MicroPython: a journey from Kickstarter to Space

Damien P. George

George Robotics Limited,
Cambridge, UK

PyCon AU, Melbourne, 13th August 2016
Motivation for MicroPython

Electronics circuits now pack an enormous amount of functionality in a tiny package.

Need a way to control all these sophisticated devices.

Scripting languages enable rapid development.

Is it possible to put Python on a microcontroller?

Why is it hard?

- Very little memory (RAM, ROM) on a microcontroller.
Why Python?

- High-level language with powerful features (classes, list comprehension, generators, exceptions, ...).
- Large existing community.
- Very easy to learn, powerful for advanced users: shallow but long learning curve.
- Ideal for microcontrollers: native bitwise operations, procedural code, distinction between int and float, robust exceptions.
- Lots of opportunities for optimisation (Python is compiled).
Why can’t we use CPython? (or PyPy?)

- Integer operations:
  Integer object (max 30 bits): 4 words (16 bytes)
  Preallocates $257 + 5 = 262$ ints $\rightarrow$ 4k RAM!
  Could ROM them, but that’s still 4k ROM.
  And each integer outside the preallocated ones would be another 16 bytes.

- Method calls:
  `led.on()`: creates a bound-method object, 5 words (20 bytes)
  `led.intensity(1000)` $\rightarrow$ 36 bytes RAM!

- For loops: require heap to allocate a range iterator.
Pyboard demo!

D.P. George

MicroPython 5/32
Crowdfunding via Kickstarter

Kickstarter is a good way to see if your idea has traction, or not.

- 30th April 2013: start!
- 17th September: flashing LED with button in bytecode Python.
- 21st October: REPL, filesystem, USB VCP and MSD on PYBv2.

1 weekend to make the video.

Kickstarter launched on 13 November 2013, ran for 30 days.
Total backers: 1,931
Total raised: £97,803 ($180k)

Officially finished 12 April 2015.

(Note: Kickstarter, since 2009, collected so far over US$1 billion in funds)
The Kickstarter journey

Post funding

Once your project's funding has ended, it will continue to be public and remain preserved as is. You'll be able to use Project Updates and Project FAQs to provide new information about your project's development post-funding.

Your responsibility

If your project is successfully funded, you are required to fulfill all rewards or refund any backer whose reward you do not or cannot fulfill. A failure to do so could result in damage to your reputation or even legal action on behalf of your backers. For more on accountability, see the FAQ.

I have read these important reminders; the Terms of Use, Privacy Policy, and the Kickstarter Project Guidelines.

Launch project now
The Kickstarter journey

Micro Python: Python for microcontrollers
by Damien George

Your project has launched!

Congratulations, your project is live on Kickstarter!

Drumroll...

Here's your project URL:

The countdown begins now. You know what to do: tell people about it!

GOOD LUCK!
Kickstarter

The Python language made lean and fast to run on microcontrollers. For beginners and experts, control your electronic project with ease.
The Kickstarter journey

Micro Python: Python for microcontrollers
by Damien George

1,930 backers
£97,749
pledged of £15,000 goal
1 second to go

Back This Project
£1 minimum pledge

This project will be funded on Friday,
Dec 13, 10:09am GMT.

The Python language made lean and fast to run on
microcontrollers. For beginners and experts, control

D.P. George
MicroPython
Manufacturing

Jaltek Systems, Luton UK — manufactured 13,000+ boards.
Boards, headers, servo motors, ...
Programming and packing

D.P. George MicroPython 12/32
Packing and shipping
Packing and shipping
It’s all about the RAM

If you ask me ‘why is it done that way?’, I will most likely answer: ‘to minimise RAM usage’.

▶ Interned strings, most already in ROM.
▶ Small integers stuffed in a pointer.
▶ Optimised method calls (thanks PyPy!).
▶ Range object is optimised (if possible).
▶ Python stack frames live on the C stack.
▶ ROM absolutely everything that can be ROMed!
▶ Garbage collection only (no reference counts).
▶ Exceptions implemented with custom setjmp/longjmp.
Internals: parser, lexer, compiler and runtime

**Import**
- Built-in modules are added to scope
- User modules are compiled and executed

**Runtime**
- Support code for executing Python code
  - Built-in types (int, float, str, tuple, list, dict, ...)
  - Built-in exceptions (TypeError, IndexError, ValueError, ...)
  - Built-in functions (max, min, range, sort, sum, ...)
  - Built-in modules (sys, os, array, math, ...)
  - Load/store global variables
  - Execute functions/methods by dispatching
  - Glue code, etc

