No-Code Programming for Biology

Beginner's Guide

Stephanie Norwood
University of Cambridge

Biotechnology and Biological Sciences Research Council
Engineering and Physical Sciences Research Council
Natural Environment Research Council

No-Code Programming for Biology

Beginner's Guide

Learn how to:

- Understand and control an Arduino board
- Programme without using code
- Use simple electronic components such as screens and sensors
- Build your own devices for use in biological research

OpenPlant Biomaker Initiative

Biomaker was started in 2014 as an interdisciplinary scheme for project-based learning and innovation, coordinated by the Synthetic Biology Interdisciplinary Research Centre at the University of Cambridge and OpenPlant, one of the UK's six National Synthetic Biology Research Centres. It has been funded mainly by contributions from the BBSRC and EPSRC research councils in the UK, and a NERC/NSF UK-US collaborative grant programme.

Biomaker provides funding for interdisciplinary team-based projects at the intersection of electronics, computer science, 3D printing, sensor technology, low cost DIY instrumentation and biology, as well as workshops and outreach events. The initiative aims to build open technologies and promote development of research skills and collaborations. It taps into existing open standards and a rich ecosystem of resources for microcontrollers, first established to simplify programming and physical computing for designers, artists and scientists. These tools allow biologists to program and develop real-world laboratory tools. The Biomaker project also provides a direct route for physical scientists and engineers to get hands-on experience with biological systems.

We aim to lower the barriers that impede interdisciplinary work, and to promote the kinds of training that are useful for building instruments and devices for biological experiments in the lab and field. We develop starter kits for no-code programming that allow biologists to build bioinstrument prototypes for measurement and control of biological systems. These have a wide range of applications including instrumentation, microscopy, microfluidics, 3D printing, biomedical devices, DNA design, plant sciences and outreach and public engagement. You can find examples of documented projects on the Biomaker website at **www.biomaker.org**.

An important aspect of Biomaker is the use of open source and low cost tools and hardware, which facilitate equitable access to fundamental knowledge and technology, encourage a collaborative environment, and support the establishment of an open, sustainable bioeconomy.

OpenPlant | UKRI Natural Environment Research Council | NSF | UKRI Engineering and Physical Sciences Research Council | UKRI Biotechnology and Biological Sciences Research Council | SynBio IRC

Welcome to the No-Code Programming for Biology Beginner's Guide

This beginner's guide has been put together by the Biomaker team to help you get to grips with the basics of biomaking and building custom instrumentation for biological research.

Designed for those with little to no experience working with coding or hardware, this guide makes use of free open-source software and low-cost hardware to introduce you the principles behind making your own instruments.

Working though this guide can be useful as a base for those with a specific challenge or task in mind, as well as for those who are simply looking to expand their biological skillset.

Whilst we will focus on learning aspects that are useful for biological research, the information in this guide can also be used for a wide range of no-code programming applications, and we hope that these skills can be applicable whatever your area of interest.

The guide will teach you how to use the free open-source, no-code programming software XOD, as well as how to use some simple low-cost hardware devices, such as LEDs, sensors and screens.

It is built to accompany the Grove All-In-One Beginner Kit for Arduino development board, designed by Seeed Studio. Alternative versions of the Arduino Uno board and accompanying components can also be used, but you will need to wire-up these components before you start.

We have chosen the Grove Beginner Kit for Arduino as this all-in-one kit will allow you to get started without having to wire-up components. At the time of writing, this kit is available from UK stockists Cool Components and Mouser Electronics for approximately £15-£23. The Biomaker team also has a number of free kits available. To request a kit, please get in touch with the Biomaker team.

AUTHORS

Stephanie Norwood
Jim Haseloff

IMAGES

Stephanie Norwood
SparkFun Electronics
Adafruit Industries
Biomaker Challenge Participants

XOD LIBRARY CREATORS

Matt Wayland
Marco Aita
Cesar Sosa
XOD user: gst
XOD user: e
XOD user: gweimer

CONTACTS

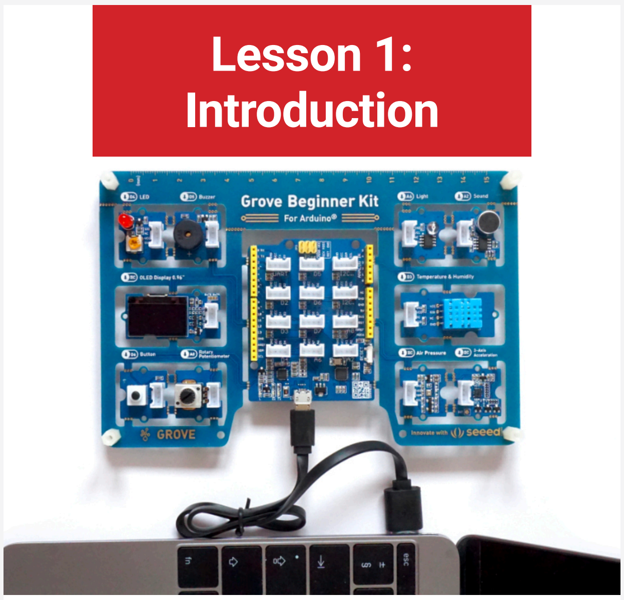
Stephanie Norwood
synbio@hermes.cam.ac.uk

Jim Haseloff
jh295@cam.ac.uk

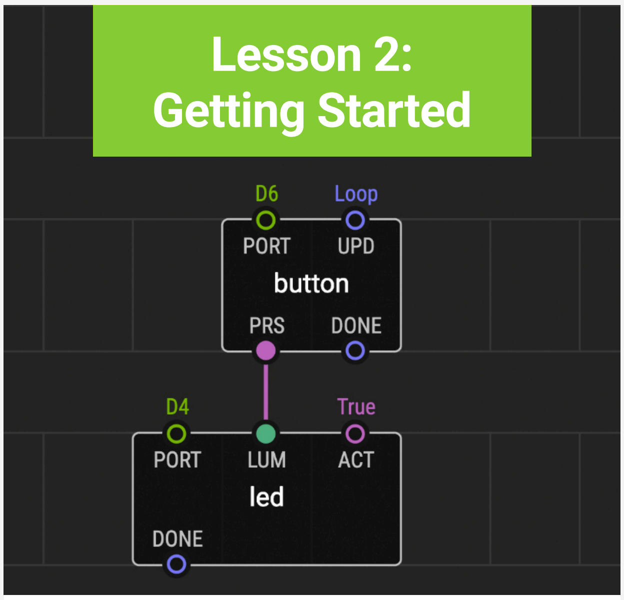
Contents

No-Code Programming for Biology

Beginner’s Guide

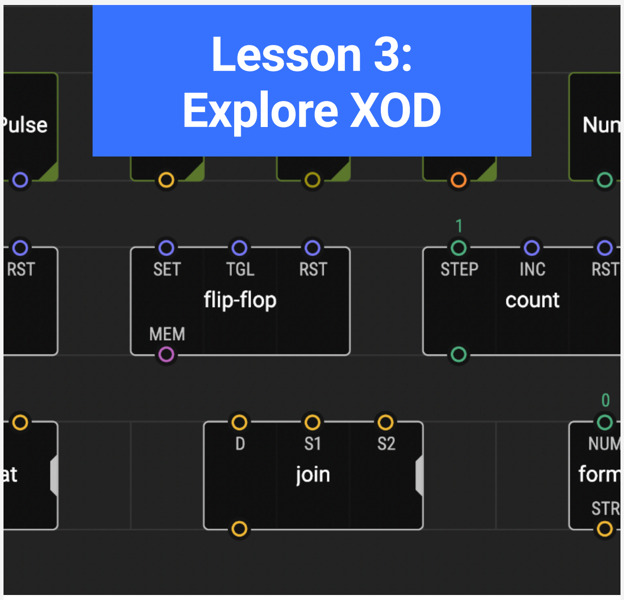


THE GUIDE
THE STARTER KIT
THE MICROCONTROLLER
THE XOD IDE



SETTING UP YOUR BOARD
TASK 1: TESTING YOUR BOARD
TASK 2: INPUT AND OUTPUT DEVICES

Lesson 3: Explore XOD

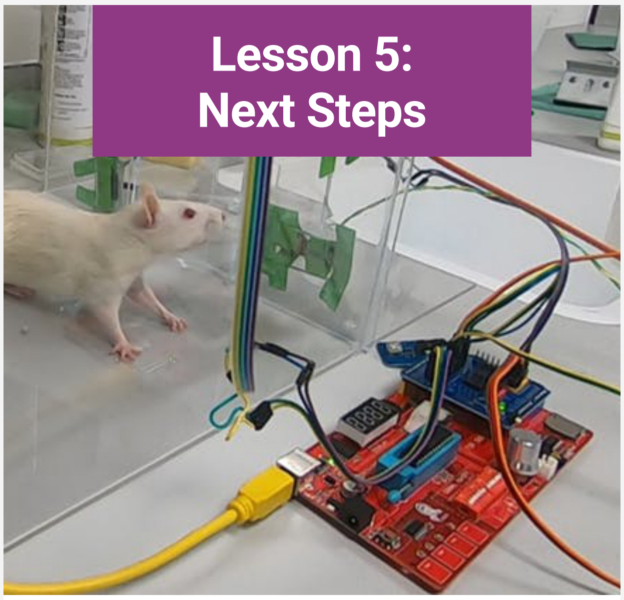


TASK 3: TWEAK AND WATCH NODES
TASK 4: FLIP, CLOCK AND COUNT NODES
TASK 5: CONCAT, JOIN AND FORMAT-NUMBER NODES

Lesson 4: Building Devices

TASK 6: CREATING NEW NODES
TASK 7: USING BUSES
TASK 8: LOGIC PROGRAMMES
TASK 9: SEQUENCES AND LOOPS

Lesson 5: Next Steps



EXPANDING YOUR CAPABILITY
CASE STUDIES
ADDITIONAL INFORMATION

Funders and Sponsors

Biotechnology and Biological Sciences Research Council
Engineering and Physical Sciences Research Council
Natural Environment Research Council
UNIVERSITY OF CAMBRIDGE
SynBio IRC
NSF
OpenPlant

BBSRC EPSRC NERC
NSF OPENPLANT SYNBIO IRC
UNIVERSITY OF CAMBRIDGE

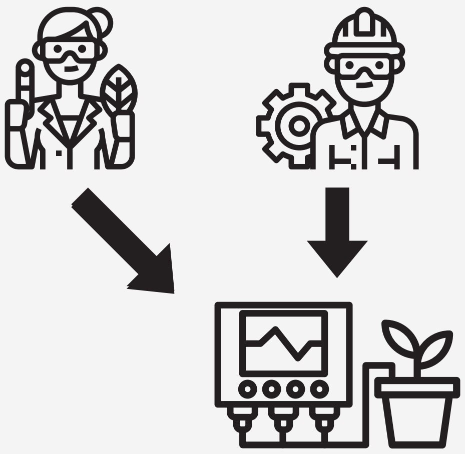
Lesson 1: Introduction

Introduction

Biomaker and No-Code Programming for Biology

The Biomaker team has put together this guide to introduce biologists, or other scientists with little formal programming training, to the basics of Biomaking, including:

- Use of Arduino-based microcontrollers
- Use of sensors, displays and actuators
- Use of XOD visual programming



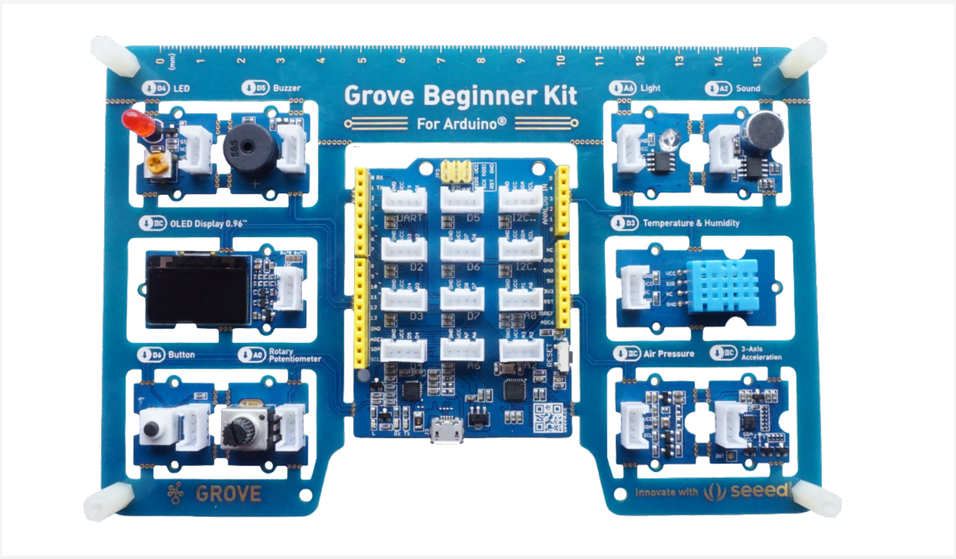
These new skills can be enabling in many ways. Scientists can gain expertise and new ways of thinking to apply to their work. Moreover, the components for this type of instrumentation are often very cheap, especially when compared with off-the-shelf commercial solutions. The use of simple hardware and software resources allow easy modification, extension and repair of custom instruments, and the use of open-source components and systems promotes sharing of information and set up of collaborative projects. This creates a growing set of resources for the community to draw from, and build upon.

We hope that you enjoy taking part in this online course, that you learn something new and that you find it useful for your future career. Most of all, we encourage anyone who is interested in developing their skills further to sign up for the Biomaker Challenge, where you can join a team of like-minded scientists and engineers to build bioinstruments for real-world applications.

In this first lesson we will cover some of the basic background information you will need to know before you start programming.

First, we will take a look at this guide and how to use it. Then we will look at the Grove board and explore each of it's built-in devices, including how they might be used. Next, we will briefly discuss microcontrollers and how to programme them, and finally we will introduce you to the XOD IDE software and some of it's terminology.

These basics will help you to become familiar with the tools we will be using throughout this guide. Many of these concepts will be covered again as we apply this knowledge to perform hands-on tasks later in this guide.



The Grove All-in-One Beginner Kit for Arduino

OBJECTIVES

By the end of this chapter you should be able to:

- Name the different parts of the Grove board and give examples of how they might be used.
- Describe the basic concepts of a microcontroller.
- Describe the steps involved in programming the Arduino board and how information is transmitted in this system.
- Name three of the most common types of electronic communication and explain the difference between them.
- List the pin (port) connections for each of the board's components.
- Recall the different parts of the XOD IDE software and describe what each part is used for.
- Recount the three key terms used in XOD programming and what they mean.
- List the data types used in XOD and give examples of each.

The Guide

The guide is split into four core lessons, each described below. These lessons are designed to be worked through in order, and start by exploring the Biomaker starter kit and the XOD integrated development environment (IDE). The rest of the guide will then take you through a series of tasks designed to introduce you to your board, as well as some key aspects of programming in XOD.

The final chapter provides some additional useful information, as well as some details about how to expand your skills and get started with designing your own devices.

In addition to the information in this guide, the XOD website also provides some useful tutorials and a community forum where you can find help at **www.xod.io**

Introduction

This chapter will give you a brief introduction to the Biomaker starter kit, including the Grove board, how to control it, and how to use XOD.

Getting Started

This chapter will take you through a few simple tasks to get started with using your board and the XOD IDE. You'll learn to use the LED, buzzer and button devices.

Explore XOD

This chapter will explore some of the most useful functions of XOD. Understanding how to use these functions will give you a great base to work from.

Building Devices

This chapter will delve into some more complex functions in XOD. By the end, you should be able to start developing your own ideas, programmes and devices!

The No-Code Programming for Biology Handbook

In addition to this beginner's guide, the OpenPlant Biomaker team has also created a range of useful beginner and advanced resources available on the Biomaker website. These resources are designed to help you learn more about the possibilities of Biomaking and to expand your capability to start building your own devices. They include additional tutorials, videos and information about commonly used hardware and expansion devices.

All of the Biomaker and No-Code Programming for Biology resources are available to download on the Biomaker website at **www.biomaker.org/resources**.

Tools to Accompany the Guide

GROVE BEGINNER KIT FOR ARDUINO

The Biomaker starter kit is composed of this beginner's guide, and the Grove Beginner Kit for Arduino. This kit is made by the open source hardware company Seeed Studio and is based on a simple Arduino microcontroller. The kit comes as an integrated PCB board with several useful input and output devices already connected and ready to go. No soldering, wiring or connecting of components means it's perfect for getting started with hardware!

The Starter Kit section (p6-7) provides a quick summary of each part of the board and what it might be used for, whilst the Microcontroller section (p8-11) gives a little background on the Arduino board.

XOD IDE

The XOD integrated development environment (IDE) is a free open-source software that allows you to programme microcontroller-based devices, such as Grove or Arduino boards, using visual 'nodes' rather than written code. Nodes can represent devices or functions, and by linking them together in different ways you can create a wide variety of different programmes. Programming visually like this can save some of the time and energy required to learn a new language and large amounts of syntax.

The XOD IDE section (p12-15) provides a quick summary of the different parts of the XOD IDE, and what you'll see when you first load the software, as well as some useful terminology used in XOD programming.

XOD WEBSITE

The XOD website (**www.xod.io**) provides plenty of useful information for beginners, including tutorials and a user guide under the 'Documentation' tab, a database of libraries under the 'Libraries' tab, and a very helpful forum under the 'Community' tab.

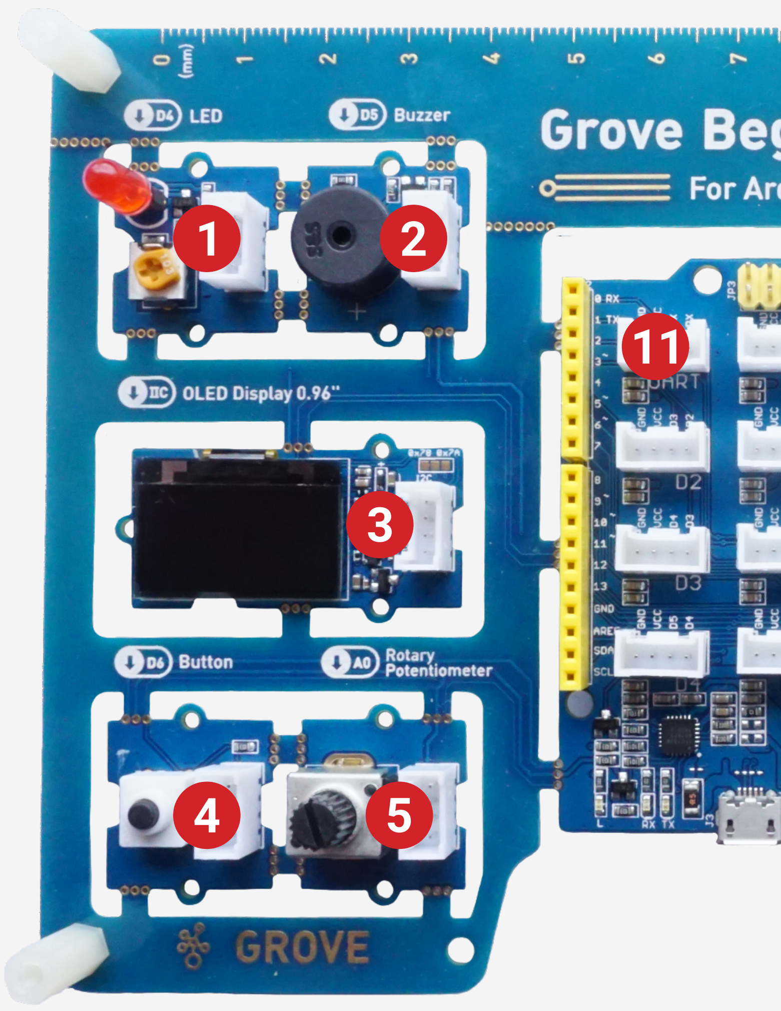
BIOMAKER WEBSITE

The Biomaker website (**www.biomaker.org**) hosts a variety of useful materials, including digital downloads of this guide, the accompanying tutorial file, and a number of other Biomaker tutorials and handbooks. These can be found under the 'Getting Started' tab.

You can also find examples of previous Biomaker projects on the website under the 'Projects' tab. With over 180 projects so far, there is plenty of inspiration for the budding Biomaker. Projects are also documented on the Biomaker Hackster Hub (**www.hackster.io/biomaker**).

The Starter Kit

The Biomaker starter kit is composed of this beginner's guide, and the Grove Beginner Kit for Arduino. This kit is made by the open source hardware company Seeed Studio and is based on a simple Arduino microcontroller.



1 **LED**

A red light emitting diode (LED). This light can be used as a notification or warning signal in devices.

2 **BUZZER**

Inbuilt piezoelectric buzzer. Can be programmed to emit tones at different frequencies.

3 **OLED SCREEN**

High quality OLED screen, which can be used to display both images and text. It is a 64x128 pixel matrix which can display desired content in monochrome (black and white).

4 **BUTTON**

A simple button that responds to user input (presses). Can be used as an on/off switch or trigger.

5 **ROTARY POTENTIOMETER**

Also known as a knob sensor as it senses the rotation angle of the knob. Can be used as a dial to change volume or brightness.

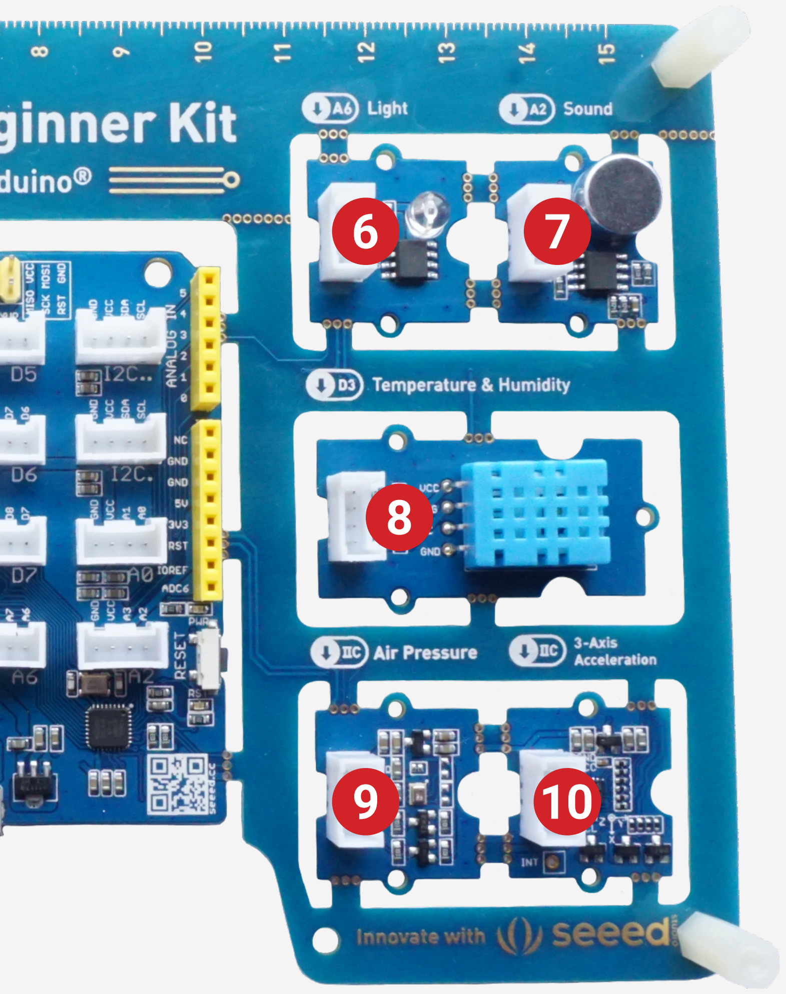
11 **MICROCONTROLLER DEVELOPMENT BOARD**

Based on the Arduino Uno and Seeeduino Lotus development board, this module is the brains of the board.

