MicroPython: a journey from Kickstarter to Space

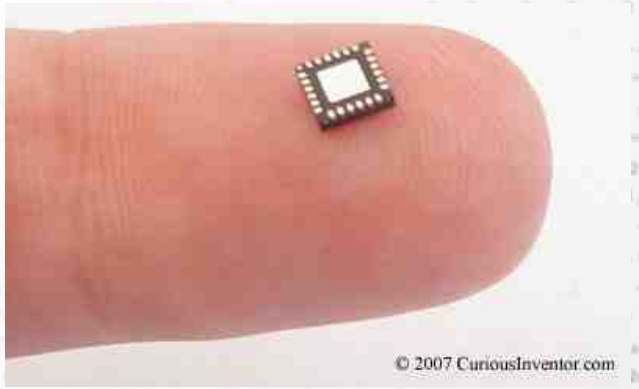
Damien P. George

George Robotics Limited,
Cambridge, UK

PyCon AU, Melbourne, 13th August 2016

Motivation for MicroPython

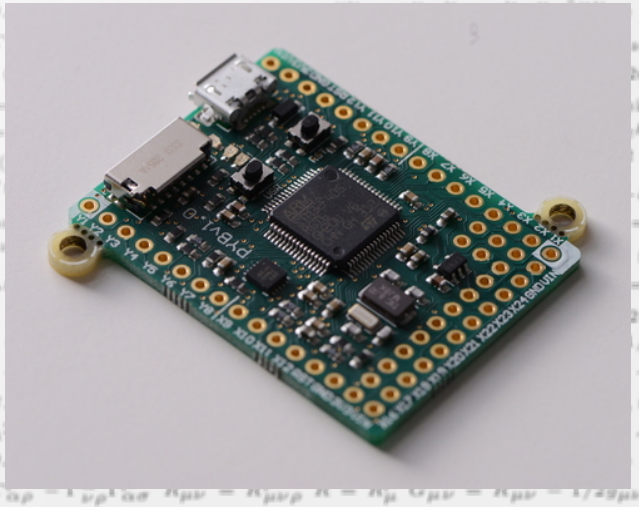
Electronics circuits now pack an enormous amount of functionality in a tiny package.



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 $\longrightarrow$ 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)

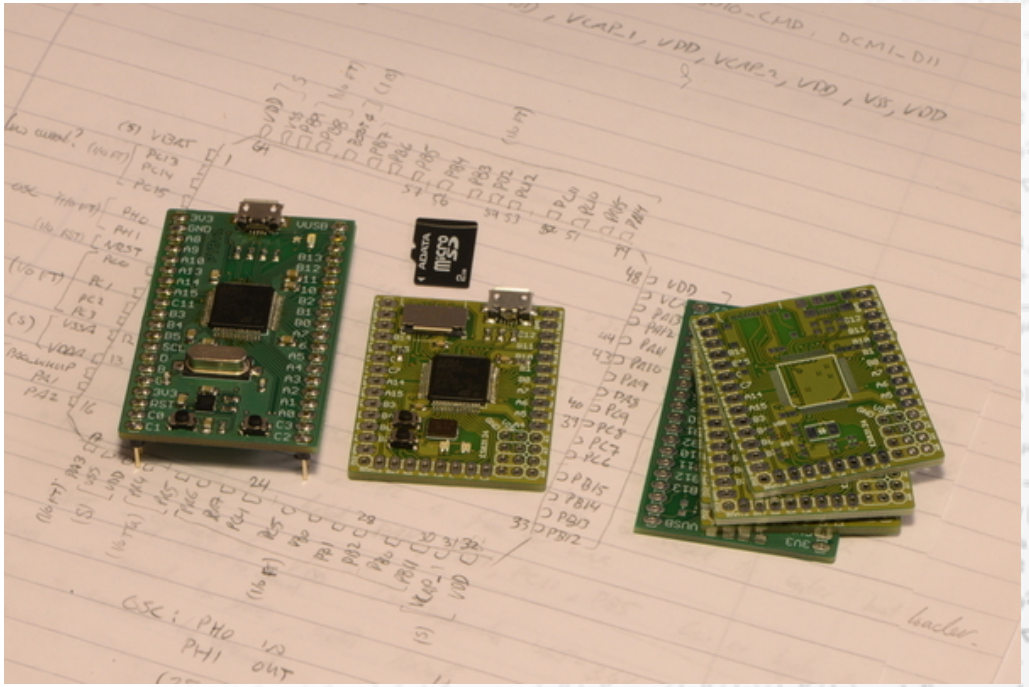
  led.intensity(1000) $\longrightarrow$ 36 bytes RAM!
- For loops: require heap to allocate a range iterator.

