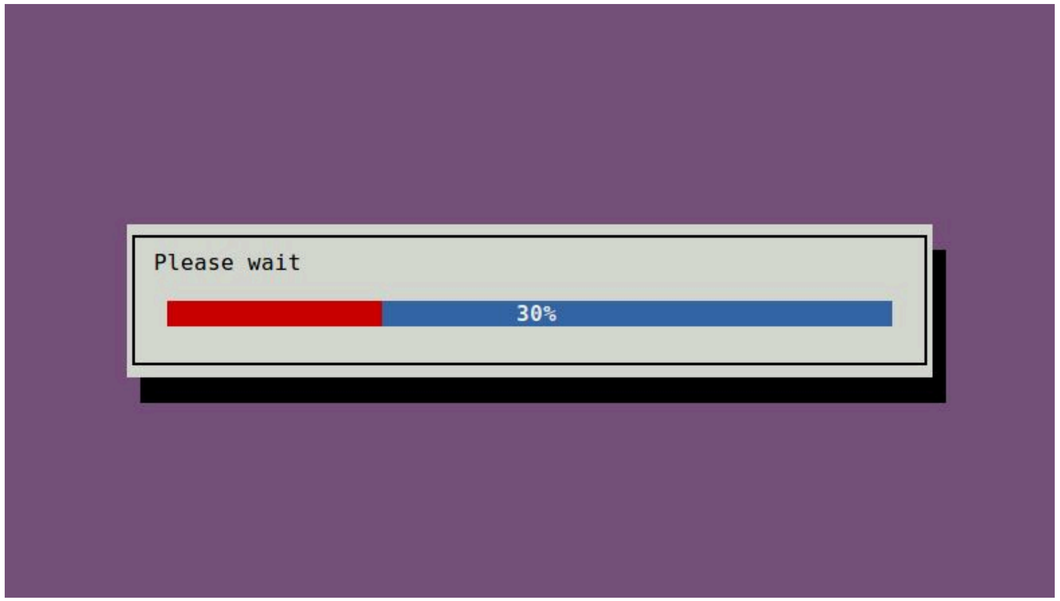
Faster Python Startup

Jeethu Rao

jeethu@fb.com

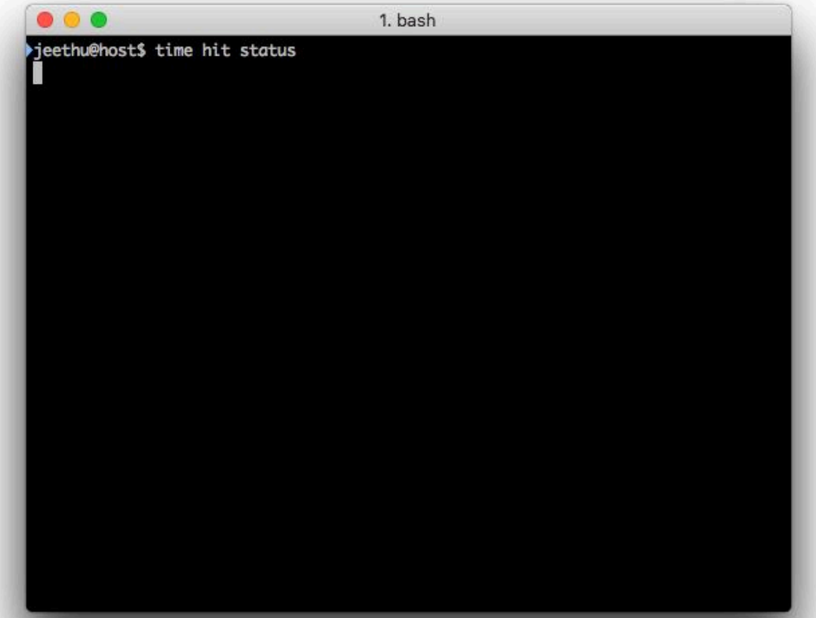
The slow start problem

The slow start problem

