

Notice Python

TODO: yourself
Test By Pycharm Jupyter...

Base Question - Python有什么不为人知的坑? (https://www.zhihu.com/question/29823322)

Satwikkansal - wtfpython (https://github.com/satwikkansal/wtfpython)

暮晨 leisurelicht - wtfpython-cn (https://github.com/leisurelicht/wtfpython-cn)

Top-10-mistakes-... - Martin Chikilian (https://www.toptal.com/python/top-10-mistakes-that-python-programmers-make)

Use For - Python-100-Days - jackfrued (https://github.com/jackfrued/Python-100-Days)

总结：
(1) 不论是驻留、折叠、哈希、销毁等，都是实现机制引起；
(2) 源代码查看注解；
(3) 精简原则，避免入坑；

001.string

```
In [1]: stra = "some_string"
        id(stra), id("some"+"_"+"string")

Out[1]: (140679647345264, 140679647345264)
```

002.string

关键词：字符串的驻留

```
In [2]: a, b = "wtf", "wtf"
        a is b

Out[2]: True

In [3]: a = "wtf!"
        b = "wtf!"
        a is b

Out[3]: False

In [4]: a, b = "wtf!", "wtf!"
        a is b

Out[4]: True
```

003.string

常量折叠(constant folding)，Python 中的一种窥孔优化(peephole optimization) 技术。

关键词：常量折叠、窥孔优化

Cpython源码 (https://github.com/python/cpython/blob/3.6/Python/peephole.c#L288)

```
In [5]: 'a' * 20 is 'aaaaaaaaaaaaaaaaaaaaa'

Out[5]: True

In [6]: 'a' * 21 is 'aaaaaaaaaaaaaaaaaaaaa'

Out[6]: False
```

004.list

关键词：哈希值

```
In [7]: some_dict = {}  
        some_dict[5.5] = "Ruby"  
        some_dict[5.0] = "JavaScript"  
        some_dict[5] = "Python"  
  
        some_dict
```

```
Out[7]: {5.5: 'Ruby', 5.0: 'Python'}
```

```
In [8]: some_dict[5.0]
```

```
Out[8]: 'Python'
```

```
In [9]: some_dict[5]
```

```
Out[9]: 'Python'
```

```
In [10]: 5.0 == 5
```

```
Out[10]: True
```

```
In [11]: hash(5) == hash(5.0)
```

```
Out[11]: True
```

005.return

关键词：返回

```
In [12]: def some_func():  
        try:  
            return 'from_try'  
        finally:  
            return 'from_finally'
```

```
In [13]: some_func()
```

```
Out[13]: 'from_finally'
```

006.class

关键词：类、销毁

```
In [14]: class A:  
        pass
```

```
In [15]: A()
```

```
Out[15]: <__main__.A at 0x7ff2886b0240>
```

```
In [16]: A() == A()
```

```
Out[16]: False
```

```
In [17]: A() is A()
```

```
Out[17]: False
```

```
In [18]: hash(A()) == hash(A())
```

```
Out[18]: True
```

```
In [19]: id(A()) == id(A())
```

```
Out[19]: True
```

007.classid

关键词：销毁时机

```
In [20]: class B(object):
          def __init__(self):
              print("I")

          def __del__(self):
              print("D")
```

```
In [21]: B(), B()
```

```
I
I
```

```
Out[21]: (<__main__.B at 0x7ff2886b0860>, <__main__.B at 0x7ff2886b09e8>)
```

```
In [22]: B() is B()
```

```
I
I
D
D
```

```
Out[22]: False
```

```
In [23]: id(B()) == id(B())
```

```
I
D
I
D
```

```
Out[23]: True
```

