The University Of Sheffield

Country of the blind: networking devices without user interfaces

COM3505, Lecture 5

Prof Hamish Cunningham

Week 5...

Congrats! When you've finished the next chunk you're half-way through!

Today we put some of the jigsaw together and start creating our first true IoT devices...

- review of firmware to date (ask qs!)
- a preview of cloudside & projects
- labs: turning *things* into *internet things*

But first a word about assessment…
(and the story of a man called Chico and his bag full of cash).

COM3505 assessment:

- each week's lab assignments bring you a max of 7% of your final mark, and
- each MOLE quiz brings you 10%

The weighting of each assignment is in relation to their complexity, but roughly speaking if we asked you to do 7 things in a week and you completely each one perfectly, then each one would be worth 1 point out of 100 towards your final mark. Optionals exercises count in the same way as core (but you can pass without doing them).

So: lots of little steps... keep plugging away!

ESP32 Firmware, the Story so Far (1)

```
// Ex01.h

#ifndef LAB_01_H
#define LAB_01_H

// MAC address ////////////////////////////////////////////////////////////
char MAC_ADDRESS[13]; // 12 char MAC, + NULL terminator
void getMAC(char *);

// LED utilities //////////////////////////////////////////////////////////
void ledOn();
void ledOff();
void blink(int = 1, int = 300);

#endif
```

```
// Ex01.ino: printing on the serial line; blinking the built-in LED
void setup01() {
  Serial.begin(115200);          // initialise the serial line
  getMAC(MAC_ADDRESS);   // store the MAC address as chip ID
  pinMode(BUILTIN_LED, OUTPUT);   // set up pin for built-in LED
  Serial.println("\nsetup01..."); // say hi
}
```

```
// ...Ex01.ino…

void loop01() {
  Serial.printf("ESP32 MAC = %s\n", MAC_ADDRESS); // print the ESP's "ID"
  blink(3);                  // blink the on-board LED...
  delay(1000);                  // ...and pause
}

// why not have getMAC return a String? we could, but see:
// https://hackingmajenkoblog.wordpress.com/2016/02/04/the-evils-of-arduino-strings/
// https://www.youtube.com/watch?v=eVGvPsCrQ_Y
void getMAC(char *buf) { // the MAC is 6 bytes, so needs careful conversion...
  uint64_t mac = ESP.getEfuseMac(); // ...to string (high 2, low 4):
  sprintf(buf, "%04X%08X", (uint16_t) (mac >> 32), (uint32_t) mac);
}
// That DOESN'T MEAN never use String! Just be aware of potential problems.

void ledOn()  { digitalWrite(BUILTIN_LED, HIGH); }
void ledOff() { digitalWrite(BUILTIN_LED, LOW); }
void blink(int times, int pause) {
  ledOff();
  for(int i=0; i<times; i++) {
    ledOn(); delay(pause); ledOff(); delay(pause);
  }
}
```

ESP32 Firmware, the Story so Far (2)

```
// Ex02.h
#ifndef LAB_02_H
#define LAB_02_H
bool useInternalLED = true; // which LED to blink
#endif


// Ex02.ino
// blinking an external LED; reading from a switch

void setup02() {
  setup01(); // include previous setup for serial, and for the internal LED
  Serial.printf("\nsetup02...\nESP32 MAC = %s\n", MAC_ADDRESS); // ESP's "ID"

  // set up GPIO pin for an external LED
  pinMode(32, OUTPUT); // set up pin 32 as a digital output

  // set up GPIO pin for a switch
  pinMode(14, INPUT_PULLUP); // pin 14: digital in, built-in pullup
}
```

```
void loop02() {
  if(digitalRead(14) == LOW) { // switch pressed
    Serial.printf("switch is pressed...\n");
    useInternalLED = ! useInternalLED;
  }

  if(useInternalLED) {
    Serial.printf("blinking internal LED...\n");
    blink(1, 500); // where did this come from? Ex01.h
  } else {
    Serial.printf("setting 32 HIGH...\n");
    digitalWrite(32, HIGH);       // on...
    delay(500);

    Serial.printf("setting 32 LOW...\n");
    digitalWrite(32, LOW);        // off...
    delay(500);
  }
}
```

ESP32 Firmware, the Story so Far (3)

```
// Ex03.h
// add more LEDs and switches; run as traffic lights or individually
#ifndef LAB_03_H
#define LAB_03_H   // LEDs are assigned to pins 32,15 and 12
const int red = 32; const int yellow = 15; const int green = 12;
boolean ON_RED = false;
#endif

// Ex03.ino
void setup03() {
  setup02();                  // previous setups...
  Serial.println("\nsetup03..."); // debug printout
  // set up further GPIO pins with nice names for additional external LEDs
  pinMode(yellow, OUTPUT);
  pinMode(green, OUTPUT);
  digitalWrite(green, HIGH); // we start the traffic light sequence on green
}

void loop03() {
  if (digitalRead(14) == LOW) { // if the switch is LOW, ie. pushed down
    Serial.printf(" ON_RED is %d; switch pressed...\n", ON_RED);
    changeLights();           // call this function to change the lights!
  }
}
```

```
void changeLights() {
  if(ON_RED) {
    // red and yellow on for 2 seconds (red is already on though)
    digitalWrite(yellow, HIGH);
    delay(2000);

    // turn off red and yellow, then turn on green
    digitalWrite(yellow, LOW);
    digitalWrite(red, LOW);
    digitalWrite(green, HIGH);
  } else {
    // green off, yellow on for 3 seconds
    digitalWrite(green, LOW);
    digitalWrite(yellow, HIGH);
    delay(3000);

    // turn off yellow, then turn red on for 5 seconds
    digitalWrite(yellow, LOW);
    digitalWrite(red, HIGH);
  }

  ON_RED = ! ON_RED;
}
```