An ATmega328P microcontroller chip lies at the core, acting as small low-power computer that can be reprogrammed to create whatever device you wish.

The kit comes as an integrated PCB board with several useful input and output devices already connected and ready to go. No soldering, wiring or connecting of components means it's perfect for getting started with hardware!

Below is a quick summary of each part of the board and what they might be used for.



6 **LIGHT SENSOR**

A photoresistor that can detect the light intensity of the environment.

7 **SOUND SENSOR**

A simple microphone that can detect the sound intensity of the environment.

8 **TEMPERATURE AND HUMIDITY SENSOR**

Also known as a hygrometer. A pre-callibrated digital sensor that can measure the temperature and humidity of the environment. Will not work below 0ºC.

9 **AIR PRESSURE SENSOR**

Also known as a barometer. A high-precision digital sensor that can measure both air pressure and temperature. Can also be used to measure altitude.

10 **3-AXIS ACCELERATION SENSOR**

Also known as an accelerometer, which senses movement of the board. It can be used to measure orientation, tilting, movement or gestures.

The white plug sockets in the centre and yellow header sockets around the edges can be used to plug in additional components.

This module also has a reset button to reset your programme at any time, and a micro USB port, to connect the board to your computer.

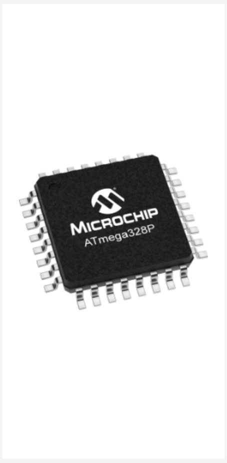
A USB cable is provided in the right-hand compartment of the Grove box, and Grove cables (to connect components to the white sockets) are provided in the left-hand compartment of the Grove box.

The Microcontroller

What is a Microcontroller?

A microcontroller is a small low-power computer encapsulated in a tiny electronic chip. In contrast to a general purpose computer like a laptop or PC, microcontrollers are often designed to complete one task and run one specific programme. They are low cost and only require small amounts of power, so they are often used in simple electronic devices such as kitchen appliances, implantable medical devices and power tools.

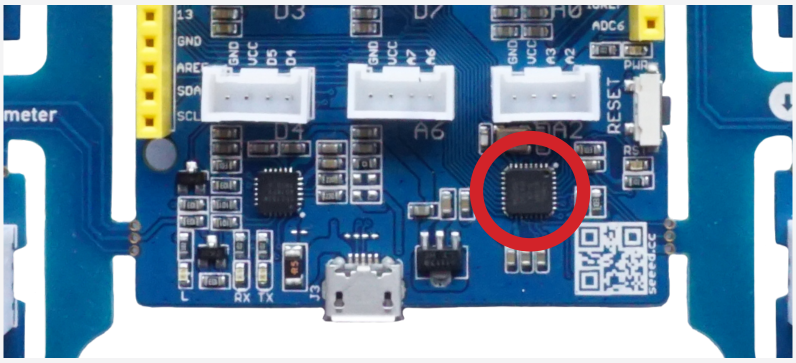
Like any other computer, a microcontroller has a Central Processing Unit (CPU), a 'long-term' memory (Electrically Programable Read Only Memory, EPROM) for holding your programme, and a 'short-term' memory (Random Access Memory, RAM) for holding and accessing user data. It communicates with the outside world via a series of metal pins that can either send (output) or receive (input) information.



The ATmega328P microcontroller used in the Arduino board

A microcontroller development board, like the Grove Arduino board, houses a microcontroller chip on a small PCB board alongside some additional parts and connections making it easy for anyone to programme and connect components to a microcontroller.

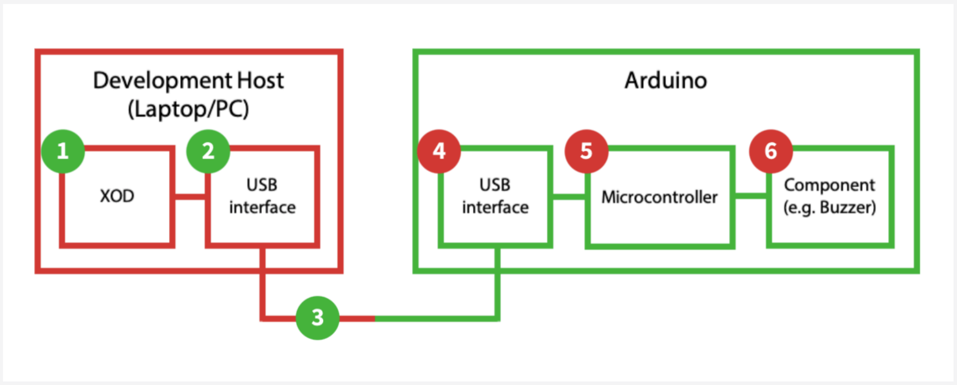
Development boards are intended to be cheap and easily accessible, and are often used for developing prototypes and custom instruments. To get an idea of the wide range of projects that are possible to achieve using an Arduino development board, take a look at the project documentation platform Hackster at **www.hackster.io/Arduino**.



Location of the ATmega328P microcontroller on the Grove board (red circle)

Controlling Your Arduino Board

In order to tell the Grove board what to do you will need to plug it into a computer. This is referred to as the development host, as it is where you will write and develop the programme you want to install. The diagram below explains how information is transferred from your computer to the board. Once the programme has been transferred, the board can be disconnected and will be able to run the desired programme independently of your computer, although it will need an alternative power supply. The board can be programmed to perform a multitude of different tasks depending on what components you want to use, and what programmes you install.



Workflow for programming your Arduino board

1 Write your programme using XOD software

2 Upload your programme

3 Information is sent from the computer to the Grove board via a USB cable

4 Information is received by the Grove board

5 The programme is written to the EPROM (long-term) memory of the microcontroller chip. This allows the board to act as its own independent computer, carrying out the specific programme you have uploaded.

6 Information is sent to the onboard components via the microcontroller pins. The programme stored in the microcontroller's memory will tell the components what to do, for example, turn on the buzzer at a certain pitch.

The Microcontroller

Types of Communication

There are several different ways for the board to communicate with your components. These are known as communication protocols, and they are the different ways in which data can be transferred between devices. Which pin is connected to which device depends on what type of communication protocol is used, and that depends on the type of device. Below we describe the three types of communication that are used on the Grove board.

It is important to be aware of these different communication types, as they will determine the board’s pin connects (see next page), as well as how to plug in any additional components you would like to add to your board.

ANALOG	Analog sockets are used to connect analog input and output devices. They can transmit signals that are continuous (meaning they can have an infinite amount of values within a given range), unlike digital sockets which can only transmit signals in two states: on and off. Most sensors are analog sensors.
DIGITAL	Digital sockets are also used to connect input and output devices. However, unlike analog devices, digital devices cannot take a range of values, they can only communicate by switching between two states: on and off. These digital sockets are used for most non-sensor components.
I2C	For devices which deal with both inputs and outputs we need duplex communication protocols, which can transmit data in both directions. The duplex protocol used on the Grove board is I2C. Inter-integrated circuit (I2C) pins provide a way to communicate with multiple devices at once. In this case, several devices are connected to the same pin, and each device is given a “name” (known as an address). Addresses are written as XXh, with XX being a two digit code of numbers and letters. For example 19h or 3Ch.

This covers the three pin types used to connect the Grove board’s inbuilt components. Arduino boards are also able to connect to devices using two other communication protocols, known as UART and SPI. We will not use these communication types in this guide, but if you would like to learn more you can find an excellent tutorial comparing I2C, UART and SPI protocols on the SparkFun website at **www.learn.sparkfun.com/tutorials/i2c**.

Pin Connections

Once your programme has been uploaded to the microcontroller chip, the chip needs to communicate with the board's components. Information can either be sent as an output from the microcontroller to the components, or received by the microcontroller as an input from a component.

Each device on the board is connected to both to the power source and to one or more of the pins (also known as ‘ports’) on the microcontroller chip. This allows the microcontroller to communicate with the components. It is important to know which component is connected to which pin, as we will need to use this information when we are programming.

The table below outlines which onboard devices are connected to which pins.

PIN	DEVICE
A0	Rotary Potentiometer
A2	Sound Sensor
A6	Light Sensor
D3	Temperature and Humidity Sensor
D4	LED
D5	Buzzer
D6	Button
I2C (19h)	Three-Axis Accelerator
I2C (77h)	Air Pressure Sensor
I2C (3Ch)	OLED Screen

The XOD IDE

The XOD integrated development environment (IDE) is a free open-source software that allows you to programme microcontroller-based devices, such as Grove or Arduino boards, using visual ‘nodes’ rather than written code.

1

YOUR PATCH

The central part of your screen displays the XOD patch you currently have open. A patch is a space for you to create your programme. It is like a file in another programme, and can be used to create one small programme, or one section of a larger programme. You can create multiple patches and store them together in a project.

When you first open the XOD IDE You will see a project called ‘welcome-to-xod’. This is a pre-installed tutorial from XOD. You can create a new project by navigating to ‘File > New Project’ in the menu bar.

2

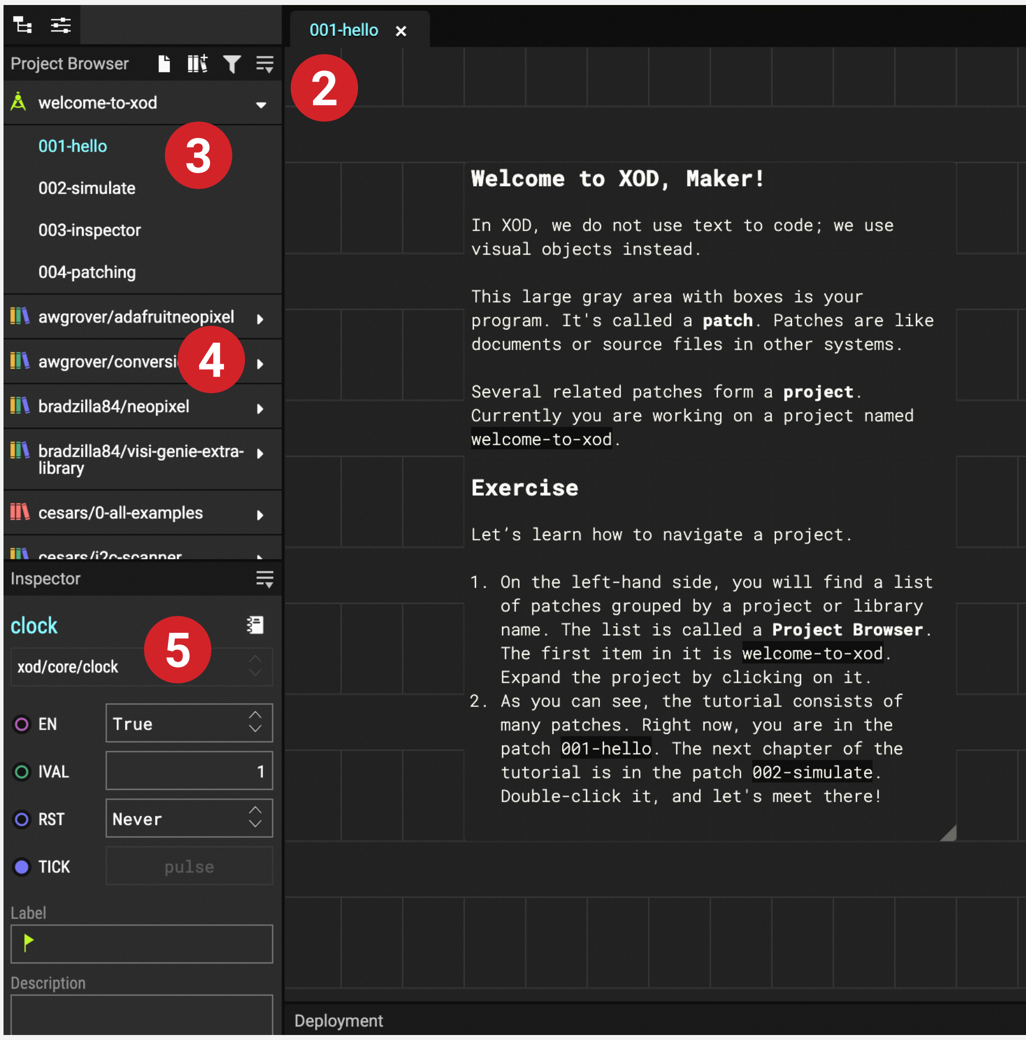
The four buttons to the left of the Project Browser represent (from left-to-right):

ADD PATCH
Lets you add a new patch to your project.

ADD LIBRARY
Lets you install new libraries. You can download libraries that other users have made to expand the number of nodes available.

FILTER
Lets you filter what libraries and nodes you can see.

PROJECT BROWSER MENU
Lets you minimise or move the Project Browser pane.



Welcome to XOD, Maker!

In XOD, we do not use text to code; we use visual objects instead.

This large gray area with boxes is your program. It's called a **patch**. Patches are like documents or source files in other systems.

Several related patches form a **project**. Currently you are working on a project named `welcome-to-xod`.

Exercise

Let's learn how to navigate a project.

1. On the left-hand side, you will find a list of patches grouped by a project or library name. The list is called a **Project Browser**. The first item in it is `welcome-to-xod`. Expand the project by clicking on it.
2. As you can see, the tutorial consists of many patches. Right now, you are in the patch `001-hello`. The next chapter of the tutorial is in the patch `002-simulate`. Double-click it, and let's meet there!

6

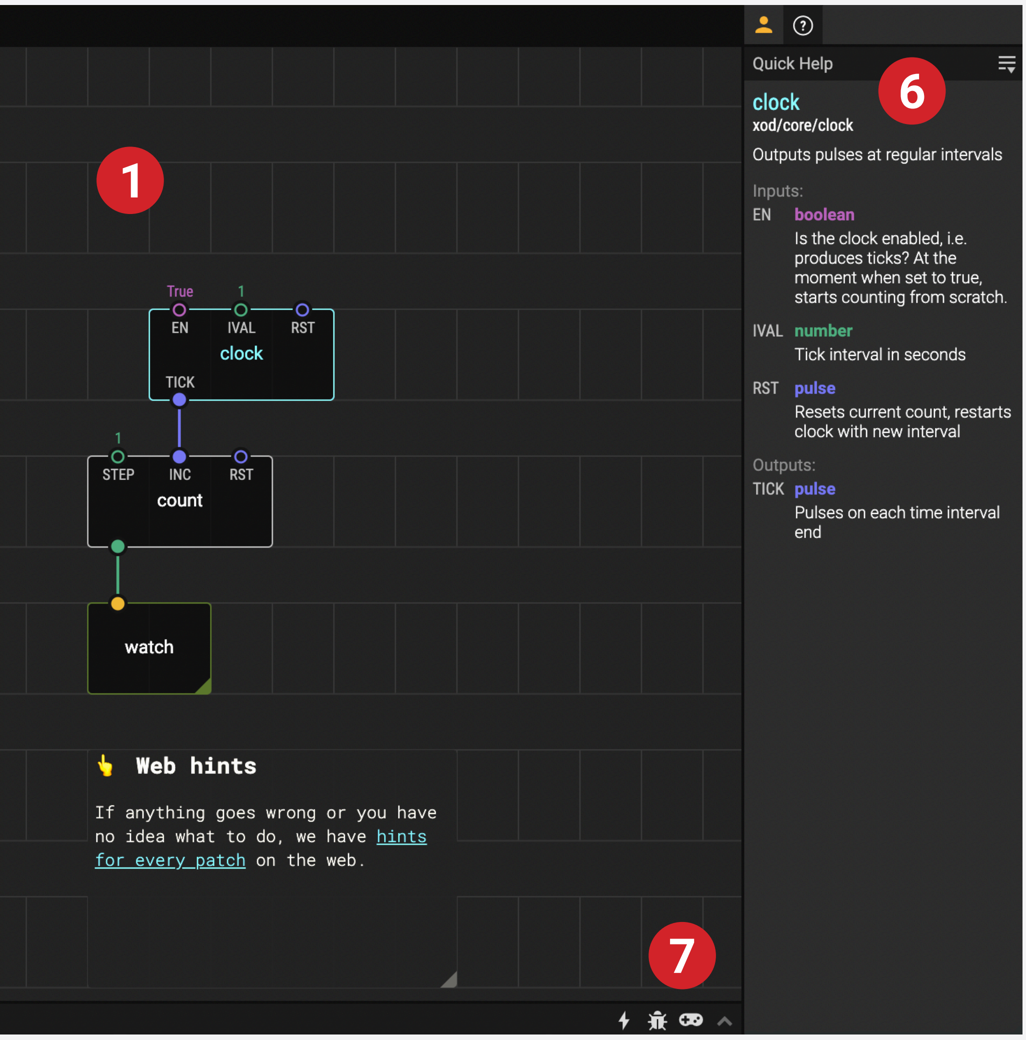
QUICK HELP

The quick help pane provides information about whatever node you have selected at the time. Click on a node to see information about what it does and what each of it’s inputs and outputs (‘pins’) do.

You can toggle this pane on and off using the question mark button at the top right of the screen.

Nodes can represent devices or functions, and by linking them together in different ways you can create a wide variety of different programmes. Programming visually like this can save some of the time and energy required to learn a new language and large amounts of syntax.

Below is a quick summary of the different parts of the XOD IDE, and what you'll see when you first load the software.



clock
xod/core/clock
Outputs pulses at regular intervals

Inputs:
EN boolean
Is the clock enabled, i.e. produces ticks? At the moment when set to true, starts counting from scratch.

IVAL number
Tick interval in seconds

RST pulse
Resets current count, restarts clock with new interval

Outputs:
TICK pulse
Pulses on each time interval end

```
👆 Web hints

If anything goes wrong or you have
no idea what to do, we have hints
for every patch on the web.
```

3

PROJECT BROWSER: PROJECT PATCHES

The top half of the Project Browser pane shows the project you have open. If you click the drop down arrow next to the project name you will be able to see all of the patches within your project.

4

PROJECT BROWSER: LIBRARIES

The bottom half of the Project Browser pane shows all of the libraries you have installed. When you first use XOD these will be limited to the basic XOD libraries (e.g. xod/bits, xod/core). Libraries with red icons have an error somewhere in their patches.

5

INSPECTOR

The Inspector pane shows information about the node or patch that you have selected at the time. This is where you will input information and change the properties of your nodes (e.g. you can tell the programme which of the microcontroller's ports your component is connected to).

This pane also contains the Label and Description boxes, which you can use to help document your programmes

7

The four buttons at the bottom of the patch represent (from left-to-right):

UPLOAD TO ARDUINO
Upload your patch to the board.

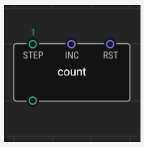
UPLOAD AND DEBUG
Upload the patch, and watch what's happening on your screen at the same time. Useful for testing programmes and debugging.

SIMULATE
Simulate your programme without hardware. Useful for getting started.

TOGGLE DEPLOYMENT PANE
Toggle the Deployment pane on and off to see how the upload is working.

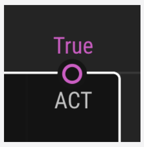
The XOD IDE

XOD Terminology



Nodes

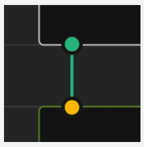
Nodes are the building blocks of a programme in XOD. Depicted as a black rectangle with white border, they will display their name in the middle, and have a number of small circular inputs and outputs on the the top and bottom, known as pins. Nodes can represent any number of things, from hardware (like a LED or sensor) to mathematical or logical operations (like add, subtract, and, or, if etc.).



Pins

In XOD, 'pin' refers to the inputs and outputs of a node, which are represented as small circles. Input pins are located on the top edge of a node, and output pins are located on the bottom edge of a node. Pins can have different data types (see next page) and are coloured accordingly. The name of a pin will appear below the circle, and the current value of a pin will appear above it.

Note: to avoid confusion with the 'pins' on the microcontroller, XOD refers to these as 'ports', i.e. what Grove refers to as 'pin A0' XOD refers to a 'port A0'.

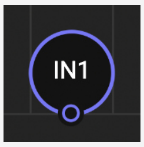


Links

Links are used to connect nodes. To create a link click on an output pin from one node, and then on an input pin from another node (or visa versa). Once a pin is linked, the circle will change to a solid colour. Pins of one data type can only be connected to certain other types (e.g. a number pin can be connected to another number pin, a string pin, or a boolean pin, but not a byte, port or pulse pin). XOD will not let you connect pin types that do not work together.

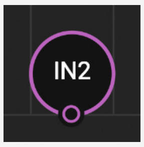
XOD Data Types

Pins in XOD can take a number of different data types depending on what they represent. Each data type is represented by a distinct colour. Below is a description of the six main data types in XOD. Custom data types are also available, but we will not discuss these here. A comprehensive explanation of each XOD data type and how they interact can be found at **www.xod.io/docs/guide/data-types** and **www.xod.io/docs/reference/data-types** respectively.



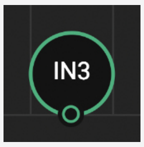
Pulse

Pulse pins are like triggers. They don't represent any specific type of data, but they are used to signify when something has happened, or to trigger something to happen at a specific time.



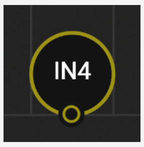
Boolean

Boolean pins represent logical information, i.e. they can only be in two states: True (on) or False (off). They can be used like a switch, and are often used when working with logic nodes such as 'and' 'or' etc.



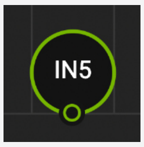
Number

Number pins are very simple: they represent numbers. Numbers can be integers or fractions in the range ±16 millions, and can display up to 6 significant digits.



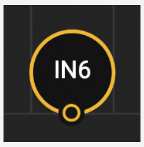
Byte

Byte pins represent bytes, a fundamental data type in computing. They can be written in a variety of ways, but we will use hexadecimals, which contains two digits (0-9,A-F) followed by h-suffix (e.g. 3Ch).



Port

Port pins represent the different 'ports' (microcontroller pins) on the board, i.e. A0-A6 and D0-D13. We will use these pins to tell hardware nodes which pin/port the hardware is connected to.



String

Strings pins represent strings of text. They are used to input text or read out text. Usually this is text input by or to be read by the user, e.g. text to be displayed on a screen.