008.dict

关键词：创建、for循环

```
for_stmt ::= "for" target_list "in" expression_list ":" suite ["else" ":" suite]
```

```
In [24]: some_string = "wtf"
          some_dict = {}
          for i, some_dict[i] in enumerate(some_string):
              pass
```

```
In [25]: # enumerate
          enumerate("wtf"), [i for i in enumerate(some_string)]
```

```
Out[25]: (<enumerate at 0x7ff2886b28b8>, [(0, 'w'), (1, 't'), (2, 'f')])
```

```
In [26]: # enumerate
          [(i, j) for i, j in enumerate(some_string)]
```

```
Out[26]: [(0, 'w'), (1, 't'), (2, 'f')]
```

```
In [27]: some_dict
```

```
Out[27]: {0: 'w', 1: 't', 2: 'f'}
```

```
In [28]: i, some_dict[i] = (0, 'w')
          i, some_dict[i] = (1, 't')
          i, some_dict[i] = (2, 'f')
          some_dict
```

```
Out[28]: {0: 'w', 1: 't', 2: 'f'}
```

```
In [29]: for i in range(4):  
        print(i)  
        i = 10
```

```
0  
1  
2  
3
```

009.time

关键词：执行时机、更新

```
In [30]: array = [1, 8, 15]  
g = (x for x in array if array.count(x) > 0)  
array = [2, 8, 22]
```

```
In [31]: array1 = [1, 8, 15]  
[x for x in array1 if array1.count(x) > 0]
```

```
Out[31]: [1, 8, 15]
```

```
In [32]: g, list(g)
```

```
Out[32]: (<generator object <genexpr> at 0x7ff288678888>, [8])
```

```
In [33]: array_1 = [1,2,3,4]  
g1 = (x for x in array_1)  
array_1 = [1,2,3,4,5]  
  
array_2 = [1,2,3,4]  
g2 = (x for x in array_2)  
array_2[:] = [1,2,3,4,5] # 更新
```

```
In [34]: list(g1), list(g2)
```

```
Out[34]: ([1, 2, 3, 4], [1, 2, 3, 4, 5])
```

010.whatis

关键词：对象、is/id、同行赋值same

```
In [35]: a = 256  
b = 256  
a is b
```

```
Out[35]: True
```

```
In [36]: a = 257  
b = 257  
a is b
```

```
Out[36]: False
```

```
In [37]: a = 257;b = 257  
a is b
```

```
Out[37]: False
```

当你启动Python 的时候，-5 到 256 的数值就已经被分配好了。
这些数字因为经常使用所以适合被提前准备好。

```
In [38]: id(256)
```

```
Out[38]: 94439004535296
```

```
In [39]: a = 256
        b = 256
        id(a), id(b)

Out[39]: (94439004535296, 94439004535296)

In [40]: id(257)

Out[40]: 140679647755312

In [41]: x = 257
        y = 257
        id(x), id(y)

Out[41]: (140679647755088, 140679647755504)

In [42]: a, b = 257, 257 # same
        id(a), id(b)

Out[42]: (140679647755600, 140679647755600)

In [43]: a = 257
        b = 257
        id(a), id(b)

Out[43]: (140679647756016, 140679647755760)
```

011.row

关键词：内存

```
In [44]: row = [""]*3
        board = [row]*3
        board

Out[44]: [['', '', ''], ['', '', ''], ['', '', '']]

In [45]: board[0]

Out[45]: ['', '', '']

In [46]: board[0][0]

Out[46]: ''

In [47]: board[0][0] = "x"
        board

Out[47]: [['X', '', ''], ['X', '', ''], ['X', '', '']]

In [48]: board = [['']*3 for _ in range(3)]
        board[0][0] = "x"
        board

Out[48]: [['X', '', ''], ['', '', ''], ['', '', '']]
```

row、board均引用了同一列表

|""|""|""| |:-:|:-:|:-:|

012.print

关键词：print、变量赋值

```
In [49]: funcs = []
results = []
for x in range(7):
    def some_func():
        return x
    funcs.append(some_func)
    results.append(some_func())

funcs_results = [func() for func in funcs]
```

```
In [50]: results
```

```
Out[50]: [0, 1, 2, 3, 4, 5, 6]
```

```
In [51]: funcs_results
```

```
Out[51]: [6, 6, 6, 6, 6, 6, 6]
```

```
In [52]: powers_of_x = [lambda x: x**i for i in range(4)]
[f(2) for f in powers_of_x]
```

```
Out[52]: [8, 8, 8, 8]
```

```
In [53]: funcs = []
results = []
for x in range(7):
    def some_func(x_=x):
        return x_
    funcs.append(some_func)
    results.append(some_func())

[func() for func in funcs]
```

```
Out[53]: [0, 1, 2, 3, 4, 5, 6]
```