Pyboard demo!

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

Guidelines | Basics | Rewards | Story | About you | Account | Review | Launch | Preview 0

funded.

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

Support

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

Launch project now

The Kickstarter journey

KICKSTARTER

Micro Python: Pytho...

Hide

Post update
Edit project
Check dashboard
View backer report
View messages
Send survey

Kickstarter School
Contact us
Creator FAQ
Log out

Micro Python: Python for microcontrollers

by Damien George

Home | Updates 0 | Backers 0 | Comments 0 | Cambridge, United Kingdom | Hardware

Your project has launched!

Congratulations, your project is live on Kickstarter!

Drumroll...

Here's your project URL:
http://www.kickstarter.com/projects/214379695/micro-python-python-for-microcontrollers

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

GOOD LUCK!
Kickstarter

Share 0 | Tweet | Embed

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

0
backers

£0.00
pledged of £15,000 goal

29
days to go

Back This Project
£1 minimum pledge

This project will only be funded if at least £15,000 is pledged by Friday Dec 13, 10:09am GMT.

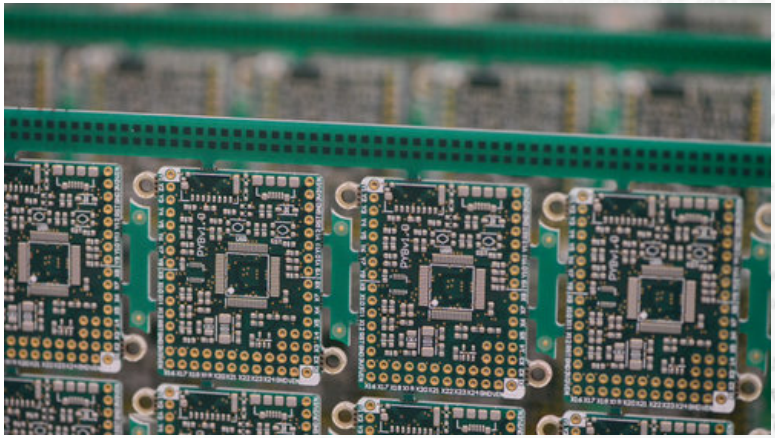
Funding period
Nov 13, 2013 - Dec 13, 2013 (30 days)

Project by
Damien George
Cambridge, United Kingdom

The Kickstarter journey

Manufacturing

Jaltek Systems, Luton UK — manufactured 13,000+ boards.



Boards, headers, servo motors, . . .

