pycom
GO INVENT

Daniel Campora
CTO & Co-Founder | @danielpycom

What is MicroPython?

Pycom

We are here to enable and inspire everyone to be an inventor.

Our Vital Stats

Founded

02 November 2015

Launched

05 January 2016

Offices

HQ in the UK with offices in the Netherlands and Paris

FTEs

12 growing to 22 in 2017

Co-founders

Fred de Haro, CEO: Fred has spent many years building pre-IPO businesses and helping them secure high growth whilst remaining true to their original DNA.

Daniel Campora, CTO. Experienced hardware and software lead used to spearheading large automotive and smart city projects and well accustomed to demanding customers.

Bettina Rubek Slater, COO: Strategic tech marketeer who brings many years of portfolio planning, strategic marketing and operational insight.

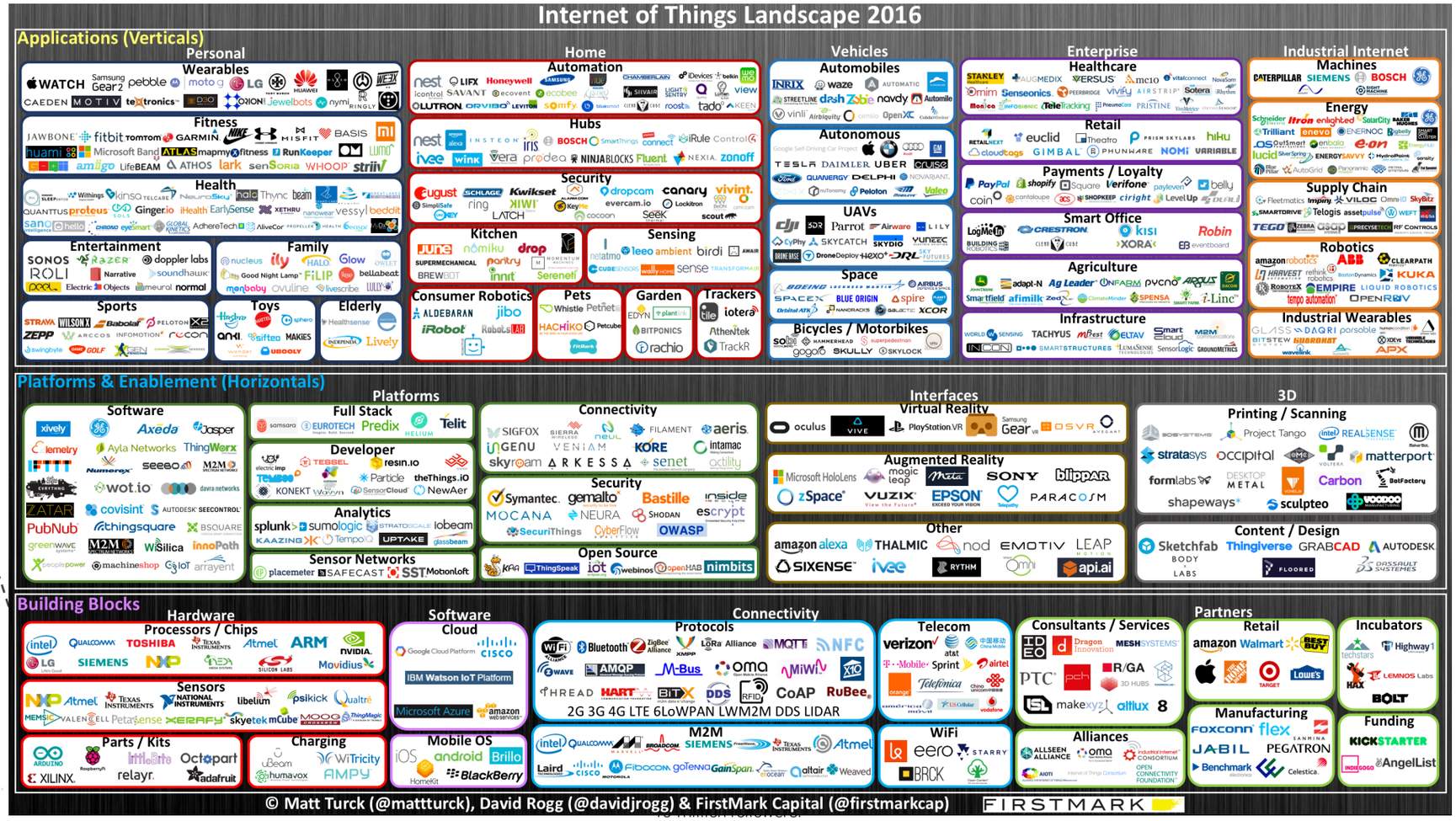
12 Months

- Portfolio of 5 boards and 4 OEM modules
- 40,000 units shipped to 70+ countries
- 6,000 customers, 68% of which are enterprises
- 6 Tier 1 Mobile network operators fully engaged
- Global Reseller Channel
- 15,000+ unique users on the Pycom Forum
- 60,000+ unique visitors to Pycom's website
- 7 events, 1400+ followers on Twitter