Lesson 2: Getting Started

Setting up Your Board

Testing Your Board

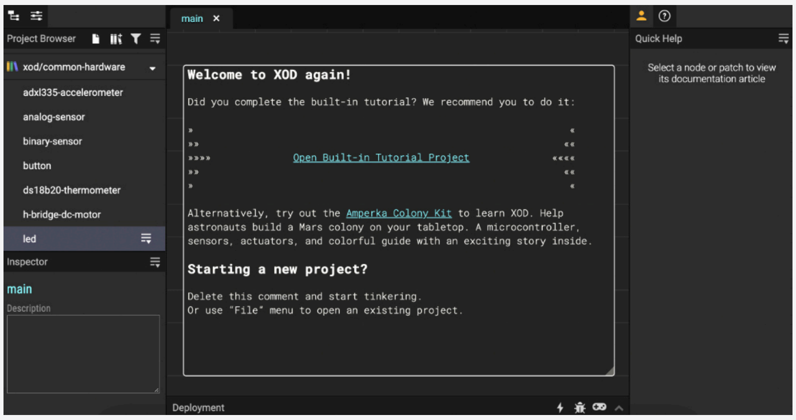
Input and Output Devices

Getting Started

This chapter is all about how to get started with no-code programming and using your board.

Beginning with ‘Setting up your Board’, you will learn how to set up your computer, including how to download the XOD IDE, USB drivers and Biomaker tutorial files.

This is followed by two tasks: ‘Testing your Board’ and ‘Input and Output Devices’. In Task 1 you will use the XOD IDE to test your connection and programme the simple LED on your board. In Task 2 you will learn how to build a simple device using the button and buzzer modules on your board, as well as expanding your knowledge of how to use XOD.



```
Welcome to XOD again!

Did you complete the built-in tutorial? We recommend you to do it:

»                                                    «
»»                                                  ««
»»»»          Open Built-in Tutorial Project      ««««
»»                                                  ««
»                                                    «

Alternatively, try out the Amperka Colony Kit to learn XOD. Help
astronauts build a Mars colony on your tabletop. A microcontroller,
sensors, actuators, and colorful guide with an exciting story inside.

Starting a new project?

Delete this comment and start tinkering.
Or use "File" menu to open an existing project.
```

The XOD welcome screen

OBJECTIVES

By the end of this chapter you should be able to:

- Prepare your Biomaker starter kit by downloading the relevant software and drivers, plugging in your board, and opening the XOD IDE.
- Name the different sections of the XOD IDE and understand what they are used for.
- Apply your knowledge of the XOD IDE to perform the following simple tasks: add a node, change pin settings, connect nodes, add a library.
- Use the XOD IDE to upload programmes to your board.
- Use the XOD IDE to clear programmes from your board.
- Use three of the inbuilt components on the Grove board: the LED, the buzzer and the button.
- Understand how to troubleshoot your programme and find additional help.
- Understand how input and output devices can be used together to build simple devices.

Setting up your Board

Downloads

XOD

To download the free XOD software, simply visit www.xod.io and download the desktop IDE from the XOD homepage. You will need to download the correct IDE for your operating system (Windows, MacOS, Linux etc.).

Note that a browser-based IDE is also available, but does not support hardware, so is not suitable for use with this guide.

USB DRIVER

For your computer to communicate with your Arduino board it will need to have the correct driver installed, in this case, a CP210 driver. Most operating systems will already have the correct drivers installed, including: Windows 7 and 10, Mac OSX v10.10.5 (Yosemite) to v10.15.5 (Catalina), Linux and Ubuntu v18.04.2, 64-bit.

If you are using one of the above systems we suggest ignoring this step and continuing with the guide.

If you are using a different operating system, or are having trouble connecting with your board, you may need to download a CP210 driver. Downloads for most common operating systems are available from Silabs at: www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers.

TUTORIALS

To accompany this guide, the Biomaker team has created a XOD tutorial file. This file will allow you to work through the tasks in this guide within the XOD environment.

You can choose to work through the tasks by using the step-by-step instructions in this book, by using the XOD tutorial file, or by using a combination of both.

If you chose to use the XOD tutorial file, we advise that you take the time to read though the introduction and information at the start of each chapter in this guide, otherwise you may miss out on useful information.

You can download the XOD tutorial file on the Biomaker website at: www.biomaker.org/nocode-programming-for-biology-handbook.

Testing your Board

Task 1: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (LED and Button modules)
- USB-A to micro USB cable

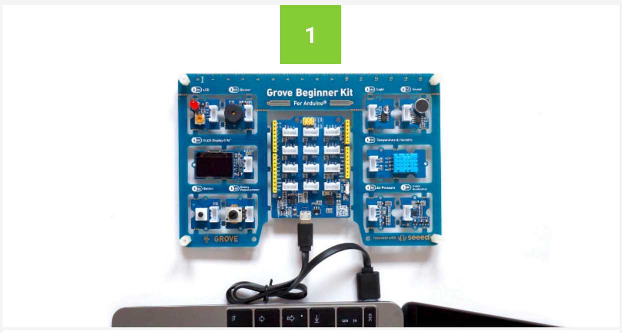
Now that you’ve downloaded the software you’re all set and ready to get stuck in!

This task will walk you through how to connect your board to the computer and upload your first programme using XOD.

We’ll be using the LED light in the top left corner of the board to test our connection.

You may notice that when you first plug in the board the OLED screen in the middle-left turns on. This is part of the inbuilt demo mode on the board. You can learn more about this in the Grove User Manual.

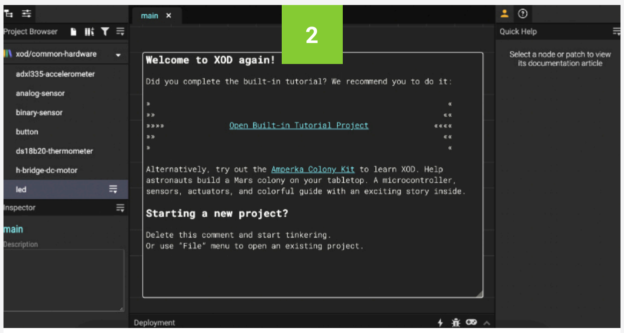
1



PLUG IN YOUR BOARD

Use the USB cable provided to plug your board into the computer. You can find this cable in the right-hand compartment of the Grove box. Plug the micro USB end into the socket at the bottom of the central section of the board, and the standard USB (USB-A) end into your computer.

2

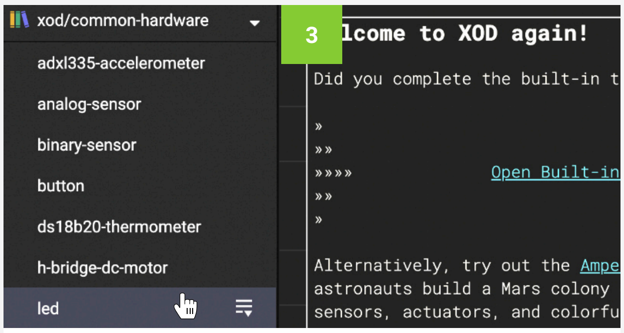


```
Welcome to XOD again!

Did you complete the built-in tutorial? We recommend you to do it:

»                                                         «
»»                                                       ««
»»»»             Open Built-in Tutorial Project        ««««
»»                                                       ««
»                                                         «

Alternatively, try out the Amperka Colony Kit to learn XOD. Help
astronauts build a Mars colony on your tabletop. A microcontroller,
sensors, actuators, and colorful guide with an exciting story inside.

Starting a new project?

Delete this comment and start tinkering.
Or use "File" menu to open an existing project.
```

OPEN THE XOD IDE

Open up the XOD software on your computer.

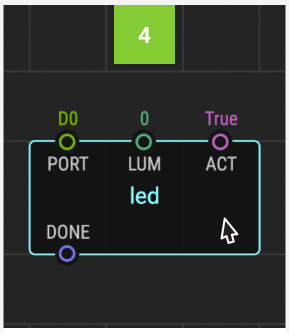
If you are using the tutorial file provided, open this file in XOD. You can follow the instructions in patches *tuto101-tuto114* to complete this task.



```
lcome to XOD again!
Did you complete the built-in t
»
»»
»»»»            Open Built-in
»»
»
Alternatively, try out the Ampe
astronauts build a Mars colony
sensors, actuators, and colorfu
```

ADD AN LED NODE

Using the Project Browser on the top left, find the *xod/ common-hardware* library, click the dropdown menu and find the node called "*led*". Click on this node and drag it into the patch. This is one of several ways to add a new node. See the Adding Nodes to your Patch box for more information on alternative methods.



SELECT THE NODE

Click on the *led* node. The outer edge will turn blue and more information will appear in the Inspector pane on the bottom left.

Adding Nodes to your Patch

There are several different ways to add a node to your XOD patch, and which one you use is completely up to you!

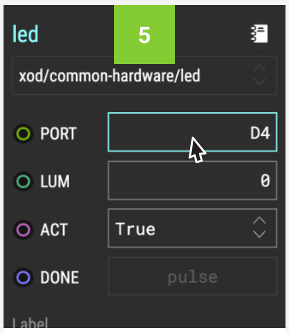
1. **DRAG FROM LIBRARIES**
   As described above. If you know the library the node is in, you can find the library in the Project Browser, click the dropdown menu, click on the node, drag it into the patch and release.

2. **DOUBLE CLICK ON THE PATCH**
   If you know the name of the node you want, or want to search for a node you can use the search bar. Double click anywhere in your patch and the search bar will appear. Start typing the name of the node and options will appear. Click on the correct node and it will insert into your patch.

3. **KEYBOARD SHORTCUT**
   Similar to double-clicking the patch. Click anywhere on the patch and press 'i' on your keyboard. This will bring up the same search bar as above.

4. **MENU BAR**
   This is a third way to bring up the search bar. Select 'Edit > Insert Node...' from the menu bar.

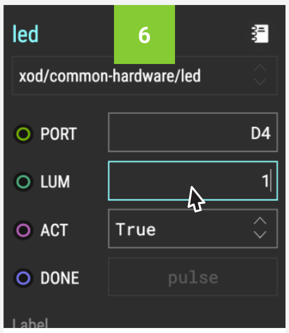
Testing your Board



led
xod/common-hardware/led
PORT D4
LUM 0
ACT True
DONE pulse
Label

SET PORT PIN

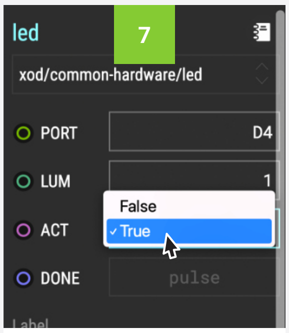
The LED on the board is connected to port D4, so click on the text box next to PORT and set this to D4 by typing 'D4'.



led
xod/common-hardware/led
PORT D4
LUM 1
ACT True
DONE pulse
Label

SET LUM PIN

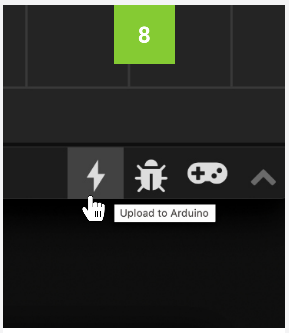
LUM stands for luminance, i.e. how bright the LED is on a scale of 0-1. Set this to 1 (brightest level) by typing '1'.



led
xod/common-hardware/led
PORT D4
LUM 1
ACT False / ✓ True
DONE pulse
Label

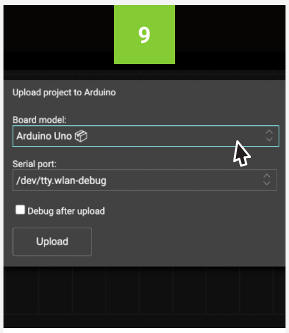
SET ACT PIN

ACT is a boolean pin that can only be true or false. Use the dropdown to set this to 'True'. This makes sure the LED responds.



UPLOAD

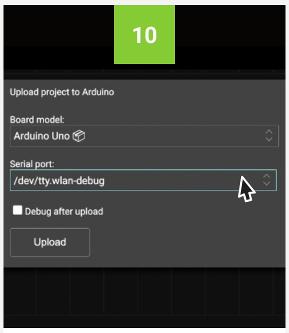
Click on the small lightening icon in the bottom right, or select 'Deploy > Upload to Arduino...' from the menu bar.



Upload project to Arduino
Board model: Arduino Uno
Serial port: /dev/tty.wlan-debug
Debug after upload
Upload

SET BOARD MODEL

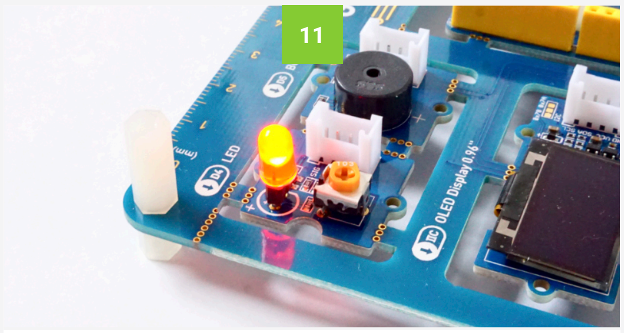
Use the dropdown menu to select 'Arduino Uno' or 'Arduino/Genuino Uno'.



Upload project to Arduino
Board model: Arduino Uno
Serial port: /dev/tty.wlan-debug
Debug after upload
Upload

SET SERIAL PORT

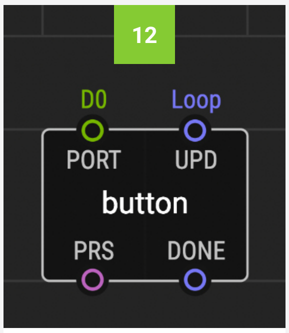
Use the dropdown menu to select the option that ends in '(Silicon Labs)'.



WATCH YOUR LED!

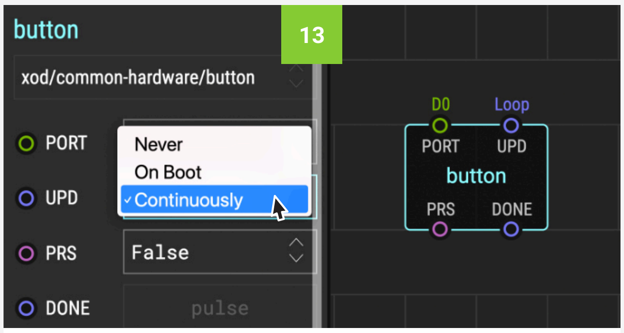
Click upload and watch the LED on your board. It should light up!

If not, don't worry! Take a look at the Troubleshooting box on the next page (p25).



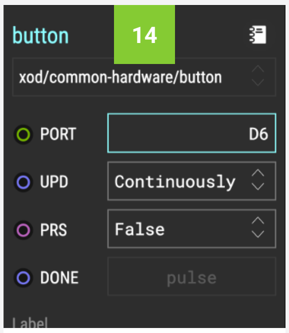
ADD A BUTTON NODE

Now let’s add another node. Using one of the ways described on p21, add a *button* node from the *xod/common-hardware* library.



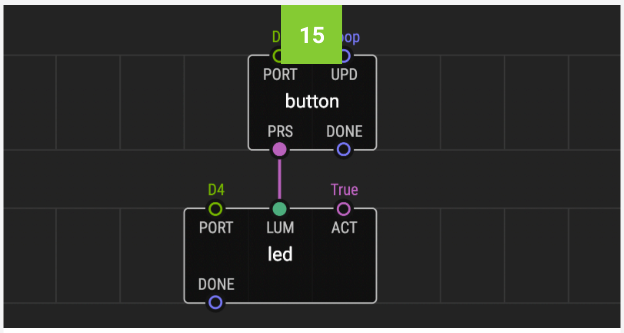
SET BUTTON PINS:
UPD PIN

The UPD pin specifies how often the programme updates. This can be set to ‘Never’, ‘On Boot (Boot)’, or ‘Continuously (Loop)’. Alternatively another node can be connected to this pin and used to determine how often it updates. Make sure this is set to ‘Continuously (Loop)’, so that whenever we press the button it is read instantly.



SET BUTTON PINS:
PORT PIN

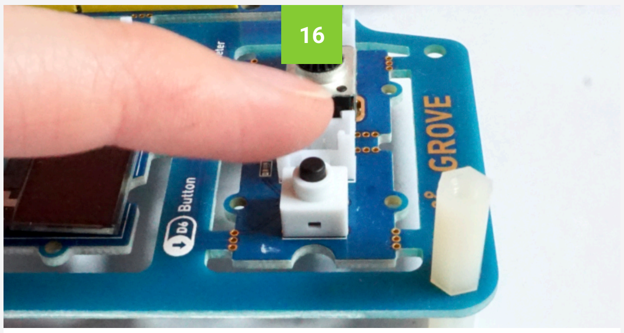
As with the LED node, PORT specifies which port the button is connected to. Set this to ‘D6’.



CONNECT THE NODES

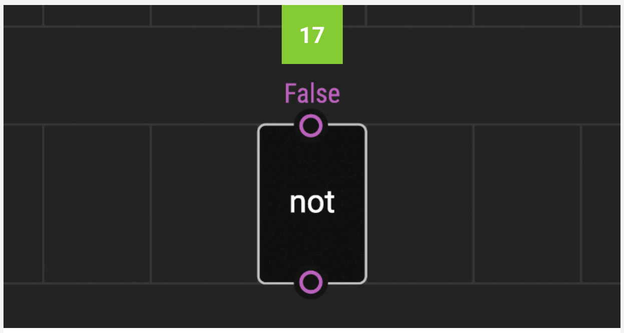
We want the LED to turn on whenever we press the button, so we need to connect the *button* output pin PRS (press) to the *led* input pin LUM. Do this by clicking on the PRS pin and then on the LUM pin. Now when you click on the *led* node you will not be able to set the LUM pin, because it’s value is determined by *button* node.

Testing your Board



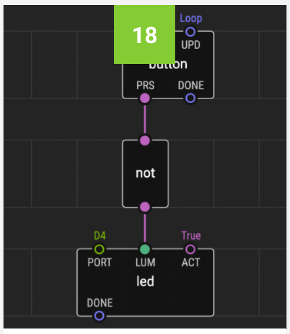
UPLOAD AND TEST

Upload the patch and see what happens. You will notice that the programme is backwards. The LED is on and turns off when you press it. This is because the board’s buttons are set to be on by default, and turn off when pressed. We can fix this programme with a logic node.



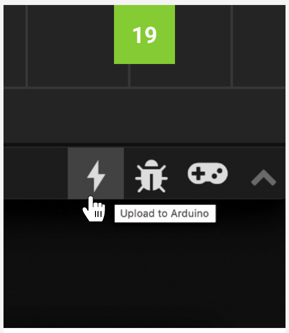
ADD A NOT NODE

To invert the signal from the button we can use a different type of node that represent a logic function, rather than a piece of hardware. Insert a *not* node from the *xod/core* library.



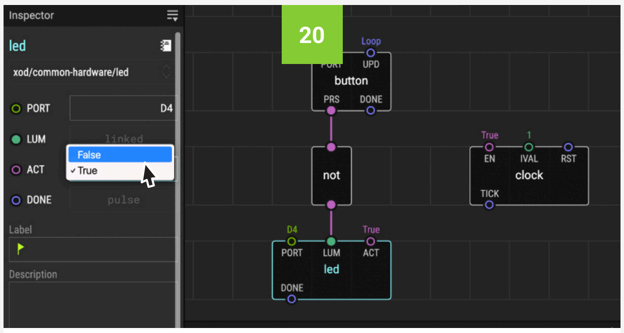
REWIRE THE PATCH

Delete the link between the *button* and *led*. Connect PRS to the *not* input pin, and the *not* output pin to LUM.



UPLOAD AND TEST

Now try uploading your programme again. This time it should work as planned.

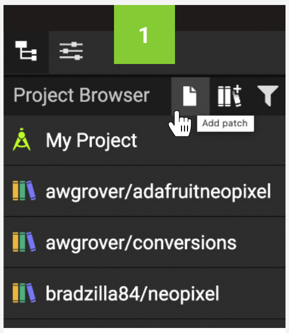


EXPERIMENT!

Congrats! you've now made a simple programme that uses an input (the button) and an output (the led) to affect change. Why not try experimenting with this patch? Play around with some pins. E.g. change the led ACT pin, or link a clock node to the button UPD pin and see what happens. See what you can achieve!

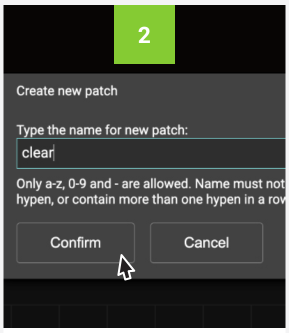
Clearing the Board

Once a programme is loaded onto your board it will remain there and restart whenever you turn on the board. Each time you upload a new programme it will write over the previous programme. You don't need to clear the board before you upload a new programme, but if you wish to reset your board you can do this manually by uploading a blank patch in XOD.



MAKE A NEW PATCH

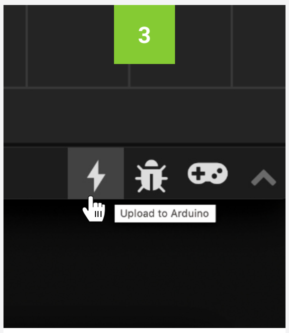
Add a new patch by clicking the ‘Add patch’ button in the project browser or selecting ‘File > New Patch...’ from the menu.



Type the name for new patch:
clear
Only a-z, 0-9 and - are allowed. Name must not
hypen, or contain more than one hypen in a row

NAME THE PATCH

Type a name for your new patch, e.g. ‘clear’ and click confirm or press the Enter key.



UPLOAD

Upload the empty patch by clicking the upload button as before. This will turn off the LED and clear the board.

Troubleshooting

If your LED doesn’t light up straight away there a few quick things to check:

1. **IS THE BOARD PLUGGED IN CORRECTLY?**
   If your board is plugged in correctly the power light on the right side of the Seeeduino module should light up. If not, make sure that the USB cable is plugged in fully.

2. **HAVE YOU SET YOUR NODE PARAMETERS CORRECTLY?**
   Setting the wrong parameters is a common mistake. In this case you should make sure the pins are set as follows:
   PORT = D4   LUM = 1   ACT = True

3. **HAVE YOU UPLOADED USING THE RIGHT BOARD MODEL AND SERIAL PORT?**
   After clicking the upload button you should make sure that you have the correct board model and serial port selected. Use the dropdown menus to select ‘Arduino Uno’/’Arduino/Genuino Uno’ and ‘dev/tty.usbserial-0001 (Silicon Labs)’.

4. **DO YOU NEED TO INSTALL A USB DRIVER?**
   If XOD is not recognising your board, you may need to install a CP210 USB driver (see page 15).

Still need help? XOD provides *watch* and *tweak* nodes which are useful for troubleshooting and debugging your programme. Read more about them on page 27 of this guide, or take a look at XOD’s guide to debugging at **www.xod.io/docs/guide/debugging**. For further help you can always contact the Biomaker team at synbio@hermes.cam.ac.uk.

Input and Output Devices

Task 2: Requirements

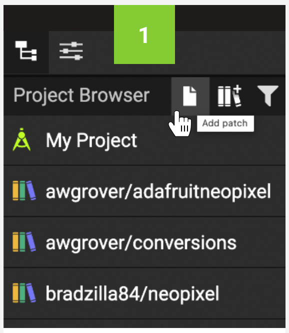
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (buzzer , button and rotary potentiometer modules)
- USB-A to micro USB cable

Great! You now understand the basic principles of using XOD, and can programme your board to control one of the onboard devices: the LED.