Programming and packing

Packing and shipping

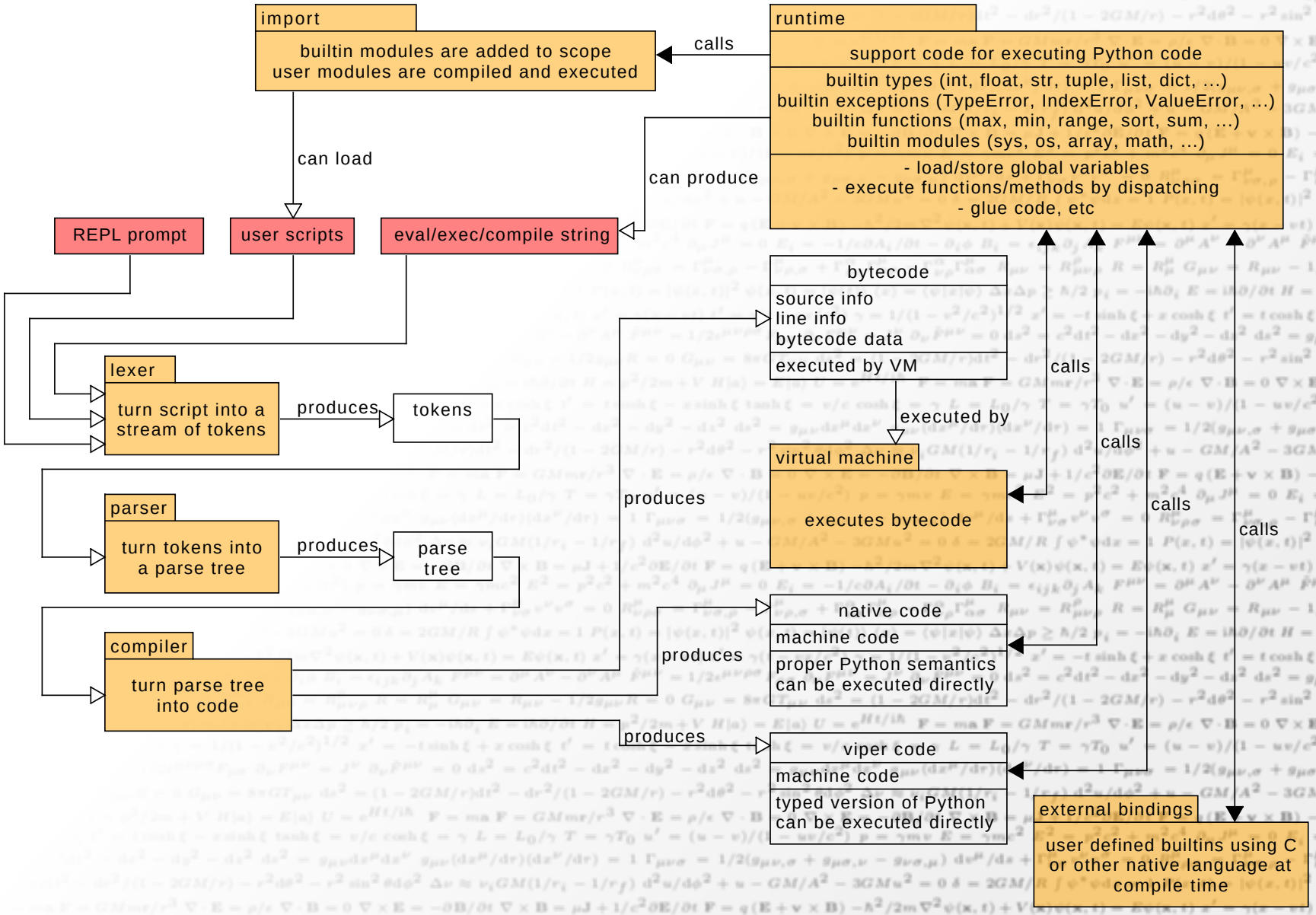
Packing and shipping

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



Object representation

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

Integers:

- xxxxxxxx xxxxxxxx xxxxxxxx xxxxxxx1
- Transparent transition to arbitrary precision integers.

Strings:

- xxxxxxxx xxxxxxxx xxxxxxxx xxxxxx10

Objects:

- xxxxxxxx xxxxxxxx xxxxxxxx xxxxxx00
- 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
```

Emitters: native

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

```
00:  e92d41fe  push   {r1, r2, r3, r4, r5, r6, r7, r8, lr}
04:  e24dd028  sub    sp, sp, #40        ; 0x28
08:  e59f7000  ldr    r7, [pc]           ; 0x10
0c:  ea000000  b      0x14
10:  080794e0  .word  0x080794e0
14:  e1a04003  mov    r4, r3
18:  e1a03002  mov    r3, r2
1c:  e1a02001  mov    r2, r1
20:  e1a01000  mov    r1, r0
24:  e3a00074  mov    r0, #116           ; 0x74
28:  e58d0000  str    r0, [sp]
2c:  e3a00080  mov    r0, #128           ; 0x80
30:  e58d0004  str    r0, [sp, #4]
34:  e3a00004  mov    r0, #4
38:  e58d0014  str    r0, [sp, #20]
3c:  e28d0000  add    r0, sp, #0
40:  e92d0010  stmfd  sp!, {r4}
44:  e1a0e00f  mov    lr, pc
48:  e597f0a0  ldr    pc, [r7, #160]     ; 0xa0
4c:  e8bd0001  ldmfd  sp!, {r0}
50:  e59d4024  ldr    r4, [sp, #36]      ; 0x24
54:  e59d5020  ldr    r5, [sp, #32]
58:  e1a02005  mov    r2, r5
5c:  e1a01004  mov    r1, r4
60:  e3a00005  mov    r0, #5
64:  e1a0e00f  mov    lr, pc
68:  e597f034  ldr    pc, [r7, #52]      ; 0x34
6c:  e28dd028  add    sp, sp, #40        ; 0x28
70:  e8bd81fe  pop    {r1, r2, r3, r4, r5, r6, r7, r8, pc}
```

Emitters: inline assembler!

