Porting Python to run without an OS

Josh Triplett
josh@joshtriplett.org

PyCon 2015

We ported Python to GRUB to run on BIOS and EFI.

We ported Python to GRUB to run on BIOS and EFI.

- Why?

We ported Python to GRUB to run on BIOS and EFI.

- Why?
- What do we provide?

We ported Python to GRUB to run on BIOS and EFI.

- Why?
- What do we provide?
- How does it work?

We ported Python to GRUB to run on BIOS and EFI.

- Why?
- What do we provide?
- How does it work?
- Platform APIs

We ported Python to GRUB to run on BIOS and EFI.

- Why?
- What do we provide?
- How does it work?
- Platform APIs
- Many demos along the way

Why?

- To test hardware, BIOS, ACPI, and EFI

Why?

- To test hardware, BIOS, ACPI, and EFI
- To replace myriad one-off tools targeting DOS or EFI

Why?

- To test hardware, BIOS, ACPI, and EFI
- To replace myriad one-off tools targeting DOS or EFI
- To avoid writing more C, or pseudo-shell with C expressions

Why?

- To test hardware, BIOS, ACPI, and EFI
- To replace myriad one-off tools targeting DOS or EFI
- To avoid writing more C, or pseudo-shell with C expressions
- As an exploratory environment

What?

- Target platforms: 32-bit on BIOS, 32-bit EFI, 64-bit EFI

What?

- Target platforms: 32-bit on BIOS, 32-bit EFI, 64-bit EFI
- CPython 2.7 (target audience familiarity)

What?

- Target platforms: 32-bit on BIOS, 32-bit EFI, 64-bit EFI
- CPython 2.7 (target audience familiarity)
- Interactive REPL (including history and tab-completion)

What?

- Target platforms: 32-bit on BIOS, 32-bit EFI, 64-bit EFI
- CPython 2.7 (target audience familiarity)
- Interactive REPL (including history and tab-completion)
- Substantial fraction of the standard library

What?

- Target platforms: 32-bit on BIOS, 32-bit EFI, 64-bit EFI
- CPython 2.7 (target audience familiarity)
- Interactive REPL (including history and tab-completion)
- Substantial fraction of the standard library
- Additional modules for platform support: CPU, SMP, ACPI, EFI...

What?

- Target platforms: 32-bit on BIOS, 32-bit EFI, 64-bit EFI
- CPython 2.7 (target audience familiarity)
- Interactive REPL (including history and tab-completion)
- Substantial fraction of the standard library
- Additional modules for platform support: CPU, SMP, ACPI, EFI...
- Test suite and exploratory tools, all written in Python

```
>>> import demo
```

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make
  - Using host Linux toolchain

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make
  - Using host Linux toolchain
  - No GNU target string (cpu-vendor-os) for GRUB

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make
  - Using host Linux toolchain
  - No GNU target string (cpu-vendor-os) for GRUB
  - No target headers in "default" path for toolchain

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make
  - Using host Linux toolchain
  - No GNU target string (cpu-vendor-os) for GRUB
  - No target headers in "default" path for toolchain
- Add all of the necessary Python source files

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make
  - Using host Linux toolchain
  - No GNU target string (cpu-vendor-os) for GRUB
  - No target headers in "default" path for toolchain
- Add all of the necessary Python source files
- Manually write `pyconfig.h`

How?

- `PyRun_InteractiveLoop(stdin, "<stdin>");`
- Can't use Python's configure and make
  - Using host Linux toolchain
  - No GNU target string (cpu-vendor-os) for GRUB
  - No target headers in "default" path for toolchain
- Add all of the necessary Python source files
- Manually write `pyconfig.h`
- Provide functions expected by Python

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc
  - Hack to seek backwards by one

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc
  - Hack to seek backwards by one
- qsort

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc
  - Hack to seek backwards by one
- qsort
- Floating-point math functions (fdlibm)

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc
  - Hack to seek backwards by one
- qsort
- Floating-point math functions (fdlibm)
- printf/sprintf

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc
  - Hack to seek backwards by one
- qsort
- Floating-point math functions (fdlibm)
- printf/sprintf
  - Mostly used GRUB's

C functions expected by Python

- fstat/isatty/lseek etc on file descriptors
  - Wrote a simple file descriptor table
- ungetc
  - Hack to seek backwards by one
- qsort
- Floating-point math functions (fdlibm)
- printf/sprintf
  - Mostly used GRUB's
  - Had to fix bugs (%%)

Performance issues

- Slow boot time
- Extra painful in CPU simulation environments

Performance issues

- Slow boot time
- Extra painful in CPU simulation environments
- Python parser reads characters and uses ungetc
- Minimal disk caching

Performance issues

- Slow boot time
- Extra painful in CPU simulation environments
- Python parser reads characters and uses ungetc
- Minimal disk caching
- Compile Python on host, use it to byte-compile into .pyc files

Performance issues

- Slow boot time
- Extra painful in CPU simulation environments
- Python parser reads characters and uses ungetc
- Minimal disk caching
- Compile Python on host, use it to byte-compile into .pyc files
- No mtime support, so zero the mtime