Like the button and the LED, most devices you will use can be grouped into two general categories: inputs and outputs.

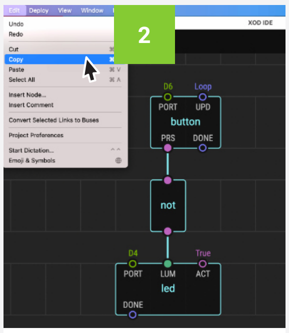
Understanding how to control different inputs and outputs, and how to combine them together is key to making useful devices.

In this task we will build on our knowledge to add two new devices to our belt. The buzzer and the rotary potentiometer (also known as a knob sensor).



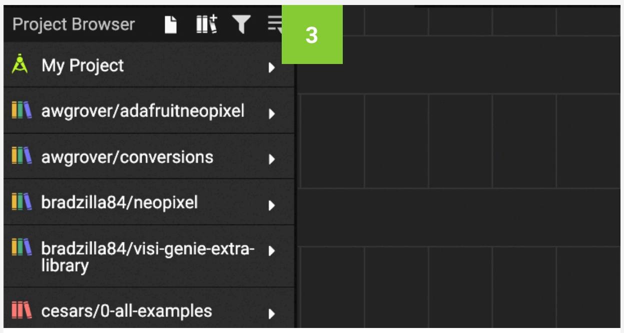
MAKE A NEW PATCH

Follow the instructions in 'Clearing the Board' to open and name a new patch. (If you are using the tutorial file, move on to tuto201.)



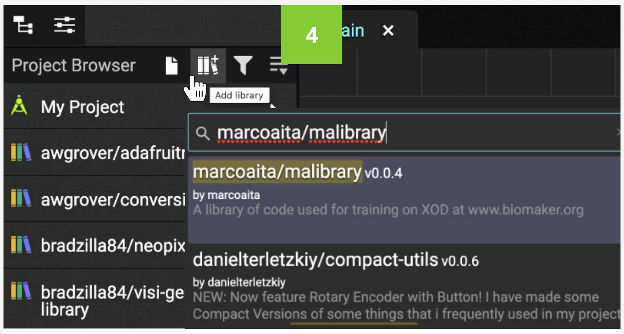
COPY YOUR PATCH

Drag a box around all three nodes in the last patch. Copy and paste into your new patch. This will save us some work!



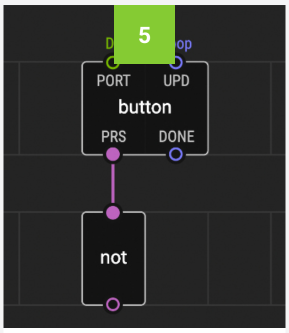
FINDING A BUZZER NODE

There is no preinstalled node to represent the buzzer, but several have been created by members of the XOD community. We will use one from the library called *marcoaita/malibrary*.


4
Project Browser
Add library
My Project
awgrover/adafruitr
awgrover/conversi
bradzilla84/neopix
bradzilla84/visi-ge library
ain ×
marcoaita/malibrary
marcoaita/malibrary v0.0.4
by marcoaita
A library of code used for training on XOD at www.biomaker.org
danielterletzkiy/compact-utils v0.0.6
by danielterletzkiy
NEW: Now feature Rotary Encoder with Button! I have made some
Compact Versions of some things that i frequently used in my project

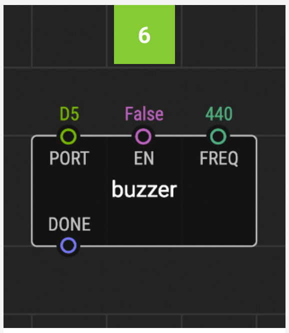
ADD A LIBRARY

Add this library by clicking on the ‘Add library’ button (next to the ‘New patch’ button at the top of the Project Browser) or by navigating to ‘File > Add Library...’. Type the full name of the library, and when it appears, click on it to install. If you get an error message asking to install dependencies, accept this.



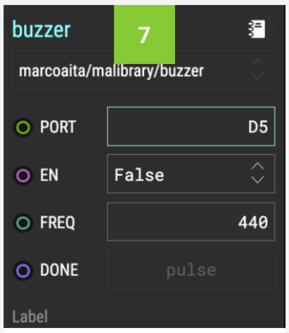
DELETE LED NODE

This time we want to use the buzzer as an output instead of the LED. Click on the led node and delete it.



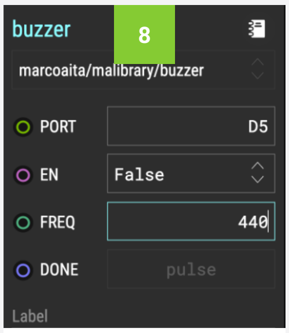
ADD A BUZZER NODE

Now that the library is installed you can search for the *marcoaita/malibrary/ buzzer* node and add it as usual.



SET BUZZER PINS: PORT

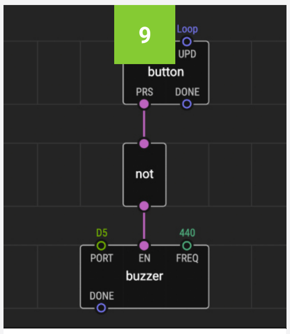
The buzzer is connected to port D5, so make sure the PORT pin is set to ‘D5’



SET BUZZER PINS: FREQ

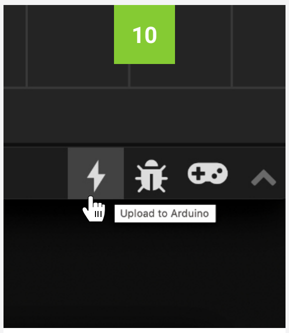
The FREQ pin sets the frequency and pitch of the buzzer. You can leave this as 440, or change it too see what happens.

Input and Output Devices



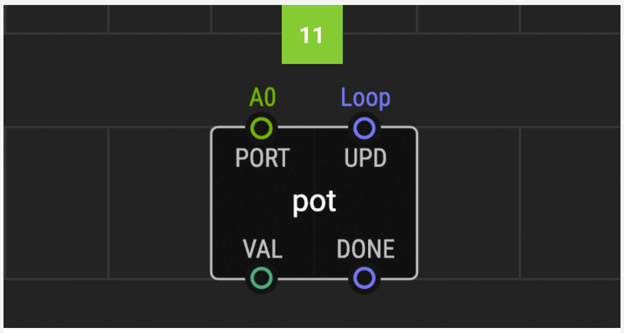
RECONNECT THE NODES

Connect the *not* output pin to the buzzer EN (enabled) pin. This will 'enable' the buzzer when the button is pressed.



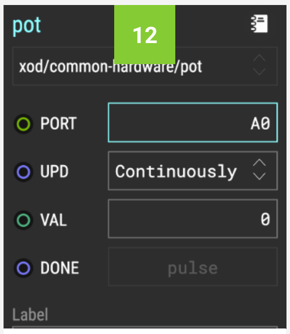
UPLOAD AND TEST

Now try uploading your programme. It should work similarly to the LED patch, i.e. the buzzer turns on when you press the button.



ADD A POT NODE

Now let's add a second input. We can use the inbuilt rotary potentiometer (knob) to adjust the frequency of the buzzer sound. To represent the potentiometer we can use the *pot* node from the *xod/common-hardware* library. Add a *pot* node to the patch.

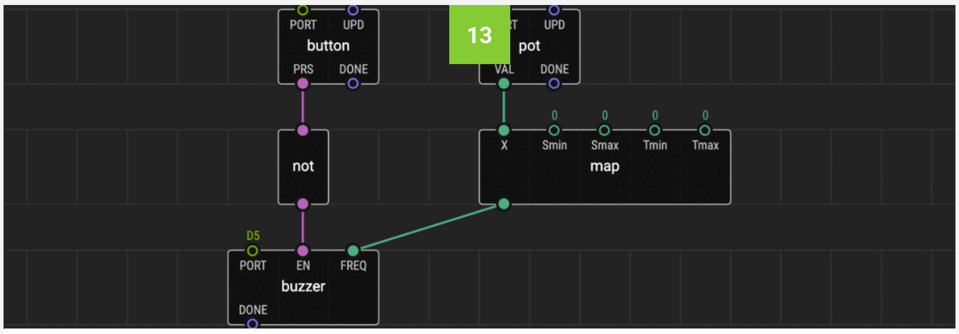


xod/common-hardware/pot

PORT A0
UPD Continuously
VAL 0
DONE pulse
Label

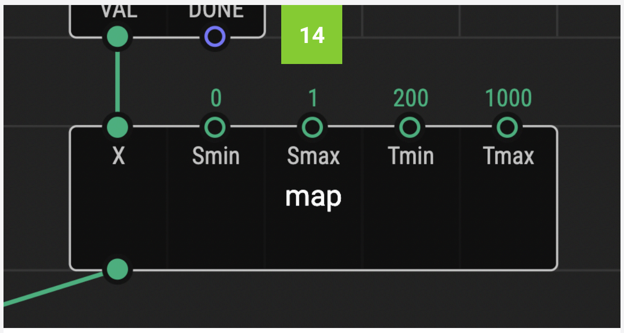
SET POT PINS

The potentiometer is connected to port A0, so set PORT to A0. Set UPD to 'Continuously'.



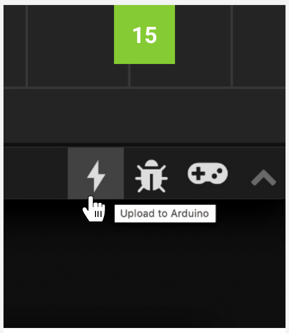
MAPPING VALUES: ADD AND CONNECT A MAP NODE

We could connect the *pot* output VAL (value) pin straight to the FREQ input pin. However, this wouldn't work well, as the VAL output ranges between 0 and 1, and frequencies emitted by the buzzer are much higher. To get around this, we can add a *map* node. This will 'map' your input range to a new output range, so we can change the 0-1 scale of the potentiometer to a larger scale that the buzzer can use. Add a *map* node from *xod/math*. Connect the *pot* VAL pin to the *map* X pin and the *map* output pin to the *buzzer* FREQ pin.



SET MAP PINS

Smin and Smax set the source range, whilst Tmin and Tmax set the target range. Set Smin to ‘0’ and Smax to ‘1’. Set Tmin to ‘200’ and Tmax to ‘1000’.



UPLOAD AND TEST

Now try uploading your programme. Use the button to turn the buzzer on and off, and the potentiometer to set the frequency.

Building Complex Devices

When building biological devices, you will need to combine a variety of inputs and outputs to create a functioning programme and device. You will often receive inputs, e.g. from an on/off button or sensor, and then use these inputs to create the desired output, e.g. displaying a reading, sending data to a computer or moving a motor.

The button buzzer and potentiometer example used here is a very simple example, but the principle applies in more complex systems too. Using XOD allows you to visualise this information flow from input to output, which can be helpful and sometimes more intuitive than traditional text-based coding.

In the next lesson we’ll be getting a better understanding of what is possible in XOD by exploring a variety of useful nodes and processes using a range of the board’s inbuilt devices.

When you are ready to explore beyond the starter kit’s capabilities, the Resources tab of the Biomaker website explores a variety of common input and output devices which are useful for building biological devices.

Lesson 3: Explore XOD

Tweak and Watch Nodes

Flip, Clock and Count Nodes

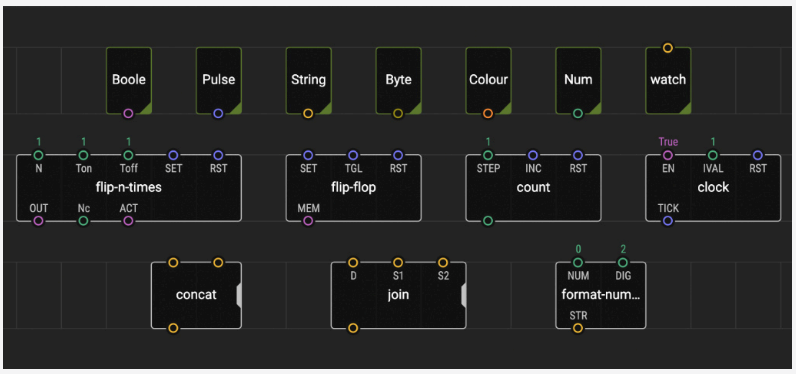
Concat, Join and Format-Number Nodes

Explore XOD

This chapter will explore some of the most useful nodes XOD has to offer. These nodes are used very commonly when building simple instruments, and will give you a good base to start from when exploring more complex devices.

The sections in this chapter are split into three tasks. First, following on from Task 2 in the previous chapter, Task 3 explores ‘Tweak and Watch Nodes’, which are useful for simulating and troubleshooting. Second, Task 4 examines ‘Flip, Clock and Count Nodes’ which are useful for ensuring correct timing of programmes. Finally, Task 5 looks at ‘Concat, Join and Format-Number Nodes’ which are useful for using and formatting text in XOD.

This chapter also encourages you to experiment with your use of XOD. It provides suggestions for how to expand the tasks, and encourages you to start thinking about the different ways in which you can achieve a desired outcome using no-code programming.



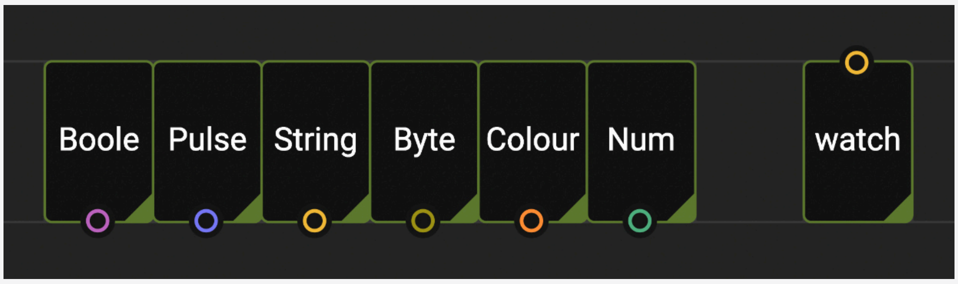
Useful XOD nodes

OBJECTIVES

By the end of this chapter you should be able to:

- Explain the functions of the following XOD nodes: *tweak*, *watch*, *flip-n-times*, *flip-flop*, *clock*, *count*, *concat*, *join*, *format-number*.
- Apply your knowledge of these nodes to start building simple programmes.
- Use the XOD IDE to create and save a new project.
- Use the XOD IDE to ‘upload and debug’ programmes, allowing you to watch and edit your programme live.
- Use two more of the inbuilt components on the Grove board: the temperature and humidity sensor and the air pressure sensor.
- Build and compare different versions of a programme to achieve different functions and outcomes.
- Experiment with the programmes you have built by changing parameters and exploring new nodes
- Understand where to find more information about the basic nodes available in XOD

Tweak and Watch Nodes



TWEAK NODES

The *tweak* nodes provided in XOD are a great way to edit your programmes whilst they are running. They are used in conjunction with the 'Simulate' and 'Upload and Debug' functions of XOD to edit programmes in real time, meaning that you don't have to reload the programme each time you want to make a small change, like altering the value of a pin.

There are multiple *tweak* nodes available depending on what type of pin you would like to change, and you will need to use the matching *tweak* node for the pin type, i.e. *tweak-boolean*, *tweak-pulse*, *tweak-byte*, *tweak-colour*, and *tweak-number*. There are also several *tweak-string* nodes depending on the size of string you want to input, e.g. *tweak-string-16* allows you you input a string of up to 16 characters, whilst *tweak-string-128* allows you to input up to 128 characters.

To use *tweak* nodes you will need to use the 'Upload and Debug' button rather than the 'Upload to Arduino' button, as this opens XOD's 'Debugger' function, which lets you live edit nodes. To edit a *tweak* node in the Debugger, click on the node and you will now be able to make changes in the Inspector whilst the programme is running.

WATCH NODE

The *watch* node is the opposite of a *tweak* node. Instead of letting you input data, it lets you view output data whilst the programme is running. Connecting a *watch* node to any output pin lets you view the current value of that pin. This is useful for being able to visualise what the programme is doing, and where any problems are occurring.

A *watch* node can be connected to any output except a pulse output. To visualise the output from a pulse pin, you can use a *count* node in combination with a *watch* node. This is discussed further on page 41.

Like *tweak* nodes, *watch* nodes need to be used in conjunction with the Debugger function. When the Debugger opens, the *watch* node will turn green and display the last value received from the connected pin.

Tweak and Watch Nodes

Task 3: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (temperature and humidity sensor module)
- USB-A to micro USB cable

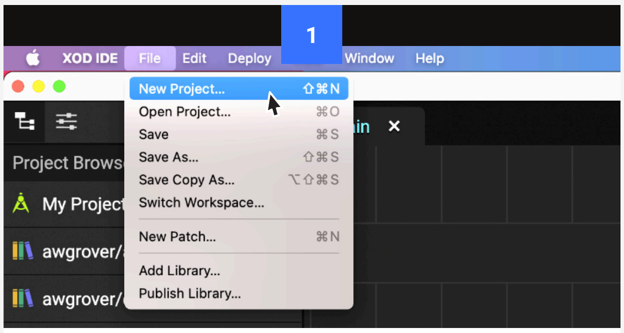
In this task we'll look at how we can use *tweak* and *watch* nodes to take readings from another of the inbuilt devices: the temperature and humidity sensor.

We'll be using a *tweak-pulse* node to act as a button and take a reading whenever we press (or 'tweak') it, and watch nodes to display our readings on the computer screen.

We'll also be using the 'Simulate/Debug' mode in XOD, which lets us watch and make changes while the code is running.

This is a great example of how *tweak* and *watch* nodes can be used to quickly and easily test a patch. They are very useful for testing and debugging patches, so you should try to get used to using them as you build.

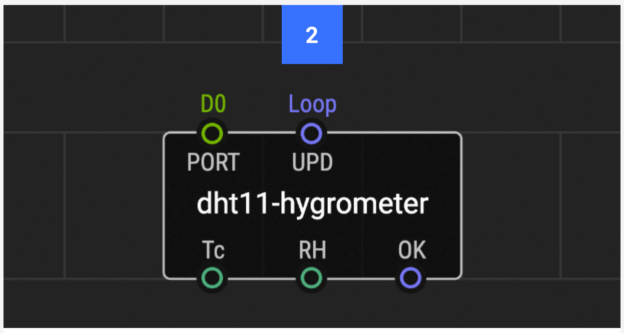
1



MAKE A NEW PROJECT

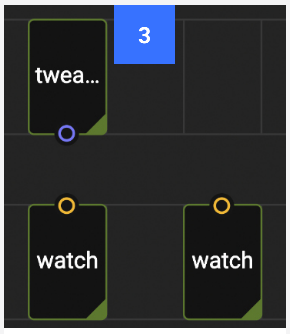
A new chapter deserves a clean slate, so let's look at how we start a new project. First, save your old project (you can't have two projects open at once). Then navigate to 'File > New Project...' in the menu bar. Finally, give your project a name, and you can get started. (If you are using the XOD tutorial file, move on to tuto301).

2



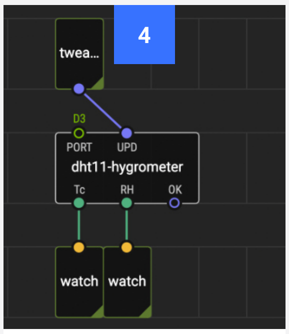
ADD AND SET HYGROMETER NODE

The onboard temperature and humidity sensor is technically known as a DHT11 hygrometer. There is a preinstalled XOD node for this device called *dht11-hygrometer*. Add this node to your patch from the *xod-dev/dht* library. Set the port pin to 'D3'



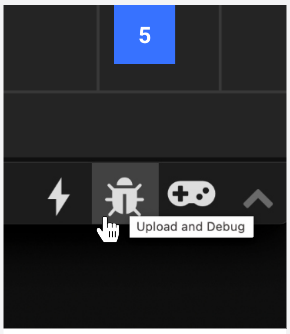
ADD TWEAK AND WATCH NODES

Add a *tweak-pulse* and two *watch* nodes from the *xod/debug* library.



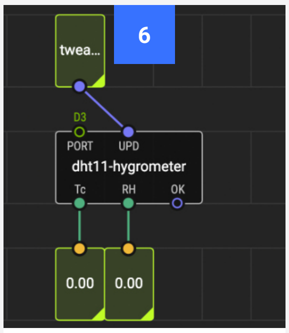
CONNECT THE NODES

Connect the *tweak-pulse* node to the UPD pin and a watch node to each of the pins Tc (temperature ºC) and RH (relative humidity).



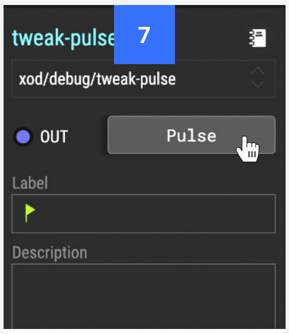
UPLOAD AND DEBUG

Click the 'Upload and Debug' button (ladybird) or use the upload button and tick the box labelled 'Debug after upload'.



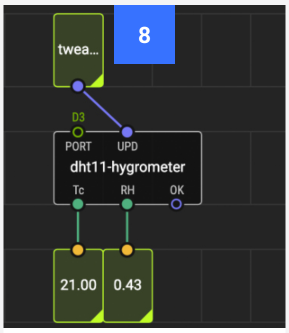
WATCH

Look at the watch node. It won't display a reading yet because we have set the UPD pin to only update when we tweak it.



'TWEAK' THE NODE

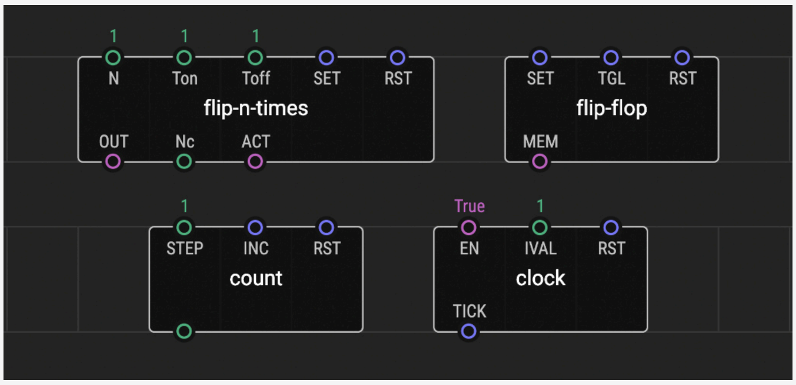
Whilst it is still running, click on the *tweak* node. In the Inspector pane is a button next to OUT that says 'pulse'. Click it.



WATCH AGAIN

Look at the *watch* nodes again. This time they should display the current temperature and humidity.

Flip, Clock and Count Nodes



FLIP NODES

Flip nodes are boolean logic nodes that switch (or 'flip') between two states: 'True' and 'False'. There are two useful flip nodes in XOD: *flip-n-times* and *flip-flop*.

The first node, *flip-n-times*, will switch between 'True' and 'False' a set number of times (N). You can determine the time spent in each state using the Ton and Toff pins, and the whole sequence will be initiated by a pulse to the SET pin. This node is useful for creating sequences and patterns.

The second node, *flip-flop*, will switch between 'True' and 'False' states each time the toggle (TGL) pin receives a pulse. This is a particularly useful node which can act as a toggle or switch in many situations.

