MicroPython and the Internet of Things

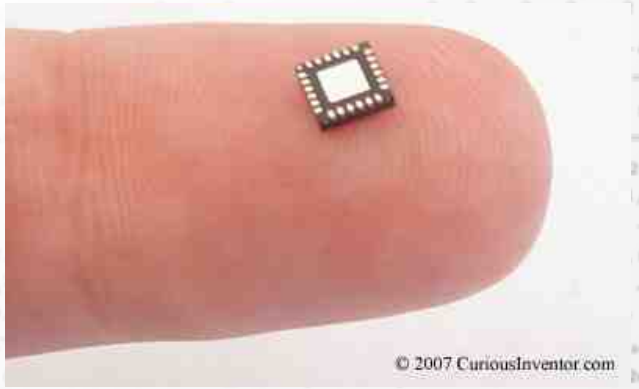
Damien P. George

George Robotics Limited,
Cambridge, UK

GOTO Amsterdam, 15th June 2016

Motivation for MicroPython

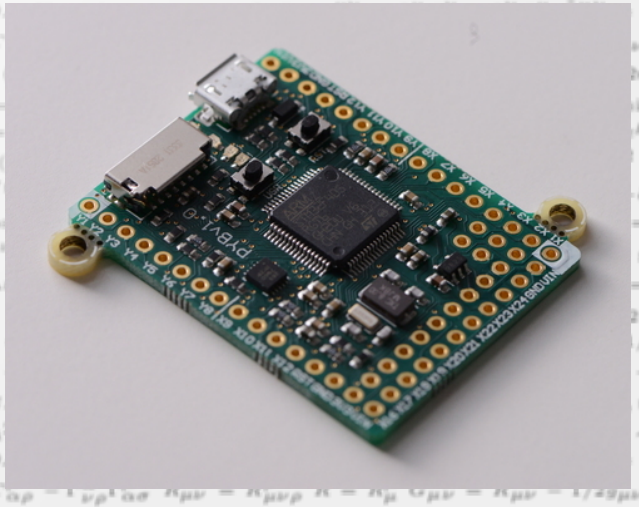
Electronics circuits now pack an enormous amount of functionality in a tiny package.



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:

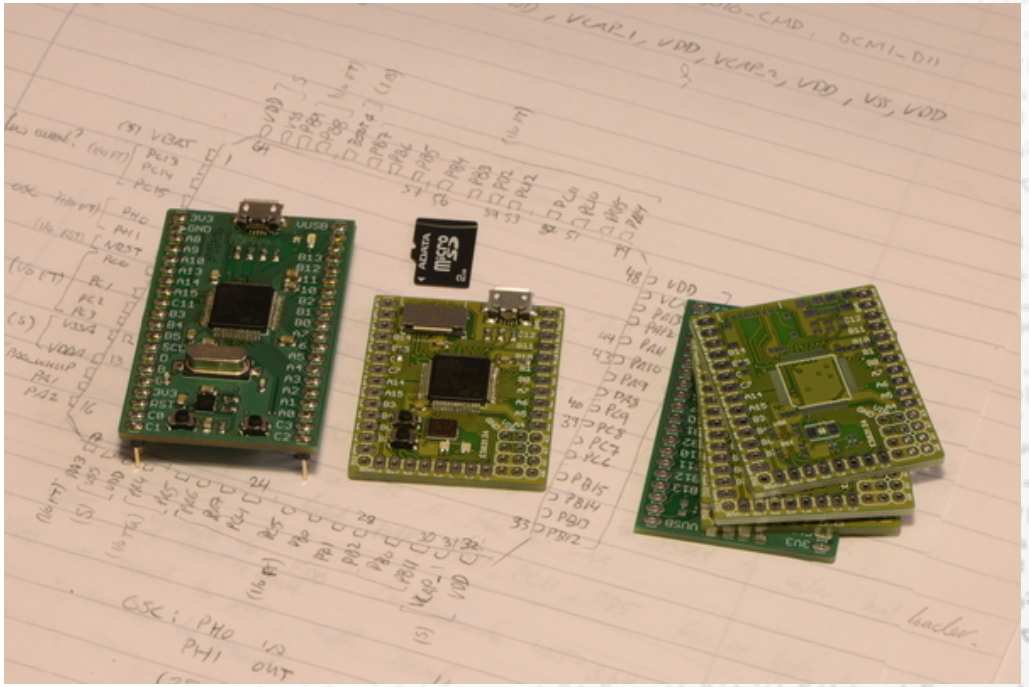
  led.on(): creates a bound-method object, 5 words (20 bytes)

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

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.