```
@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'))`

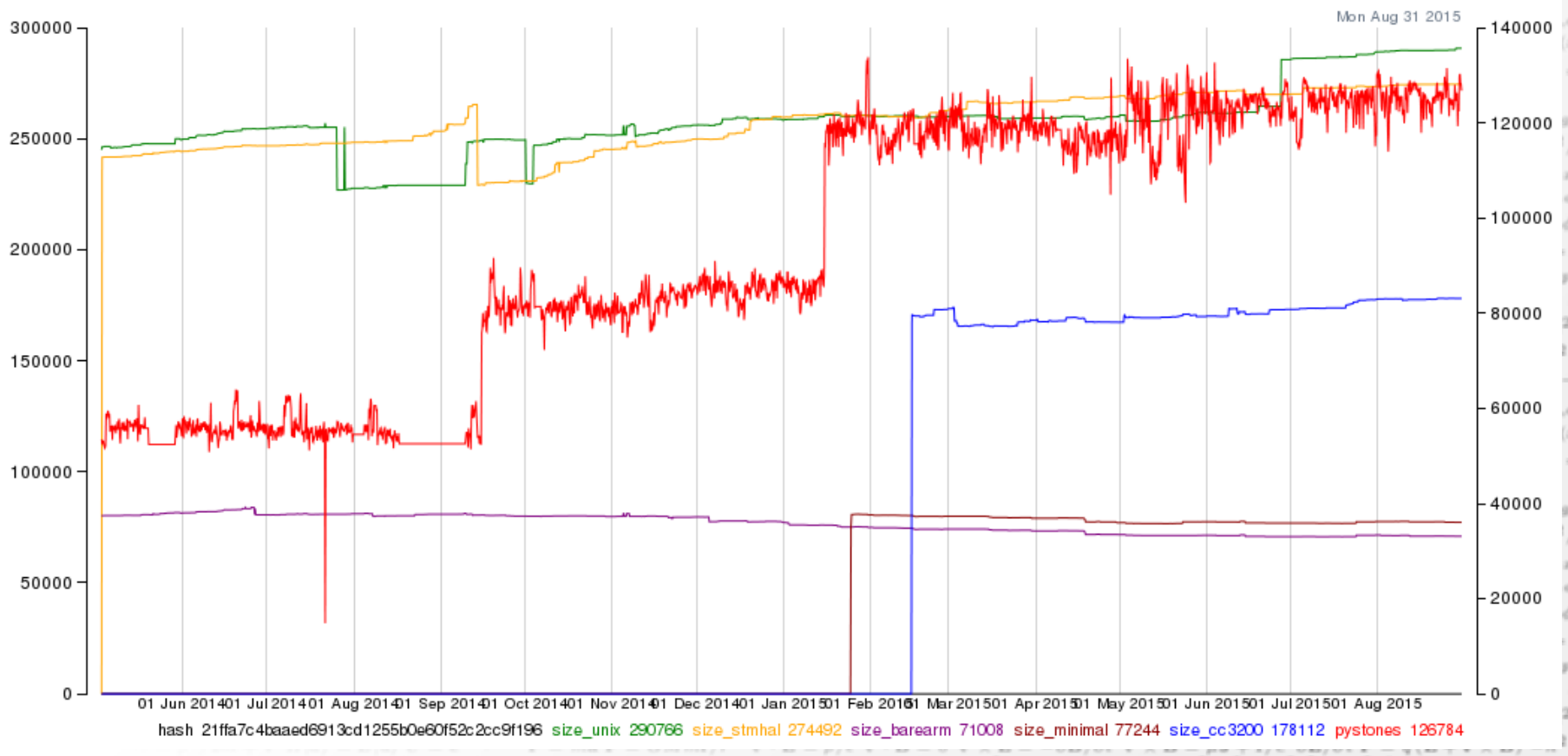
Coding style

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.