CLOCK NODE

Like the *flip* nodes, the *clock* node is also useful for controlling the timing of your programmes. However, instead of giving a boolean 'True'/'False' output, the *clock* node sends pulses at a specific time interval. This node creates a regular 'ticking' of pulses, which can be used to control your programme.

COUNT NODE

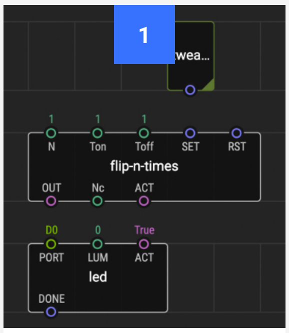
The *count* node is complimentary to the *flip* and *clock* nodes, and acts as a measure of how many times a pulse or boolean 'True' signal has been sent. This is useful for keeping track of your programme and it's progress. The *clock* node is also very useful, when used in conjunction with a *watch* node, to visualise the output from a pulse pin (see **Task 4, Steps 15-16** for an example of this use).

Task 4: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (LED module)
- USB-A to micro USB cable

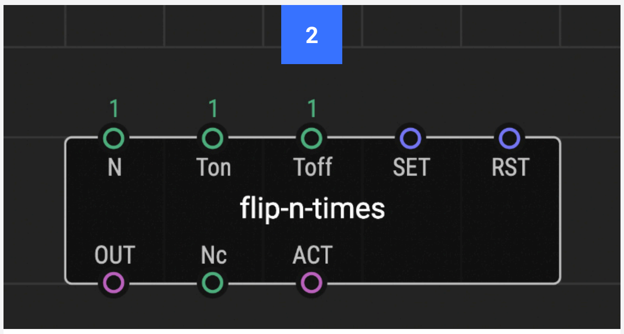
In this task we'll be experimenting with *flip*, *clock* and *count* nodes to control the behaviour of the inbuilt LED, making it flash.

The *flip* and *clock* nodes can be useful for modifying and timing the behaviour of nodes, whilst the *count* node can be useful for monitoring these behaviours. In the context of biological devices, these nodes are very useful for fine-tuning devices and for building larger programmes.



NEW PATCH AND ADD NODES

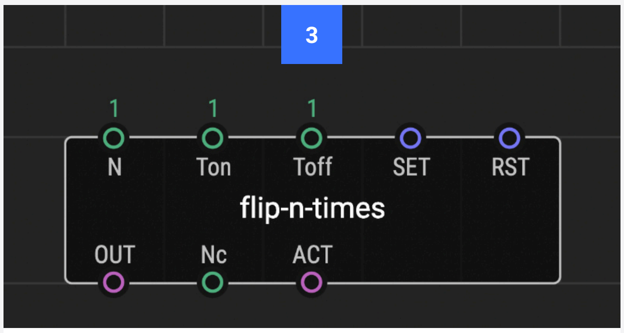
Open a patch (or go to tuto401). Add these nodes: *tweak-pulse* (*xod/debug*), *flip-n-times* (*xod/core*) and *led* (*xod/common-hardware*).



FLIP-N-TIMES NODE INPUTS

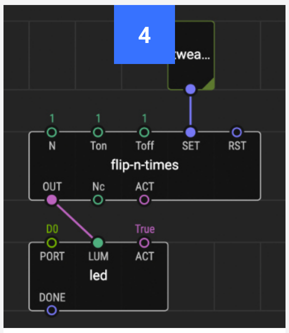
This node has 5 input pins: SET, RST (reset), N (number), Ton (time on) and Toff (time off). This node defines a sequence that will switch between true and false N number of times. Ton and Toff define the duration of each on and off state. A pulse to SET will start the sequence, and a pulse to RST will reset the node.

Flip, Clock and Count Nodes



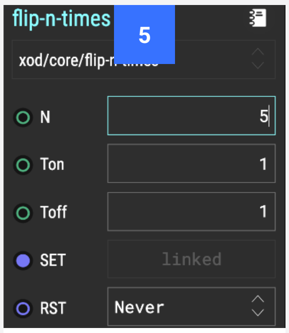
FLIP-N-TIMES NODE OUTPUTS

The *flip-n-times* node has three outputs. OUT reads the current state of the node (true/false). Nc reads the number of times cycled. ACT reads whether the sequence is currently running or not. If you'd like to get a better idea of how these outputs work, you can always add watch nodes, to help see what's going on.



CONNECT THE NODES

Connect the *tweak-pulse* node to the *flip-n-times* SET pin, and *flip-n-times* OUT pin to the *led* LUM pin.

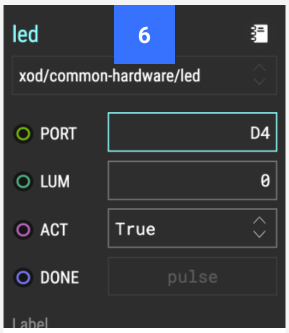


flip-n-times
xod/core/flip-n-times

N	5
Ton	1
Toff	1
SET	linked
RST	Never

SET FLIP-N-TIMES PINS

Set N to '5', Ton and Toff to '1', and RST to 'Never'. You can also add a *tweak-pulse* node to the RST pin to test how it works.



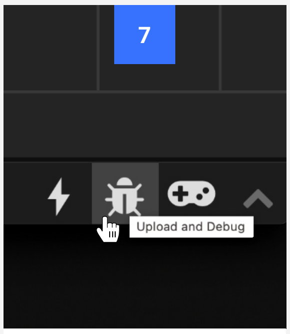
led
xod/common-hardware/led

PORT	D4
LUM	0
ACT	True
DONE	pulse

Label

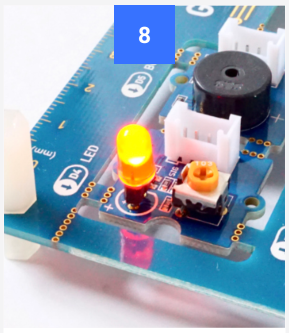
SET LED PINS

As in **Task 1 Steps 5-7** (p22), set the PORT pin to 'D4' and the ACT pin to 'True'.



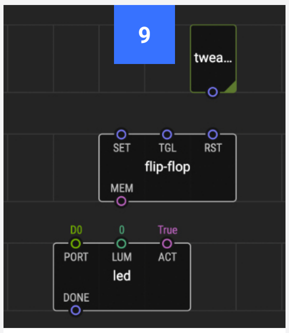
UPLOAD AND DEBUG

Upload and debug the programme using the ladybird button.



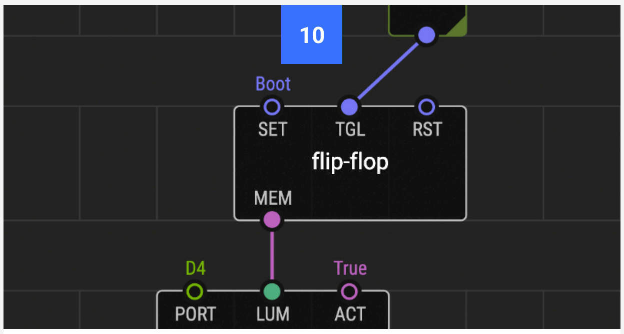
TWEAK AND WATCH

Press the *tweak-pulse* node and watch what happens to the LED. Each time you press, the light should flash 5 times.



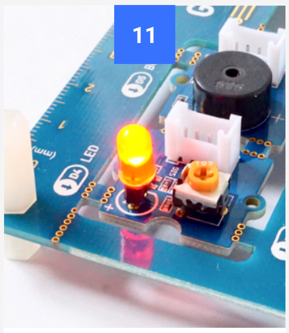
FLIP-FLOP NODE

Now lets try a different flip node. Delete the *flip-n-times* node and add a *flip-flop* node (*xod/core*).



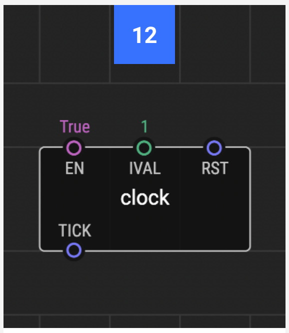
CONNECT AND SET FLIP-FLOP PINS

The *flip-flop* node has 3 input pins: SET, RST and TGL (toggle). TGL switches the node between true and false each time it receives a pulse. Connect the *tweak-pulse* node to TGL. Set SET to ‘On Boot’ and RST to ‘Never’. The MEM (memory) output pin reads out the latest state of the node. Connect this to the LUM pin of the *led*.



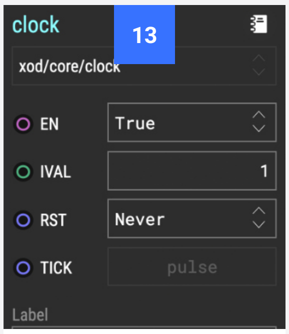
TEST THE PATCH

Upload the program and pulse the *tweak-pulse* node. The LED should switch between on and off each time you press it.



CLOCK NODE

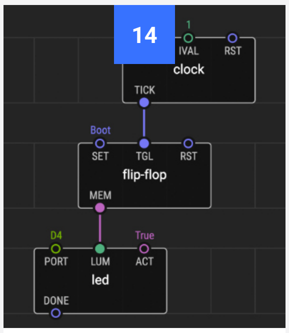
The *flip-flop* node can also flash the LED when combined with a *clock* node. Add a *clock* node (*xod/core*) to the patch.



EN	True
IVAL	1
RST	Never
TICK	pulse

SET CLOCK PINS

Set EN (enabled) to ‘True’ and RST to ‘Never’. IVAL determines how often the clock ticks (in secs). Set this to ‘1’.

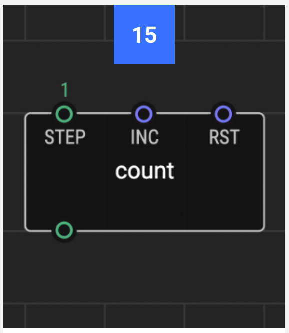


RECONNECT

Delete the *tweak-pulse* node and connect the *clock* node in its place by linking the TICK pin to the TGL pin.

Flip, Clock and Count Nodes

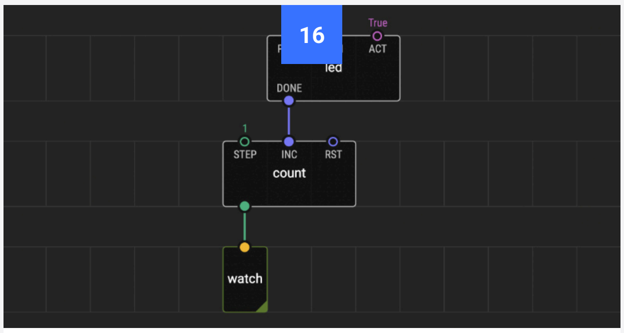
15



COUNT NODE

We’ll also add a *count* node to this patch so that we can monitor the number of times the LED flashes. Add a *count* node (*xod/core*).

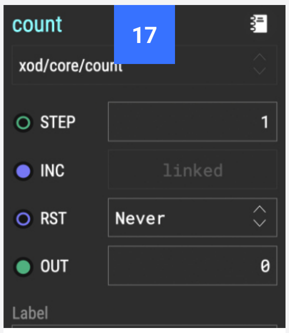
16



CONNECT COUNT NODE

Connect the *led* DONE pin to the *count* INC (increase) pin. The DONE pin pulses each time the LED turns on or off, and the INC pin increases the count each time it is pulsed. Connect the *count* output pin to a *watch* node so we can see the count. This will let us see on the screen each time the LED pulses.

17


count
xod/core/count

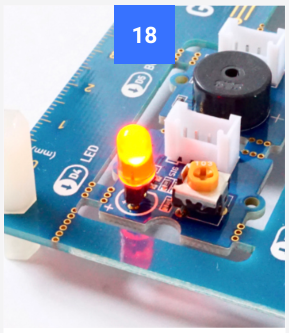
STEP	1
INC	linked
RST	Never
OUT	0

Label

SET COUNT PINS

Set RST to ‘Never’. STEP determines how much the count increases by with each pulse. Set this to ‘1’.

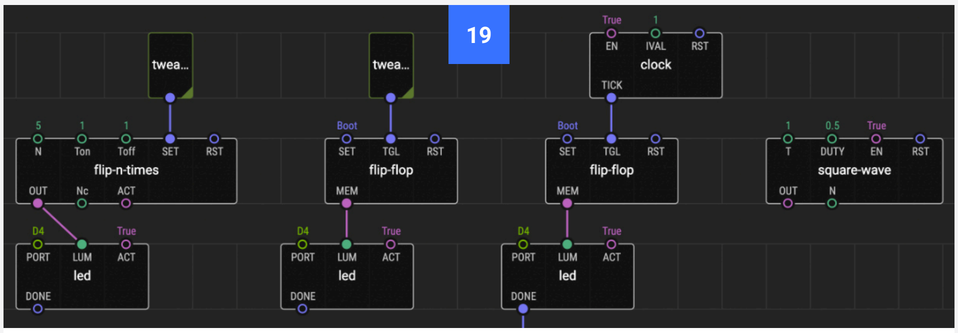
18



TEST THE PATCH

Upload the program. Watch the LED and count. The LED should flash and the count will increase with each flash.

19



EXPERIMENT!

As with all programming, there is always more than one way to achieve a similar outcome, and different methods may suit different applications. Here we have tested two different ways of making the LED flash, but there are plenty of other ways you can experiment with. Why not try using a *square-wave* node to make the LED flash? See if you can work it out using the help pane and XOD website. Or you can just try playing around with the nodes you’ve already tried. Try experimenting with different timings and patterns of flashing.

Watching Pulse Pins: Combining Count and Watch Nodes

Combining *count* and *watch* nodes is a really useful way to visualise the output of a pulse pin. Unfortunately, a pulse output can not be directly connected to a *watch* node in XOD, as the data types (pulse and string) clash.

We can get around this by connecting a pulse output to the INC pin of a *count* node, and then connecting a *watch* node to the *count* output pin (as we did in **Steps 15-16** of this Task). In this setup, the *count* node increases with each pulse sent, and we can visualise this in the debugger with the *watch* node.

Discovering New Nodes

The last step of this task encourages you to try out a new node: the *square-wave* node. We have not provided specific information about this node here, but it is useful to practice discovering new nodes for yourself.

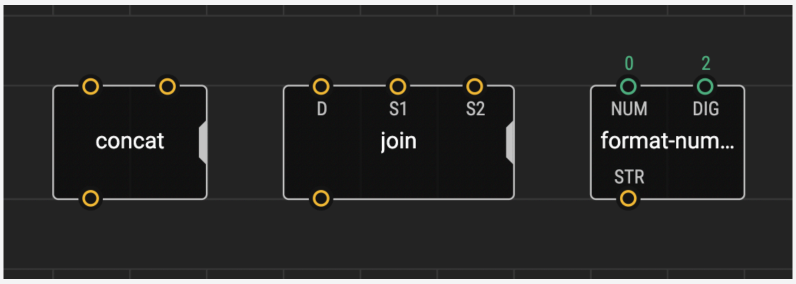
To work out how a new node works, it is best to start with understanding what its pins do. In most cases, the Quick Help pane will give a brief explanation of the node and what each of its pins does. This is often enough information to work out the node's function and how to use it.

You can also find additional documentation about each node on the XOD website. You can access this by clicking on the document button next to the node name in the Inspector pane. Or by visiting **www.xod.io/libs** and using the search function.

When the documentation for a node is not sufficient, you can usually still work out its function by adding *tweak* nodes to its inputs and *watch* nodes to it's outputs. Then simulate or 'Upload and Debug' your patch. Try editing each of the inputs in turn and watch how this affects the outputs. In this way you can often determine a node's function experimentally.

If you are still having trouble, you can always find help on the XOD forum at **www.forum.xod.io**.

Concat, Join and Format-Number Nodes



CONCAT NODE

The *concat* node allows you to join two or more sets of strings together. This is useful for combining different inputs for display or storage. E.g. combining a number reading from a sensor with the symbol for it's units. Concat will join the inputs directly, so if you require a space between them you will need to input this in your string.

Concat is a variadic node. Meaning you can change the number of inputs. You can do this by dragging out the white tab on the right side of the node.

JOIN NODE

The *join* node is similar to the *concat* node, but has the additional feature of allowing you to chose how the different string inputs are joined together. The delimiter (D) pin determines what character is used to separate inputs, e.g. a space, comma or colon etc. This is particularly useful for storing data readings, as you can separate values with a comma or tab to create comma-separated (.csv) or tab-separated (.tsv) files. Like *concat*, the *join* node is variadic, and can take as many inputs as necessary.

FORMAT-NUMBER NODE

As the name suggests, the *format-number* node allows you to format number outputs. With this node, you can determine the amount of decimal places displayed, which can be very useful if you would like to display a sensor reading, for example. This node also converts the format of the input from a number to a string.

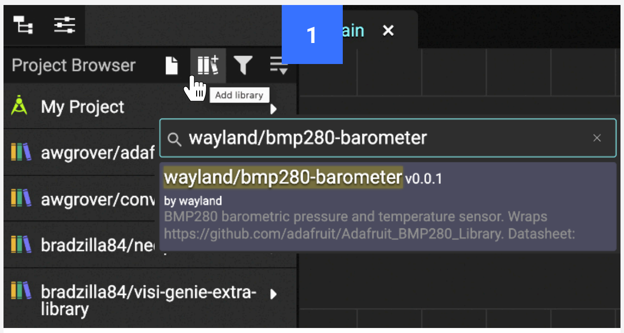
Other nodes useful for formatting numbers include *number-split-to-*digit, from the library *gst/number-split-to-digit*), *dec-to-2digits* and *dec-to-4digits*, both from the library *cesars/utils*.

Task 5: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (air pressure sensor module)
- USB-A to micro USB cable

In this task we'll look at using *concat*, *join* and *format-number* nodes. These nodes are especially useful when working with data and displays.

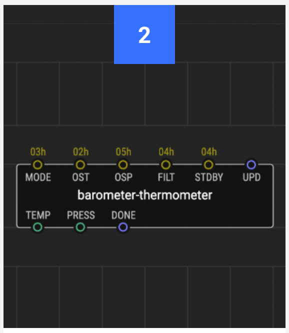
The *concat* and *join* nodes are both used to combine information in the form of strings (text). The *format-number* node is used to set the number of decimal points displayed in a number. In combination, these nodes are useful for formatting the outputs of sensor modules, both for data storage, and for display on a screen.



BMP280 barometric pressure and temperature sensor. Wraps https://github.com/adafruit/Adafruit_BMP280_Library. Datasheet:

NEW PATCH AND ADD LIBRARY

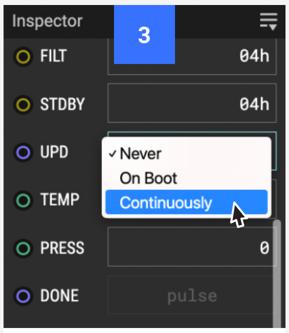
Open a new patch (or move on to tuto501). To work with the air pressure sensor (also known as a barometer) you will need to install the library *wayland/bmp280-barometer*.



ADD BAROMETER-THERMOMETER NODE

Add a *barometer-thermometer* node (*wayland/bmp280-barometer*) to the patch.

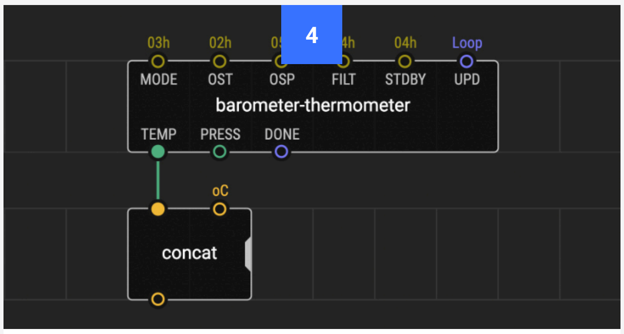
Concat, Join and Format-Number Nodes



FILT 04h
STDBY 04h
UPD Continuously
PRESS 0

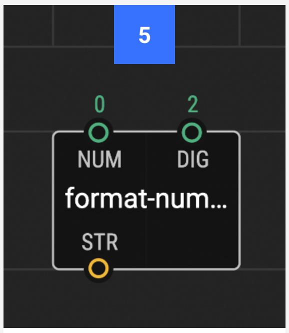
SET BAROMETER-THERMOMETER NODE

Set UPD to 'Continuously'. Leave other inputs as they are. Outputs are temperature (TEMP) and pressure (PRESS).



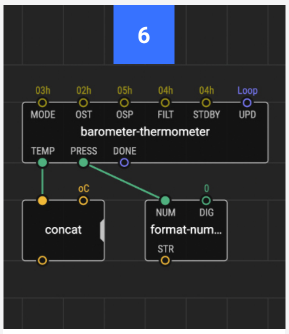
ADD AND SET CONCAT NODE

Add a *concat* node (*xod/core*). This node combines multiple strings from the input pins into a single output. This node is 'variadic' meaning you can expand the node by pulling on the tab on the right, letting you increase the number of inputs. Connect the first input to the TEMP pin. Set the second input to 'oC' (degrees centigrade).



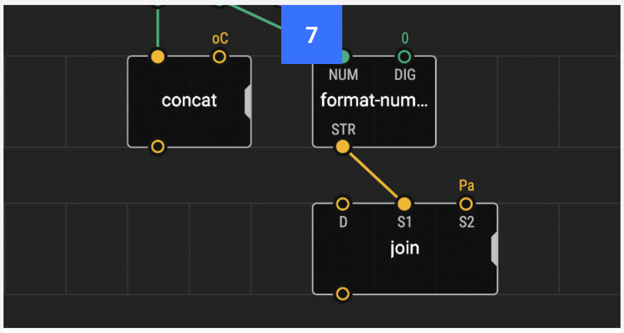
ADD FORMAT-NUMBER NODE

Add a *format-number* node (*xod/core*). This node lets you format the number of decimal places in a number.



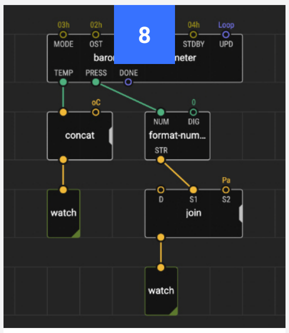
SET FORMAT-NUMBER NODE

Link the *barometer-thermometer* PRESS pin to the *format-number* NUM (number) pin. Set DIG (digits) to '0' decimal places.



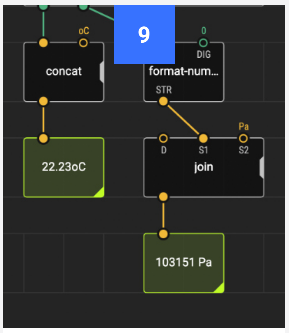
ADD AND SET JOIN NODE

Add a *join* node (*xod/core*). This is similar to *concat*, but has a D (delimiter) pin. D determines how inputs are joined (e.g. via a space or colon). It's automatically set to be a space. Leave it as this. Connect the first input (S1) to the *format-number* STR (string) pin. Set the second input to 'Pa' (pascals, the unit of air pressure).