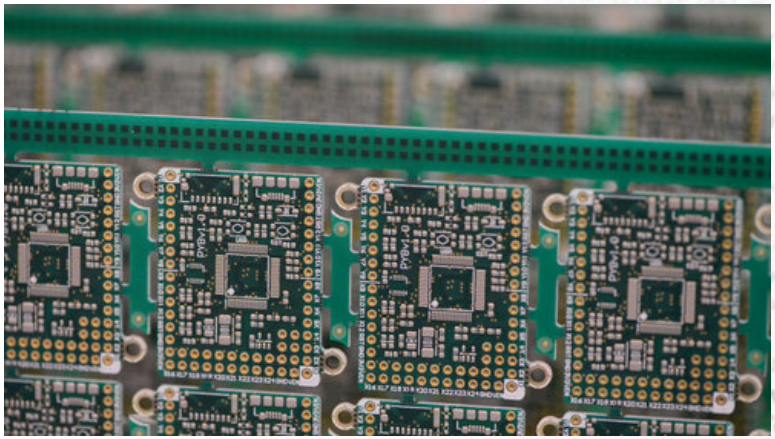
Total backers: 1,931
Total raised: £97,803

Officially finished 12 April 2015.

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

Manufacturing

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



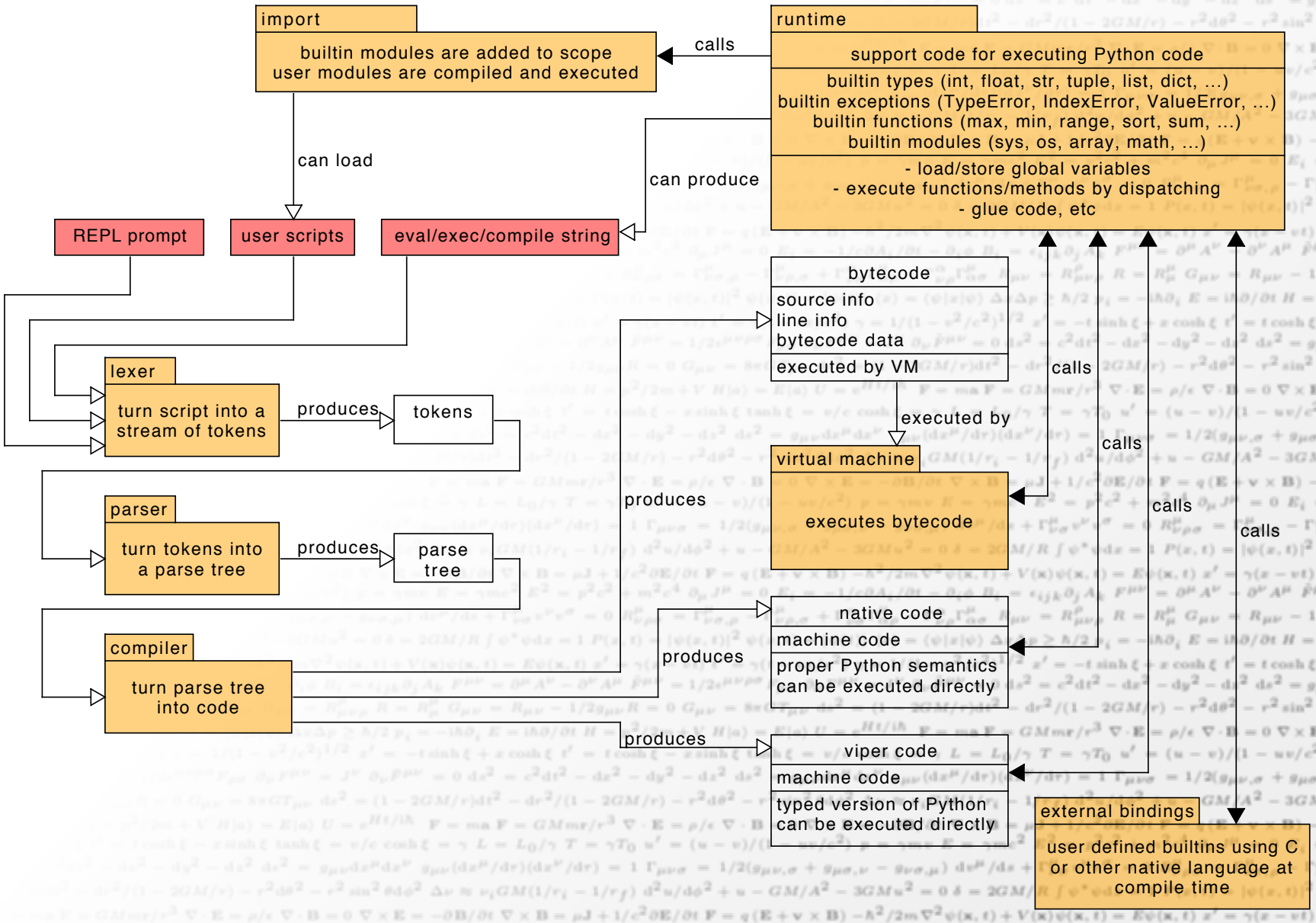
Reward fulfilment and shipping

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


import
builtin modules are added to scope
user modules are compiled and executed
calls
runtime
support code for executing Python code
builtin types (int, float, str, tuple, list, dict, ...)
builtin exceptions (TypeError, IndexError, ValueError, ...)
builtin functions (max, min, range, sort, sum, ...)
builtin modules (sys, os, array, math, ...)
- load/store global variables
- execute functions/methods by dispatching
- glue code, etc
can load
can produce
REPL prompt
user scripts
eval/exec/compile string
bytecode
source info
line info
bytecode data
executed by VM
calls
executed by
virtual machine
executes bytecode
calls
calls
calls
lexer
turn script into a stream of tokens
produces
tokens
parser
turn tokens into a parse tree
produces
parse tree
produces
compiler
turn parse tree into code
produces
native code
machine code
proper Python semantics can be executed directly
produces
viper code
machine code
typed version of Python can be executed directly
external bindings
user defined builtins using C or other native language at compile time