ESP32 Firmware, the Story so Far (4)

```
// Ex04.h: debugging infrastructure
#ifndef LAB_04_H
#define LAB_04_H

// setting different DBGs true triggers prints
#define dbg(b, s) if(b) Serial.print(s)
#define dln(b, s)  if(b) Serial.println(s)
#define startupDBG      true
#define loopDBG         true
#define monitorDBG      false
#define netDBG          true
#define miscDBG         true
#define analogDBG       false

#endif
```

```
// Ex04.ino

// setup and loop ////////////////////////////////////////////////////////
void setup04() {
  setup03();          // previous setups: get the MAC, set up GPIO pins, ...
  dln(startupDBG, "\nsetup04..."); // debug printout on serial, with newline
}

void loop04() {
  dbg(loopDBG, "ESP32 MAC = "); // print the...
  dln(loopDBG, MAC_ADDRESS);    //   ...ESP's "ID"
  blink(3);                     // blink the on-board LED...
  delay(1000);                  //   ...and pause
}
```

ESP32 Firmware, the Story so Far (5)

```
// Ex05.h: loop slicing
#ifndef LAB_05_H
#define LAB_05_H
unsigned long firstSliceMillis; // what time did we start?
unsigned long lastSliceMillis;  // what time did we last run this action?
int loopIteration = 0; // a control iterator for slicing up the main loop
const int LOOP_ROLLOVER = 25000000; // # loops per action seq
#endif

// Ex05.ino
void setup05() {
  setup04();          // previous setups: get MAC, set up GPIO pins, ...
  dln(startupDBG, "\nsetup05..."); // debug printout on serial, with newline
  firstSliceMillis = millis();       // remember when we began
  lastSliceMillis = firstSliceMillis; // an approximation to use in first loop
}
void loop05() { // Q what types of timings do different slices equate to?
  int sliceSize = 1000000;
  int TICK_DO_SOMETHING = 1;
  int TICK_DO_SOMETHING_ELSE = 9999999;
  // actions on individual iterations
  ...
  if(loopIteration == TICK_DO_SOMETHING) { // do task for this iteration
    dbg(loopDBG, "doing something, loopIteration number = ");
    dln(loopDBG, loopIteration);
  } else if(loopIteration == TICK_DO_SOMETHING_ELSE) { // other task...
    dbg(loopDBG, "doing something else, loopIteration number = ");
    dln(loopDBG, loopIteration);
  }
  // actions on each X slices
  if(loopIteration % sliceSize == 0) { // a slice every sliceSize iterations
    dbg(loopDBG, "loopIteration number = "); dbg(loopDBG, loopIteration);
    dbg(loopDBG, ", slice lasted "); dbg(loopDBG, millis() - lastSliceMillis);
    dbg(loopDBG, " milliseconds"); dbg(loopDBG, "; slice size is ");
    dln(loopDBG, sliceSize);
    lastSliceMillis = millis();
  }
  // roll over (alternative: just let the counter overflow...)
  if(loopIteration++ == LOOP_ROLLOVER) {
    loopIteration = 0;
    dbg(loopDBG, "loopIteration rolling over; ");
    dbg(loopDBG, LOOP_ROLLOVER); dbg(loopDBG, " loops lasted ");
    dbg(loopDBG, millis() - firstSliceMillis);
    dln(loopDBG, " milliseconds...; rolling over");
  }
} // loop05()
```

ESP32 Firmware, the Story so Far (6)

```
// Ex06.h: starting up a wifi access point and web server
#ifndef LAB_06_H
#define LAB_06_H
#include <WiFi.h>
#include <ESPWebServer.h>
String apSSID;                // SSID of the AP
ESPWebServer webServer(80);     // a simple web server
#endif

// Ex06.ino
void setup06() {
  setup05();           // previous setups...
  dln(startupDBG, "\nsetup06..."); // debug printout on serial
  startAP();           // fire up the AP...
  startWebServer();     // ...and the web server
}
void loop06() {
  // things to try: how responsive is it with different time slicing options?
  // or what if we add in "loop02();"...? what if there are mulitple requests?
  // how might this compare with responsiveness on the ESP8266?
  if(! (loopIteration++ % 1000 == 0)) return; // a slice every 1k iterations
  dln(netDBG, "calling webServer.handleClient()...");
  webServer.handleClient(); // deal with any pending web requests
}
```

```
void startAP() {
  apSSID = String("Thing-");   apSSID.concat(MAC_ADDRESS);
  if(! WiFi.mode(WIFI_AP_STA)) dln(startupDBG, "set Wifi mode failed");
  if(! WiFi.softAP(apSSID.c_str(), "dumbpassword"))
    dln(startupDBG, "failed to start soft AP");
}
void startWebServer() { // register callbacks to handle different paths
  webServer.on("/", handleRoot);        // ".../"
  /* ... */  webServer.begin();
}
String getPageTop() { // page creation utils (for better ones see Ex07!)
  return "<html><head><title>COM3506 IoT [ID: " + apSSID + "]</title>\n"
/* ... */
void handleNotFound() { // webserver handler callbacks ...
   webServer.send(200, "text/plain", "URI Not Found");
}
void handleRoot() {
  String toSend = getPageTop();
  toSend += getPageBody();
  toSend += getPageFooter();
  webServer.send(200, "text/html", toSend);
}
void handleHello() { /* ... */ }
```