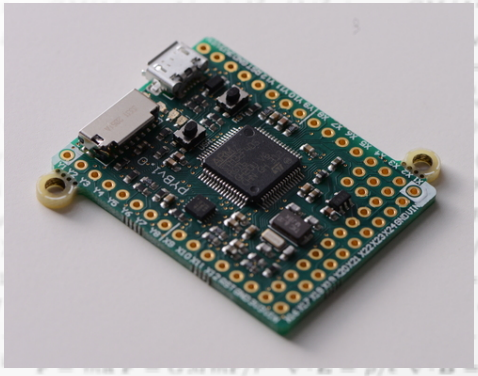
A big project needs many contributors, and open-source allows such projects to exist.

GitHub stars — all C/C++ projects (Sept 2015)

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

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 Xtensa 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
  $\longrightarrow$ easy transition to "physical computing"

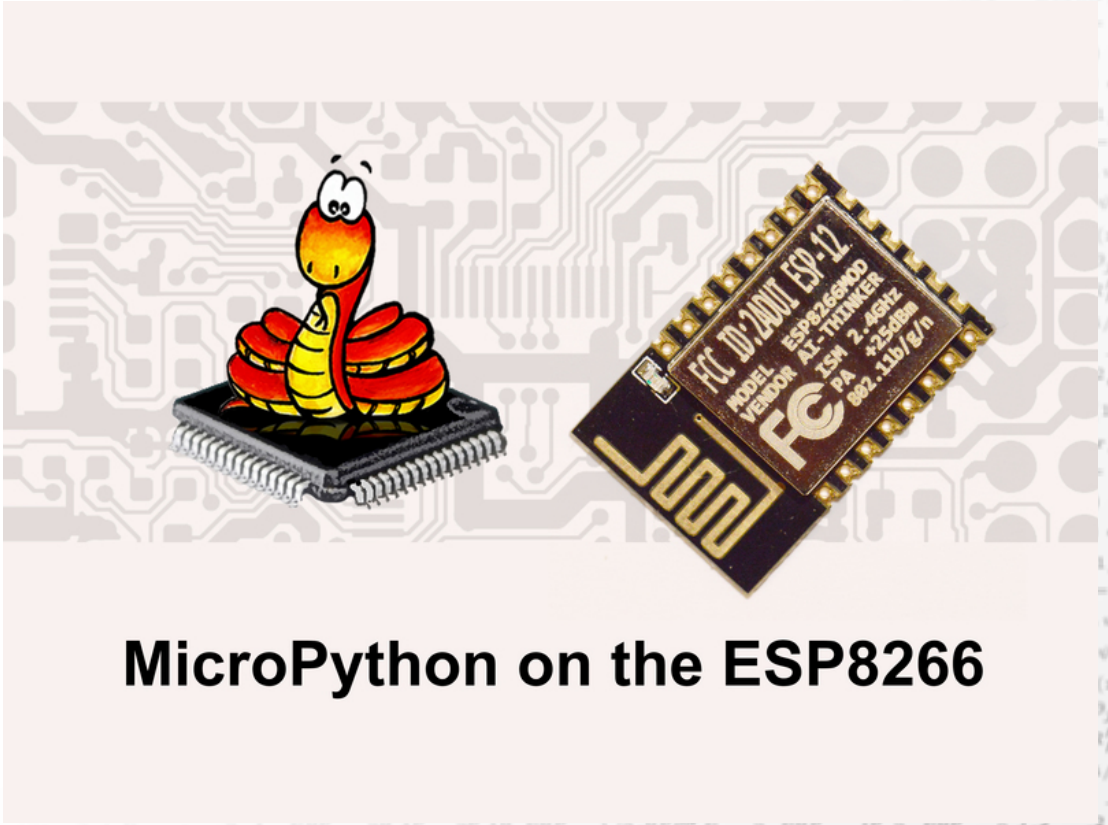
micro:bit demo!

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

micropython.org

forum.micropython.org

github.com/micropython