

Faster Python Startup

Jeethu Rao

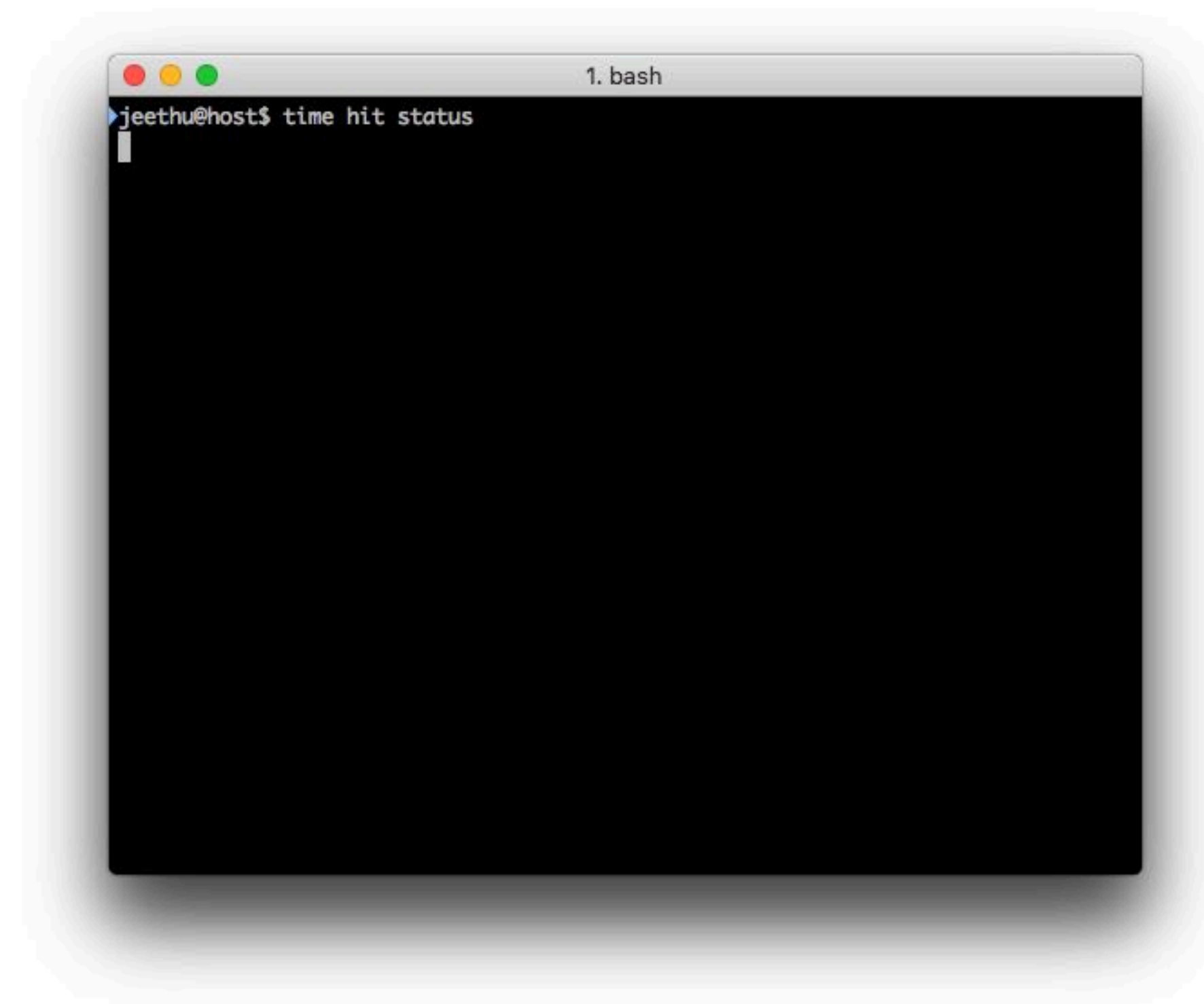
jeethu@fb.com

The slow start problem

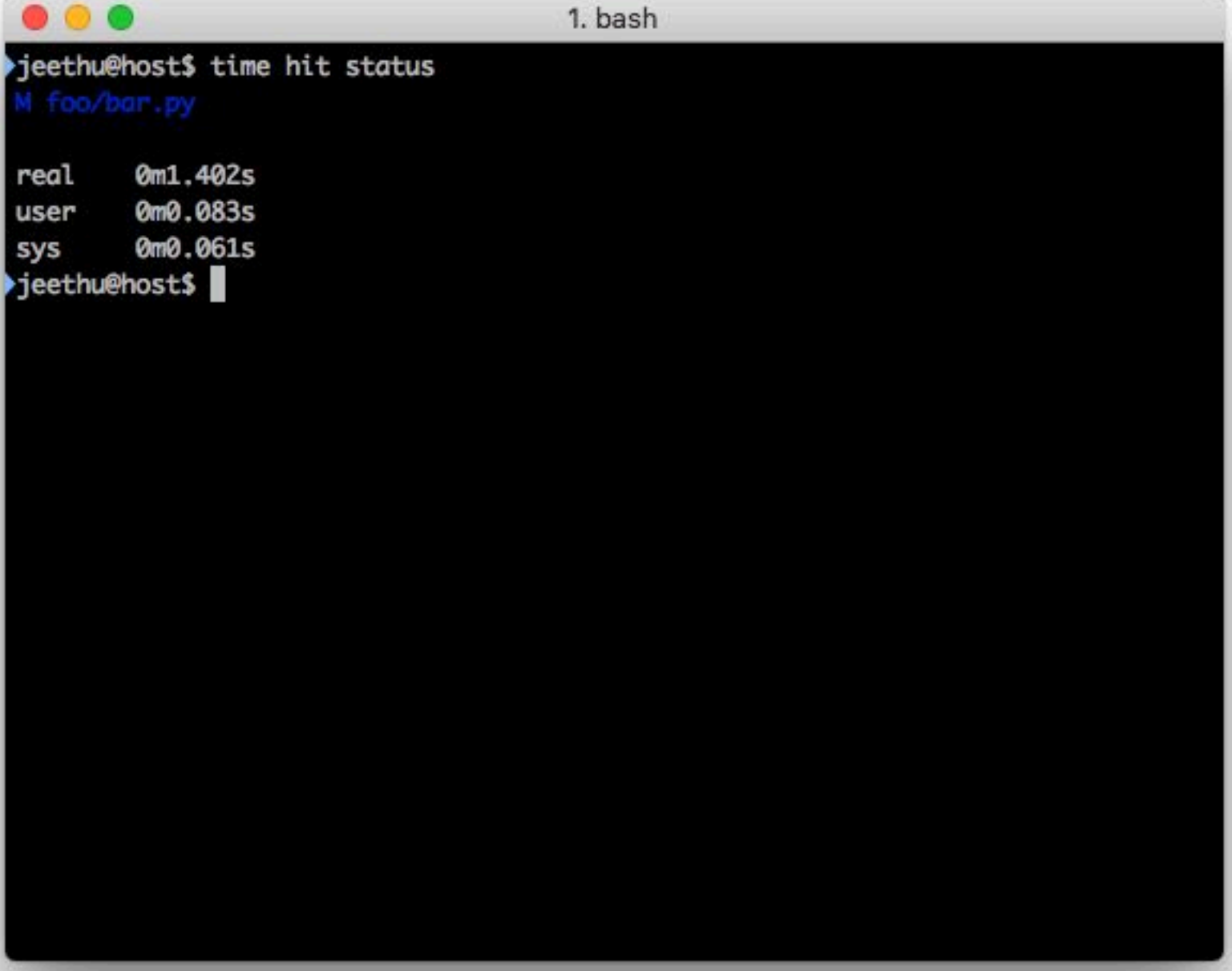
The slow start problem



The slow start problem



The slow start problem



```
1. bash
jeethu@host$ time hit status
M foo/bar.py

real    0m1.402s
user    0m0.083s
sys     0m0.061s
jeethu@host$
```

A terminal window titled "1. bash" showing the execution of the command "time hit status". The output displays timing statistics for the command: real time 0m1.402s, user time 0m0.083s, and system time 0m0.061s. The prompt "jeethu@host\$" is visible before and after the command execution.