Pycom Positioning

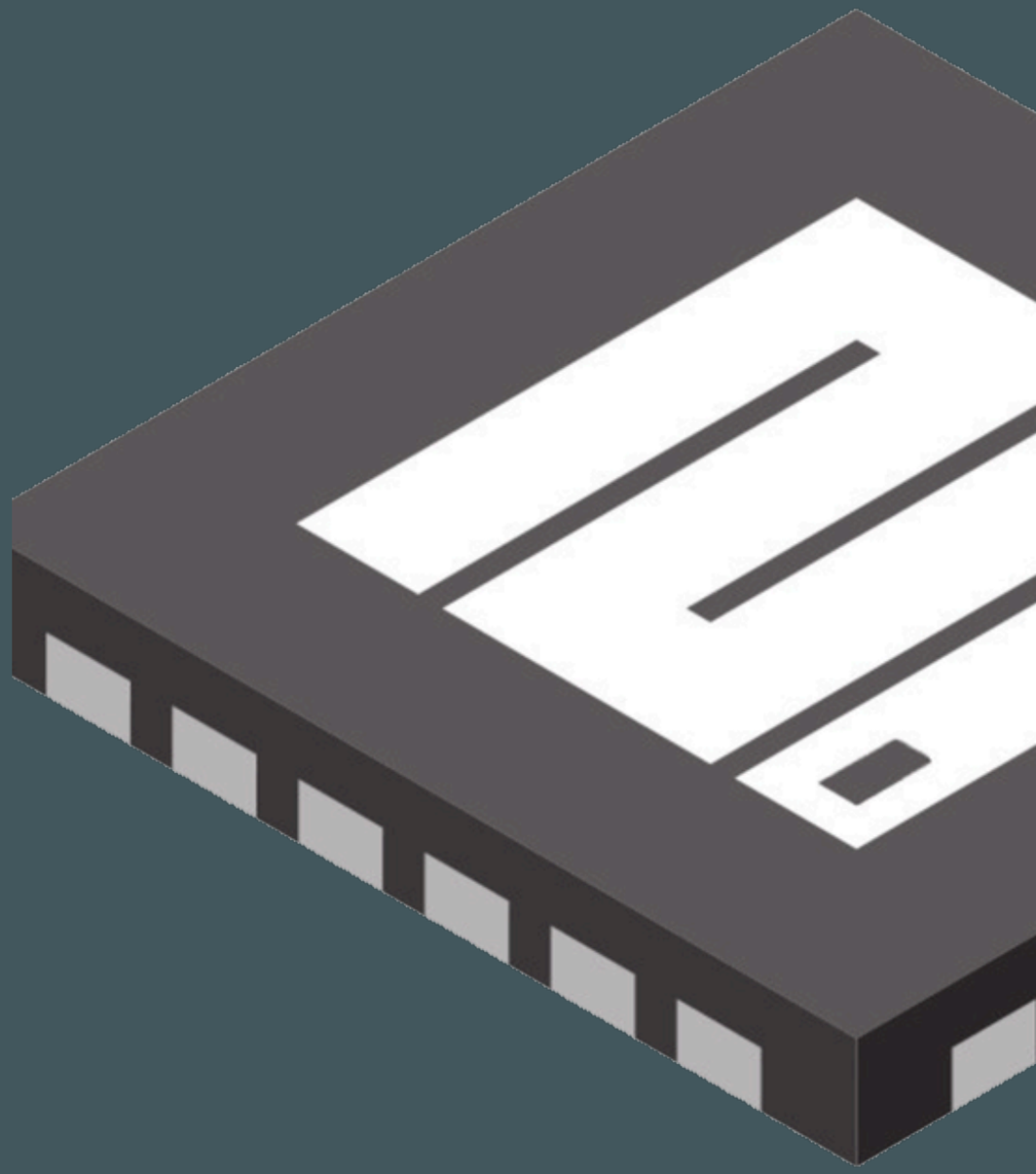
FULL IOT STACK

- Dashboard
- Cloud / Data Management Middleware
- Wireless Networks
- Gateways
- Sensors



