Practices of the Zambian Pythonista Pro

October 27, 2020

0.0.1 The content of this notebook contains recipes for "Transforming Code into Beautiful, Idiomatic Python" Please keep these practices on your fingertips as an expert python engineer.

This is enticed from an established concept that `code` is often read than written. hence to make it an exciting experience for the future readers and maintainers of your code, abide by these industry pythonic practices.

Content delivery by: Alison Mukoma … sonlinux

0.1 Contents

- How to open and close files
- Concise expressive one-liners

0.1.1 Looping over a range of functions

[85]:
```
# make a list and loop over the list
# python for is not the same as other language
# it uses the iterator protocal

for i in [0,1,2,3,4,5]:
    print (i**2)
```

```
0
1
4
9
16
25
```

[86]:
```
# the output of range is the list above

for i in range(6):
    print (i**2)
```

```
0
1
4
9
16
25
```

range is removed and xrange (iterator based range) has substituted it in Python 3

0.1.2 Looping over a collection

[87]:
```
hostels = ['kariba', 'angola', 'kenya', 'tanzania']
```

[88]:
```
# How would a C programmer do it?

for i in range(len(hostels)):
    print (hostels[i])
```

```
kariba
angola
kenya
tanzania
```

[89]:
```
# Pythonic way

for hostel in hostels:
    print (hostel)
```

```
kariba
angola
kenya
tanzania
```

0.1.3 Looping backwards

[90]:
```
# start from the back, step -1
# C , C++, Java programmer

for i in range(len(hostels)-1, -1, -1):
    print (hostels[i])
```

```
tanzania
kenya
angola
kariba
```

[91]:
```
# pythonic way

for hostel in reversed(hostels):
    print (hostel)
```

```
tanzania
kenya
angola
kariba
```

0.1.4 Looping over a collection of indicies

[92]:
```
# C programmer

for i in range(len(hostels)):
    print (i, '--->', hostels[i])
```

```
0 ---> kariba
1 ---> angola
2 ---> kenya
3 ---> tanzania
```

[93]:
```
# pythonic way

for i, hostel in enumerate(hostels):
```

```
    print (i, '--->', hostel)
```

```
0 ---> kariba
1 ---> angola
2 ---> kenya
3 ---> tanzania
```

0.1.5 Looping over two collections

[94]:
```
student_names = ['esther', 'doreen', 'jachin']
hostels = ['angola', 'mozambique', 'kariba', 'tanzania']
```

[95]:
```
# c programmer

n = min(len(student_names), len(hostels))
for i in range(n):
    print (student_names[i], '--->', hostels[i])
```

```
esther ---> angola
doreen ---> mozambique
jachin ---> kariba
```

[96]:
```
# pythonic way

for student_names, hostel in zip(student_names, hostels):
    print (student_names, '--->', hostel)
```

```
esther ---> angola
doreen ---> mozambique
jachin ---> kariba
```

zip menifests a third list in memory, the third list consists of tuples. It does not scale. Until Python 3 where zip was removed and replaced with izip which uses the iterator property.

0.1.6 Looping in sorted order

[97]:
```
hostels = ['kenya', 'kariba', 'angola', 'tanzania']
```

[98]:
```
for hostel in sorted(hostels):
    print (hostel)
```

```
angola
kariba
kenya
tanzania
```

[99]:
```
for hostel in sorted(hostels, reverse=True):
    print (hostel)
```

```
tanzania
kenya
kariba
angola
```

0.1.7 Custom sort order

[100]:
```
hostels = ['kenya', 'kariba', 'angola']
```

[101]:
```
def compare_length(c1, c2):
    if len(c1) < len(c2): return -1
    if len(c1) > len(c2): return 1
    return 0

# print (sorted(colors, cmp=compare_length))
```

[102]:
```
print (sorted(hostels, key=len))
```

```
['kenya', 'kariba', 'angola']
```

Key functions will be shorter and faster and they are no longer in python3. For any comparison function there is a key function

0.1.8 Call a function until a sentinel value

[103]:
```
# blocks = []
# while True:
#     block = f.read(32)
#     if block == '':
#         break
#     blocks.append(block)
```

[104]:
```
# blocks = []
# for block in iter(partical(f.read, 32), ''):
#     blocks.append(block)
```

iter?