The bottleneck

The bottleneck

- With some profiling, we found that up to about 20% of the interpreter startup time was spent on importing modules.

The bottleneck

- With some profiling, we found that up to about 20% of the interpreter startup time was spent on importing modules.
 - Disk I/O

The bottleneck

- With some profiling, we found that up to about 20% of the interpreter startup time was spent on importing modules.
 - Disk I/O
 - Allocating and initializing many tiny objects in memory.

One possible solution

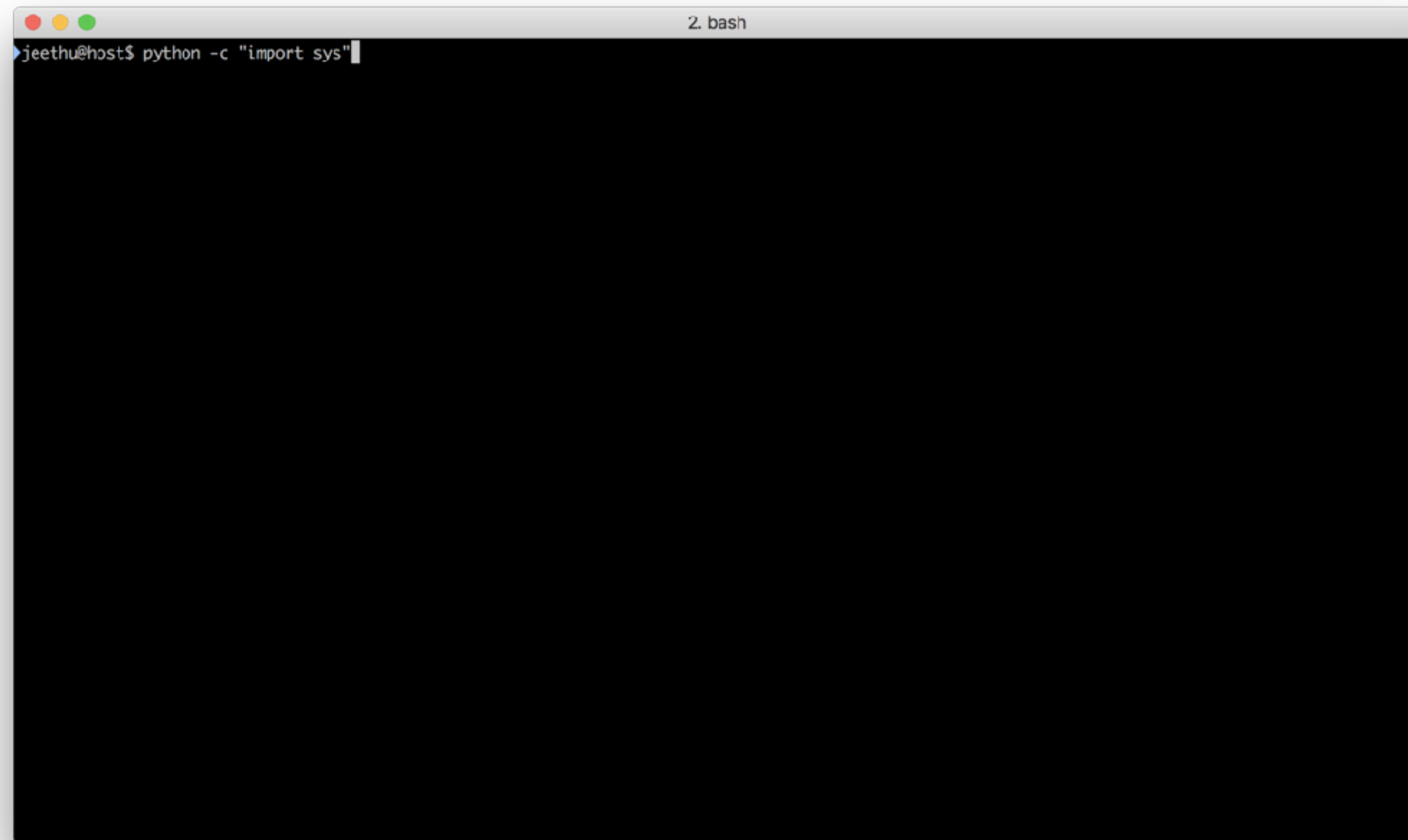
One possible solution

GO



Another (contrived) example

Another (contrived) example

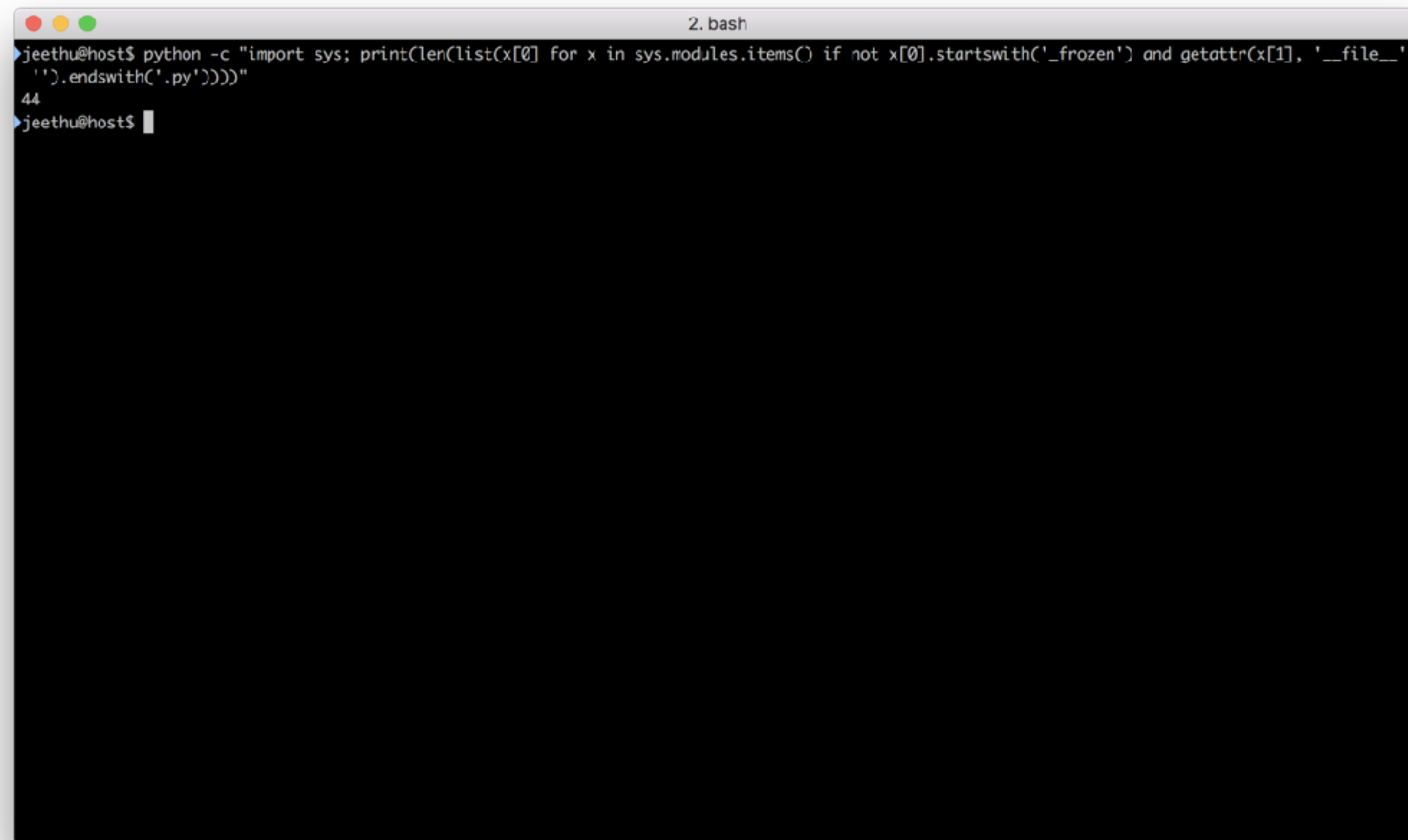
A terminal window titled "2. bash" with a dark background. The prompt "jeethu@host\$" is visible, followed by the command "python -c 'import sys'" and a cursor. The rest of the terminal is empty.

```
jeethu@host$ python -c "import sys"
```