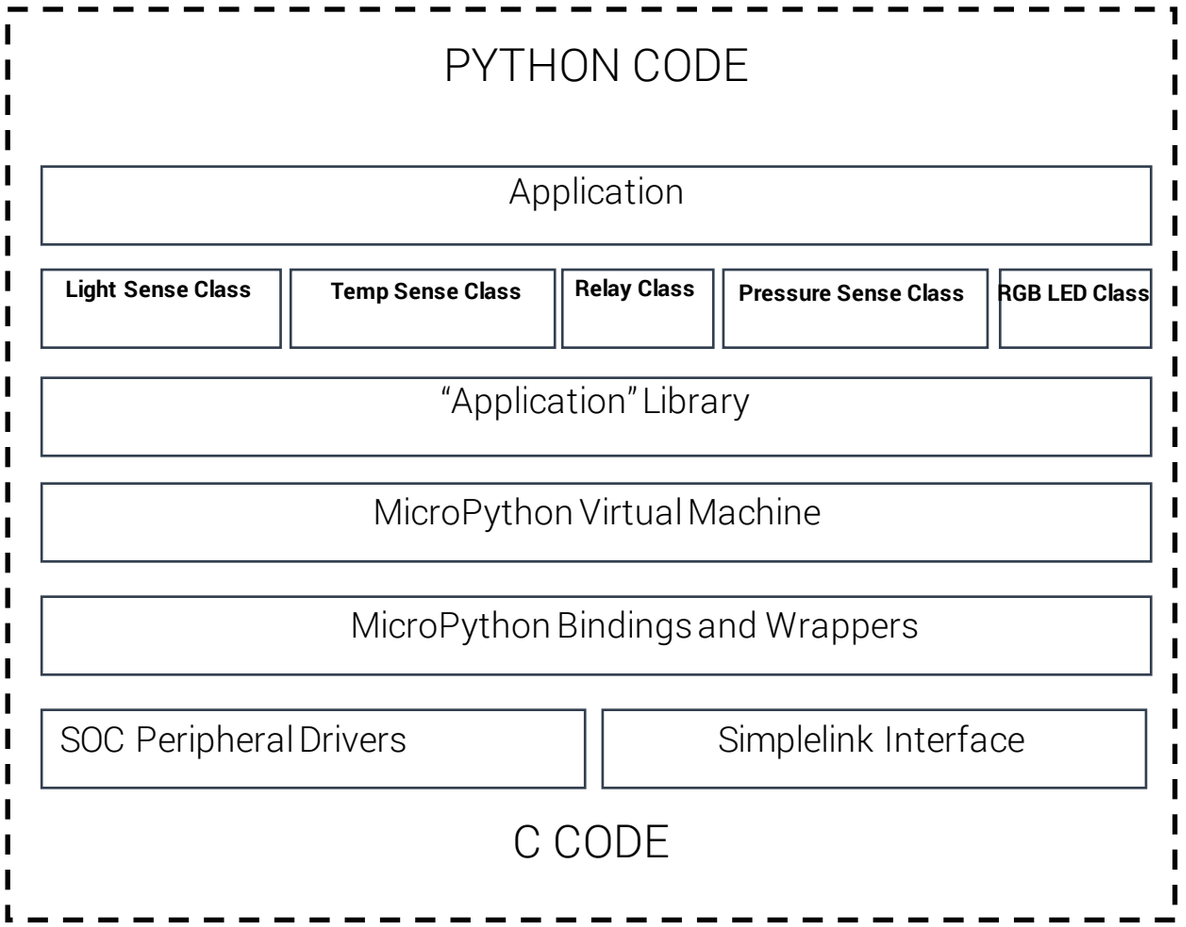
MicroPython

- A lean and efficient implementation of CPython 3.5
- Includes a small subset of the Python Standard Library
- Optimised to run on Microcontrollers
- Perfect for small and constrained environments
- Mostly compatible with ‘normal’ Python (CPython 3.5) allowing easy transfer of code



What's Inside?

- A full Python compiler and runtime
- Runs on the bare-metal
- Interactive prompt (the REPL) to execute commands immediately,
- Runs and imports scripts from the built-in filesystem
- The REPL has history, tab completion, auto-indent and paste mode
- A selection of core Python libraries plus modules such as "machine" for accessing low-level hardware.

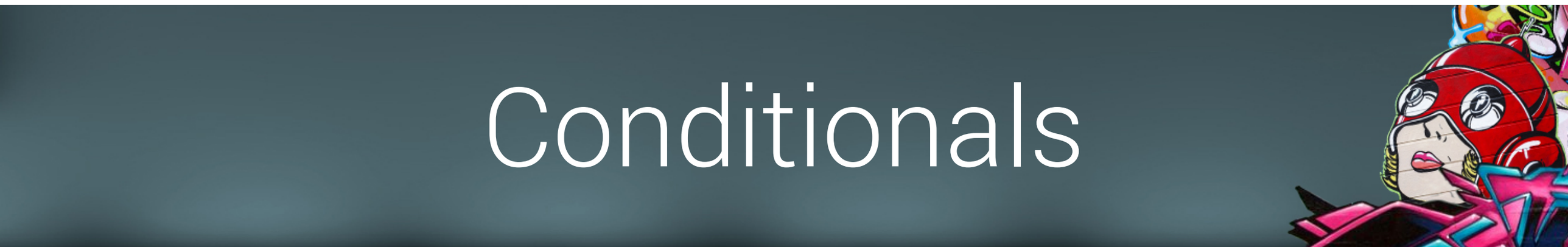


Examples

MicroPython

If, Elif & Else

```
target = 23
temperature = getTemperature()

if temperature == target:
    # New block starts here