Docstring: iter(iterable) -> iterator iter(callable, sentinel) -> iterator

Get an iterator from an object. In the first form, the argument must supply its own iterator, or be a sequence. In the second form, the callable is called until it returns the sentinel. Type: builtin_function_or_method

iter's second parameter takes in sentinel

In order for it to work, the function has to have no arguements, partial takes in function of many arguments to small arguments

Partial Function

[105]:
```
def func(one, two, three):
    print ('{} {} {}'.format (one, two, three))
```

[106]:
```
func('a', 'b', 'c')
```

```
a b c
```

[107]:
```
from functools import partial

test = partial(func, 'a')
```

[108]:
```
test('b', 'c')
```

```
a b c
```

0.1.9 Distinguishing multiple exit points in loops

[109]:
```
def find(seq, target):
    found = False
    for i, value in enumerate(seq):
        if value == target:
            found = True
            break
    if not found:
        return -1
    return i
```

[110]:
```
print (find ('monkey brains', 'o'))
```

```
1
```

[111]:
```
#### Try to avoid flags as much as possible
```

[112]:
```
def find(seq, target):
    for i, value in enumerate(seq):
        if value == target:
            break
    else:
        return -1
    return i
```

[113]:
```
print (find ('monkey brains', 'o'))
```

```
1
```

Remember else in for like you remember ‘nobreak’

0.1.10 Looping over dictionary keys

[114]:
```
d = {'patricia': 'kenya', 'micheal': 'kariba', 'gasiano': 'tanzania'}
```

[115]:
```
# printing k

for k in d:
    print (k)
```

```
patricia
micheal
gasiano
```

[116]:
```
for k in d.keys():
    print (k)
```

```
patricia
micheal
gasiano
```

[117]:
```
# one way of printing key and values

for k in d:
    print (k, '--->', d[k])
```

```
patricia ---> kenya
micheal ---> kariba
gasiano ---> tanzania
```

[118]:
```
# better way/pythonic

for k, v in d.items():
    print (k, '--->', v)
```

```
patricia ---> kenya
micheal ---> kariba
gasiano ---> tanzania
```

items was removed and replaced with iteritems as of py 3

0.1.11 Construct a dictionary from pairs

[119]:
```
names = ['patricia', 'micheal', 'esther']
hostels = ['kenya', 'kariba', 'angola']
```

[120]:
```
d = dict(zip(names, hostels))
```

[121]:
```
d
```

[121]:
```
{'patricia': 'kenya', 'micheal': 'kariba', 'esther': 'angola'}
```

zip was replaced by izip as of py3

0.1.12 Counting with dictionaries

[122]:
```
hostels = ['kenya', 'angola', 'kariba', 'tanzania', 'mozambique']
```

[123]:
```
my_hostel_names = {}
```

[124]:
```
# basic method of doing it

for hostel in hostels:
    if hostel not in my_hostel_names:
        my_hostel_names[hostel] = 0
    my_hostel_names[hostel] += 1
```

[125]:
```
print(my_hostel_names)
```

```
{'kenya': 1, 'angola': 1, 'kariba': 1, 'tanzania': 1, 'mozambique': 1}
```

[126]:
```
my_hostel_names = {}
for hostel in hostels:
    my_hostel_names[hostel] = my_hostel_names.get(hostel, 0) + 1
```

[127]:
```
print(my_hostel_names)
```

```
{'kenya': 1, 'angola': 1, 'kariba': 1, 'tanzania': 1, 'mozambique': 1}
```

[128]:
```
from collections import defaultdict
my_hostel_names = defaultdict(int)
for hostel in hostels:
    my_hostel_names[hostel] += 1
```

[129]:
```
print(my_hostel_names)
```

```
defaultdict(<class 'int'>, {'kenya': 1, 'angola': 1, 'kariba': 1, 'tanzania': 1,
'mozambique': 1})
```