Another (contrived) example

```
2. bash
jeethu@host$ python -c "import sys; print(list(x[0] for x in sys.modules.items() if not x[0].startswith('_frozen') and getattr(x[1], '__file__', '')
.endswith('.py')))"
['encodings', 'codecs', 'encodings.aliases', 'encodings.utf_8', 'encodings.latin_1', 'io', 'abc', '_weakrefset', 'site', 'os', 'stat', 'posixpath',
'genericpath', 'os.path', '_collections_abc', '_sitebuiltins', 'sysconfig', '_sysconfigdata_m_darwin_darwin', '_osx_support', 're', 'enum', 'types',
'functools', 'collections', 'operator', 'keyword', 'heapq', 'reprlib', 'weakref', 'collections.abc', 'sre_compile', 'sre_parse', 'sre_constants', '
copyreg', '_bootlocale', 'importlib', 'importlib._bootstrap', 'importlib._bootstrap_external', 'warnings', 'importlib.util', 'importlib.abc', 'import
lib.machinery', 'contextlib', 'encodings.cp437']
jeethu@host$
```

Another (contrived) example



```
2. bash
jeethu@host$ python -c "import sys; print(len(list(x[0] for x in sys.modules.items() if not x[0].startswith('_frozen') and getattr(x[1], '__file__', '').endswith('.py'))))"
44
jeethu@host$
```

Agenda

Agenda

- Overview of module loading in Python

Agenda

- Overview of module loading in Python
- Improving startup performance

Agenda

- Overview of module loading in Python
- Improving startup performance
- Prior art

Agenda

- Overview of module loading in Python
- Improving startup performance
- Prior art
- Future work

Overview of module loading in Python

Contents of a .pyc file

Overview of module loading in Python

Contents of a .pyc file

pyc file	
4 bytes	magic
4 bytes	modification date
4 bytes	file size
n bytes	marshaled code object

Overview of module loading in Python

The marshal module

Overview of module loading in Python

The marshal module

- Python's low level serialization module and format.

Overview of module loading in Python

The marshal module

- Python's low level serialization module and format.
- Supports serializing and unserializing 13 types and 3 singleton objects.

Overview of module loading in Python

The marshal module

- Python's low level serialization module and format.
- Supports serializing and unserializing 13 types and 3 singleton objects.
- Marshaled object graphs in .pyc files are made up of a subset of the types that marshal supports.

Overview of module loading in Python

The marshal module

- Python's low level serialization module and format.
- Supports serializing and unserializing 13 types and 3 singleton objects.
- Marshaled object graphs in .pyc files are made up of a subset of the types that marshal supports.
 - 8 types, None and Ellipsis objects.

Improving startup performance

The plan

Improving startup performance

The plan

- Bake in frequently used modules into the data segment of the compiled binary.

Improving startup performance

The plan

- Bake in frequently used modules into the data segment of the compiled binary.
- cPython already does this (differently) for importlib bootstrap.

Improving startup performance

The plan

- Bake in frequently used modules into the data segment of the compiled binary.
- cPython already does this (differently) for importlib bootstrap.
- Get rid of the overheads of un-marshaling these modules

Improving startup performance

The plan

- Bake in frequently used modules into the data segment of the compiled binary.
 - cPython already does this (differently) for importlib bootstrap.
- Get rid of the overheads of un-marshaling these modules
 - Use the C99 designated initializers feature for this.

Improving startup performance

The plan

- Bake in frequently used modules into the data segment of the compiled binary.
 - cPython already does this (differently) for importlib bootstrap.
- Get rid of the overheads of un-marshaling these modules
 - Use the C99 designated initializers feature for this.
- Inject these modules into the `sys.modules` dictionary at startup.