ESP32 Firmware, the Story so Far (7)

```
#ifndef LAB_07_H
#define LAB_07_H
const char *boiler[] = { // boilerplate: constants & pattern parts of template
  "<html><head><title>",                                          // 0
  "default title",                                                // 1
  "</title>\n",                                                   // 2
  "<meta charset='utf-8'>",                                       // 3
  // adjacent strings in C are concatenated:
  "<meta name='viewport' content='width=device-width, initial-scale=1.0'>\n"
  "<style>body{background:#FFF; color: #000; font-family: sans-serif;", // 4
  "font-size: 150%;}</style>\n",                                  // 5
  "</head><body>\n<h2>",                                          // 6
  "Welcome to Thing!",                                            // 7
  "</h2>\n<p><a href='/'>Home</a>   </p>\n",       // 8
  "</body></html>\n\n",                                           // 9
};


typedef struct { int position; const char *replacement; } replacement_t;
void getHtml(String& html, const char *[], int, replacement_t [], int);
#define ALEN(a) ((int) (sizeof(a) / sizeof(a[0]))) // only in definition scope!
#endif
```

```
void setup07() { setup06(); dln(startupDBG, "\nsetup07..."); }
void loop07() {
  webServer.handleClient(); // serve pending web requests every loop
  if(! (loopIteration++ % 500000 == 0)) return;
  for(int i = 0; i < ALEN(boiler); i++) dbg(miscDBG, boiler[i]);
  replacement_t repls[] = { // the elements to replace in the boilerplate
    { 1, "a better title" },
    { 7, "Eat more green vegetables!" },
  };
  String htmlPage = ""; // a String to hold the resultant page
  getHtml(htmlPage, boiler, ALEN(boiler), repls, ALEN(repls));
  dbg(miscDBG, htmlPage.c_str()); // print the result
}
void getHtml( // turn array of strings & set of replacements into a String
  String& html, const char *boiler[], int boilerLen,
  replacement_t repls[], int replsLen
) {
  for(int i = 0, j = 0; i < boilerLen; i++) {
    if(j < replsLen && repls[j].position == i)
      html.concat(repls[j++].replacement);
    else
      html.concat(boiler[i]);
  }
}
```

ESP32 Firmware, the Story so Far: Summary

Weeks 1 — 3:

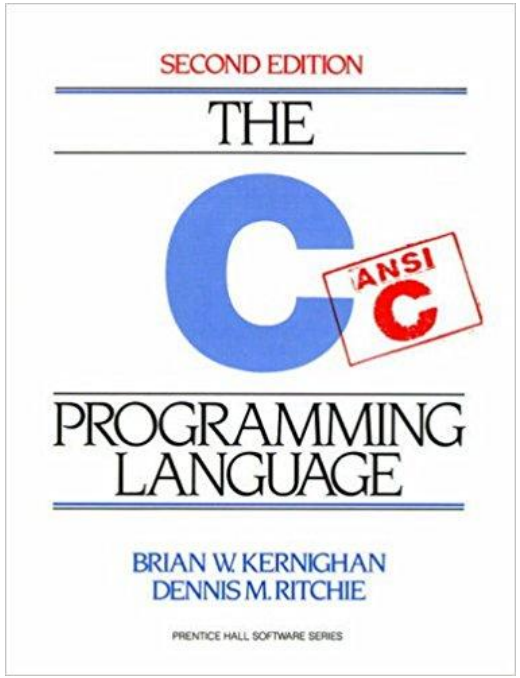
- Thing.ino: code structure; lab exercise runner sketch
- Ex01: printing on the serial line; blinking the built-in LED
- Ex02: blinking an external LED; reading from a switch
- Ex03: add more LEDs and switches; run as traffic lights or individually [OPT]
- Ex04: debugging infrastructure [OPT]
- Ex05: loop slicing

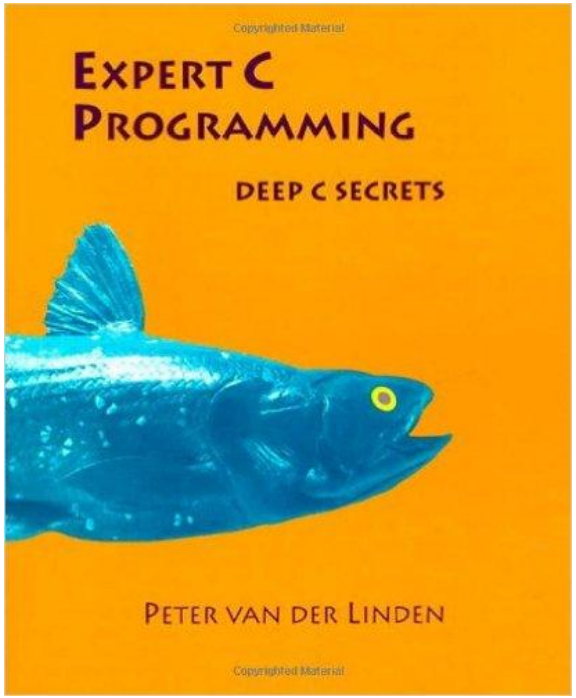
Week 4:

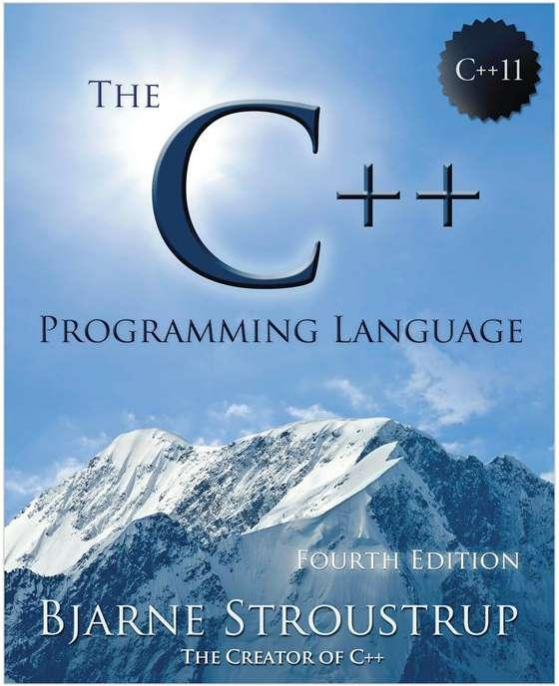
- Ex06: becoming a wifi access point and web server
  - using WiFi.h and ESPWebServer.h fire up an access point and serve pages
- Ex07 [OPT]: simple utilities for representing and manipulating HTML elements and serving pages

The University Of Sheffield

Deep C Secrets







Moving cloudside with Adafruit.io and IFTTT

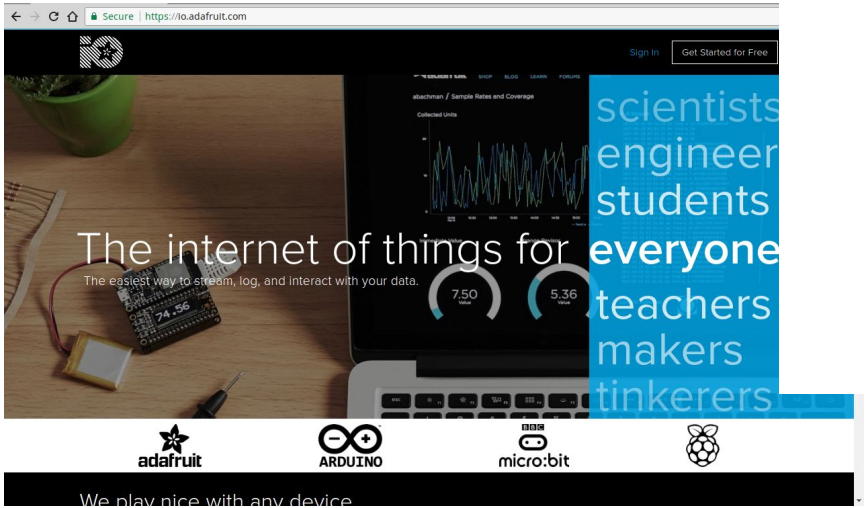
- Adafruit IO and IFTTT are both cloud services that can be used to connect with IoT devices.
- They both offer a ‘Freemium’ model that allows for basic use for free, with the option of paying for a premium service.
- Adafruit IO offers data-logging, graphing and dashboard facilities.
- IFTTT offers event-based connections to hundreds of other services such as SMS, twitter, phone notifications etc.

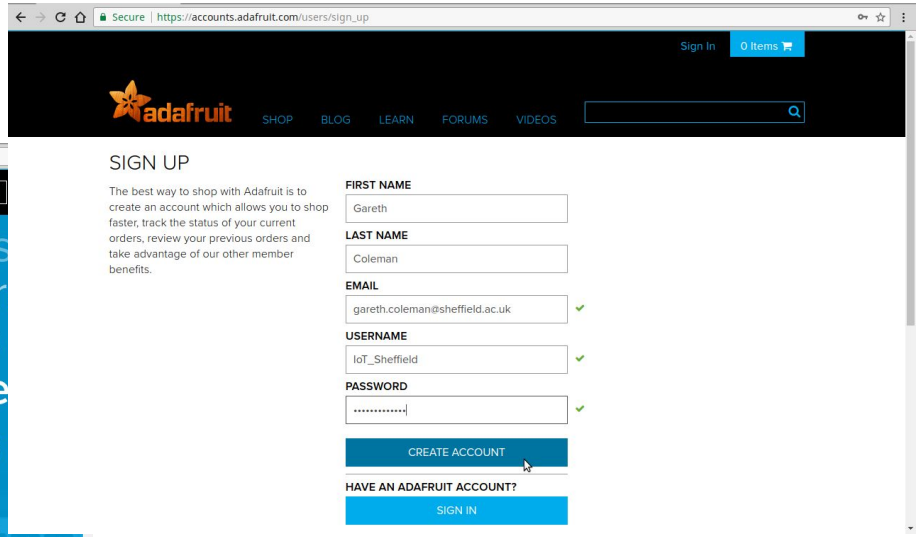
IFTTT: If This, Then That integration platform

Our ESPs can gather all sorts of data and push it into the cloud. IFTTT allows us to react to particular events in that data, triggering actions on a diverse set of platforms.