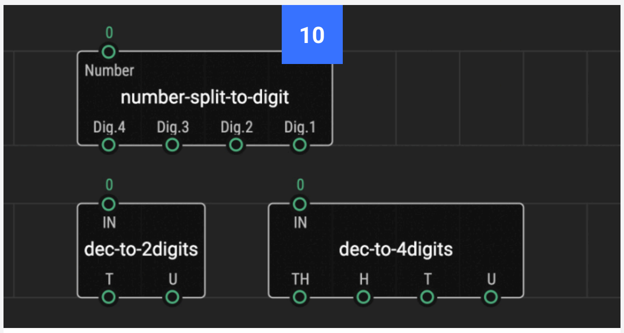
ADD WATCH NODES

Add two *watch* nodes (*xod/debug*). Link one to the output of the *concat* node, and one to the output of the *join* node.



TEST THE PATCH

Upload and debug. Look at the *watch* nodes. You may need to expand the *watch* nodes by pulling on the bottom right corner.



EXPERIMENT!

Play around with the nodes in your patch to see how you can format the sensor output in different ways. Try exploring other nodes available for formatting numbers in XOD. For example, *number-split-to-digit* (from *gst/ number-split-to-digit*), *dec-to-2digits* or *dec-to-4digits* (both from *cesars/utils)*.

More Information on Basic Nodes

This section has covered a number of basic nodes that we have found useful in building simple biological devices. However, there are plenty of other useful nodes out there, both pre-installed in XOD, and created by XOD users such as yourself.

In the rest of this guide we will continue to explore useful nodes and techniques in XOD, but if you'd like to explore for yourself, here are a few useful resources for getting to grips with XOD:

XOD TUTORIAL
Each time you open XOD it will offer you the option of following its inbuilt tutorial. Working through this is a great way to learn more about what XOD can do. It is also available online at **www.xod.io/docs/tutorial**.

XOD GUIDE
The XOD user guide provides advice on some more complex concepts, as well as some case studies to work through. It is available online at **www.xod.io/docs/guide**.

XOD CORE LIBRARY
Taking a look through the nodes in the XOD core library will help you understand the most basic nodes available and what they do. Find xod/core in the Project Browser.

Lesson 4: Building Devices

Creating New Nodes

Using Buses

Logic Programmes

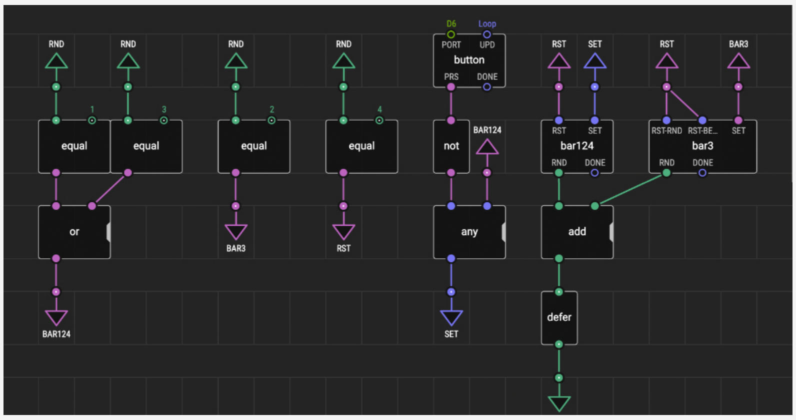
Sequences and Loops

Building Devices

So far in this guide we have explored how to use a few useful nodes to perform some simple tasks, like watching the readings from a sensor, or flashing an LED. However, we often want to perform more complex tasks, like reading and storing data, displaying information on a screen, or creating complex logical programmes.

This lesson will build on what we have learned already, and explore some more complex concepts in XOD. This will help you to build larger, more complex programmes and devices in a neat and efficient way.

The first two tasks in this lesson, Task 6 and Task 7 will cover how to make new nodes and how to use buses respectively. These skills are useful for creating tidy, and compartmentalised programmes. Task 8 and Task 9 will then explore how we can use these skills to build logic-based programmes, and how we can introduce sequences and loops, which are useful for biological devices.



XOD patch from Task 9: Sequences and Loops

OBJECTIVES

By the end of this chapter you should be able to:

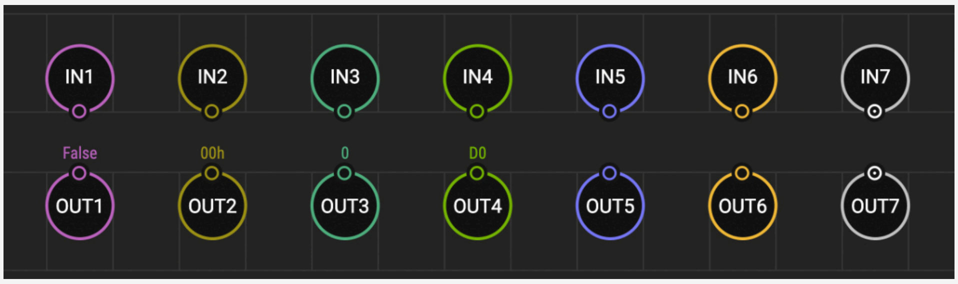
- Describe the function of XOD terminal nodes and how they are used.
- Create new nodes in XOD by combining existing and terminal nodes.
- Test and use the new nodes you have created in programmes.
- Describe the function of buses, their advantages, and how to use them.
- Use maths and logic nodes to create logic programmes in XOD.
- Recall at least two different methods for creating sequences in XOD.
- Implement programming loops in XOD using the *defer* node.
- Use the remaining components on the Grove board: the sound sensor, light sensor, 3-axis acceleration sensor and the OLED screen.
- Programme the OLED screen to display graphics such as text and shapes.
- Recall how to document your nodes correctly, including describing the node and it's pins and adding comment boxes.
- Recall how to publish nodes, or collections of patches a library.

Creating New Nodes

Why Create New Nodes?

If we want to build programmes capable of more complex functions than producing a simple input and output, we will need to add a few more skills to our repertoire. More complex programmes often require more nodes, and the multitude of nodes and links can quickly become confusing. The good news is that XOD provides several ways of reducing the complexity of your patches and keeping your programmes neat and tidy. This is good practice so that you can keep track of what you're doing, and also for others who may need to understand your programme.

One way simplify a complex programme is to make your own nodes. This means that you can encapsulate specific functions within your programme into neat little packages that can be easily connected to each other. It also has the advantage that they can easily be shared with the wider XOD community, making useful new nodes available to everyone.



XOD terminal nodes

Creating your own nodes is much easier than it sounds. It is essentially the same as creating any other patch, but we need to add special nodes called 'terminals' to allow our new node to communicate with other nodes.

There are two types of terminals: inputs and outputs. Like *tweak* nodes, they come in different types based on their data type. The nodes above from left-to-right, top-to-bottom are: *input-boolean*, *input-byte*, *input-number*, *input-port*, *input-pulse*, *input-string*, *input-t1* (custom input type), *output-boolean*, *output-byte*, *output-number*, *output-port*, *output-pulse*, *output-string*, *output-t1* (custom input type).

In the next task we'll explore in more detail how to use these terminals to create your own nodes, using the inbuilt OLED screen on your board as an example.

The XOD website also provides some excellent information about how to make your own nodes at www.xod.io/docs/guide/nodes-for-xod-in-xod.

Creating New Nodes

Task 6: Requirements

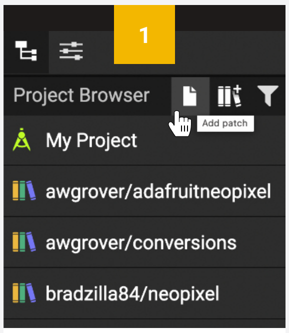
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (sound sensor and OLED screen modules)
- USB-A to micro USB cable

In this task we’ll learn how to make a new node that will allow us to write text on our OLED screen.

Instructing the OLED screen to display text is a slightly more complex task than we have done so far, and involves several nodes to represent the screen rather than one.

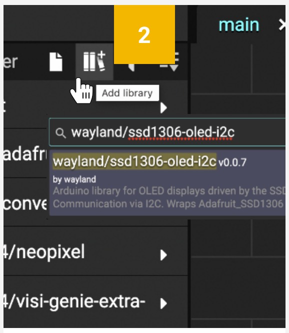
In this task we’ll be combining these multiple nodes into one, which we will then use to display the readings from our onboard sound sensor.

Creating nodes like this is useful as it helps to simplify the patch, and we can also save new nodes for later use in different programmes.



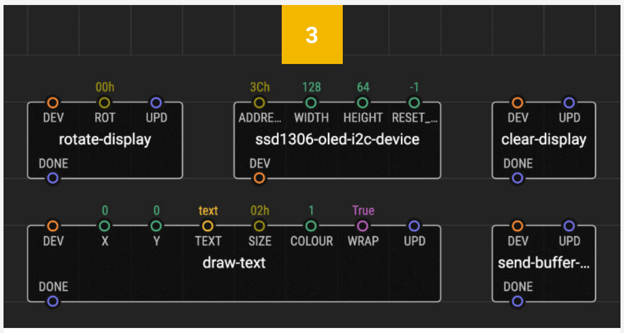
NEW PROJECT AND NEW PATCH

Save your project and create a new one (or move on to tuto601). Add a new patch to the project, and name it ‘*write-text-to-oled*’.


2
main
Add library
wayland/ssd1306-oled-i2c
wayland/ssd1306-oled-i2c v0.0.7
by wayland
Arduino library for OLED displays driven by the SSD
Communication via I2C. Wraps Adafruit_SSD1306
/neopixel
/visi-genie-extra-

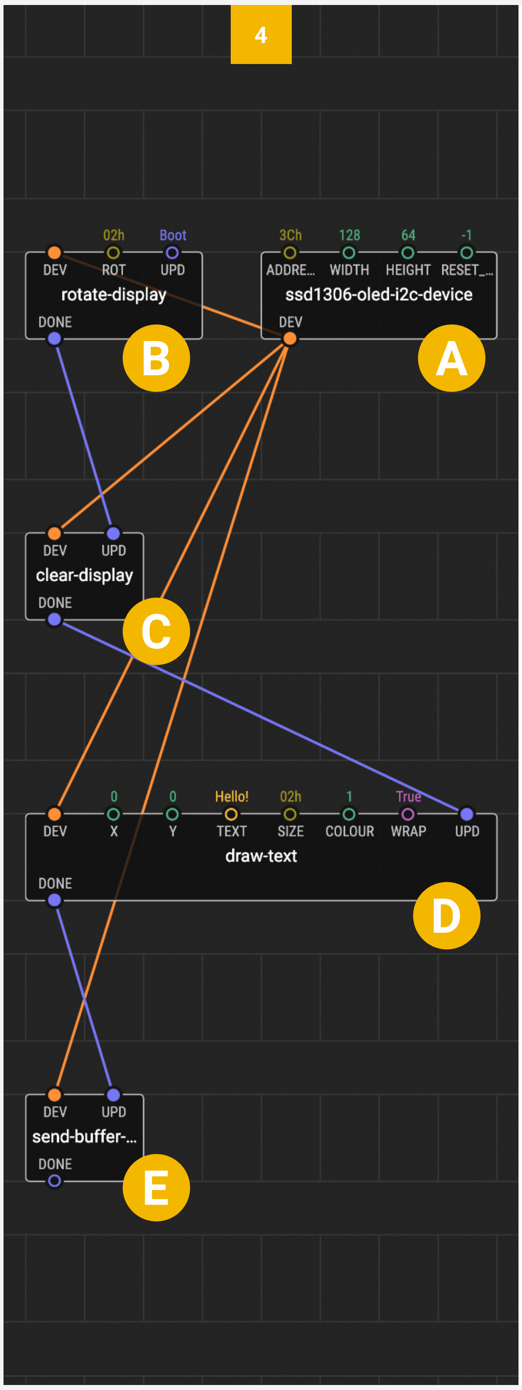
ADD LIBRARY

To work with the OLED screen you will need to install the library *wayland/ssd1306-oled-i2c*.



ADD NODES

From the *wayland/ssd1306-oled-i2c* library, add the following nodes to your patch: *ssd1306-oled-i2c-device*, *rotate-display*, *clear-display*, *draw-text*, *send-buffer-to-display*.



SETTING UP THE OLED SCREEN

Setting up the OLED screen requires several connections between these nodes, so let's take it step by step.

A **SSD1306-OLED-I2C-DEVICE**

This node represents the OLED device. WIDTH and HEIGHT set the dimensions of the screen in pixels. Leave these as '128' and '64'. ADDRESS identifies the port, leave this as '3Ch'. RESET represents the screen's reset pin. Leave this as '-1' as our board does not have a dedicated screen reset pin. **The output of this node, DEV (device) needs to be connected to each of the other nodes' DEV input pins.**

B **ROTATE-DISPLAY**

You can change the screen orientation using this node. Set ROT to '02h', which is correct for our screen. Set UPD to 'On Boot' so that the screen updates when the programme starts. This node starts a sequence that allows us to display items on the screen. **In this sequence, each DONE pin connects to the next UPD pin.** Connect the *rotate-screen* DONE pin to the *clear-display* UPD pin.

C **CLEAR-DISPLAY**

This node should be used before displaying anything on the screen. Connect the DONE pin to the *draw-text* UPD pin.

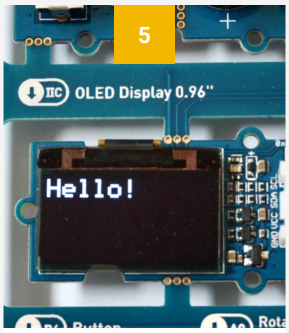
D **DRAW-TEXT**

This node inputs the text we want displayed. X and Y determine the position of the text by coordinates. Leave these as '0', '0'. TEXT is where you enter your text. Use 'Hello!' as a test. SIZE determines the size of the text. Leave this as '02h'. COLOUR determines the colour of the text (black or white). Set this to '1'. WRAP determines whether the text is wrapped within the boundaries of the screen. Leave this as 'True". Connect the DONE pin to the *send-buffer-to-display* UPD pin.

E **SEND-BUFFER-TO-DISPLAY**

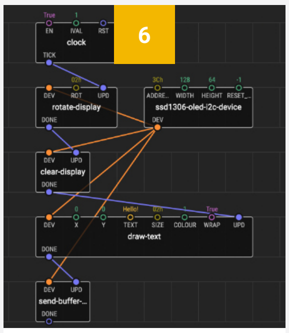
So far we have written information to the microchip's memory, but we haven't actually sent it to the screen. This node is the final step that sends this data. It needs to be used whenever you want to display something on the screen.

Creating New Nodes



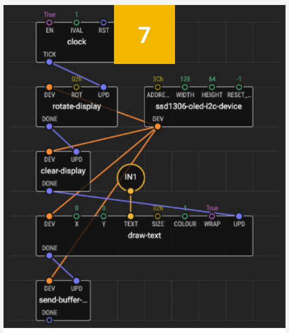
TEST THE PATCH

Upload the patch and watch your OLED screen. White text should appear in the top left-hand corner of the screen.



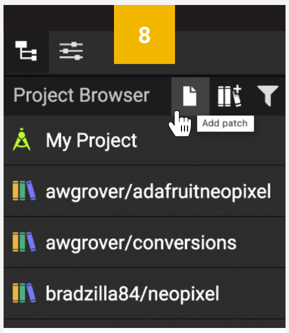
ADD A CLOCK NODE

Add a *clock* node (*xod/core*) and link the TICK pin to *rotate-display* UPD. This will make your screen update once a second.



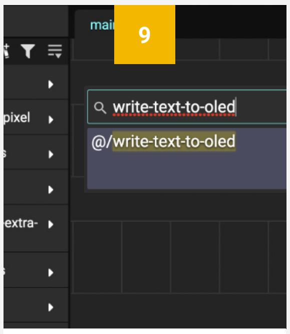
ADD AN INPUT-STRING NODE

Add an *input-string* node (*xod/patch-nodes*) and connect it to *draw-text* TEXT.



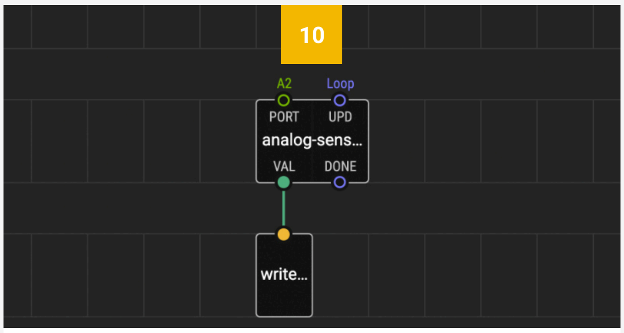
MAKE A NEW PATCH

Now we've made our new node, let's try adding it to another patch. Add a new patch and name it 'sound-sensor'.



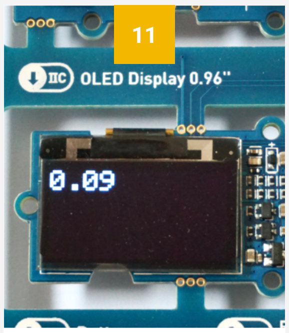
ADD WRITE-TEXT-TO-OLED NODE

To add your new node to a patch, simply search for it as usual, or drag the patch from the Project Browser into the patch.



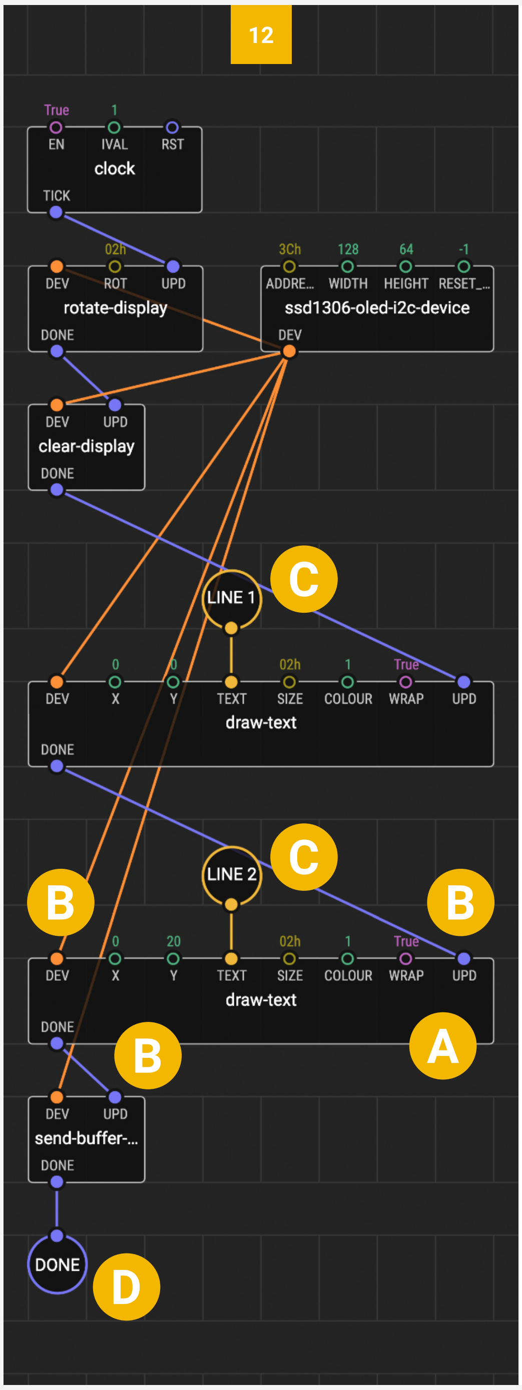
ADD AND SET ANALOG-SENSOR NODE

Now that we have a node that will write text to the screen, let's use it to display a sensor reading. For many common analog sensors (including the inbuilt sound sensor) you can use the simple XOD *analog-sensor* node (*xod/common-hardware*). Add this node, set PORT to 'A2', and connect VAL to the *write-text-to-oled* node.



TEST THE PATCH

Upload the patch, and you should see the readings from the sound sensor displayed on your screen.



MODIFY YOUR NODE

Writing a line of text directly to the OLED screen is great, but what if we need a more complicated node? For example, one that takes multiple lines of text, or one that sends a signal once the text has been uploaded?

Lets go back to our *write-text-to-display* node. You can do this by opening the patch in the Project Browser, or by double clicking on the node in your patch.

Make the changes listed below to expand the capabilities of your node.

A **SECOND DRAW-TEXT NODE**

Add a second *draw-text* node below the first. We will use this to draw a second line of text. Set the Y pin of this node to 20. This will move the text down by 20 pixels, creating a new line.

B **RECONNECT**

Delete the link between the first *draw-text* node and *send-buffer-to-display*. Link the first *draw-text* DONE pin to the second *draw-text* UPD pin. Link the second *draw-text* DONE pin to the *send-buffer-to-display* UPD pin. Connect *ssd1306-oled-i2c-device* DEV to the second *draw-text* DEV pin.

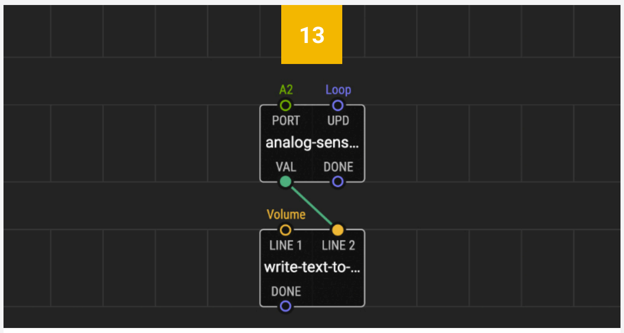
C **LABEL INPUT-STRING NODES**

Add a second *input-string* node and connect it to the TEXT pin of the second *draw-text* node. We will now have more than one input into our node, so we need give them labels, to avoid confusion. Click on each *input-string* node in turn. Use the Inspector pane to name your nodes by typing in the 'Label' text box. Name your nodes 'LINE 1' and 'LINE 2' respectively.

D **ADD OUTPUT-PULSE NODE**

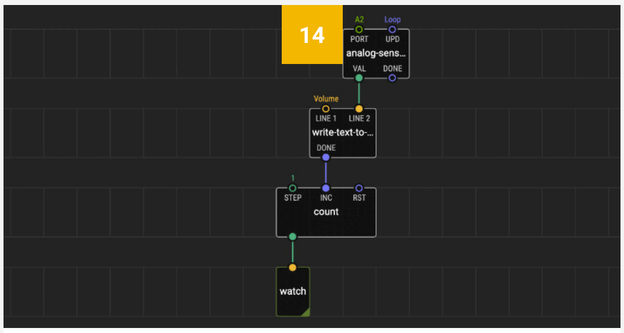
Add an *output-pulse* node to the patch, and connect it to the *send-buffer-to-display* DONE pin. Use the Inspector to label this node 'DONE'.

Creating New Nodes



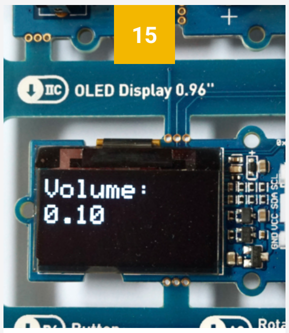
MODIFYING YOUR PATCH

Return to your 'sound-sensor' patch using the tab at the top, or the Project Browser. You will notice that the *write-text-to-oled* node has changed. It now has two inputs and an output. Delete the link between VAL and LINE 1 and link VAL to LINE 2 instead. Use the Inspector to set LINE 1 to 'Volume:'.