Performance issues

- Slow boot time
- Extra painful in CPU simulation environments
- Python parser reads characters and uses ungetc
- Minimal disk caching
- Compile Python on host, use it to byte-compile into .pyc files
- No mtime support, so zero the mtime
- Still slow due to stat

Performance issues

- Slow boot time
- Extra painful in CPU simulation environments
- Python parser reads characters and uses ungetc
- Minimal disk caching
- Compile Python on host, use it to byte-compile into .pyc files
- No mtime support, so zero the mtime
- Still slow due to stat
- zipimport

readline

- Wanted history and completion
- readline depends heavily on POSIX and tty
- Didn't want to write a pile of C code

readline

- Wanted history and completion
- readline depends heavily on POSIX and tty
- Didn't want to write a pile of C code
- Wrote Python's readline in Python

readline

- Wanted history and completion
- readline depends heavily on POSIX and tty
- Didn't want to write a pile of C code
- Wrote Python's readline in Python
- Implemented line editing, history, and completion in pure Python
- Set `PyOS_ReadlineFunctionPointer` to a C function that calls a previously set Python callback

(python) filesystem

- Wanted to construct dynamic menus in GRUB

(python) filesystem

- Wanted to construct dynamic menus in GRUB
- GRUB has disk and filesystem providers for (hd0), (cd)

`(python)` filesystem

- Wanted to construct dynamic menus in GRUB
- GRUB has disk and filesystem providers for `(hd0)`, `(cd)`
- Added a `(python)` device and filesystem
- `(python)` implementation calls Python callbacks to read
- Python code can add arbitrary in-memory files

`(python)` filesystem

- Wanted to construct dynamic menus in GRUB
- GRUB has disk and filesystem providers for `(hd0)`, `(cd)`
- Added a `(python)` device and filesystem
- `(python)` implementation calls Python callbacks to read
- Python code can add arbitrary in-memory files
- `configfile (python)/menu.cfg`

bits module

- Various functions to access hardware functionality:
- CPUID
- MSRs
- Memory-mapped I/O
- I/O ports

SMP

- Need to collect or modify state from all CPUs

SMP

- Need to collect or modify state from all CPUs
- Wake up all CPUs at startup
- Put them in a power-efficient sleep (mwait) waiting for work to do
- Various functions to wake up CPUs and run specific functions on them

SMP

- Need to collect or modify state from all CPUs
- Wake up all CPUs at startup
- Put them in a power-efficient sleep (mwait) waiting for work to do
- Various functions to wake up CPUs and run specific functions on them
- Python can easily correlate data across CPUs (dict, set)

ACPI

- Advanced Configuration and Power Interface
- Configuration format used in PC firmware
- Static data tables and evaluatable bytecode methods

ACPI

- Advanced Configuration and Power Interface
- Configuration format used in PC firmware
- Static data tables and evaluatable bytecode methods
- Ported ACPICA reference implementation to BITS
- Added Python bindings
  - Evaluate arbitary ACPI methods
  - Arguments converted from Python to ACPI
  - Result converted from ACPI to Python

ACPI

- Advanced Configuration and Power Interface
- Configuration format used in PC firmware
- Static data tables and evaluatable bytecode methods
- Ported ACPICA reference implementation to BITS
- Added Python bindings
  - Evaluate arbitary ACPI methods
  - Arguments converted from Python to ACPI
  - Result converted from ACPI to Python
- `>>> import acpi`
- `>>> acpi.dump("_HID")`
- More detailed hardware exploration demoed elsewhere

EFI

- Extensible Firmware Interface
- Replacement for classic PC BIOS
- “Extensible”:
  - Everything’s a “protocol”
  - Protocols include native C functions to call
- `>>> import efi`

ctypes and libffi

- ctypes: Interface to C types and functions from Python
- libffi: Foreign Function Interface, implements function calling conventions

ctypes and libffi

- ctypes: Interface to C types and functions from Python
- libffi: Foreign Function Interface, implements function calling conventions
- Ported libffi to run in GRUB
- Added support for the EFI calling convention

ctypes and libffi

- ctypes: Interface to C types and functions from Python
- libffi: Foreign Function Interface, implements function calling conventions
- Ported libffi to run in GRUB
- Added support for the EFI calling convention
- Declare EFI protocols and functions in Python
- Call them from Python

ctypes and libffi

- ctypes: Interface to C types and functions from Python
- libffi: Foreign Function Interface, implements function calling conventions
- Ported libffi to run in GRUB
- Added support for the EFI calling convention
- Declare EFI protocols and functions in Python
- Call them from Python
- No C code required

efi_file

- File-like object built on `EFI_FILE_PROTOCOL`
- Make directories, write files
- `efi.get_boot_fs().mkdir("dir").create("f").write("Hi")`

EFI Graphics Output Protocol

- GOP provides functions to read and write screen contents.

EFI Graphics Output Protocol

- GOP provides functions to read and write screen contents.
- Such as presentation slides.
- Hello from EFI, BITS, and the bits.present module!
- No new C code needed to implement this presentation and demo.

EFI Graphics Output Protocol

- GOP provides functions to read and write screen contents.
- Such as presentation slides.
- Hello from EFI, BITS, and the bits.present module!
- No new C code needed to implement this presentation and demo.

BIOS Implementation Test Suite (BITS)

`http://biosbits.org/`