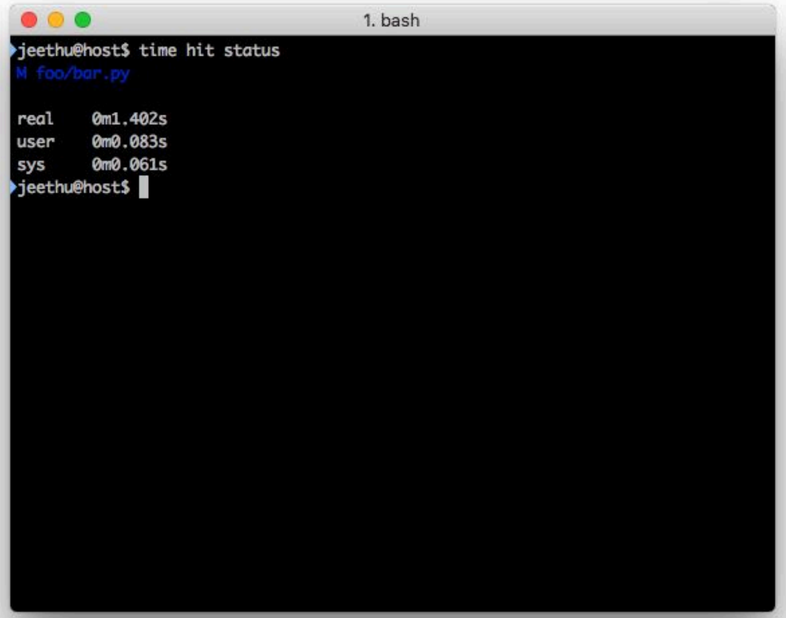


Source: https://www.flickr.com/photos/xmodulo/15829923031

The slow start problem



The slow start problem



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