Adafruit IO: log data, visualise it, trigger IFTTT

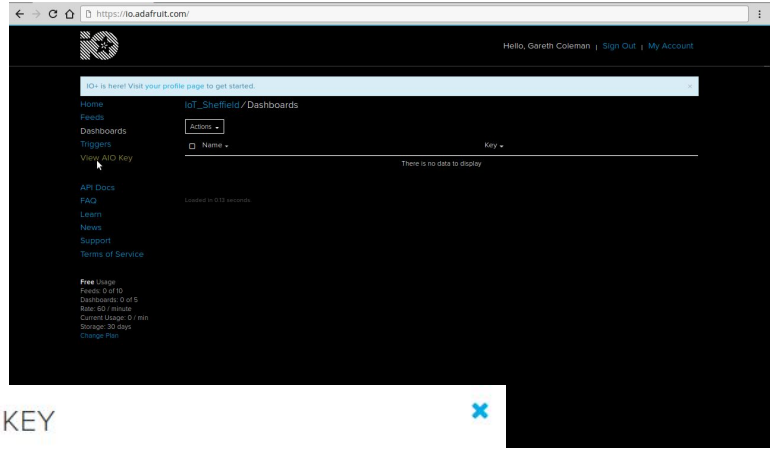
- visit: io.adafruit.com
- create account

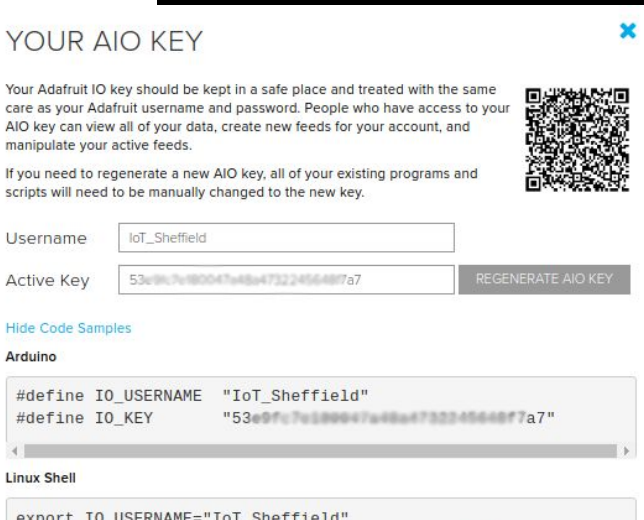




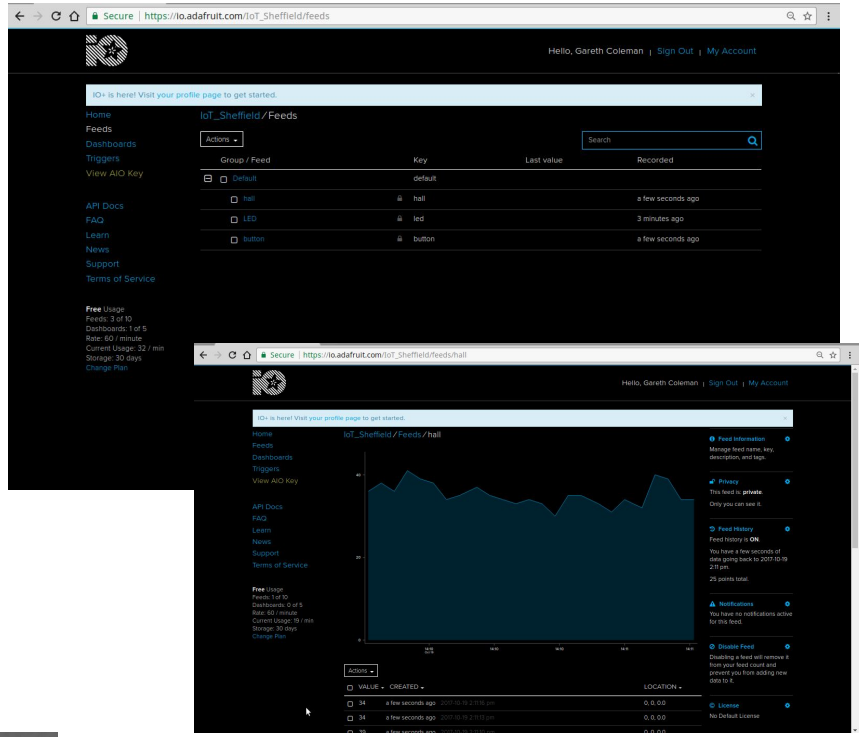
The keys to the kingdom

- In order to process messages sent to cloud platforms, user-unique API keys are commonly used
- Keep your key safe and private
- With Adafruit IO your key is accessible from the main page, ready for pasting into a sketch