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}
```

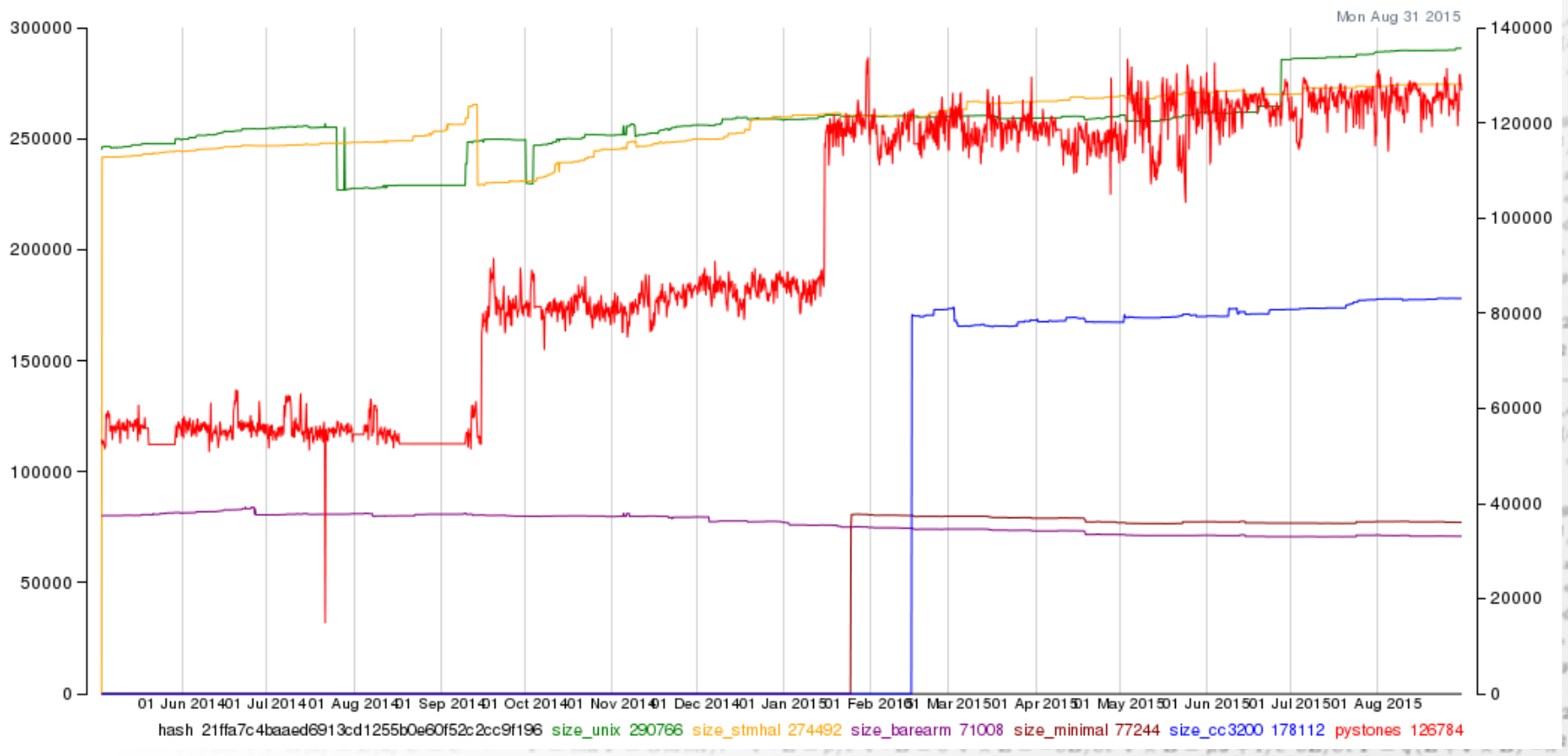
Coding style

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`

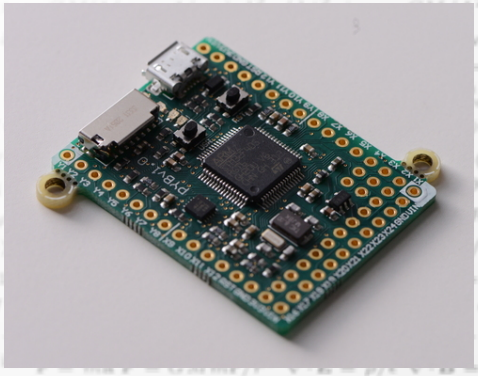
MicroPython is a *public* project on GitHub.

- A global coding conversation.
- Anyone can clone the code, make a fork, submit issues, make pull requests.
- MicroPython has over 3300 "stars" (top 0.02%), and more than 670 forks.
- Contributions come from many people (110+), 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.

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)

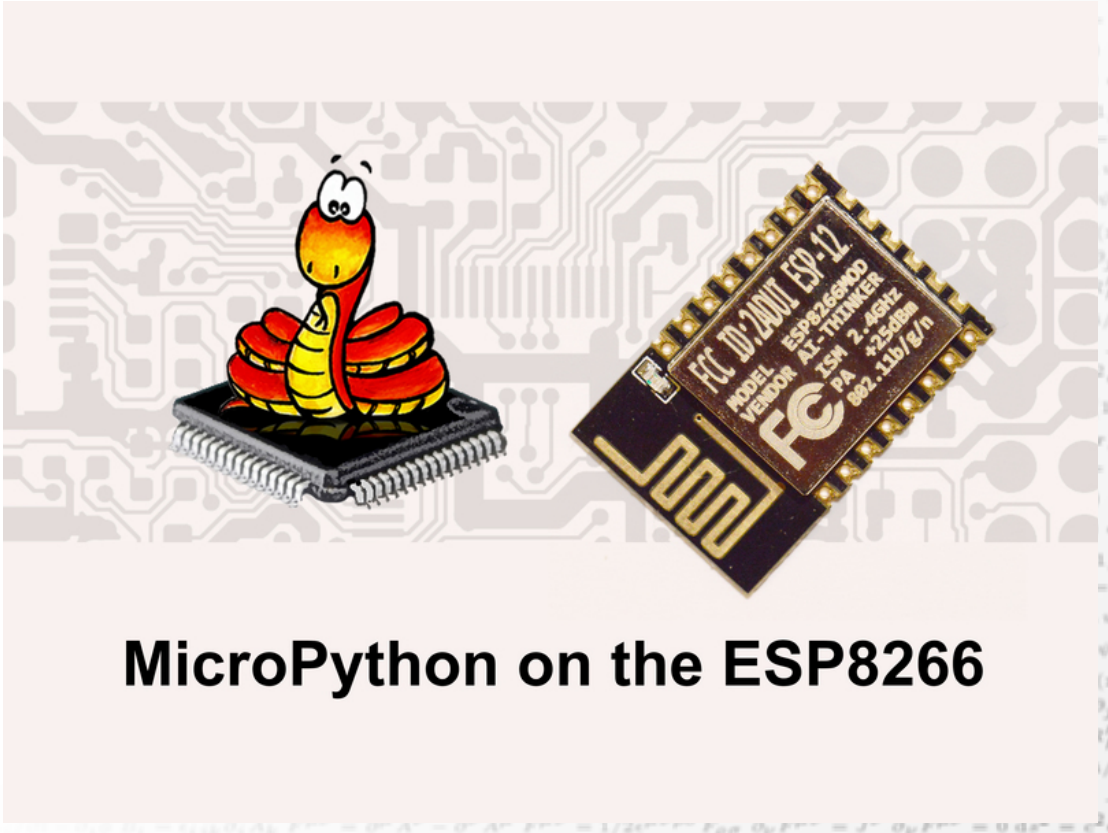
The port to LEON/SPARC/RTEMS for Space

- 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 for satellite control (app. layer)

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"

And then went back for more...



Kickstarter #2 was a *pure software* campaign.

Finished on 2nd March 2016 with 1384 backers, £28,334.

Live Demo!

MicroPython brings Python to resource-limited systems.

It allows rapid development of IoT applications.

Future development:

- continued development of ESP8266 port
- improved (micro)asyncio support
- multithreading support
- more features for the micro:bit
- further work with ESA
- easier embedding

micropython.org

forum.micropython.org

github.com/micropython