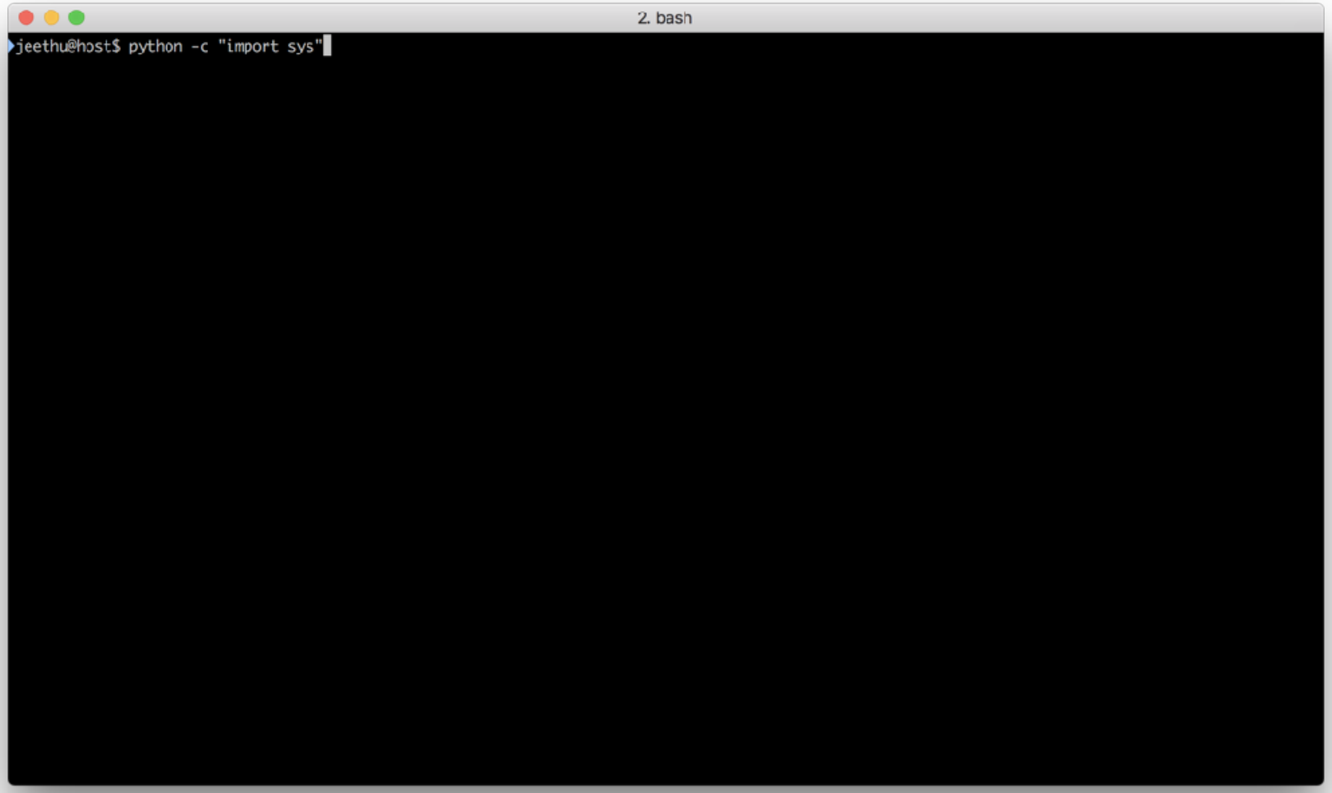
One possible solution

GO

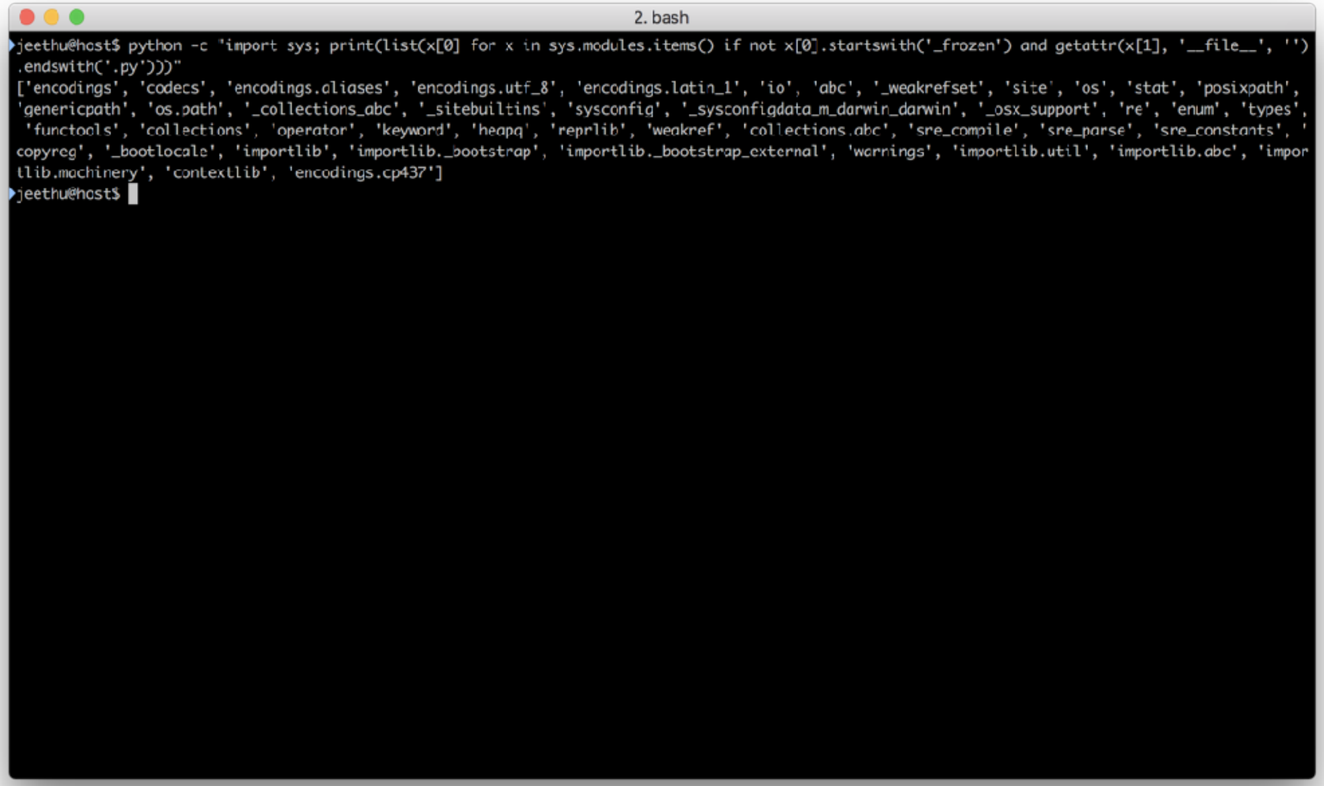
Source: https://www.rust-lang.org/logos/rust-logo-256x256-blk.png

