counter({1: 8, 3: 4, 2: 2, 4: 1})

423 print(result.most_common(2))#[(1, 8), (3, 4)]
```