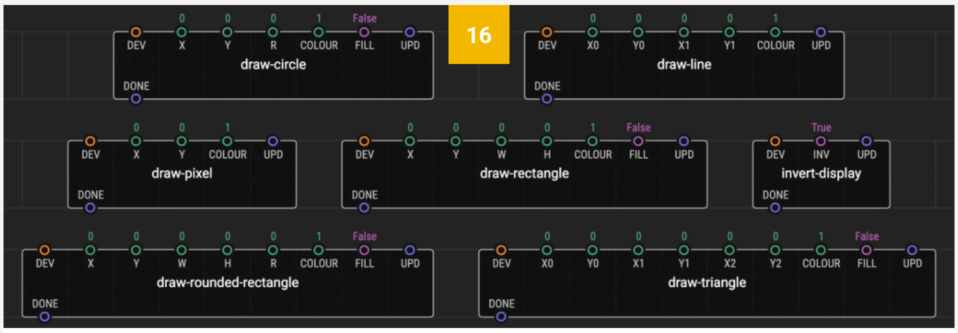
ADD COUNT AND WATCH NODES

Add a *count* node (*xod/core*) and *watch* node (*xod/debug*). Link the *write-text-to-oled* DONE pin to the *count* INC pin, and the *watch node* to the *count* output pin. A *watch* node cannot be directly linked to a pulse pin, so this is a useful trick if you want the watch the output from a pulse pin.



TEST THE PATCH

Upload and debug. You should see two lines of text on your screen, and the watch node will count when the screen updates.



EXPERIMENT!

The OLED screen is a really useful device, and can be used in a multitude of different ways. Try playing around with your new node by adding another line, or changing the position and size of the text. Or you can try using the OLED to display data from a different sensor. You can also experiment with some of the other nodes in the *wayland/ssd1306-oled-i2c* library. This library contains lots of useful nodes for drawing different objects on the screen, as well as several example patches to show you how they work.

Sharing Nodes and Publishing Libraries

One of the great advantages of XOD is the growing community of contributors, who are generating an ever-expanding range of nodes and libraries for other to use. When you create new nodes that you think might be useful for others, you can easily share these with the XOD community by publishing them as a library.

There are no strict rules about what constitutes a library, so even if you only create one node, this can still be a library. Publishing allows others to use your nodes, but is also useful for reusing your own work, as you can download your own libraries for use in all of your projects.

Creating a library is essentially just making a project with a patch for each node. You can also include patches with example of how to use the nodes, as there are in the *wayland-ssd1306-oled-i2c* library. Once you've created your project, you need to set the metadata. Do this by navigating to 'Edit > Project Preferences' in the menu bar. Here you should enter a name a description for you library, as well as a licence type (e.g. GNU, CC-BY etc. more info. at **www.opensource.org/licenses**).

To publish your library, go to 'File > Publish Library' in the menu, click 'Publish', and you're done! You can find out more about publishing libraries at **www.xod.io/docs/guide/creating-libraries**.

Documenting Nodes

When publishing your work it is good practice to make sure that your nodes are well documented. This helps to remind yourself what you've done, and allows others to get an idea of how the node can be used.

Before you publish you library, make sure that you have described the node and each of it's pins. To write a description of the node, click a blank space on the patch and a 'Description' box will appear in the Inspector pane. Write a brief description about what the node does and what its used for, e.g. "*This node writes two lines of text to the ssd1306 OLED screen*".

To write a description of the pins, click on an *input* or *output* node and you will see the 'Description' box at the bottom of the Inspector pane. Write a brief description of the pin, e.g. "*String to display on the first line of text. Text will appear at coordinates 0:0*".

You can also provide more information about how your node works by adding comments to the patch. To do this, navigate to 'Edit > Insert Comment' in the menu bar. You can find out more about documenting nodes at **www.xod/docs/guide/documenting-nodes**.

Using Buses

Task 7: Requirements

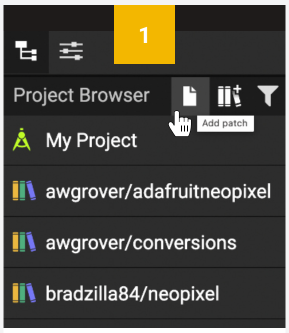
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (3-axis acceleration and OLED screen modules)
- USB-A to micro USB cable

In this task we'll look at another way to simplify our patches: using buses. Buses are a way to link pins 'invisibly' so that you don't have too many link intersections that make the data flow confusing.

Buses are a little like *input* and *output* nodes. They come in two types, *to-bus* and *from*-bus, and they automatically take the data type of the pin they're connected to.

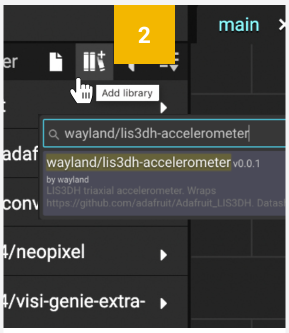
The *to-bus* node is used like an *output* node and sends information from an output to a bus. The *from-bus* node acts like an *input* node and retrieves information from the bus of the same name.

We'll practice using buses by displaying the output of our 3-axis acceleration sensor (also known as an accelerometer or tilt sensor) on our OLED screen.



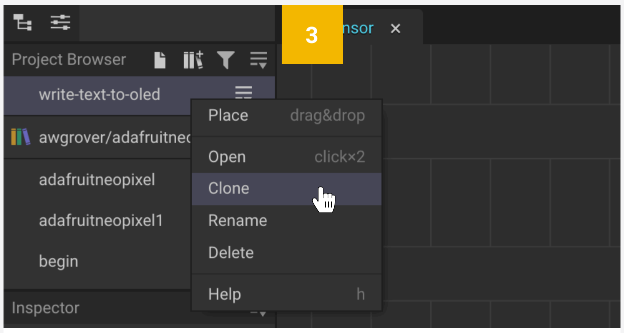
MAKE A NEW PATCH

Add a new patch to the project, and name it '*tilt-sensor*' (or move on to tuto701).


2
main
Add library
wayland/lis3dh-accelerometer
wayland/lis3dh-accelerometer v0.0.1
by wayland
LIS3DH triaxial accelerometer. Wraps https://github.com/adafruit/Adafruit_LIS3DH. Datas
/neopixel
/visi-genie-extra-

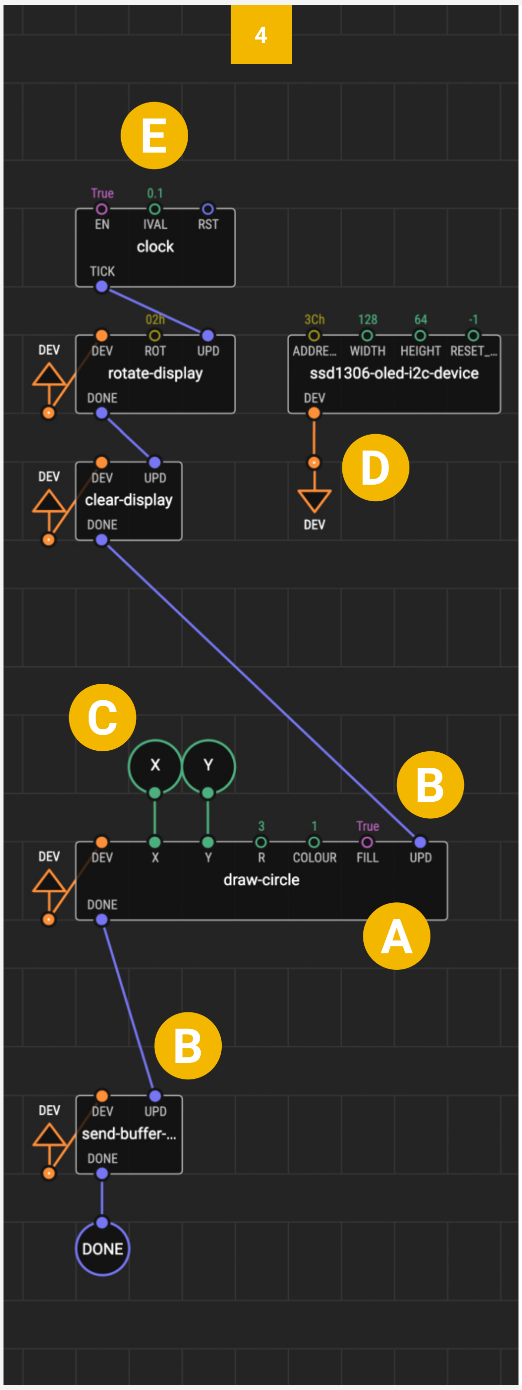
ADD LIBRARY

To work with the accelerometer you will need to install the library *wayland/lis3dh-accelerometer*.



CLONE WRITE-TEXT-TO-OLED PATCH

For this task we'll be using the OLED display again, but in a slightly different way. So that we don't have to start again, we can clone the *write-text-to-oled* patch by right clicking on the patch in the Project Browser and selecting 'Clone'. Rename the new patch from '*write-text-to-oled-copy*' to '*write-dot-to-oled*'.



MODIFY YOUR NODE

This time we would like to draw a small circle on the screen instead of text. The circle will move around the screen as you tilt the board.

Follow the instructions below to modify the *write-dot-to-oled* patch for this new purpose.

A **ADD AND SET DRAW-CIRCLE NODE**

Delete both *draw-text* nodes along with their associated *input* nodes. Add a *draw-circle* node. Set R (radius) to '3' to make a circle 3 pixels wide. Leave colour as '1'. Set FILL to 'True' so that we get a solid circle rather than an outline.

B **RECONNECT**

Link the *clear-display* DONE pin to the *draw-circle* UPD pin. Link the *draw-circle* DONE pin to the *send-buffer-to-display* UPD pin.

C **ADD INPUT-NUMBER NODES**

Add two *input-number* nodes and connect them to the X and Y pins. Use the inspector to label these 'X' and 'Y'.

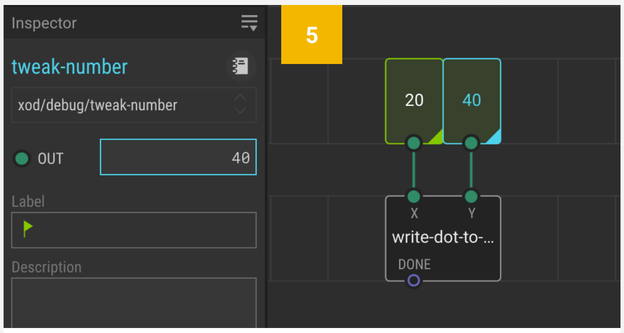
D **REPLACE LINKS WITH BUSES**

Although the OLED patch worked, it was very messy, with links criss-crossing, and it would be easy to miss a connection. Let's improve this by replacing these links with a bus. First, delete all of the orange links between the ssd1306-oled-i2c-device and the other nodes. Add a *to-bus* node (*xod/patch-nodes*) and link it to the output of ssd1306-oled-i2c-device. Use the inspector to label this node 'DEV'. Add a *from-bus* node (*xod/patch-nodes*). Make sure this *from-bus* node is also labelled 'DEV' as buses can only communicate if they have the same name. Repeat this process, or copy and paste the DEV *from-bus* node until you have four in total. Connect these to each of the DEV input pins on the nodes *rotate-display*, *clear-display*, *draw-circle* and *send-buffer-to-display*.

E **CHANGE CLOCK TIMING**

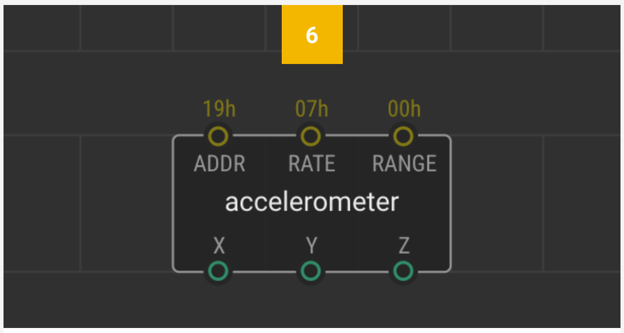
Set the *clock* IVAL pin to 0.1, so that it updates more frequently.

Using Buses



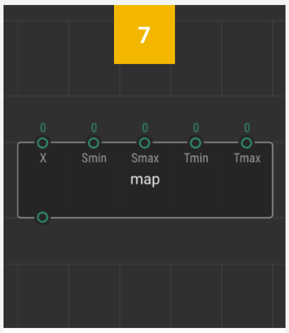
TEST THE NODE

Test the node by returning to your *tilt-sensor* patch, adding a '*write-dot-to-oled*' node, and connecting two *tweak-number* nodes to the X and Y inputs. Upload and debug. Click on the *tweak-number* nodes and use the Inspector to change their values. Watch how this shifts the dot around the screen.



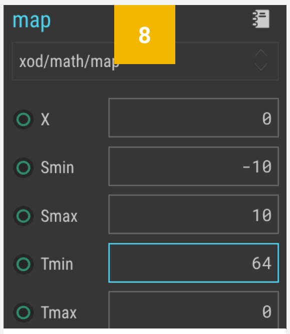
ADD AN ACCELEROMETER NODE

Add an *accelerometer* node (*wayland/lis3dh-accelerometer*). We want to use the output from the accelerometer to set the location of the dot on the screen. You could connect the *accelerometer* X and Y pins directly to the *write-dot-to-oled* X and Y pins, but it wouldn't work as the nodes' ranges don't match up.



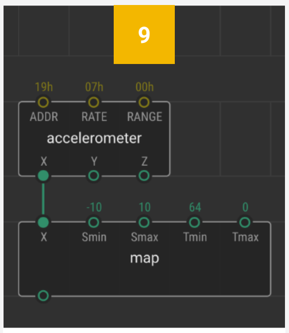
ADD A MAP NODE

To fix this, add a *map* node (*xod/math*). This node lets us map the *accelerometer* output range to the *write-dot-to-oled* input range.



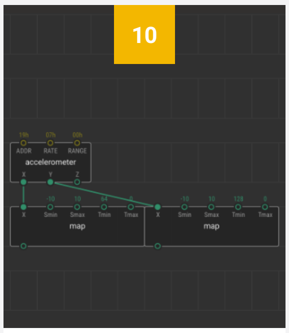
SET MAP NODE

Set Smin to '-10' and Smax to '10' (the range of the *accelerometer*). Set Tmin to '64' (the screen height) and Tmax to '0'.



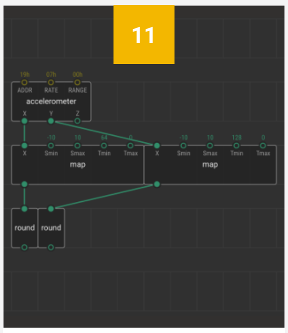
CONNECT MAP NODE

Link the *accelerometer* X and *map* X pins. The node now converts the *accelerometer* X range to values within the height of the screen.



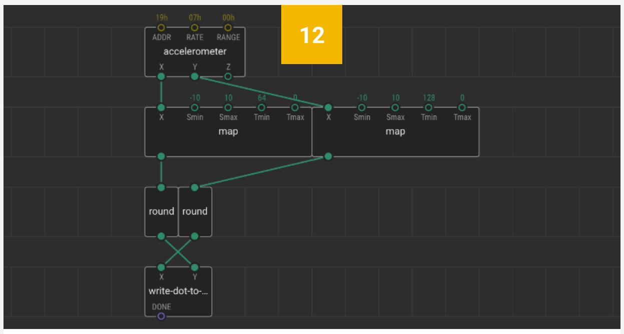
REPEAT MAP NODE

Add a second map node add link it to the Y output. Set Smin to '-10', Smax to '10', Tmin to '128' (the screen width) and Tmax to '0'.



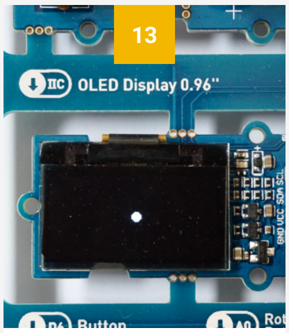
ADD ROUND NODES

Add two *round* nodes (*xod/math*). Connect them to the outputs of the *map* nodes. This will round the outputs to whole numbers.



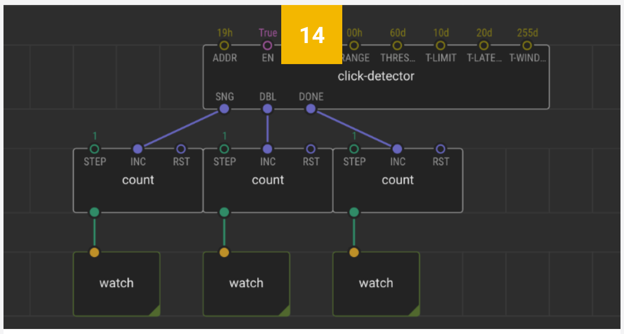
CONNECT ACCELEROMETER TO OLED

Delete the *tweak* nodes from the *write-dot-to-oled* node. Connect the *round* output linked to the *accelerometer* X output to the Y input pin, and the *round* output linked to the *accelerometer* Y output to the X pin. This seems counter-intuitive, but is due to the settings of the different nodes.



TEST THE PATCH

Upload the patch. Tilt your board left and right, backwards and forwards, and watch the little dot on the screen move!



EXPERIMENT!

Experiment with the nodes in this patch. Can you get some text to appear when the dot lands in the middle? It is also worth exploring the *wayland/lis3dh-accelerometer* library, as it has useful nodes and plenty of demonstrations. For example, the *click-detector* node above, which detects taps of the sensor.

Logic Programmes

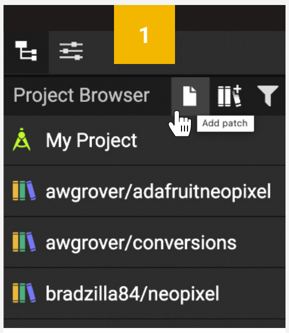
Task 8: Requirements

- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (light sensor and OLED screen modules)
- USB-A to micro USB cable

In this task we'll take some of the skills we've learned so far in this lesson and use them to create a more complex programme that uses logic to instruct the board what to do.

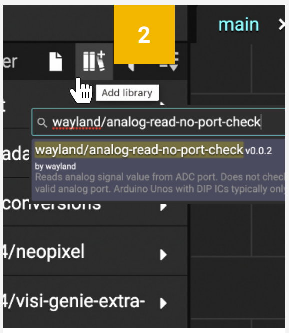
We'll be using the light sensor and OLED screen modules of the board to create a simple light sensing device.

Whilst this is still a fairly simple device, it contains a lot of the basic functions that you can use for your own instruments: an input in the form of a sensor; a logic programme that instructs the board what to do based on the value of this input; and an output that changes something, in this case the text displayed on a screen.



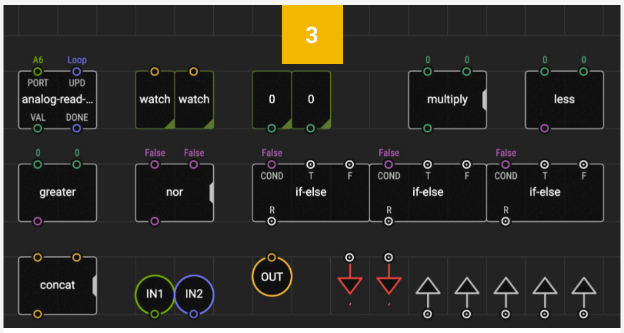
MAKE A NEW PATCH

Add a new patch to the project, and name it '*light-sensor*' (or move on to tuto801).


2
main
Add library
wayland/analog-read-no-port-check
wayland/analog-read-no-port-check v0.0.2
by wayland
conversions
/neopixel
/visi-genie-extra-

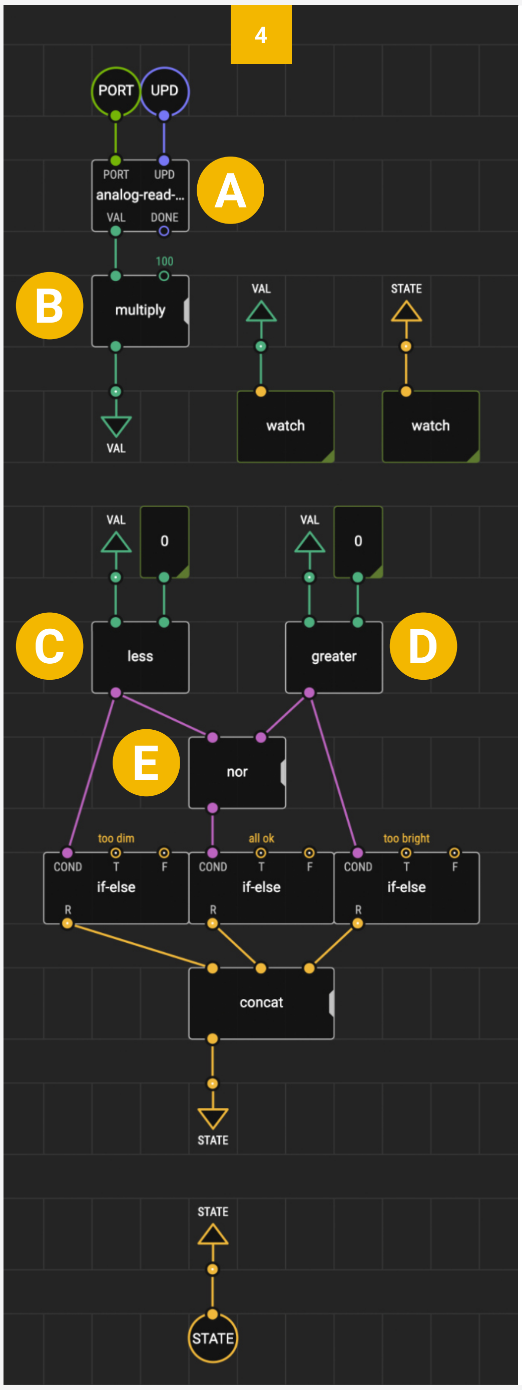
ADD LIBRARY

To work with the light sensor you will need to install the library *wayland/analog-read-no-port-check*.



ADD NODES

Add the following nodes to the *light-sensor* patch: *analog-read-no-port-check* (*wayland/analog-read-no-port-check*), *watch* x2, *tweak-number* x2 (*xod/debug*), *multiply*, *less*, *greater*, *nor*, *if-else* x3, *concat* (*xod/core*), *input-port*, *input-pulse*, *output-string*, *to-bus* x2, *from-bus* x5 (*xod/patch-nodes*).



SET UP YOUR LIGHT-SENSOR NODE (PART 1)

We want our node to return one of three readings depending on the light intensity: 'too dim', 'all ok' or 'too bright'. We'll use a combination of simple logic functions to programme this capability.

Follow the instructions below to set up your patch.

A **ANALOG-READ-NO-PORT-CHECK**

This node represents the light sensor. Connect the VAL output to the first input of the *multiply* node.

B **MULTIPLY**

We will use the *multiply* node to scale up the output of the *sensor* node. The node will multiply the input values, so let's set the second input '100'. Connect one of the *to-bus* nodes to the *multiply* output and label it 'VAL'.

C **LESS**

This node takes the first input value and compares it to the second input value. It will return 'True' if the first value is less than the second, and 'False' if not. Connect one of the *from-bus* nodes to the first input and label it 'VAL' so it receives the value output by the *multiply* node. Connect one of the *tweak-number* nodes to the second input so you can tweak the lower limit later.