013.plus_eq

```
In [54]: t = ([],[])
t
```

```
Out[54]: ([], [])
```

```
In [55]: try:
    t[0] += [1]
    print(t)
except Exception as e:
    print(f"Exception: {e}")
```

```
Exception: 'tuple' object does not support item assignment
```

014.locals/exec

```
In [56]: var = 100

def f1():
    locals()['var'] = 200
    print(var)

def f2():
    exec("locals()['var'] = 300")
    print(var)

f1()
f2()
```

```
100
100
```

015.list_without

```
In [57]: l1 = [  
        'foo'  
        # without ,  
        'bar'  
        ]  
  
        l2 = [  
        'foo'  
        ,  
        'bar'  
        ]  
  
        l1, l2
```

```
Out[57]: (['foobar'], ['foo', 'bar'])
```

016.true_false

```
In [58]: True is True is True
```

```
Out[58]: True
```

```
In [59]: False is False is False
```

```
Out[59]: True
```

```
In [60]: False is (True is False)
```

```
Out[60]: True
```

```
In [61]: (False is True) is False
```

```
Out[61]: True
```

```
In [62]: (False is False) is False
```

```
Out[62]: False
```

```
In [63]: False is (False is False)
```

```
Out[63]: False
```

```
In [64]: (False is True) and (True is False)
```

```
Out[64]: False
```

017.SyntaxError

```
In [65]: a = 5  
        , , ,  
  
        wf = wave.open(r"C:\Users\Notepad++\xbox\ubuntu.wav", 'rb')  
        , , ,  
  
        print(a)
```

```
File "<ipython-input-65-7c12888d9cb0>", line 5  
    , , ,  
    ^
```

```
SyntaxError: (unicode error) 'unicodeescape' codec can't decode bytes in position 20-21: truncated \UXX  
XXXXXX escape
```

```
In [66]: a = 5  
        r' , , ,  
        wf = wave.open(r"C:\Users\Notepad++\xbox\ubuntu.wav", 'rb')  
        , , ,  
        print(a)
```

```
5
```

018.function

```
In [67]: def foo(bar=[]):  
        bar.append("baz")  
        return bar
```

```
In [68]: foo()
```

```
Out[68]: ['baz']
```

```
In [69]: foo()
```

```
Out[69]: ['baz', 'baz']
```

```
In [70]: foo()
```

```
Out[70]: ['baz', 'baz', 'baz']
```

```
In [71]: foo()
```

```
Out[71]: ['baz', 'baz', 'baz', 'baz']
```

```
In [72]: foo()
```

```
Out[72]: ['baz', 'baz', 'baz', 'baz', 'baz']
```

```
In [73]: def foo(bar=None):  
        if bar is None:  
            bar = []  
        bar.append("baz")  
        return bar
```

```
In [74]: foo()
```

```
Out[74]: ['baz']
```

```
In [75]: foo()
```

```
Out[75]: ['baz']
```

```
In [76]: foo()
```

```
Out[76]: ['baz']
```

019.class_x

```
In [77]: class A(object):  
        x = 1  
  
        class B(A):  
            pass  
  
        class C(A):  
            pass  
  
A.x, B.x, C.x
```

```
Out[77]: (1, 1, 1)
```

```
In [78]: B.x = 2  
A.x, B.x, C.x
```

```
Out[78]: (1, 2, 1)
```

```
In [79]: A.x = 3  
A.x, B.x, C.x
```

```
Out[79]: (3, 2, 3)
```



```
In [80]: class A(object):  
        x = 1  
  
        class B(A):  
            pass  
  
        class C(A):  
            pass  
  
A.x, B.x, C.x
```

Out[80]: (1, 1, 1)

```
In [81]: A.x = 3  
A.x, B.x, C.x
```

Out[81]: (3, 3, 3)

020.GIL

Global Interpreter Lock - 全局解释器锁 (https://blog.csdn.net/megustas_jjc/article/details/79110284)