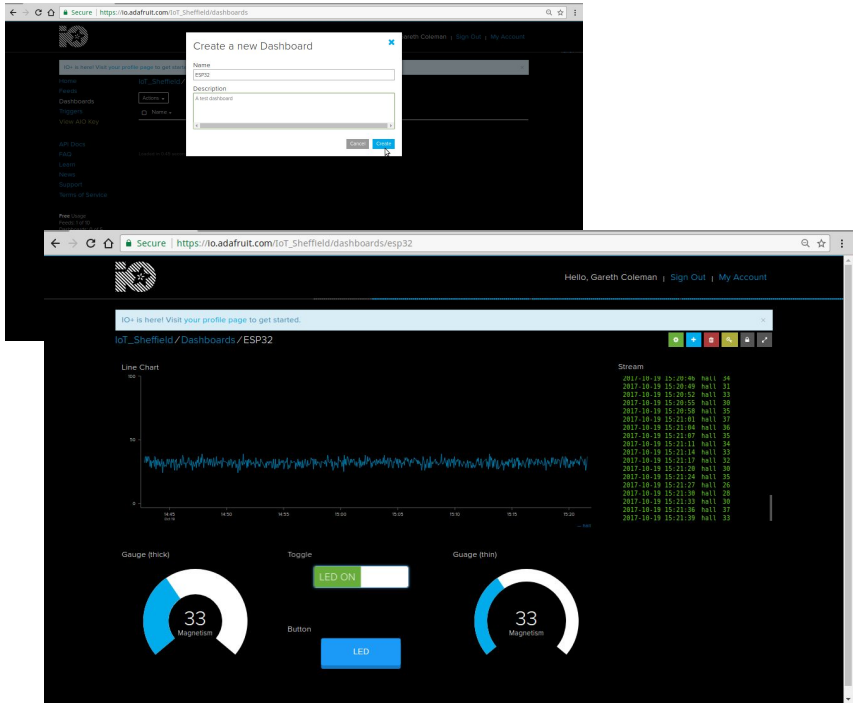


Feeds in Adafruit IO



- The basic unit of input and output in Adafruit IO is the feed
- Feeds receive data and store it
- Feed data is stored for 30 days
- Feeds have rate limit of 30 per min
- You can have up to 10 feeds

Dashboards in Adafruit IO

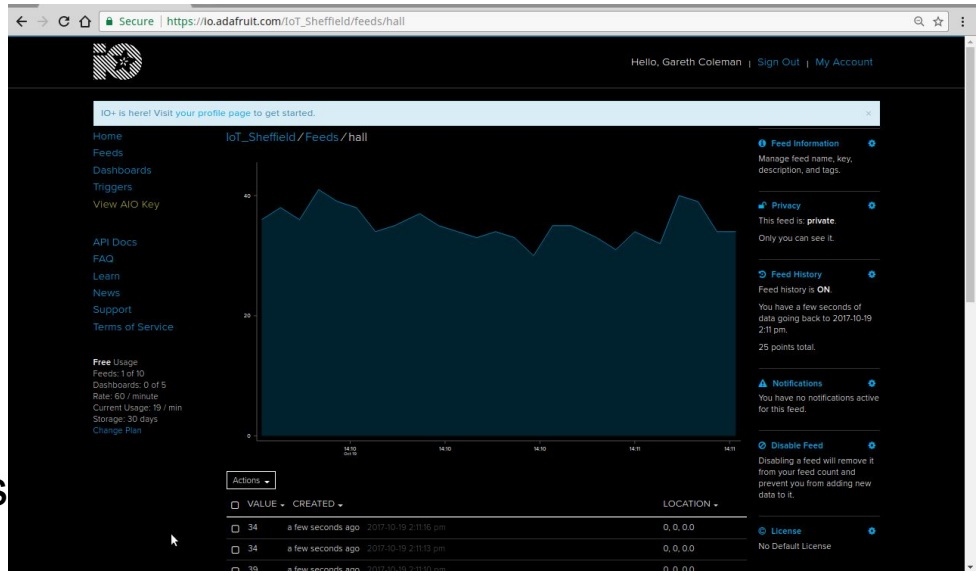


- Dashboards contain blocks that output data, receive input or do both
- Blocks are configured to connect to one or many feeds
- As with everything Adafruit, clear documentation and tutorials are provided:

learn.adafruit.com/adafruit-io-basics-dashboards

Adafruit IO and IFTTT in weeks 7-12

- We'll be generating data and pushing it to:
  - Adafruit.io
  - com3505.gate.ac.uk
- We'll generate images that describe (or *visualise*) the data, e.g. ========>
- We'll trigger events, either via IFTTT or using physical actuators like motors or radio-controlled switches
- We'll do useful stuff like….