    print('ideal temperature achieved')
    # New block ends here
elif temperature < target:
    # Another block
    print('temperature is too low')
    # You can do whatever you want in a block ...
else:
    print('temperature is too high')
    # you must have temperature > target to reach here

print('done')
# This last statement is always executed,
# after the if statement is executed.
```



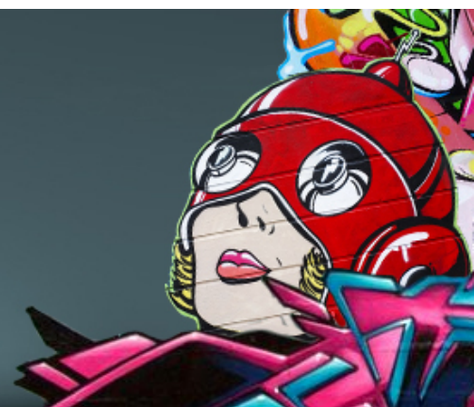
While

```
target = 23
running = True

while running:
    light = getLight()

    if light == target:
        print('On!')
        # this causes the while loop to stop
        running = False
    elif light < target:
        print('Too Low')
    else:
        print('Too High')
else:
    print('The while loop is over.')
    # Do anything else you want to do here

print('Done')
```

For

```
for i in range(1, 5):
    print(i)
else:
    print('The for loop is over')
```

