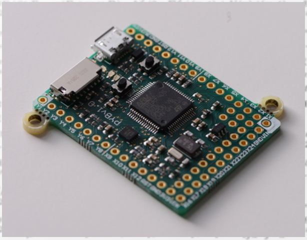
MicroPython: REPLs everywhere

Damien P. George

DAMTP & The Cavendish,
University of Cambridge

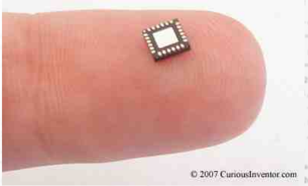
Inria (Lille), 2nd September 2015



Part I: Shrinking Python down to run on a microcontroller

Motivation for MicroPython

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



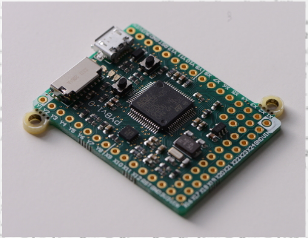
Need a way to control all these sophisticated devices.

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 (this may sound surprising, but 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.

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`.

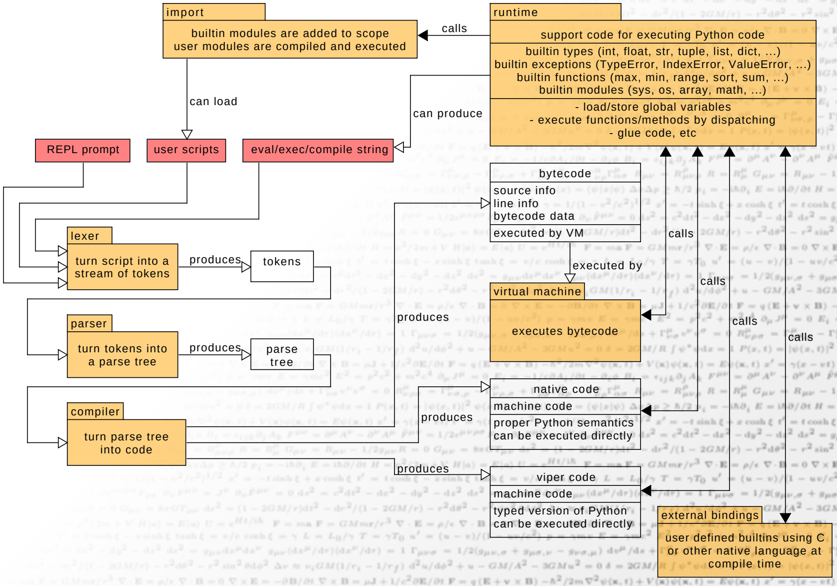
PC demo

Internals

- Lexical analyser: simple tokeniser that interns all identifiers and (most) strings.
- Parser: grammar represented in table form, evaluated by a single, non-recursive function.
- Compiler: passes over the parse tree 3-4 times.
- Code emitter: emits bytecode, or machine code.
- Virtual machine: interprets bytecode.
- Runtime: many helper functions, implementing functionality for the objects.

A given port (eg Linux, bare metal) provides specific RAM and I/O hooks, a REPL, and custom modules.

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
lexer
turn script into a stream of tokens
produces
tokens
executed by
calls
virtual machine
executes bytecode
produces
calls
calls
parser
turn tokens into a parse tree
produces
parse tree
native code
machine code
proper Python semantics
can be executed directly
produces
compiler
turn parse tree into code
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
- Certain strings are not interned.

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, ...).

Note: still room for packing truncated floats.

