CPYTHON INTERNALS 101

@dawranliou

Exploring is never boring

AGENDA

1. Overview
2. Walkthrough
3. Demo

SCOPE

WHY BOTHER?

1. Curiosity
2. Research
3. Makes you a better programmer
4. Showoff
5. Solve Problems

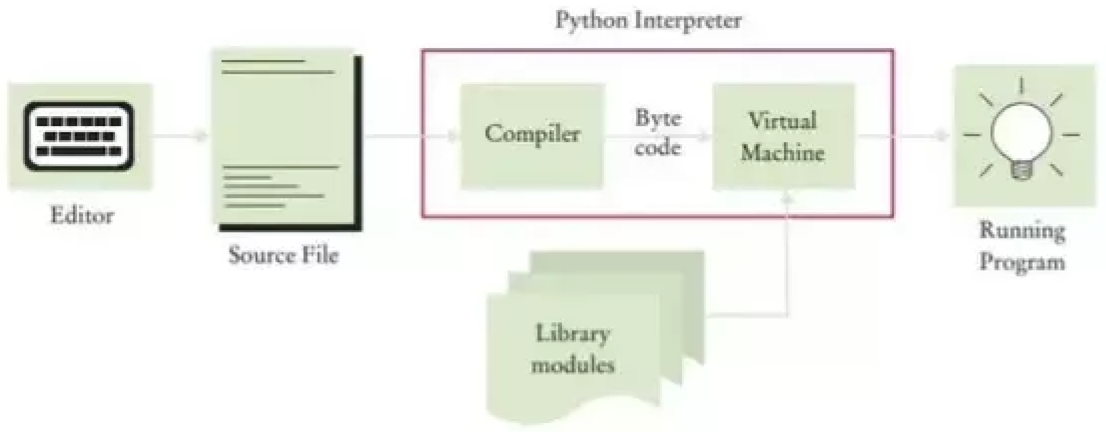
```
>>> False is False is False
>>> (False is False) is False
>>> False is False is False
True
```

```
>>> False is False is False

  2           0 LOAD_GLOBAL              0 (False)
              3 LOAD_GLOBAL              0 (False)
              6 DUP_TOP
              7 ROT_THREE
              8 COMPARE_OP               8 (is)
             11 JUMP_IF_FALSE_OR_POP    21
             14 LOAD_GLOBAL              0 (False)
             17 COMPARE_OP               8 (is)
             20 RETURN_VALUE
        >>   21 ROT_TWO
             22 POP_TOP
             23 RETURN_VALUE
```

```
>>> 1 < 2 < 3

  2           0 LOAD_CONST               1 (1)
              3 LOAD_CONST               2 (2)
              6 DUP_TOP
              7 ROT_THREE
              8 COMPARE_OP               0 (<)
             11 JUMP_IF_FALSE_OR_POP    21
             14 LOAD_CONST               3 (3)
             17 COMPARE_OP               0 (<)
             20 RETURN_VALUE
        >>   21 ROT_TWO
             22 POP_TOP
             23 RETURN_VALUE
```

WHAT TO LEARN?

How The Python Interpreter Works



1. Frames
2. Functions
3. Scope
4. PyObject
5. Iterators
6. Class
7. Generators

HOW?

1. 10-hour CPython Internals Walkthrough
2. github.com/dawran6/cpython-internals-learn
3. Disassemble python code
4. PythonTutor
5. (PyPy)

WALKTHROUGH

PYTHON SOURCECODE TREE

1. `Include/` - all the .h files
2. `Objects/` - all the .c files representing python objects
3. `Python/` - the main runtime
4. `Modules/` - built-in modules implemented in C
5. `Libs/` - standard libraries implemented in Python

WHAT'S USEFUL

1. `Python/ceval.c`
2. `Include/` - object definition
3. `Objects/` - object implementation