0.1.13 Grouping with dictionaries

[130]:
```
names = ['esther', 'patricia', 'jachin', 'christopher', 'felix', 'micheal',␣
↪'joseph', 'doreen', 'douglas']
```

[131]:
```
d = {}
for name in names:
    key = len(name)
    if key not in d:
```

```
        d[key] = []
    d[key].append(name)
```

[132]:
```
d
```

[132]:
```
{6: ['esther', 'jachin', 'joseph', 'doreen'],
 8: ['patricia'],
 11: ['christopher'],
 5: ['felix'],
 7: ['micheal', 'douglas']}
```

[133]:
```
# better way
# just like get but has a side effect of missing key
# also the word is bad

d = {}
for name in names:
    key = len(name)
    d.setdefault(key, []).append(name)
```

[134]:
```
d
```

[134]:
```
{6: ['esther', 'jachin', 'joseph', 'doreen'],
 8: ['patricia'],
 11: ['christopher'],
 5: ['felix'],
 7: ['micheal', 'douglas']}
```

[135]:
```
# modern way
d = defaultdict(list)
for name in names:
    key = len(name)
    d[key].append(name)
```

[136]:
```
d
```

[136]:
```
defaultdict(list,
            {6: ['esther', 'jachin', 'joseph', 'doreen'],
             8: ['patricia'],
             11: ['christopher'],
             5: ['felix'],
             7: ['micheal', 'douglas']})
```

This is the new idiom for grouping in python

0.1.14 Is a dictionary pop() atomic?

[137]:
```
d = {'jachin': 'kariba', 'doreen': 'angola', 'mwewa': 'tanzania'}
```

[138]:
```
while d:
    key, value = d.popitem()
    print (key, '--->', value)
```

```
mwewa ---> tanzania
doreen ---> angola
jachin ---> kariba
```

You do not have to put locks around it so it can be used in threads

0.1.15 Linking dictionaries

[139]:
```
a = {'name': 'Patricia'}
b = {'name': 'Jachin', 'email': 'jachin@pythonngeeks.web'}
c = {'name': 'Doreen', 'email': 'doreen@ladieswhocode.net', 'candidate_id':␣
 ↪'222'}
d = {'name': 'Esther', 'email': 'esther@iautomatemachines.com', 'candidate_id':␣
 ↪'205'}
e = {'name': 'felix', 'email': 'felix@pyinventor.zm', 'engineer_codex': '551'}
```

[140]:
```
from collections import ChainMap
ChainMap(a, b, c, d, e)
```

[140]:
```
ChainMap({'name': 'Patricia'}, {'name': 'Jachin', 'email':
'jachin@pythonngeeks.web'}, {'name': 'Doreen', 'email':
'doreen@ladieswhocode.net', 'candidate_id': '222'}, {'name': 'Esther', 'email':
'esther@iautomatemachines.com', 'candidate_id': '205'}, {'name': 'felix',
'email': 'felix@pyinventor.zm', 'engineer_codex': '551'})
```

0.1.16 Clarify function calls with keyword arguments

[141]:
```
def twitter_search(name, retweets, numtweets, popular):
    return 0
```

[142]:
```
# without keyword arguments
twitter_search('obama', False, 20, True)
```

[142]:
```
0
```

[143]:
```
# with keyword arguments
twitter_search(name='obama', retweets=False, numtweets=20, popular=True)
```

[143]:
```
0
```

0.1.17 Clarify multiple return values with named tuples

use namedtuple instead of tuple

[144]:
```
from collections import namedtuple
```

[145]:
```
TestResults = namedtuple('TestResults', ['failed', 'attempted'])
```

[146]:
```
TestResults(0, 1)
```

[146]:
```
TestResults(failed=0, attempted=1)
```

0.1.18 Unpacking sequences

[147]:
```
p = 'Alison', 'Mukoma', 0x30, 'python@learnerscorner.ehc.zm'
```

[148]:
```
p
```

[148]:
```
('Alison', 'Mukoma', 48, 'python@learnerscorner.ehc.zm')
```

[149]:
```
# instead of doing this

fname = p[0]
lname = p[1]
age = p[2]
email = p[3]
```