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: viper

```
@micropython.viper
def add(x:int, y:int) -> int:
    return x + y
```

Compiles to:

```
00:   e92d41fe   push   {r1, r2, r3, r4, r5, r6, r7, r8, lr}
04:   e59f7000   ldr    r7, [pc]        ; 0xc
08:   ea000000   b      0x10
0c:   080794e0   .word  0x080794e0
10:   e1a04000   mov    r4, r0
14:   e1a05001   mov    r5, r1
18:   e1a01004   mov    r1, r4
1c:   e0811005   add    r1, r1, r5
20:   e1a00001   mov    r0, r1
24:   e8bd81fe   pop    {r1, r2, r3, r4, r5, r6, r7, r8, pc}
```

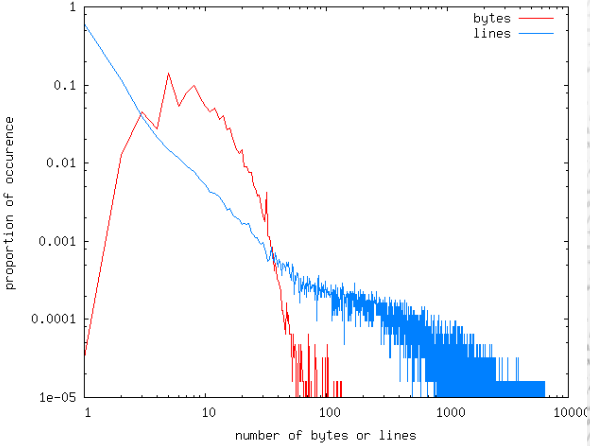
Emitters: inline assembler

```
@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'))`

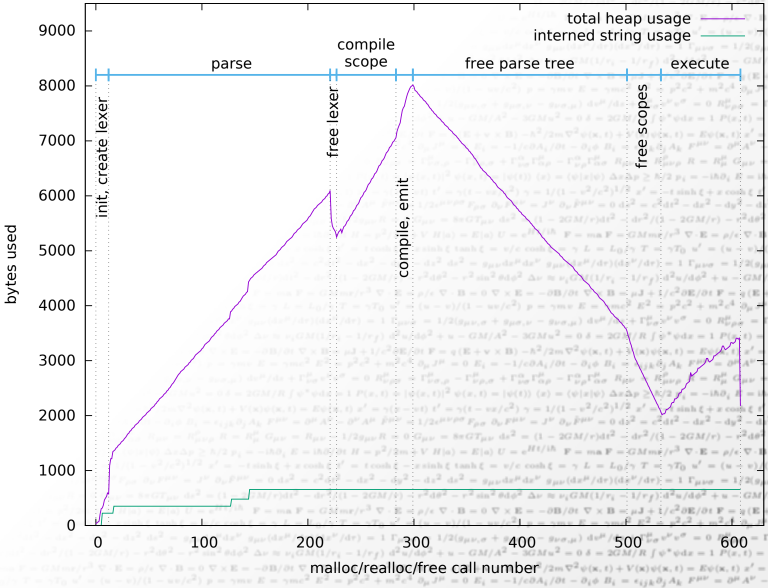
Encoding of source-bytecode map

Histogram showing frequency of having to skip n bytes and n lines.



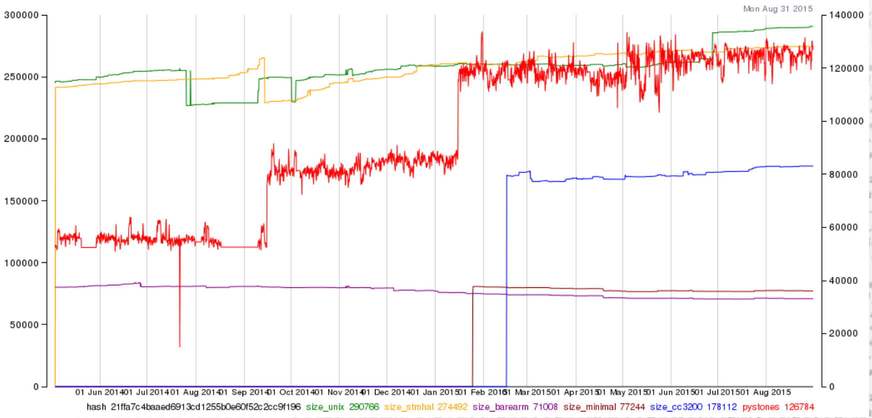
Encoding optimised for this (very simple Huffman-like).

Example heap usage



Code dashboard

http://micropython.org/resources/code-dashboard/



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.

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 2250 “stars”, and more than 410 forks.
- Contributions come from many people, 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.