OPCODE

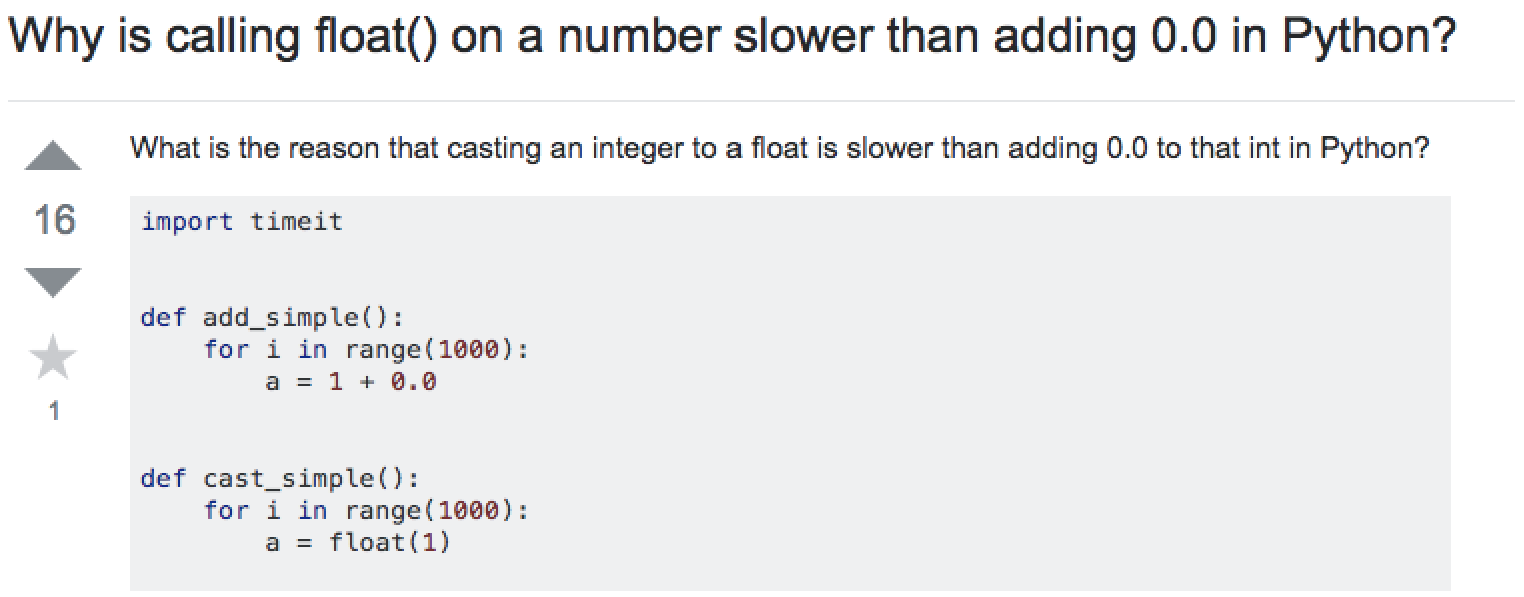
```
  1           0 LOAD_CONST               0 (1)
              3 STORE_NAME               0 (x)

  2           6 LOAD_CONST               1 (2)
              9 STORE_NAME               1 (y)

  3          12 LOAD_NAME                0 (x)
             15 LOAD_NAME                1 (y)
             18 BINARY_ADD
             19 STORE_NAME               2 (z)

  4          22 LOAD_NAME                2 (z)
             25 PRINT_ITEM
             26 PRINT_NEWLINE
             27 LOAD_CONST               2 (None)
```

Q&A

Action

Thanks!