Another (contrived) example

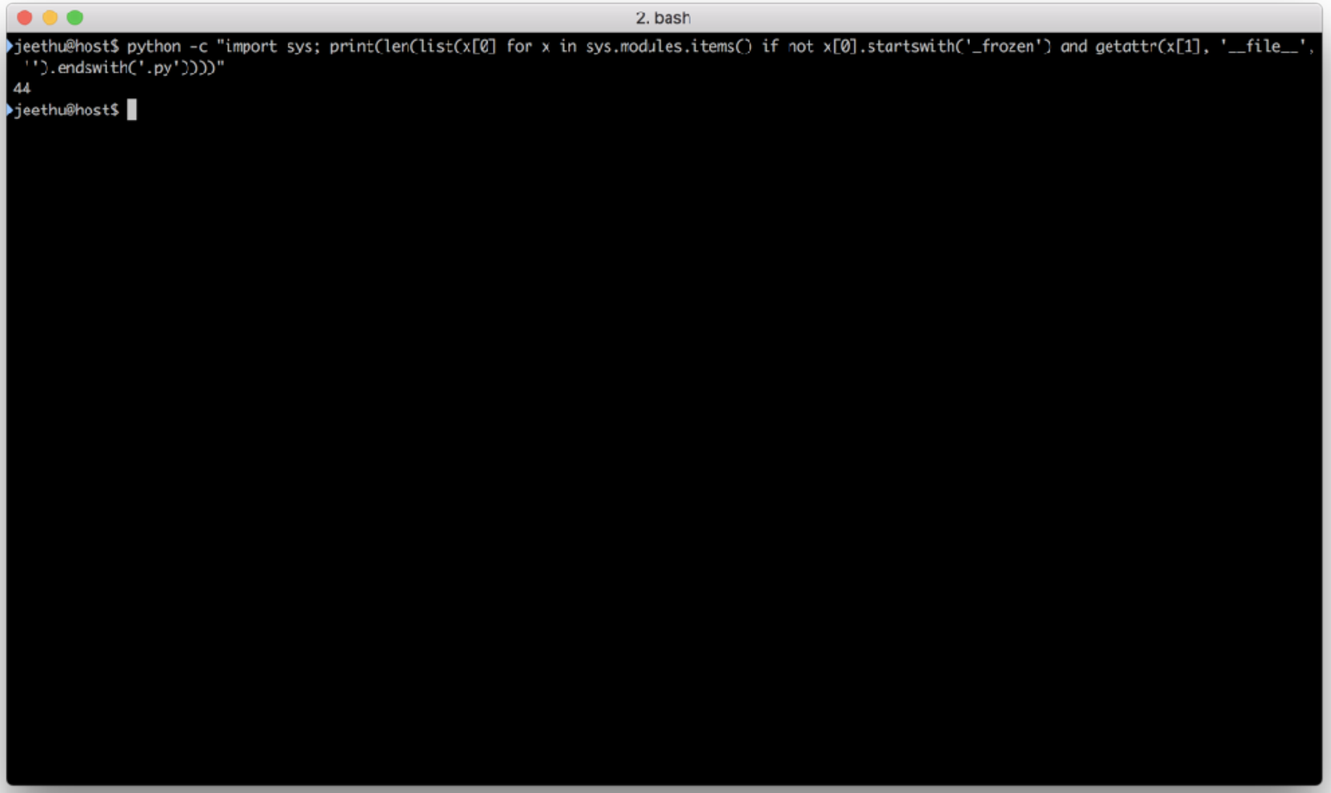
Another (contrived) example



Another (contrived) example



Another (contrived) example



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

```c
#include <Python.h>

#define PyVarObject_HEAD_INIT_RC(rc, type, size)      { PyObject_HEAD_INIT_RC(rc, type) size },

/* 0 */
static struct _longobject _long_obj_1 = {PyVarObject_HEAD_INIT_RC(2, &PyLong_Type, 0) {} };

/* 1 */
static struct {
PyObject_VAR_HEAD digit ob_digit[1];
} _long_obj_2 = {PyVarObject_HEAD_INIT_RC(2, &PyLong_Type, 1) {1} };
```

Improving startup performance

Generated C code for small string objects

```
#include <Python.h>

#define PyObject_HEAD_INIT_RC(rc, type)      { _PyObject_EXTRA_INIT rc, type },

/* '' */
static struct {
  PyASCIIObject data;
  uint8_t extra_data[1];
} _unicode_obj_1 = {{PyObject_HEAD_INIT_RC(2, &PyUnicode_Type) 0, -1,
                     {SSTATE_NOT_INTERNED, PyUnicode_1BYTE_KIND, 1, 1, 1}, NULL },
                    {0}};

/* 'hello world' */
static struct {
  PyASCIIObject data;
  uint8_t extra_data[12];
} _unicode_obj_2 = {{PyObject_HEAD_INIT_RC(2, &PyUnicode_Type) 11, -1,
                     {SSTATE_NOT_INTERNED, PyUnicode_1BYTE_KIND, 1, 1, 1}, NULL },