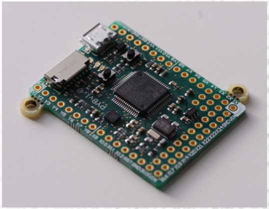
GitHub stars — all 10 million+ projects

1	twbs/bootstrap	CSS	85,841	
2	vhf/free-programming-books	None	42,432	
3	angular/angular.js	JavaScript	42,102	
4	mbostock/d3	JavaScript	41,149	
5	nodejs/node-v0.x-archive	JavaScript	38,004	
6	jquery/jquery	JavaScript	35,780	
7	FortAwesome/Font-Awesome	HTML	35,663	
8	h5bp/html5-boilerplate	JavaScript	30,974	
9	meteor/meteor	JavaScript	27,843	
10	rails/rails	Ruby	27,531	
	...			
1766	apache/couchdb	JavaScript	2,279	
1767	lifesinger/lifesinger.github.com	JavaScript	2,276	
1768	googlesamples/android-topeka	Java	2,274	
1769	addyosmani/es6-tools	None	2,274	
1770	activerecord-hackery/ransack	Ruby	2,274	
1771	sciactive/pnotify	CSS	2,273	
1772	addyosmani/basket.js	JavaScript	2,273	
1773	PostgresApp/PostgresApp	Objective-C	2,273	
1774	micropython/micropython	C	2,272	<<<<<
1775	seanpowell/Email-Boilerplate	HTML	2,271	
1776	lipka/piecon	JavaScript	2,271	
1777	alfajango/jquery-dynatable	JavaScript	2,271	
	...			

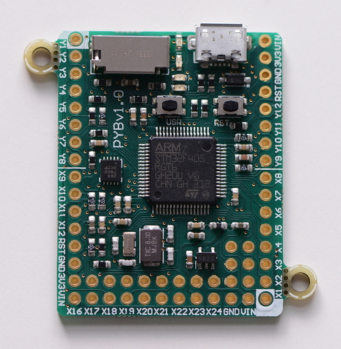
GitHub stars — all C/C++ projects

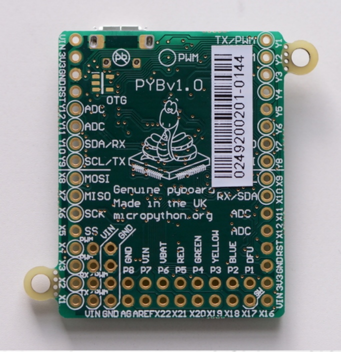
1	torvalds/linux	C	25,156	
2	nwjs/nw.js	C++	24,175	
3	atom/electron	C++	15,777	
4	ariya/phantomjs	C++	15,084	
5	antirez/redis	C	14,701	