Loops



Structures

```
# Tuple
months = ('January','February','March','April','May','June',\
'July','August','September','October','November','  December')

# List
cats = ['Tom', 'Snappy', 'Kitty', 'Jessie', 'Chester']

# Dictionary
phonebook = {'Andrew Parson':8806336, 'Emily Everett':6784346,\
'Lewis Lane':1122345}
```

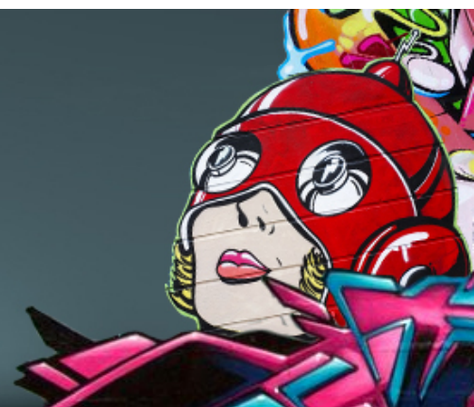
Operators

```
a = 60              # 60 = 0011 1100
b = 13              # 13 = 0000 1101
c = 0

c = a & b;          # 12 = 0000 1100
print "Line 1 - Value of c is ", c

c = a | b;          # 61 = 0011 1101
print "Line 2 - Value of c is ", c

c = a ^ b;          # 49 = 0011 0001
print "Line 3 - Value of c is ", c

c = ~a;             # -61 = 1100 0011
print "Line 4 - Value of c is ", c

c = a << 2;         # 240 = 1111 0000
print "Line 5 - Value of c is ", c

c = a >> 2;         # 15 = 0000 1111
print "Line 6 - Value of c is ", c
```

Conversions

```
import binascii

# Convert binary data to hexadecimal
# representation. Returns bytes string.
Key_Hex = binascii.hexlify('0b1011000')

# Convert hexadecimal data to binary
# representation. Returns bytes string.
# (i.e. inverse of hexlify)
Key_Bin = binascii.unhexlify('2B 7E 15 16\
28 AE D2 CF 4F 3C'.replace(' ',''))

# replace() can be used to replace
# elements of the string
```

Bit Manipulation & Number Conversion



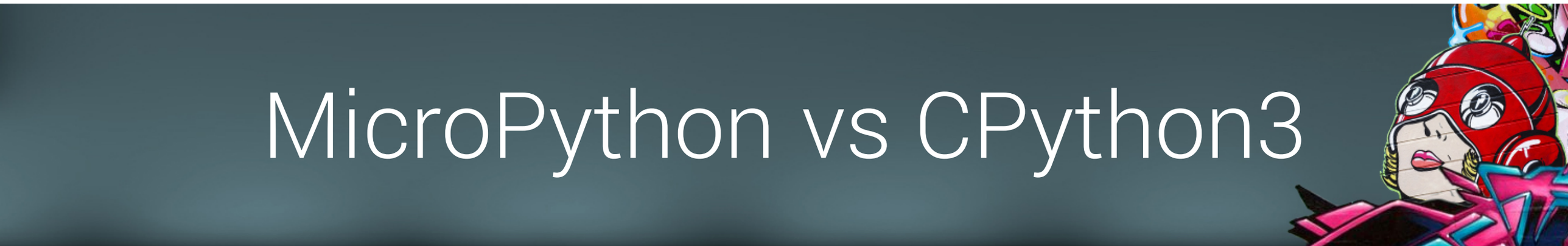
Major Design Differences

1. MicroPython **does not ship with an extensive standard library** of modules.

2. Unlike CPython3, which uses reference-counting, **MicroPython uses garbage collection** as the primary means of memory management.

3. MicroPython **does not implement complete CPython object data model**, but only a subset of it.

4. MicroPython implements **only subset of functionality and parameters of particular functions and classes**. Each specific issue may be treated as "implementation difference" and resolved.