In CPython, the global interpreter lock, or GIL, is a mutex that prevents multiple native threads from executing Python bytecodes at once. This lock is necessary mainly because CPython's memory management is not thread-safe. (However, since the GIL exists, other features have grown to depend on the guarantees that it enforces.)

```
while True:  
    acquire GIL  
    for i in 1000:  
        do something  
    release GIL  
    /* Give Operating System a chance to do thread scheduling */
```

```
In [82]: # single_thread  
  
from threading import Thread  
import time  
  
def my_counter():  
    i = 0  
    for _ in range(int(1e6)):  
        i = i + 1  
    return True  
  
def main():  
    thread_array = {}  
    start_time = time.time()  
    for tid in range(2):  
        t = Thread(target=my_counter)  
        t.start()  
        t.join()  
    end_time = time.time()  
    print("Total time: {}".format(end_time - start_time))  
  
if __name__ == '__main__':  
    main()
```

Total time: 0.09841322898864746

```
In [83]: # multi_thread

from threading import Thread
import time

def my_counter():
    i = 0
    for _ in range(int(1e6)):
        i = i + 1
    return True

def main():
    thread_array = {}
    start_time = time.time()
    for tid in range(2):
        t = Thread(target=my_counter)
        t.start()
        thread_array[tid] = t
    for i in range(2):
        thread_array[i].join()
    end_time = time.time()
    print("Total time: {}".format(end_time - start_time))

if __name__ == '__main__':
    main()
```

Total time: 0.09646821022033691

021.py2py3

Python是世界上最好的两门语言。

print、print()

```

[root@izhp3cguqxi2d0j4qrm94bz ~]# python2
Python 2.7.5 (default, Jun 20 2019, 20:27:34)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print 1
1
>>> print 1, 2
1 2
>>> print(1)
1
>>> print(1, 2)
(1, 2)
>>> help("print")
#
# The ``print`` statement
# *****
#
#   print_stmt ::= "print" ([expression ("," expression)* [","]]
#                       | ">>" expression [("," expression)+ [","]])
#
# ``print`` evaluates each expression in turn and writes the resulting
# object to standard output (see below). If an object is not a string,
# it is first converted to a string using the rules for string
# conversions. The (resulting or original) string is then written. A
# space is written before each object is (converted and) written, unless
# the output system believes it is positioned at the beginning of a
# line. This is the case (1) when no characters have yet been written
# to standard output, (2) when the last character written to standard
# output is a whitespace character except ``' ``'``, or (3) when the last
# write operation on standard output was not a ``print`` statement. (In
# some cases it may be functional to write an empty string to standard
# output for this reason.)
#
# Note: Objects which act like file objects but which are not the built-in
# file objects often do not properly emulate this aspect of the file
# object's behavior, so it is best not to rely on this.
#
# A ``'\n'`` character is written at the end, unless the ``print``
# statement ends with a comma. This is the only action if the statement
# contains just the keyword ``print``.
#
# Standard output is defined as the file object named ``stdout`` in the
# built-in module ``sys``. If no such object exists, or if it does not
# have a ``write()`` method, a ``RuntimeError`` exception is raised.
#
# ``print`` also has an extended form, defined by the second portion of
# the syntax described above. This form is sometimes referred to as
# "``print`` chevron." In this form, the first expression after the
# ``>>>`` must evaluate to a "file-like" object, specifically an object
# that has a ``write()`` method as described above. With this extended
# form, the subsequent expressions are printed to this file object. If
# the first expression evaluates to ``None``, then ``sys.stdout`` is
# used as the file for output.
#
>>> quit()

```

```
[root@izhp3cguqxi2d0j4qrm94bz ~]# python3
Python 3.6.8 (default, Apr 25 2019, 21:02:35)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-36)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> print 1
      File "<stdin>", line 1
        print 1
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(1)?
>>> print 1, 2
      File "<stdin>", line 1
        print 1, 2
          ^
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(1, 2)?
>>> print(1)
1
>>> print(1, 2)
1 2
>>> help(print)
#
# Help on built-in function print in module builtins:
#
# print(...)
#     print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
#
#     Prints the values to a stream, or to sys.stdout by default.
#     Optional keyword arguments:
#     file: a file-like object (stream); defaults to the current sys.stdout.
#     sep:   string inserted between values, default a space.
#     end:   string appended after the last value, default a newline.
#     flush: whether to forcibly flush the stream.
#
>>> quit()
```