Preview weeks 7-12: projects. P1: air quality monitor

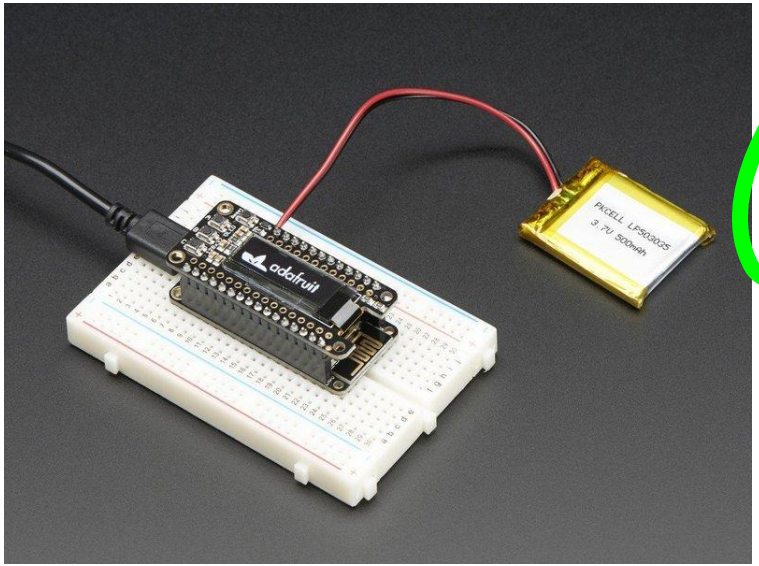
Who the…?

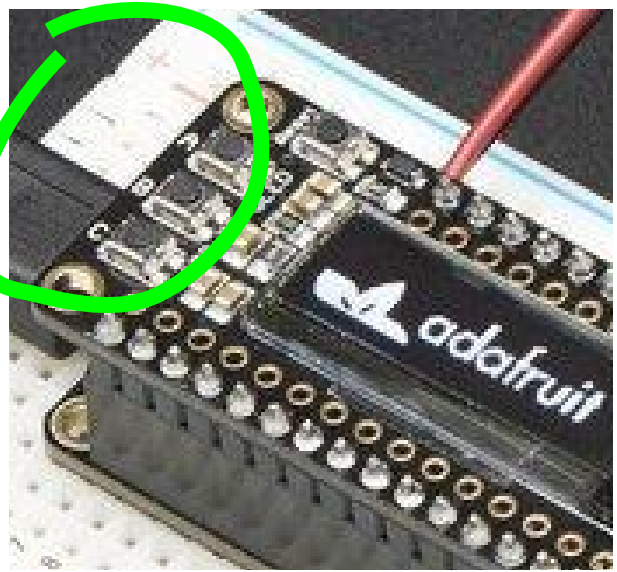
- how do we get better air?! (move to Dubai? or sue…?!)
- (also: intensive sustainable farming, project 4., 11m15s)

P2: Campus panic button

Issues:

- location
- trust
- spoofing
- visual indicators of “temperature”
- robustness
- battery life

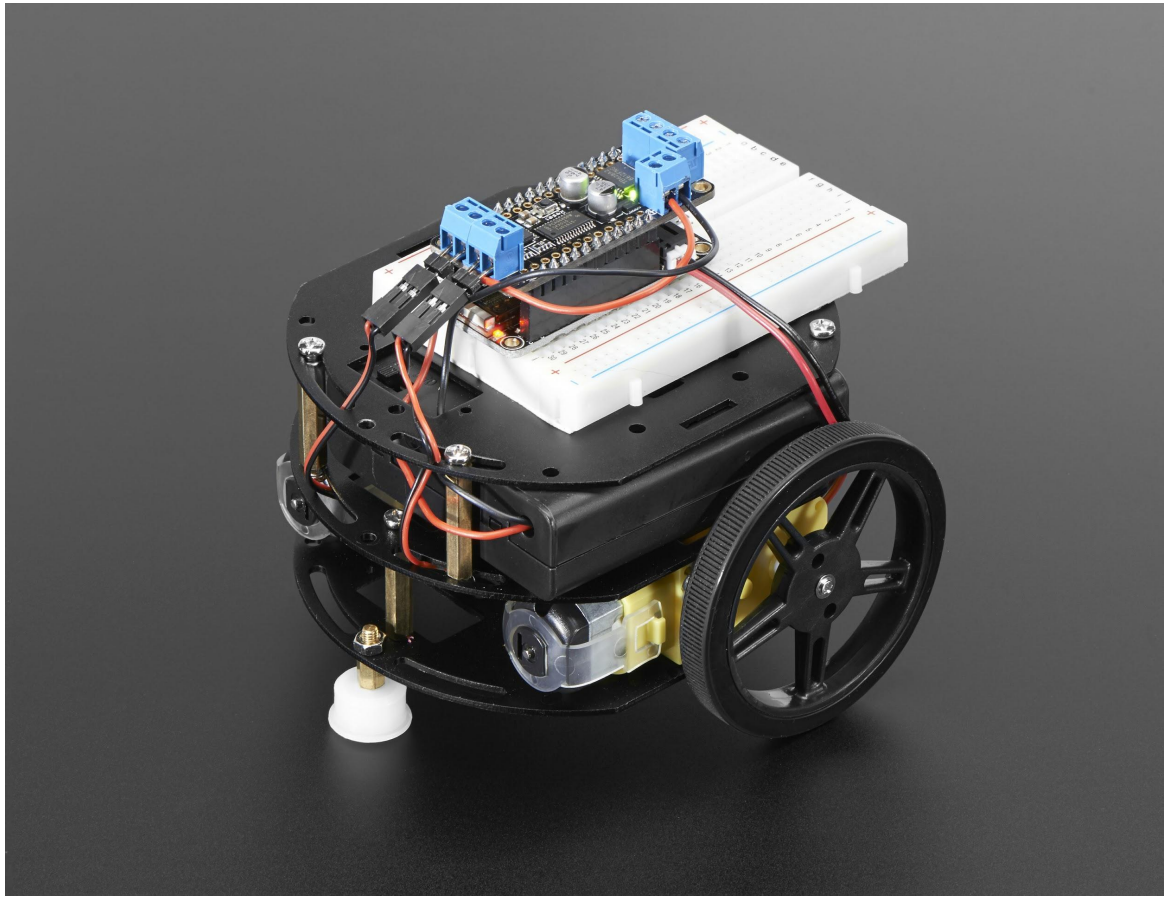




P3: RoboThing

Mobile platform for other projects?