**Virtual Machine**
- Executes bytecode
  - Source info
  - Line info
  - Bytecode data
  - Executed by VM

**Compiler**
- Turns parse tree into code

**Parser**
- Turns tokens into a parse tree

**Lexer**
- Turns script into a stream of tokens

**REPL prompt**
- User scripts

**Eval/Exec/Compile String**
- Can load
- Can produce
- Calls

**Evaluate/Execute/Compile String**
- Produces tokens
- Produces parse tree
- Produces virtual machine

**External Bindings**
- User defined builtins using C
- Or other native language at compile time

**Types of Code**
- Native code
- Machine code
- Proper Python semantics can be executed directly
- Viper code
- Machine code
- Typed version of Python can be executed directly

**D.P. George MicroPython 16/32**
Object representation

A MicroPython object is a machine word, and has 3 different forms.

Integers:
- xxxxxxxxx xxxxxxxxx xxxxxxxxx xxxxxxxx1
- Transparent transition to arbitrary precision integers.

Strings:
- xxxxxxxxx xxxxxxxxx xxxxxxxxx xxxxxxxxx10

Objects:
- xxxxxxxxx xxxxxxxxx xxxxxxxxx xxxxxxxx00
- A pointer to a structure.
- First element is a pointer to a type object.
- ROMable (type, tuple, dictionary, function, module, . . .).

Optional 30-bit single-precision FP stuffing.

Work on LEON port added representation for 64-bit NaN boxing.
Emitters: bytecode

@micropython.bytecode
def add(x, y):
    return x + y

Compiles to:

00:  b0  LOAD_FAST_0
01:  b1  LOAD_FAST_1
02:  db  BINARY_OP_ADD
03:  5b  RETURN_VALUE
@micropython.native

def add(x, y):
    return x + y
Emitters: inline assembler!

```python
@micropython.asm_thumb
def sum_bytes(r0, r1):
    mov(r2, 0)
    b(loop_entry)
label(loop1)
ldrb(r3, [r1, 0])
add(r2, r2, r3)
add(r1, r1, 1)
sub(r0, r0, 1)
label(loop_entry)
cmp(r0, 0)
    bgt(loop1)
mov(r0, r2)
```

Call as normal: `print(sum_bytes(4, b'abcd'))`
Coding style

MicroPython does not follow traditional software engineering practices:

- optimise first;
- creative solutions and tricks;
- sacrifice clarity to get smaller code;
- sacrifice efficiency to get smaller code (esp. less-used features);
- use of goto not discouraged;
- optimise to minimise stack usage;
- make decisions based on analysis.
Code dashboard

http://micropython.org/resources/code-dashboard/
GitHub and the open-source community

https://github.com/micropython

MicroPython is a *public* project on GitHub.

- A global coding conversation.
- Anyone can clone the code, make a fork, submit issues, make pull requests.
- MicroPython has over 3500 “stars” (top 0.02%), and more than 740 forks.
- Contributions come from many people (120+), with many different systems.
- Leads to: more robust code and build system, more features, more supported hardware.
- Hard to balance inviting atmosphere with strict code control.