Backup



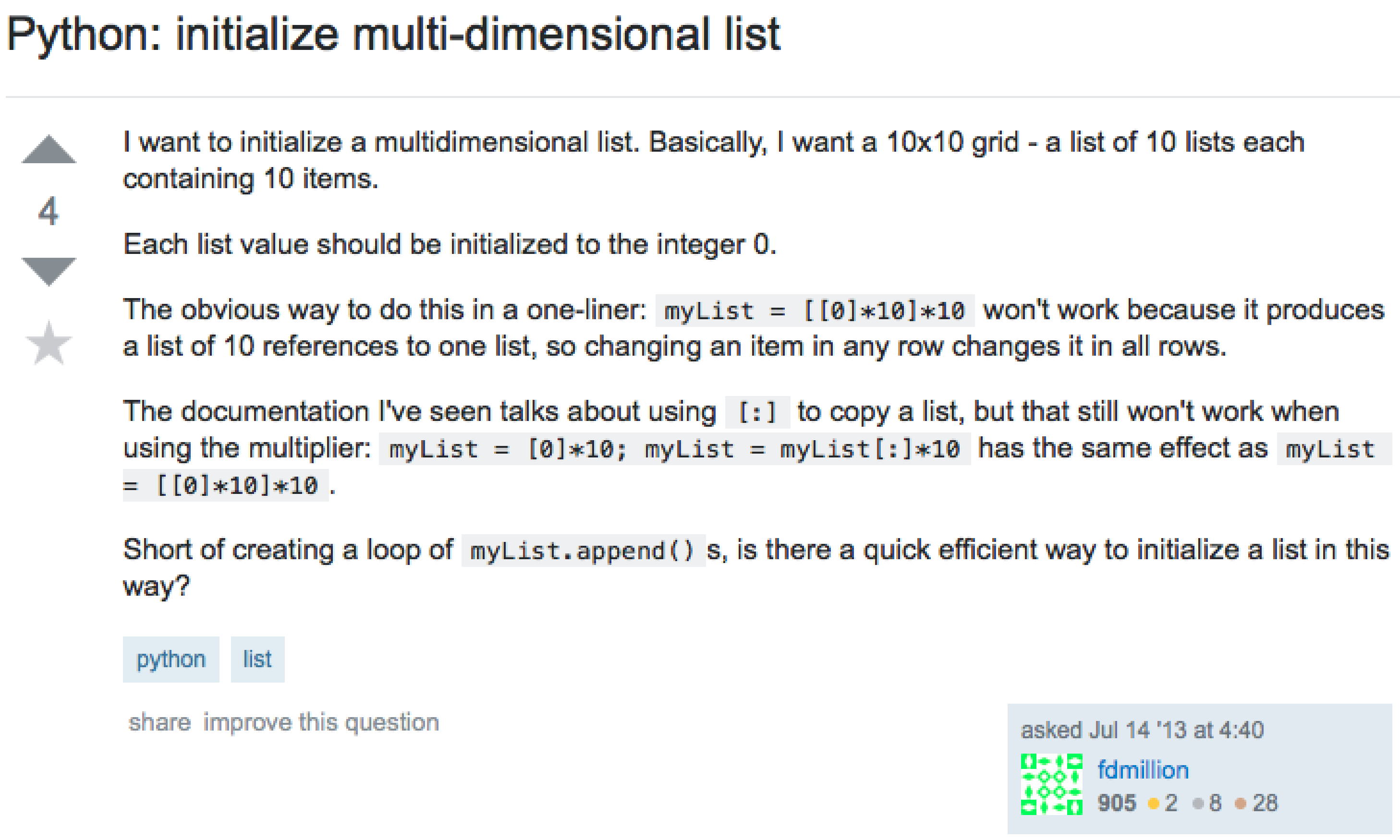
Why is calling float() on a number slower than adding 0.0 in Python?

16

1

What is the reason that casting an integer to a float is slower than adding 0.0 to that int in Python?

```python
import timeit


def add_simple():
    for i in range(1000):
        a = 1 + 0.0


def cast_simple():
    for i in range(1000):
        a = float(1)
```



Python: initialize multi-dimensional list

4

I want to initialize a multidimensional list. Basically, I want a 10x10 grid - a list of 10 lists each containing 10 items.

Each list value should be initialized to the integer 0.

The obvious way to do this in a one-liner: `myList = [[0]*10]*10` won't work because it produces a list of 10 references to one list, so changing an item in any row changes it in all rows.

The documentation I've seen talks about using `[:]` to copy a list, but that still won't work when using the multiplier: `myList = [0]*10; myList = myList[:]*10` has the same effect as `myList = [[0]*10]*10`.

Short of creating a loop of `myList.append()` s, is there a quick efficient way to initialize a list in this way?

python list

share improve this question

asked Jul 14 '13 at 4:40
fdmillion
905 ● 2 ● 8 ● 28