6	facebook/hhvm	C++	12,497	
7	textmate/textmate	C++	10,490	
8	git/git	C	10,082	
	...			
98	jonas/tig	C	2,361	
99	swoole/swoole-src	C	2,319	
100	raspberrypi/linux	C	2,310	
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	

Part II: The pyboard hardware



The pyboard





- STM32F405RG: 192k RAM, 1M ROM, 168MHz, Cortex M4F.
- USB micro connector for device (and host).
- Micro SD card.
- 3-axis accelerometer (MMA7660).
- Real-time clock, 4 LEDs, 2 switches.
- 30 GPIO: symmetric pin layout, plus extra pins.
- Internal file system. ”/flash” and ”/sd”.

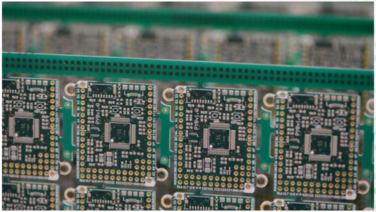
Pyboard usage

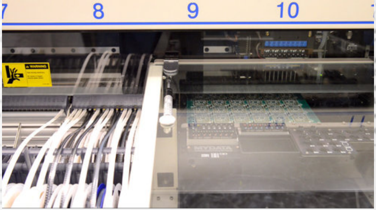
- Standard Python prompt (REPL) over USB serial device, or UART.
- Raw prompt: reads a Python script until EOF, executes it, then sends back the result.
- A script running from the flash/SD. Serial connection becomes stdin/stdout.
- Powered by USB or battery.

pyboard demo

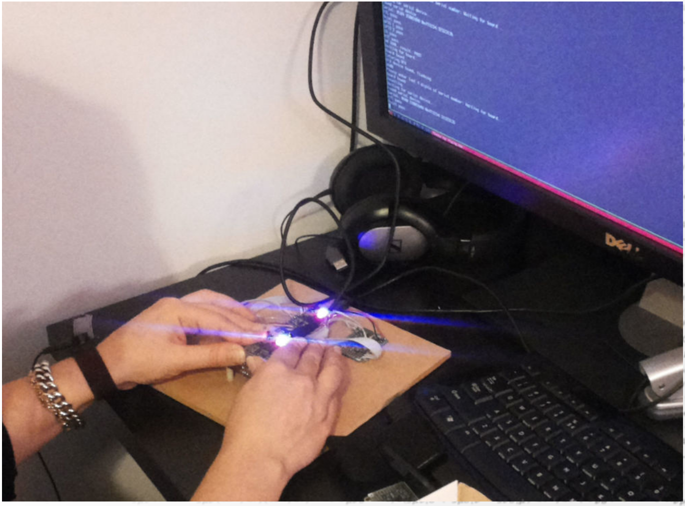
Manufacturing

Jaltek Systems, Luton UK — manufactured 7000+ boards.





Testing and programming



MicroPython Live — `http://micropython.org/live/`

pyboard is running your job!

Current code

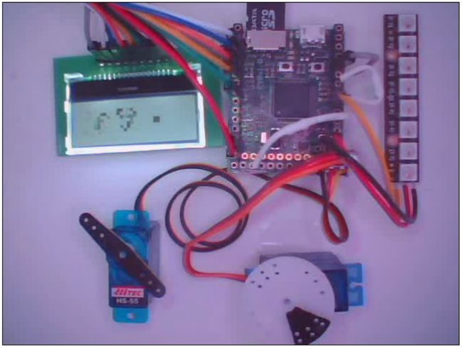
```
# Conway's Game of Life
import pyb