[150]:
```
# do this

fname, lname, age, email = p
```

0.1.19 Updating multiple state variables

[151]:
```
# fibonacci generator

def fibonacci(n):
    x = 0
    y = 1
    for i in range(n):
        yield x
        t = y
        y = x + y
        x = t
```

[152]:
```
for f in fibonacci(5):
    print (f)
```

```
0
1
1
2
3
```

[153]:
```
# Update states at ones

def fibonacci(n):
    x, y = 0, 1
    for i in range(n):
        yield x
        x, y = y, x+y
```

[154]:
```
for f in fibonacci(5):
    print (f)
```

```
0
1
1
2
3
```

0.1.20 Simultaneous state updates

This is one of the biggest causes of bug caused by states.

0.1.21 Concatenating strings

[155]:
```
names = ['patricia', 'jachin', 'ganizani', 'esther', 'felix', 'doreen',␣
↪'joseph', 'micheal', 'douglas']
```

[156]:
```
# do not use +
# this is quadratic behaviour

s = names[0]
for name in names[1:]:
    s += ', ' + name
print (s)
```

```
patricia, jachin, ganizani, esther, felix, doreen, joseph, micheal, douglas
```

[157]:
```
# do this

print (', '.join(names))
```

```
patricia, jachin, ganizani, esther, felix, doreen, joseph, micheal, douglas
```

0.1.22 Updating sequences

[158]:
```
names = ['patricia', 'jachin', 'ganizani', 'esther', 'felix', 'doreen',␣
 ↪'joseph', 'micheal', 'douglas']
```

[159]:
```
del names[0]
names.pop(0)
names.insert(0, 'mark')
```

[160]:
```
names
```

[160]:
```
['mark',
 'ganizani',
 'esther',
 'felix',
 'doreen',
 'joseph',
 'micheal',
 'douglas']
```

[161]:
```
from collections import deque
names = deque( ['patricia', 'jachin', 'ganizani', 'esther', 'felix', 'doreen',␣
 ↪'joseph', 'micheal', 'douglas'])
```

[162]:
```
names
```

[162]:
```
deque(['patricia',
       'jachin',
       'ganizani',
       'esther',
       'felix',
       'doreen',
       'joseph',
       'micheal',
       'douglas'])
```

[163]:
```
del names[0]
names.popleft()
names.appendleft('muyunda')
```

[164]:
```
names
```

[164]:
```
deque(['muyunda',
       'ganizani',
       'esther',
       'felix',
       'doreen',
```

```
        'joseph',
        'micheal',
        'douglas'])
```

deque is very efficient for updating sequences

0.1.23 Using decorators to factor-out administrative logic

[165]:
```
def web_lookup(url, saved={}):
    if url in saved:
        return saved[url]
    page = urllib.urlopen(url).read()
    saved[url] = page
    return page
```

[166]:
```
#@cache
def web_lookup(url):
    return urllib.urlopen(url).read()
```

0.1.24 Caching decorator

[167]:
```
def cache(func):
    saved = {}
    @wraps(func)
    def newfunc(*args):
        if args in saved:
            return newfunc(*args)
        result = func(*args)
        saved[args] = result
        return result
    return newfunc
```

0.1.25 How to open and close files

[168]:
```
# do not do this
f = open('sth.sth')
try:
    data  = f.read()
finally:
    f.close()
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-168-449cbd360b78> in <module>
      1 # do not do this
----> 2 f = open('sth.sth')
```

```
      3 try:
      4     data  = f.read()
      5 finally:

FileNotFoundError: [Errno 2] No such file or directory: 'sth.sth'
```

[ ]:
```
# do this
with open('sth.sth') as f:
    data = f.read()
```

0.1.26 Concise expressive one-liners

0.1.27 List Comprehensions

[ ]:
```
[x ** 2 for x in range(10)]
```

[ ]:
```
gen = (x ** 2 for x in range(10))
```

[ ]:
```
for x in gen:
    print (x)
```