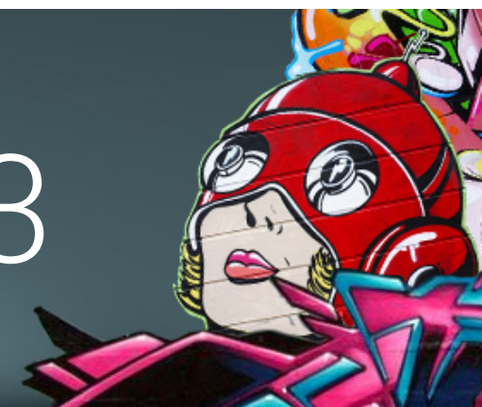


Major Design Differences

5. MicroPython supports only **minimal subset of introspection and reflection features** (such as object names, docstrings, etc.).

6. **print() function does not check for recursive data structures in the same way CPython does**. There's however checks for stack usage, so printing recursive data structure won't lead to crash due to stack overflow.

8. MicroPython **optimizes local variable handling** and does not record or provide any introspection info for them, e.g. locals() doesn't have entries for locals.



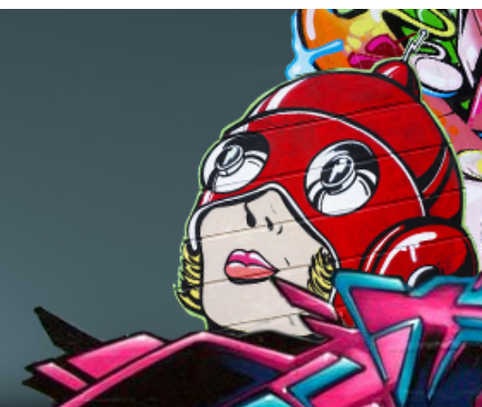
I2C Protocol

```
from machine import Pin, I2C

# create an I2C bus
i2c = I2C(scl=Pin('X1'), sda=Pin('X2'))

# scan for list of attached devices
dev_list = i2c.scan()

# write to and read from a device
i2c.writeto(0x42, b'4')
data = i2c.readfrom(0x42, 4)

# memory transactions
i2c.writeto_mem(0x42, 0x12, b'')
data = i2c.readfrom_mem(0x42, 0x12, 2)
```

GPIO Interface

```
from machine import Pin

p_out = Pin('X1', Pin.OUT_PP)
p_out.high()
p_out.low()

p_in = Pin('X2', Pin.IN, Pin.PULL_UP)
p_in.value() # get value, 0 or 1
```

Hardware Interfaces



Advanced Features

(Further Reading)

- Highly configurable due to many compile-time configuration options
- Support for many architectures (x86, x86-64, ARM, ARM Thumb, Xtensa)
- Extensive test suite with over 590 tests, and more than 18,500 individual testcases
- Code coverage at 98.4% for the core and at 96.3% for the core plus extended modules

Advanced Features

- Fast start-up time from boot to loading of first script (150 microseconds to get to boot.py, on PYBv1.1 running at 168MHz)
- A simple, fast and robust mark-sweep garbage collector for heap memory
- A MemoryError exception is raised if the heap is exhausted
- A RuntimeError exception is raised if the stack limit is reached
- Support for running Python code on a hard interrupt with minimal latency

Advanced Features (Cont.)

- Errors have a backtrace and report the line number of the source code
- Constant folding in the parser/compiler
- Pointer tagging to fit small integers, strings and objects in a machine word
- Transparent transition from small integers to big integers
- Support for 64-bit NaN boxing object model
- Support for 30-bit stuffed floats, which don't require heap memory

Advanced Features (Cont.)