PIXEL_SIZE = const(4)

# we use big pixels so it's easier to see
@micropython.viper
def big_pixel(x:int, y:int, col):
    pixel = lcd.pixel
```

Current output

```
MicroPython running Conway's Game of Life!
```



My code

running... | clear | ← | example | →

```
# Conway's Game of Life
import pyb

PIXEL_SIZE = const(4)

# we use big pixels so it's easier to see
@micropython.viper
def big_pixel(x:int, y:int, col):
    pixel = lcd.pixel
    for i in range(PIXEL_SIZE):
        for j in range(PIXEL_SIZE):
            pixel(x + i, y + j, col)

# do 1 iteration of Conway's Game of Life
@micropython.viper
def conway_step(lcd):
    get = lcd.get
    for x in range(0, 128, PIXEL_SIZE):
        for y in range(0, 32, PIXEL_SIZE):
            self = int(get(x, y))
```

My output

```
MicroPython running Conway's Game of Life!
```

Other hardware

MicroPython runs on lots of other hardware:

- STM32F4xx discovery boards,
- Espruino Pico (STM32F401),
- CC3200 wi-fi SoC (WiPy),
- ESP8266 wi-fi SoC,
- 16-bit dsPIC33F,
- ...

Part III: The Kickstarter campaign

Crowd funding

Pitch an idea, get the public to fund it, give them something in return.

Started by ArtistShare in 2003.
IndieGoGo in 2008, Kickstarter in 2009, plus many others.

- Musicians, photographers, writers, video games, hardware, ...
- Science projects
- Roll your own

2012: US$2.7 billion, more than one million individual campaigns

2013: crowdfunding industry grew to over $5.1 billion

Kickstarter: collected so far over US$1 billion in funds

The stages

Stage 1: the idea!

- Prototyping.
- Fear that someone will beat you.
- Rush to get it online.