A big project needs many contributors, and open-source allows such projects to exist.
<table>
<thead>
<tr>
<th>Rank</th>
<th>Project Name</th>
<th>Language</th>
<th>Stars</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>torvalds/linux</td>
<td>C</td>
<td>25,156</td>
</tr>
<tr>
<td>2</td>
<td>nwjs/nw.js</td>
<td>C++</td>
<td>24,175</td>
</tr>
<tr>
<td>3</td>
<td>atom/electron</td>
<td>C++</td>
<td>15,777</td>
</tr>
<tr>
<td>4</td>
<td>ariya/phantomjs</td>
<td>C++</td>
<td>15,084</td>
</tr>
<tr>
<td>5</td>
<td>antirez/redis</td>
<td>C</td>
<td>14,701</td>
</tr>
<tr>
<td>6</td>
<td>facebook/hhvm</td>
<td>C++</td>
<td>12,497</td>
</tr>
<tr>
<td>7</td>
<td>textmate/textmate</td>
<td>C++</td>
<td>10,490</td>
</tr>
<tr>
<td>8</td>
<td>git/git</td>
<td>C</td>
<td>10,082</td>
</tr>
<tr>
<td>98</td>
<td>jonas/tig</td>
<td>C</td>
<td>2,361</td>
</tr>
<tr>
<td>99</td>
<td>swoole/swoole-src</td>
<td>C</td>
<td>2,319</td>
</tr>
<tr>
<td>100</td>
<td>raspberrypi/linux</td>
<td>C</td>
<td>2,310</td>
</tr>
<tr>
<td>101</td>
<td>SFML/SFML</td>
<td>C++</td>
<td>2,282</td>
</tr>
<tr>
<td>102</td>
<td>philipl/pifs</td>
<td>C</td>
<td>2,281</td>
</tr>
<tr>
<td>103</td>
<td>micropython/micropython</td>
<td>C</td>
<td>2,272</td>
</tr>
<tr>
<td>104</td>
<td>sqlitebrowser/sqlitebrowser</td>
<td>C++</td>
<td>2,269</td>
</tr>
<tr>
<td>105</td>
<td>rswier/c4</td>
<td>C</td>
<td>2,255</td>
</tr>
<tr>
<td>106</td>
<td>philsquared/Catch</td>
<td>C++</td>
<td>2,228</td>
</tr>
<tr>
<td>107</td>
<td>joyent/http-parser</td>
<td>C</td>
<td>2,225</td>
</tr>
<tr>
<td>108</td>
<td>nanomsg/nanomsg</td>
<td>C</td>
<td>2,220</td>
</tr>
<tr>
<td>109</td>
<td>ivansafrin/Polycode</td>
<td>C++</td>
<td>2,195</td>
</tr>
<tr>
<td>110</td>
<td>libuv/libuv</td>
<td>C</td>
<td>2,192</td>
</tr>
<tr>
<td>111</td>
<td>mpv-player/mpv</td>
<td>C</td>
<td>2,173</td>
</tr>
<tr>
<td>112</td>
<td>arut/nginx-rtmp-module</td>
<td>C</td>
<td>2,158</td>
</tr>
<tr>
<td>113</td>
<td>numpy/numpy</td>
<td>C</td>
<td>2,157</td>
</tr>
</tbody>
</table>

...
Microcontroller hardware

- **pyboard**: 192k RAM, 1Mb flash; 168 MHz Cortex M4F MCU
- **WiPy**: 256k RAM+code; 80 MHz Cortex M3 MCU
- **ESP8266**: 96k RAM, 1Mb+ flash; 80-160 MHz Xtenssa CPU

Larger computers

- no real limit on RAM
- good for testing
- useful for a light-weight Python interpreter (eg OpenWrt)
The BBC micro:bit project

- Nordic BLE device: 256k flash ROM, 16k RAM
- 5x5 LED matrix, accelerometer, compass
- 700k+ devices given free to UK year 7's
- Python already taught in schools
  → easy transition to “physical computing”
micro:bit demo!
The port to LEON/SPARC/RTEMS for Space

- separation of the VM and compiler
- cross compiler and persistent bytecode
- 64-bit NaN-boxing object model
- understanding of determinism
- support for SPARC v8 architecture
- multiple VMs in the same address space

Use-case: satellite control, the application layer.
And then went back for a second Kickstarter!

Kickstarter #2 was a pure software campaign.

 Finished on 2nd March 2016 with 1384 backers, £28,334 ($50k).
Challenges

Technical challenges:

▶ slow feature creep while remaining “micro”
▶ configurability of code is getting harder
▶ memory management, always hard

Biggest challenges are non-technical:

▶ managing growth of community
▶ high expectations from the community from free software
▶ lots of contributions on GitHub, takes a lot of time to review them
▶ monetising the project to keep it alive
A powerful and modern language, large community, powerful tools — now available for constrained/embedded systems!

Applications:
▶ schools/teaching (micro:bit, pyboard, high-schools and universities)
▶ hobbyists and hackers
▶ embedded engineers, to make prototyping easier
▶ rapid development of IoT applications
▶ light-weight Internet servers
▶ in Space: general purpose application-language for payloads

Continued software/hardware development:
▶ Python 3.5 support, and improved compatibility with CPython
▶ continued development of ESP8266 port
▶ multithreading support on pyboard
▶ more features for the micro:bit
▶ further work with ESA
▶ easier embedding
▶ development of new boards