- sniffing air q in different locations?
- checking for leaks around a greenhouse?
- …?



P4: WaterElf: promoting sustainable food tech

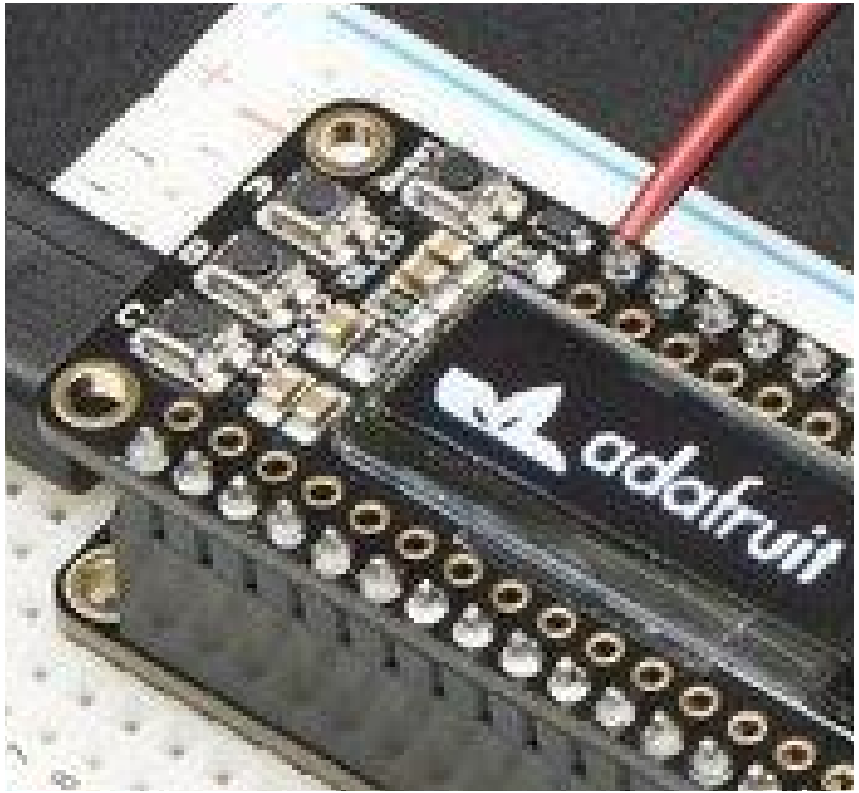
Aquaponics: save the world with fish poo!

Intensive agriculture without fertilisers.

Can be complex to control (3 organisms in balance); monitoring and control can help… Hence **WaterElf**

P5: **Peer-to-Peer** Voting Systems

- just how bad *is* your lecturer?
- *audiscians*: blending audience and performers
- a device for liquid democracy?
- **special guest star: Jared O'Mara MP (Friday Nov 10th 1.30pm DIA LT07)**



P6, P7….: other options

- water level warning system?
- off-grid with solar panels?
- various home automation projects
- define your own…?

NOTE: week 8 we're in the project space; 3D print or laser cut a case for your project?

Labs: how to network a device without a UI?

To turn a *thing* (embedded electronics based on microcontrollers, sensors and actuators) into an *internet thing* we need to connect to the network.

If the device has no user interface, how do we allow users to configure the network? (Cf. Sigfox, LoraWAN, ...)

- **Ex08**: become a web client and send your email & the MAC address of your ESP to our server (see lecture 1 slide 13 for the URL)
  - use the WifiClient class from the WiFi library
- **Ex09**: adapting Ex07/08 to allow connection of device to arbitrary networks
  - in Ex07 you (or **model code**!) created a wifi access point and served HTML pages
  - using the WiFi.scanNetworks() method, serve HTML to list available APs, and allow user to choose one, enter key, and have the ESP connect to that network
- **Ex10**: secure(ish) over-the-air (OTA) update for the ESP's firmware [**OPT**]
- **Ex11**: add captive portal functionality to Ex09 [**OPT**]

Your **TODO** list for week 5:

- Update your git repository clone as usual
- Read and digest Notes/Week05.mkd
- Do the exercises
- Review the lecture (tinyurl.com/com3505l5), do the reading
  - next week is **READING WEEK**! lie in bed all week! dye your hair! drink more than ever before!
  - alternatively: deepen knowledge by experimenting with ESP32; widen understanding by reading context; review lectures, model code and reading lists
- **Labs this week**:
  - two hours in NC_PC on **Tues at 10am**
  - one hour scheduled in the electronics lab on **Weds at 1pm** (and on Wednesday afternoons you can hang around there as long as you like!)