- Making the campaign, including video.

Stage 2: the campaign.

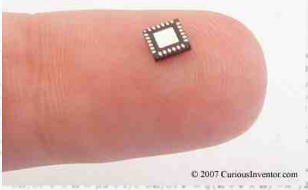
Stage 3: fulfillment.

- Spend all the money, buy lots of stuff.
- Finalise your idea and mass produce it.
- Lots of delays.
- Work out how to post 1000s of parcels.
- Packing and shipping.
- Returns, complaints, support, etc.

Stage 1: the idea

Idea for MicroPython

- System on a Chip: CPU, RAM, flash memory, timers, USB, Ethernet.
- Sensors: touch, accelerometer, gyroscope, compass, barometer.
- Outputs: LEDs, LCDs, DC motors, servos.



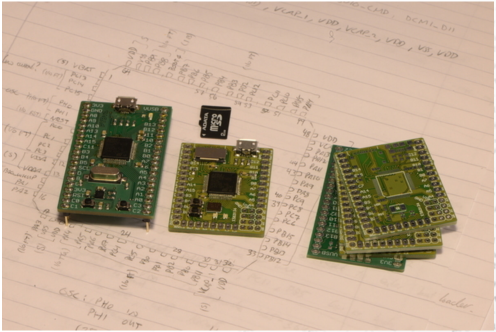
Goal: Make it easy for people to program their electronics projects.

Idea: Shrink Python to run on a microcontroller.

KICKSTARTER

Initial development

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



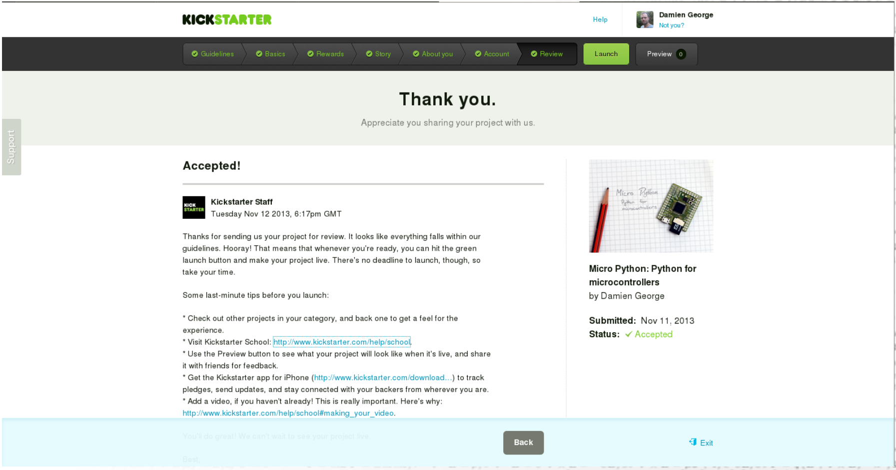
1 weekend to make the video.

Kickstarter launched on 13 November 2013, ran for 30 days.

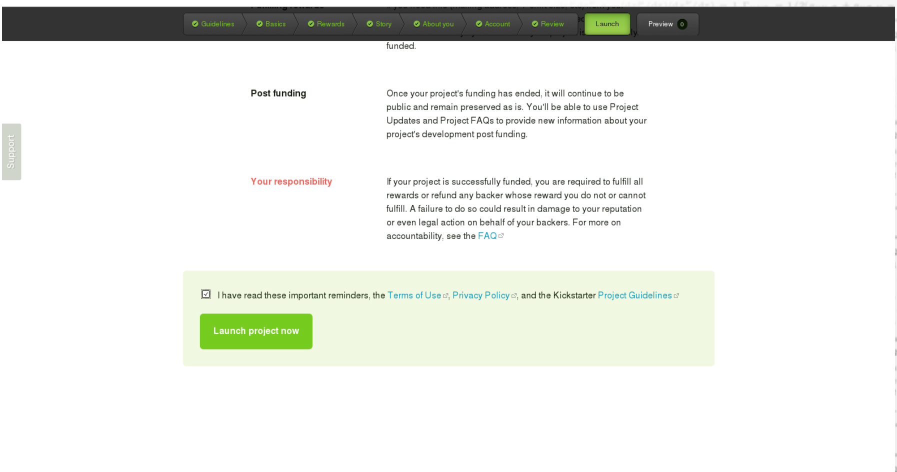
Officially finished 12 April 2015.

Stage 2: the campaign

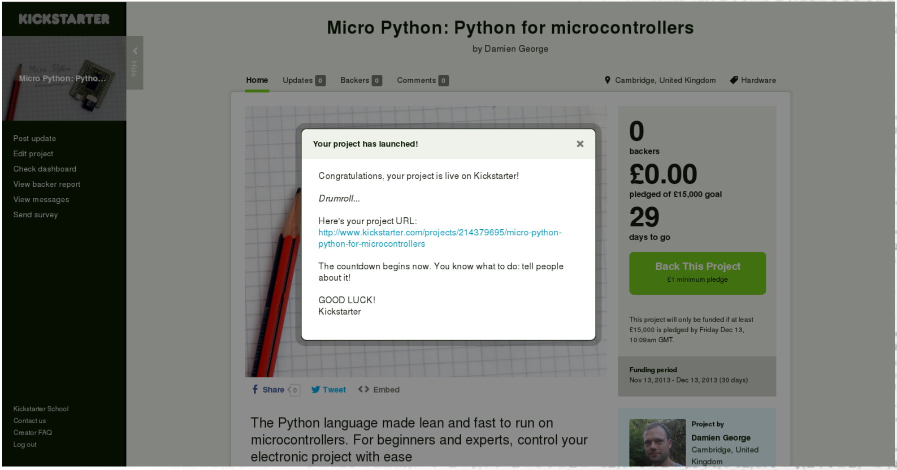
The Kickstarter journey



The Kickstarter journey



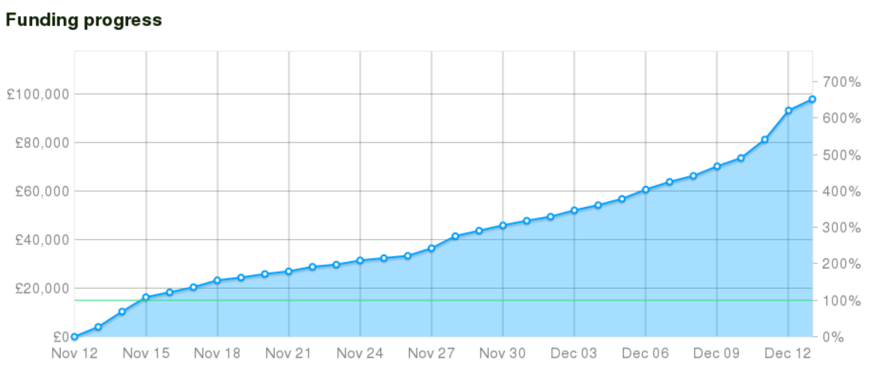
The Kickstarter journey



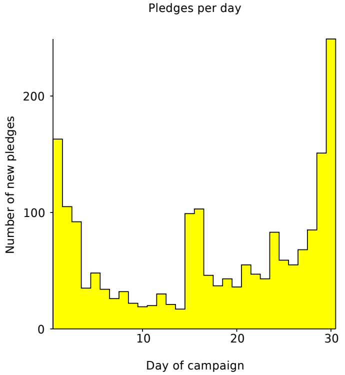
The Kickstarter journey

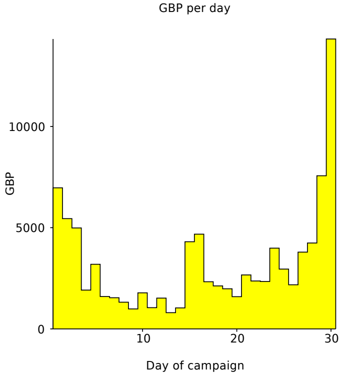
The Kickstarter journey

Campaign timeline



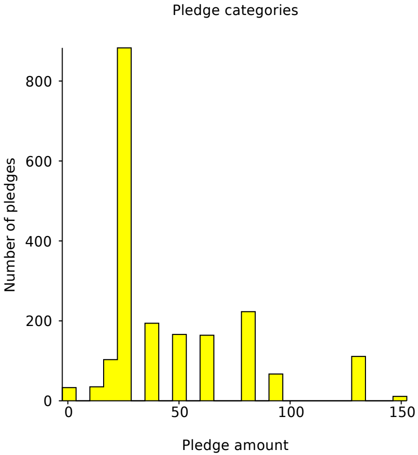
Campaign timeline





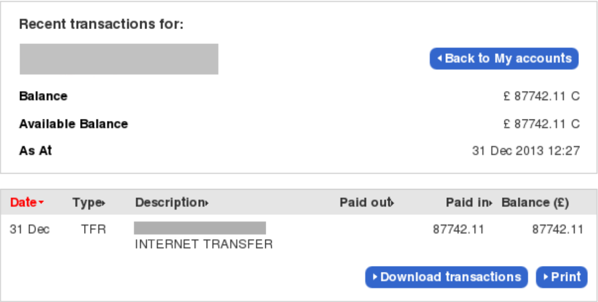
Total backers: 1,931 Total raised: £97,803

Pledge levels

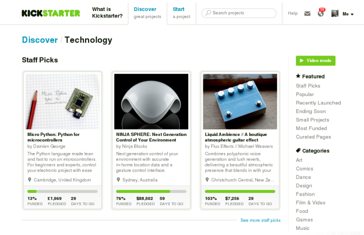


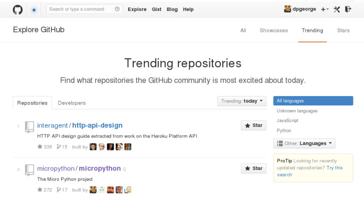
- No reward, just donation.