- A cross-compiler and frozen bytecode, to have pre-compiled scripts that don't take any RAM (except for any dynamic objects they create)
- Multithreading via the "_thread" module, with an optional global-interpreter-lock (still work in progress, only available on selected ports)
- A native emitter that targets machine code directly rather than the bytecode virtual machine
- Inline assembler (currently Thumb and Xtensa instruction sets only)

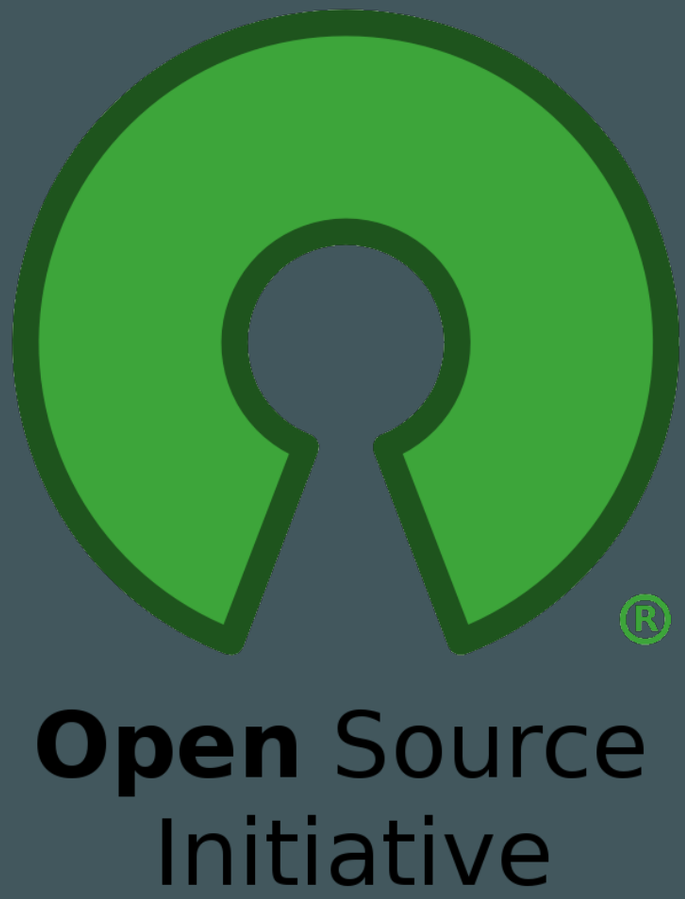
Advanced Features (Cont.)

Open Source

- MicroPython is written in C99
- The entire MicroPython core is available for general use under the very liberal MIT license
- Developed on open GitHub repo
- Source code is available on GitHub
- Everyone is welcome to contribute to the project.

Pycom contributes to MicroPython development and has funded features such as

- Garbage collector
- Multi-threading
- The ESP32 stack



https://github.com/micropython/micropython

https://github.com/pycom/pycom-micropython

pycom

GO INVENT

WiPy, LoPy, SiPy, FiPy & GPy

Pycom Devices

WiPy

ESP32
WiFi
Bluetooth

LoPy

ESP32
WiFi
Bluetooth
LoRa (Semtech, LoRaWAN certified)

SiPy

ESP32
WiFi
Bluetooth
Sigfox (Texas instruments, Sigfox certified)

GPy

ESP32
WiFi
Bluetooth
LTE-M (NBIoT & CAT-M1)
(Sequans Monarch)

FiPy

ESP32
WiFi
Bluetooth
LoRa
Sigfox
LTE-M (NBIoT & CAT-M1)

- Multi-network boards and modules (WiFi, BLE, LoRa, Sigfox, LTE-M, NB-IoT, etc.)
- Write code once and use on all Pycom boards and OEM modules
- One footprint for all boards (Fits our expansion board + Pysense & Pytrack)
- MicroPython is super fast to program
- All boards are designed for low power and long battery
- ESP32 has a dual core chip architecture, enabling easy networking

Meet the LoPy

The world's first **triple network** LoRa, WiFi and Bluetooth IoT development board featuring the ESP32 and MicroPython.

Super fast development, fit for fully scaled deployment, low power and it doubles up as a Nano Gateway.

...and here's the rest of what we've been up to

FULL IOT STACK

Dashboard: Pybytes Desktop Application and Pymate Mobile App

Middleware: Pybytes cloud-based Middleware

Networks: WiFi, Bluetooth, LoRa, Sigfox, LTE-M

Gateways:
WiPy, LoPy, SiPy, GPy and FiPy + OEM versions

Sensors: Pysense, Pytrack, and Pyscan (Q317)

Live Demo

Examples with the Pycom Devices