Improving startup performance

Reduction in the number of *stat()* and *open()* calls on Linux

```
$ strace -c -e open,stat ./python -c "" # Without patch
```

% time	seconds	usecs/call	calls	errors	syscall
55.16	0.000123	1	98	8	stat
44.84	0.000100	3	29	2	open
100.00	0.000223		127	10	total

```
$ strace -c -e open,stat ./python-patched -c "" # With patch
```

% time	seconds	usecs/call	calls	errors	syscall
71.70	0.000038	1	38	8	stat
28.30	0.000015	1	14	2	open
100.00	0.000053		52	10	total

python-dev thread: <https://mail.python.org/pipermail/python-dev/2018-May/153367.html>

Improving startup performance

Reduction in the number of *stat()* and *open()* calls on Linux

```
$ strace -c -e open,stat ./python -c "import difflib" # Without patch
```

% time	seconds	usecs/call	calls	errors	syscall
72.85	0.000770	4	188	11	stat
27.15	0.000287	6	46	2	open
100.00	0.001057		234	13	total

```
$ strace -c -e open,stat ./python-patched -c "import difflib" # With patch
```

% time	seconds	usecs/call	calls	errors	syscall
80.77	0.000105	2	58	11	stat
19.23	0.000025	1	17	2	open
100.00	0.000130		75	13	total

Improving startup performance

The resulting performance improvement benchmarked on OSX

```
jeethu-mbp:cpython jeethu$ bench "./python.exe -c ''' # Without patch
```

```
benchmarking ./python.exe -c '''
```

```
time          31.46 ms   (31.24 ms .. 31.78 ms)
                1.000 R²   (0.999 R² .. 1.000 R²)
mean          32.08 ms   (31.82 ms .. 32.63 ms)
std dev       778.1 µs   (365.6 µs .. 1.389 ms)
```

```
jeethu-mbp:cpython jeethu$ bench "./python-patched.exe -c ''' # With patch
```

```
benchmarking ./python-patched.exe -c '''
```

```
time          24.86 ms   (24.62 ms .. 25.08 ms)
                0.999 R²   (0.999 R² .. 1.000 R²)
mean          25.58 ms   (25.36 ms .. 25.94 ms)
std dev       592.8 µs   (376.2 µs .. 907.8 µs)
```

Improving startup performance

The resulting performance improvement benchmarked on OSX

```
jeethu-mbp:cpython jeethu$ bench "./python.exe -c 'import difflib'" # Without patch
```

```
benchmarking ./python.exe -c 'import difflib'
```

```
time           32.82 ms   (32.64 ms .. 33.02 ms)
                1.000 R²   (1.000 R² .. 1.000 R²)
mean           33.17 ms   (33.01 ms .. 33.44 ms)
std dev       430.7 µs   (233.8 µs .. 675.4 µs)
```

```
jeethu-mbp:cpython jeethu$ bench "./python-patched.exe -c 'import difflib'" # With patch
```

```
benchmarking ./python-patched.exe -c 'import difflib'
```

```
time           25.30 ms   (25.00 ms .. 25.55 ms)
                0.999 R²   (0.998 R² .. 1.000 R²)
mean           26.78 ms   (26.30 ms .. 27.64 ms)
std dev       1.413 ms   (747.5 µs .. 2.250 ms)
```

Improving startup performance

Generated C code for small int objects

```
1 #include <Python.h>
2
3 #define PyVarObject_HEAD_INIT_RC(rc, type, size)      { PyObject_HEAD_INIT_RC(rc, type) size },
4
5 /* 0 */
6 static struct _longobject _long_obj_1 = {PyVarObject_HEAD_INIT_RC(2, &PyLong_Type, 0) {} };
7
8 /* 1 */
9 static struct {
10 PyObject_VAR_HEAD digit ob_digit[1];
11 } _long_obj_2 = {PyVarObject_HEAD_INIT_RC(2, &PyLong_Type, 1) {1} };
```