- A single board, £20 and £24.
- Multiple boards at £50 and £90.
- Kits at £40, £60, £80 and £130.
- Special items at £150, £300 and £1,000.

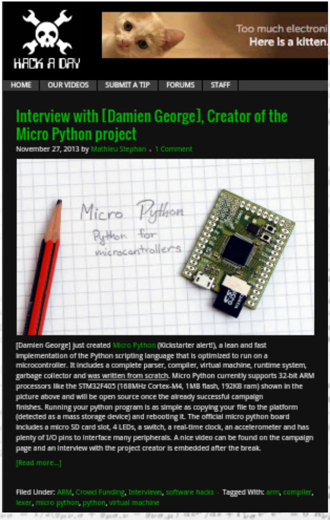
A few weeks later . . .



Internet coverage

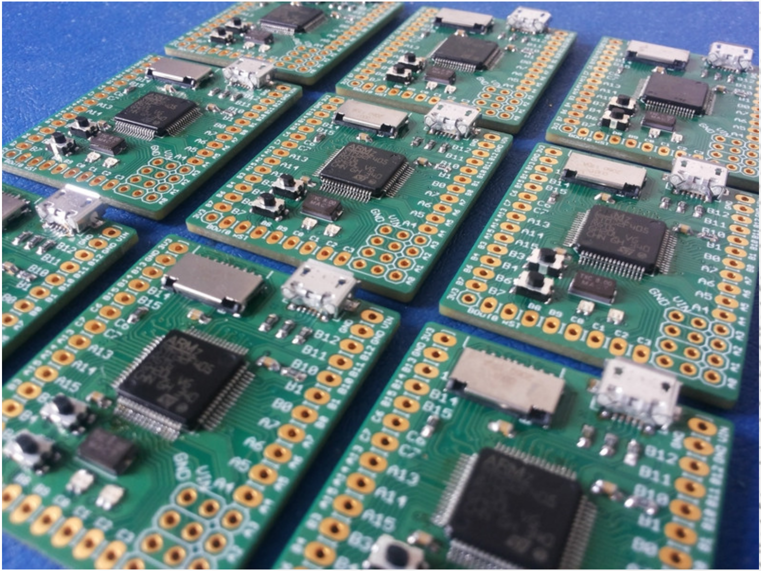






Stage 3: fulfillment

Hand made boards



Boards, headers, servo motors, . . .

Programming and packing

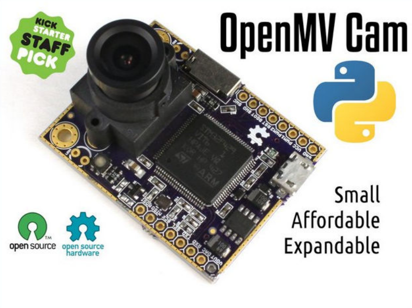
Stamps!

Packing and shipping

Packing and shipping

Other Kickstarter campaigns

Machine Vision With Python



Add machine vision to your projects by scripting Python. Small, affordable, and expandable with shields.

Follow along!

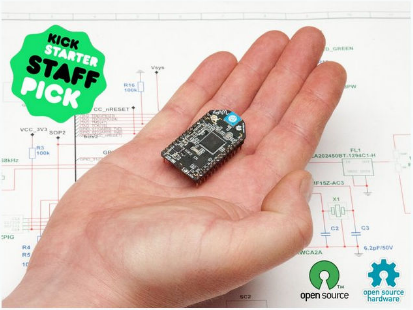
Created by

Bot Thoughts LLC

900 backers pledged $104,498 to help bring this project to life.

Other Kickstarter campaigns

The WiPy: The Internet of Things Taken to the Next Level



The IoT development platform that runs Python in real time, and features the perfect blend of power, friendliness and flexibility.

www.wipy.io

Created by

The WiPy

1,382 backers pledged €75,564 to help bring this project to life.

Lessons and tips

- Why do want to do a crowd funding campaign (fun, money, have an idea, start a business)?
- Work quick to bring it to the public:
  - might be beaten!
  - learn earlier if it's not a good idea
  - spend too long and the idea gets stale
- Make a video that shows yourself, your idea and your excitement.
- Plan your rewards carefully and offer a range (people like options) but don't make them too complicated (for your own sake).
- Don't over-promise; have a core single idea and deliver on that as best you can.