022.Exception

```
In [84]: try:
          l = ["a", "b"]
          int(l[2])
except ValueError, IndexError:
          pass

      File "<ipython-input-84-2e06a7321efd>", line 5
        except ValueError, IndexError:
              ^
SyntaxError: invalid syntax
```

```
In [85]: try:
          l = ["a", "b"]
          int(l[2])
except (ValueError, IndexError) as e:
          pass
```

023.Global

In [86]: x = 10

```
def foo():
    x += 1
    print(x)
```

foo()

```
-----
UnboundLocalError                                Traceback (most recent call last)
<ipython-input-86-c1d0ccf36861> in <module>()
      6     print(x)
      7
----> 8 foo()
```

```
<ipython-input-86-c1d0ccf36861> in foo()
      3
      4 def foo():
----> 5     x += 1
      6     print(x)
      7
```

UnboundLocalError: local variable 'x' referenced before assignment

In [87]: lst = [1, 2, 3]

```
def fool():
    lst.append(5)
```

fool()
lst

Out[87]: [1, 2, 3, 5]

In [88]: lst = [1, 2, 3]

```
def foo2():
    lst += [5]
```

foo2()
lst

```
-----
UnboundLocalError                                Traceback (most recent call last)
<ipython-input-88-95ff760e3b0d> in <module>()
      5     lst += [5]
      6
----> 7 foo2()
      8 lst
```

```
<ipython-input-88-95ff760e3b0d> in foo2()
      3
      4 def foo2():
----> 5     lst += [5]
      6
      7 foo2()
```

UnboundLocalError: local variable 'lst' referenced before assignment

024.Tab

In [89]: **def** f():
 print(1)
 f()

1

In [90]: **def** f():
 print(1)
 f()

1

```
In [92]: def f():
        print(1)
        print(1)
        f()
```

```
File "<ipython-input-92-343d7c639e87>", line 4
    print(1)
    ^
```

IndentationError: unindent does not match any outer indentation level

025.bytes/str

```
In [93]: a = 'lq2we4er56tyuio'
        b = b'6yt54e3w2we45t'
        a, b, type(a), type(b)
```

```
Out[93]: ('lq2we4er56tyuio', b'6yt54e3w2we45t', str, bytes)
```

```
In [94]: bytes(a, encoding='utf-8'), b.decode()
```

```
Out[94]: (b'lq2we4er56tyuio', '6yt54e3w2we45t')
```

026.Chinese

diff in py2、py3

```
[root@izhp3cguqxi2d0j4qrm94bz ~]# python
Python 2.7.5 (default, Aug 7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "中"
>>> a
'\xe4\x b8\xad'
>>> print(a)
中
>>> print(1, a)
(1, '\xe4\x b8\xad')
>>> exit()
```

```
[root@izhp3cguqxi2d0j4qrm94bz ~]# py3
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = "中"
>>> a
'中'
>>> print(a)
中
>>> print(1, a)
1 中
>>> quit()
```

027.Cal_frac

diff in py2、py3, **5/2**

```
[root@izhp3cguqxi2d0j4qrm94bz ~]# python
Python 2.7.5 (default, Aug 7 2019, 00:51:29)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> 5/2
2
>>> 5.0/2
2.5
>>> 5//2
2
>>> 5.0//2.0
2.0
>>> quit()
```

```
[root@izhp3cguqxi2d0j4qrm94bz ~]# py3
Python 3.6.5 |Anaconda, Inc.| (default, Apr 29 2018, 16:14:56)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 5/2
2.5
>>> 5.0/2
2.5
>>> 5//2
2
>>> 5.0//2.0
2.0
>>> quit()
```

028.list_split

```
In [1]: l = ['a', 'b', 'c', 'd', 'e']
l
```

```
Out[1]: ['a', 'b', 'c', 'd', 'e']
```

```
In [2]: l[10:]
```

```
Out[2]: []
```

```
In [3]: l[5]
# IndexError: list index out of range.
```

```
-----
IndexError                                Traceback (most recent call last)
<ipython-input-3-b7acd94015f6> in <module>()
      1
----> 2 l[5]

IndexError: list index out of range
```

029.TODO