KV6006 2022 - IOT Practical

Jonathan Sanderson, Northumbria University
2022-11-16 v1.0

> *“If it squirms, it’s Biology.*
> *If it stinks, it’s Chemistry.*
> *If it doesn’t work, it’s Physics.*
> *And if you can’t understand it, it’s Mathematics” .*
> *— Dr. Magnus Pyke*

Introduction

In this session, you’ll write some short scripts which will pull data from a range of sources (web APIs and sensor devices), manipulate the data, then display or output it in some way.

Some of the examples are trivial or silly, but they’re intended to get you thinking about how you structure data, how it’s moved around a system, and where processing takes place. Many networked systems can be thought of as “*small pieces, loosely joined*.” If the network is reliable and data is structured in a consistent way, sources can publish data without too much regard for how it’s being consumed. Similarly, data sinks (output destinations) can be combined in novel and unexpected ways.

Since neither source nor sink knows how it’s being employed, it’s also possible to build systems which are relatively resilient to partial network outages, sensor failures, or other issues which might afflict a long-running aggregate system.

Terminology

We’re going to use three main pieces of jargon:

- **API**: ‘Application Program Interface.’ An informal definition would be: a documented specification which describes how somebody else’s program can interface with yours. Developers might query for data from your API, or pass data to your software through it. As long as your code behaves as you’ve documented, the other developer doesn’t need to know anything about your implementation details. Good API design is a science (or possibly art form) in itself.
- **JSON** is a common data interchange format, often used for IOT applications. Data is structured in key/value pairs, and deeper related structures. You likely encountered JSON in KF5002 Web Programming.

- **MQTT** is a simple message-passing protocol, which relies on a central server (or 'broker') to handle data. Programs can register themselves with a broker and *publish* message through it, or *subscribe* to messages from it. Each message has two components: the *topic* (which you can think of as a channel), and the *payload* (the actual data).

Platform

We're going to use Raspberry Pis, which are simple desktop computers running a version of the Linux operating system. We'll be writing code in Python, using the Thonny editor. You could use any desktop or laptop computer and pretty much any language, but getting all the libraries installed can be a bit of a faff, not to mention working around the university firewall.

If you've not used Python before it can be a bit of a jolt, but you'll pick it up as you go along. The main things to remember are:

1. Indentation is significant. Where you've used curly braces `{...}` in other languages to denote blocks of code, in Python you indent by four spaces. Python is picky about spaces vs. tabs; Thonny will do the right thing.
2. You have to define classes and functions before you can use them. In C and similar languages function declarations are often *hoisted* so they execute first: in Python the `def` has to appear first.

Microcontrollers

We'll be communicating with networked microcontroller devices. Some are ESP8266 boards, running the Arduino platform (C++), while others are Raspberry Pi Pico W boards running MicroPython. They're limited in processing power and RAM, but are nevertheless surprisingly capable.

Microcontrollers typically lack an operating system as such, and store their program in flash memory. Apply power, and they'll immediately start running whatever code they were last flashed with. Even cheap hobbyist devices are astonishingly robust: we've had boards which have sat on a shelf for years between uses, without issue, and others which have run continuously for several years.

Pico W and ESP8266 microcontrollers cost about $6 each.