- Be prepared to respond to a lot of emails!

Part IV: The future

Post Kickstarter

We now have a private limited company.

Continue to sell the pyboard and accessories online.

CERTIFICATE OF INCORPORATION
OF A
PRIVATE LIMITED COMPANY

Company Number 8861687

The Registrar of Companies for England and Wales, hereby certifies that

GEORGE ROBOTICS LIMITED

is this day incorporated under the Companies Act 2006 as a private company, that the company is limited by shares, and the situation of its registered office is in England and Wales.

Given at Companies House, Cardiff, on 27th January 2014.

Code improvements

- Option for 64-bit NaN boxing model.
- Option for 30-bit floats packed into current pointer model.
- Allow pre-compiled bytecode to be loaded.
- 100% test coverage (at 94% right now).
- Define a consistent hardware API for all MCUs / boards.

New hardware

- Cheaper board.
- More powerful board (eg Ethernet, SDRAM).
- More add-on boards (all with a standard form factor).
- Any new, interesting MCU: make a board out of it!

Future of MicroPython

Continue to improve documentation, write tutorials, support more peripherals, build a community.

Not just for bare metal: works very well on standard systems and anyone can build and run it easily on *nix/Windows/Mac.

Great potential for Internet of Things (IoT): much easier to develop a small internet connected device using Python than C.

Embedding MicroPython in games and mobile apps.

Industrial use — use in space? Determinism is a key point.

micropython.org

forum.micropython.org

github.com/micropython