                    {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

```c
#include <Python.h>

#define PyObject_HEAD_INIT_RC(rc, type) { _PyObject_EXTRA_INIT rc, type },

#define PyVarObject_HEAD_INIT_RC(rc, type, size) \
        { PyObject_HEAD_INIT_RC(rc, type) size },

#define FROM_GC(g) ((PyObject *)(((PyGC_Head *)g)+1))

#define _GC_UNTRACKED     ((0 & ~_PyGC_REFS_MASK) | \
        (((size_t)(_PyGC_REFS_UNTRACKED)) << _PyGC_REFS_SHIFT))

/* 1 */
static struct {
PyObject_VAR_HEAD digit ob_digit[1];
} _long_obj_1 = {PyVarObject_HEAD_INIT_RC(1, &PyLong_Type, 1) {1} };

/* 'test' */
static struct {
  PyASCIIObject data;
  uint8_t extra_data[5];
} _unicode_obj_1 = {{PyObject_HEAD_INIT_RC(1, &PyUnicode_Type) 4, -1,
                     {SSTATE_NOT_INTERNED, PyUnicode_1BYTE_KIND, 1, 1, 1}, NULL },
                    {0x74, 0x65, 0x73, 0x74, 0x0}};

/* (True, 1, "test", None) */
static struct {
    struct {
      void *next;
      void *prev;
      Py_ssize_t gc_refs;
    };
    PyObject_VAR_HEAD;
    PyObject *ob_item[4];
} _tuple_obj_1 = {{NULL, NULL, _GC_UNTRACKED},
                  PyVarObject_HEAD_INIT_RC(2, &PyTuple_Type, 4)
                  {Py_True, (PyObject*) &_long_obj_1,
                   (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!