Improving startup performance

Generated C code for small string objects

```
1 #include <Python.h>
2
3 #define PyObject_HEAD_INIT_RC(rc, type)      { _PyObject_EXTRA_INIT rc, type },
4
5 /* ' ' */
6 static struct {
7     PyASCIIObject data;
8     uint8_t extra_data[1];
9 } _unicode_obj_1 = {{PyObject_HEAD_INIT_RC(2, &PyUnicode_Type) 0, -1,
10                    {SSTATE_NOT_INTERNERD, PyUnicode_1BYTE_KIND, 1, 1, 1}, NULL },
11                    {0}};
12
13 /* 'hello world' */
14 static struct {
15     PyASCIIObject data;
16     uint8_t extra_data[12];
17 } _unicode_obj_2 = {{PyObject_HEAD_INIT_RC(2, &PyUnicode_Type) 11, -1,
18                    {SSTATE_NOT_INTERNERD, PyUnicode_1BYTE_KIND, 1, 1, 1}, NULL },
19                    {0x68, 0x65, 0x6c, 0x6c, 0x6f, 0x20, 0x77, 0x6f, 0x72, 0x6c, 0x64, 0x0}};
```

Improving startup performance

Generated C code for a tuple containing a bool, int, str and None

```
1 #include <Python.h>
2
3 #define PyObject_HEAD_INIT_RC(rc, type) { PyObject_EXTRA_INIT rc, type },
4
5 #define PyVarObject_HEAD_INIT_RC(rc, type, size) \
6     { PyObject_HEAD_INIT_RC(rc, type) size },
7
8 #define FROM_GC(g) ((PyObject *)(((PyGC_Head *)g)+1))
9
10 #define _GC_UNTRACKED      ((0 & ~_PyGC_REFS_MASK) | \
11     ((size_t)_PyGC_REFS_UNTRACKED) << _PyGC_REFS_SHIFT))
12
13 /* 1 */
14 static struct {
15 PyObject_VAR_HEAD digit ob_digit[1];
16 } _long_obj_1 = {PyVarObject_HEAD_INIT_RC(1, &PyLong_Type, 1) {1} };
17
18 /* 'test' */
19 static struct {
20     PyASCIIObject data;
21     uint8_t extra_data[5];
22 } _unicode_obj_1 = {{PyObject_HEAD_INIT_RC(1, &PyUnicode_Type) 4, -1,
23     {SSTATE_NOT_INTERNEDED, PyUnicode_1BYTE_KIND, 1, 1, 1}, NULL },
24     {0x74, 0x65, 0x73, 0x74, 0x0}};
25
26 /* (True, 1, "test", None) */
27 static struct {
28     struct {
29         void *next;
30         void *prev;
31         Py_ssize_t gc_refs;
32     };
33     PyObject_VAR_HEAD;
34     PyObject *ob_item[4];
35 } _tuple_obj_1 = {{NULL, NULL, _GC_UNTRACKED},
36     PyVarObject_HEAD_INIT_RC(2, &PyTuple_Type, 4)
37     {Py_True, (PyObject*) &_long_obj_1,
38     (PyObject*) &_unicode_obj_1, Py_None}};
```


Improving startup performance

A minor snag - hash randomization and frozenset objects

```
#define PySet_MINSIZE 8

typedef struct {
    PyObject *key;
    Py_hash_t hash;          /* Cached hash code of the key */
} setentry;

typedef struct {
    PyObject_HEAD

    Py_ssize_t fill;        /* Number active and dummy entries*/
    Py_ssize_t used;        /* Number active entries */

    Py_ssize_t mask;

    setentry *table;
    Py_hash_t hash;        /* Only used by frozenset objects */
    Py_ssize_t finger;     /* Search finger for pop() */

    setentry smalltable[PySet_MINSIZE];
    PyObject *weakreflist; /* List of weak references */
} PySetObject;
```

Prior art

Prior art

- Startup snapshots in v8

Prior art

- Startup snapshots in v8
- Emacs unexec() call

Prior art

- Startup snapshots in v8
- Emacs unexec() call
- Image based languages

Prior art

- Startup snapshots in v8
- Emacs unexec() call
- Image based languages
 - SBCL (Common Lisp)
 - SmallTalk

Future work

Future work

- Remove need for a C compiler.

Future work

- Remove need for a C compiler.
- Custom Finder and Loader to load serialized modules.

Thank you!