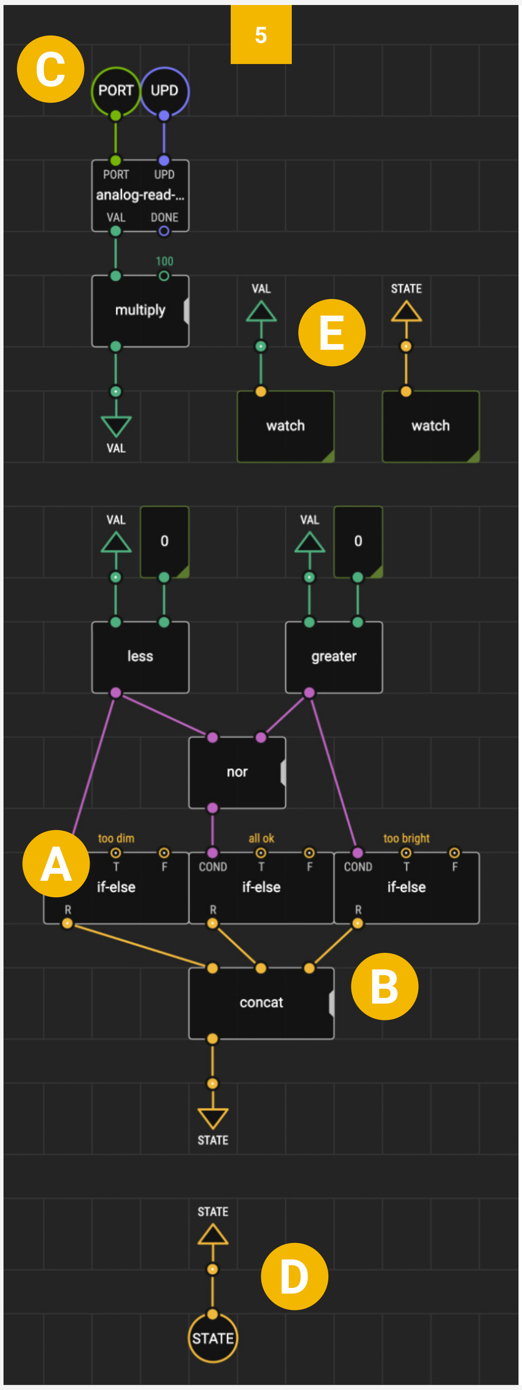
D **GREATER**

The *greater* node is very similar to the *less* node, but it will return 'True' only when the first input is greater than the second input. Repeat the same connections for the *greater* node. Connect a *from-bus* node to the first input and label it 'VAL'. Then connect a *tweak-number* node to the second input to let you tweak the upper limit.

E **NOR**

The *nor* node will only return 'True' if both inputs read 'False'. We want a third state that triggers if the light intensity is neither less than the lower limit, nor greater than the upper limit, and we will use the *nor* node to achieve this. Connect the output of *less* node to the first *nor* input, and the output of the *greater* node to the second *nor* input.

Logic Programmes



SET UP YOUR LIGHT-SENSOR NODE (PART 2)

You should now have completed the first half of the patch.

Continue setting up the second half by following the instructions below.

A **IF-ELSE**

The *if-else* node will output one value (T) if the condition (COND) it receives is true, and another (F) if it is False. We want to set up the three *if-else* nodes so that each of the above conditions (less than the lower limit, greater than the upper limit, or neither) returns as different line of text.

Set the T pin of the first *if-else* node to ‘too dim’ and connect the COND pin to the *less* output. Set the T pin of the second *if-else* node to ‘all ok’ and connect the COND pin to the *nor* output. Set the T pin of the third *if-else* node to ‘too bright’ and connect the COND pin to the *greater* output. We will leave the F pins blank, so that nothing is returned when the conditions are false.

B **CONCAT**

Use the tab on the variadic *concat* node to expand it to three inputs. Connect all three of the *if-else* outputs to the *concat* inputs. This will combine all three responses into one string. Due to the logic conditions, only one string will returned at a time and each of the other two nodes will return a blank value. Connect the second *to-bus* node to the *concat* output and label it ‘STATE’.

C **INPUT NODES**

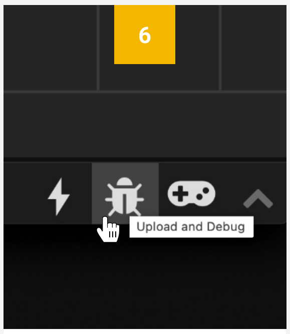
Connect the *input-port* node to the PORT pin of the *analog-read-no-port-check* node and label it ‘PORT’. Connect the *input-pulse* pin to the UPD pin of the *analog-read-no-port-check* node and label it ‘UPD’.

D **OUTPUT NODE**

Connect a *from-bus* node to the *output-string* node and label both nodes ‘STATE’. This may seem redundant, but will help us with our next step.

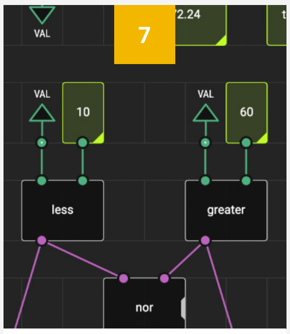
E **WATCH NODES**

Connect a *from-bus* node to each *watch* node. Label one ‘VAL’ and one ‘STATE’. This will link one to the 100x multiplied output of the sensor node so that you can see the sensor reading, and one to the final output of the *concat* node so that you can see the current state. By adding buses here we can put the two *watch* nodes together and easily view the outputs side by side, rather than having to move around the screen.



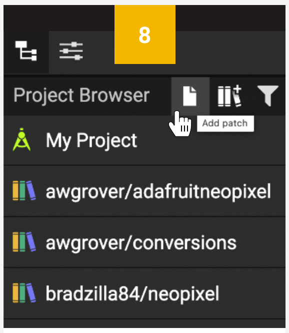
UPLOAD AND DEBUG

Upload and debug the *light-sensor* patch using the ladybird button.



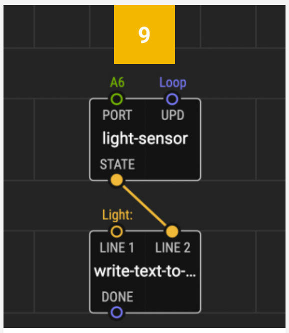
SET RANGE

Use the *tweak-number* nodes to find a range that works. We have used 10-60 but this may need adjusting to your environment.



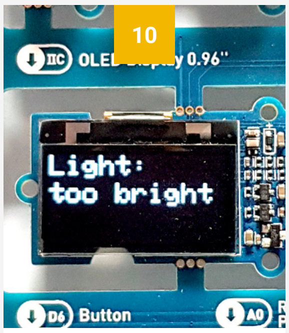
MAKE A NEW PATCH

Add a new patch to the project, and name it ‘*light-sensor-display*’ (or move on to tuto810).



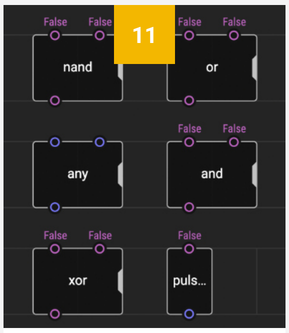
ADD AND CONNECT NODES

Add your *light-sensor* node and a *write-text-to-oled* node to the patch. Set LINE1 to ‘Light:’ and connect STATE to LINE2.



TEST THE PATCH

Upload the patch. Change the light intensity by moving or covering the board. Watch how this affects the screen output.



EXPERIMENT!

Explore some of the other logic nodes from the *xod/core* library. Or try using *pulse-on-true* with *if-else* to control a programme’s timing.

Sequences and Loops

Task 9: Requirements

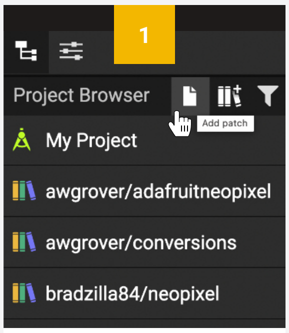
- Computer running MacOS, Windows or Linux (XOD software and USB driver installed)
- Grove Beginner Kit for Arduino (buzzer module)
- USB-A to micro USB cable

In this final task we’ll explore one of the most useful skills for building biological devices: creating sequences and loops.

By programming a sequence of events, using logic and introducing loops we can make devices that are useful for tasks such as automation, monitoring and response to environmental conditions.

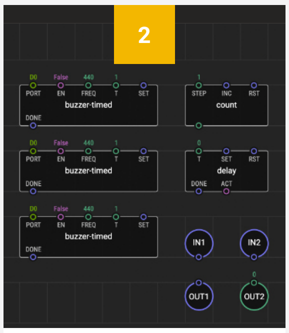
This task will introduce these skills by using the buzzer to play a simple tune ‘hot cross buns’ (an English nursery rhyme).

This will involve creating two separate sequences and using logic nodes to instruct the programme when to play them. We will use two different methods to make the sequences, and will create a separate node for each one.



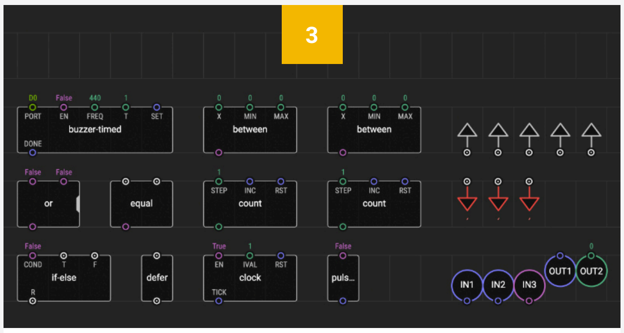
MAKE THREE NEW PATCHES

Add three new patches to the project, and name them ‘*bar124*’, ‘*bar3*’ and ‘*play-tune*’ (or move on to tuto901).



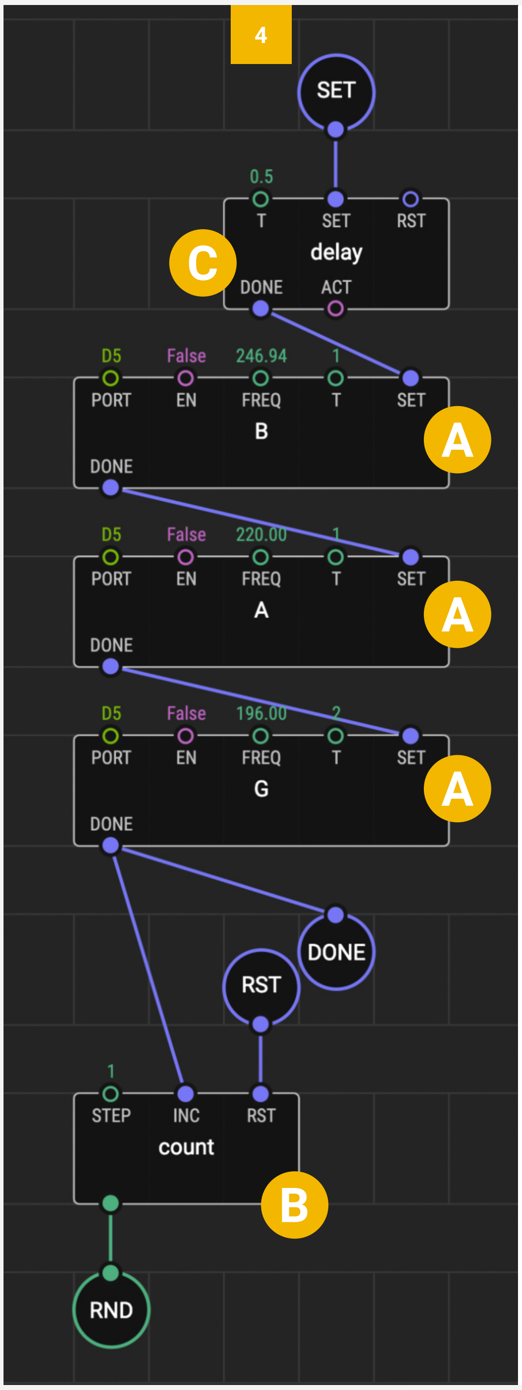
ADD NODES TO BAR124

Add: *buzzer-timed* x3 (*marcoaita/malibrary*), *delay*, *count* (*xod/core*), *input-pulse* x2, *output-pulse*, *output-number* (*xod/patch-nodes*).



ADD NODES TO BAR3

Add the following nodes to the *bar3* patch: *buzzer-timed* (*marcoaita/malibrary*), *clock*, *count* x2, *or*, *defer*, *if-else*, *equal*, *pulse-on-true* (*xod/core*), *between* x2 (*e/comparison* - you will need to install this library) *input-pulse* x2, *input-boolean*, *output-pulse*, *output-number*, *to-bus* x3, *from-bus* x5 (*xod/patch-nodes*).



SET UP YOUR BAR124 NODE

In this patch we will create the sequence of notes (B, A, G) that make up bars 1, 2 and 4 of the tune.

A

BUZZER-TIMED NODES

Set each of the *buzzer-timed* nodes to play a different note by changing their frequency pins. Set one to '246.94' Hz and change its name to '*B*' using the label field in the Inspector. Set one to '220' Hz and name it '*A*'. Set one to '196' Hz and name it '*G*'. Set PORT to 'D5' and EN to 'False' on all three. Set T (time) to '1' on *B* and *A* and to '2' on *G*. This will produce a longer final note. Connect the DONE pin of *B* to the SET pin of *A* and the DONE pin of *A* to the SET pin of *G*. Add the *output-pulse* node to the DONE pin of *G* and name it 'DONE', so that we can see when the sequence is complete.

B

COUNT

Connect the DONE pin of *G* to the INC pin of *count* so that the count increases each time the sequence is complete. Add the *output-number* node to the output pin and name it 'RND' (round), so that we can see the number of times the sequence has played. Add an *input-pulse* node to the RST (reset) pin and name it 'RST'. We will use this to reset the count when the tune finishes.

C

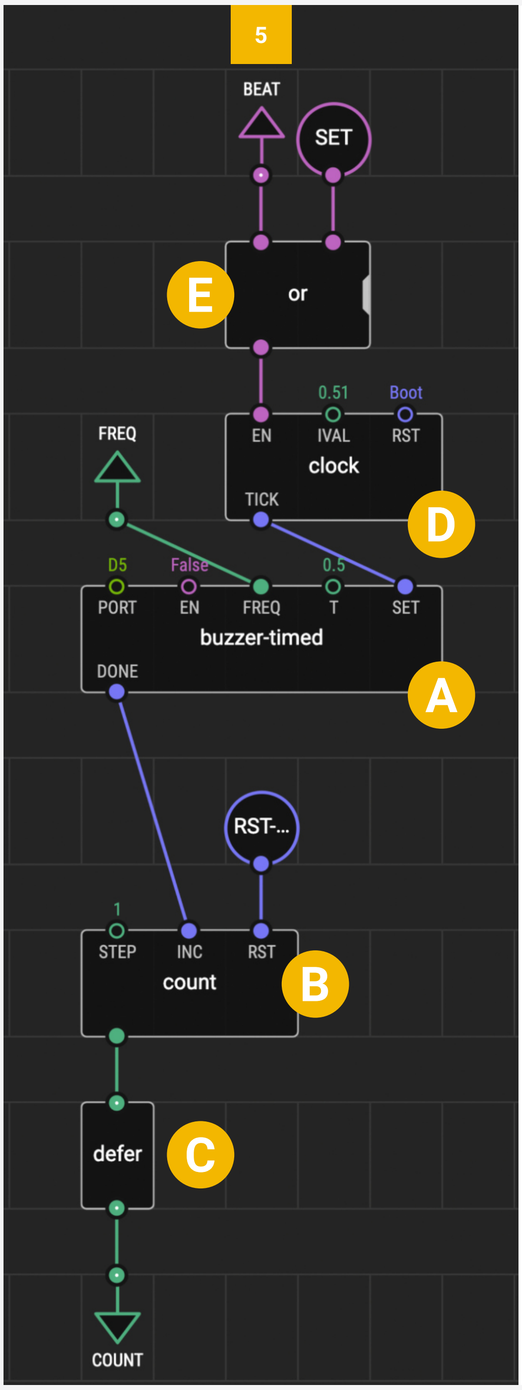
DELAY

We will use the *delay* node to introduce a half second gap between each bar. Set T (time) to 0.5, and connect the DONE pin of *delay* to the SET pin of *B*. Add an *input-pulse* node to the SET pin of *delay* and name it 'SET'. We will use this input to initiate the sequence.

To test this node, you can add a *tweak-pulse* node to the SET pin of *delay*. Upload and debug, then use the *tweak-pulse* node to initiate the sequence.

This patch uses a series of nodes with connected SET and DONE pins to create a sequence, similar to how we created our *write-text-to-oled* node in **Task 6** (p49). The patch for bar 3 will create a sequence in a slightly different way.

Sequences and Loops



SET UP YOUR BAR3 NODE (PART 1)

In this patch we will create the sequence of notes (G x 4, A x 4) that makes up bar 3 of the tune.

We will do this in a slightly different way to the *bar134* node. First we will set up a sequence to time the notes, then we will use logic nodes to control the frequency and number of the notes. This will create a loop that feeds information coming out of the programme back into the sequence. We will use buses to connect the two halves of this node.

A **BUZZER-TIMED**

Set the port to ‘D5’ and EN to ‘False’. Set T (time) to 0.5. Add a *from-bus* node to the FREQ pin and name it ‘FREQ’. This will allow our logic nodes to set the frequency of the note depending on the beat number.

B **COUNT**

Connect the *buzzer-timed* DONE pin to the *count* INC pin, so the the count increases each time the buzzer sounds. Add an *input-pulse* node to the RST pin and name it ‘RST-BEAT’. We will use this to reset the count at the end.

C **DEFER**

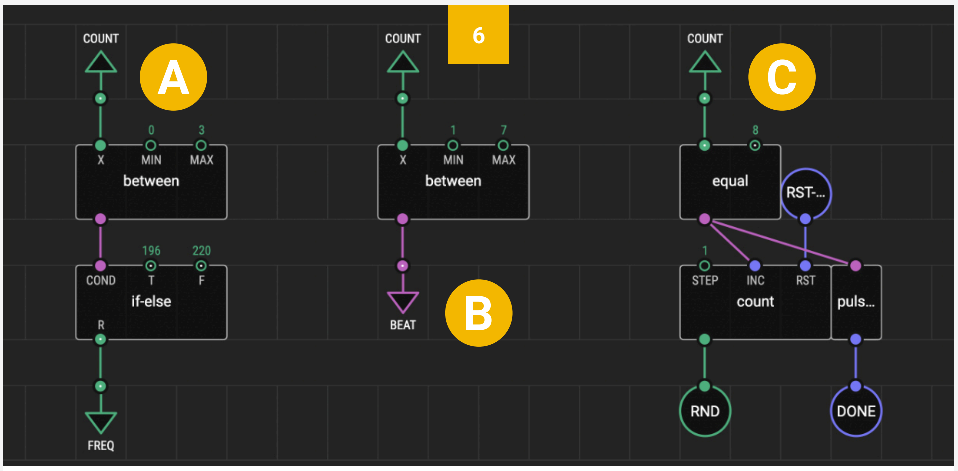
The *defer* node is the key to creating loops in XOD. In this case, we are creating a loop that reads out the beat number, and changes the frequency of the note and decides whether to repeat based on this. Connect the *count* output pin to the *defer* input pin to inform XOD of this loop. Add a *to-bus* node to the output pin and name it ‘COUNT’. This will feed into our logic nodes.

D **CLOCK**

We want the buzzer to sound a short note repeatedly. To do this, connect the *clock* node to the *buzzer-timed* SET pin. Set RST to ‘On Boot’ and set the IVAL (interval) pin to 0.51, which is slightly longer than the buzzer sounding time.

E **OR**

Using the *clock* node we’e made the buzzer sound regularly, but we don’t want it to sound all the time. We need to add conditions specifying when the clock is enabled. The buzzer should sound EITHER at the start of the third bar, OR when the sequence has started but not yet finished, i.e. after beats 1-7, but not after beat 8. Connect the *or* node to the EN pin of the *clock* node. Add an *input-boolean* node to one of the input pins and name it ‘SET’. We will use this to initiate the sequence at the start of bar 3. Add a *from-bus* node to the other input pin and name it ‘BEAT’. This will continue the sequence after beats 1-7.



SET UP YOUR BAR3 NODE (PART 2)

Follow the instructions below to complete the second half of the loop.

A **FREQUENCY**

This part switches the note between G and A depending on the count number. Use one of the *between* nodes. Add a *from-bus* node to the X pin and name it 'COUNT' so that it receives the count number from the end of our sequence. Set MIN to '0' and MAX to '3'. Connect the output to the COND pin of the *if-else* node. Set T to '196' so that the buzzer plays a G for the first four notes (count between 0-3). Set F to '220' so that the buzzer plays an A otherwise. Add a *to-bus* node to the *if-else* output pin and name it 'FREQ'. This will feed back into the loop to set the FREQ pin of the *buzzer-timed* node.

B **BEAT**

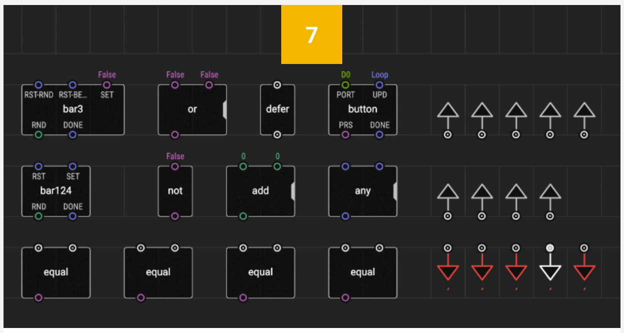
This part enables the buzzer pulse after beats 1-7. Use the second *between* node. Add a *from-bus* node to X and name it 'COUNT'. Set MIN to '1' and MAX to '7'. Add a *to-bus* node to the output pin and name it 'BEAT'. This will feed back into the loop as one of the conditions that will enable the clock.

C **ROUND**

This part records when the sequence is finished. Connect a *from-bus* node to the first input pin of the *equal* node and name it 'COUNT'. Set the second *equal* input pin to '8'. Connect the output pin to the *count* INC pin so that the count increases when the sequence is done. Connect the *input-pulse* node to the *count* RST pin and name it 'RST-RND'. We will use this to reset the count at the end. Add the *output-number* node to the *count* output and name it 'RND' so that we can see the number of times the sequence has played. Connect the *pulse-on-true* node to the *equal* output and then add the *output-pulse* node and name it 'DONE'. This will let us see when the sequence is complete.

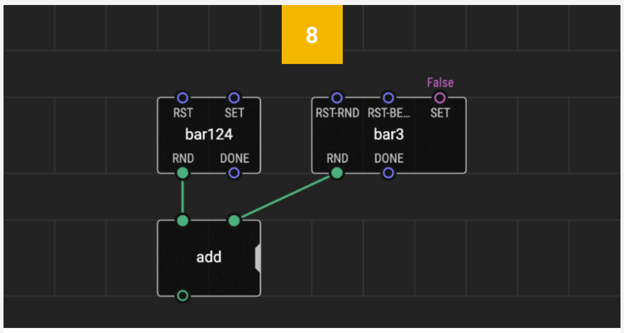
Your *bar3* node is now complete. To test this node you can add a *flip-flop* and *tweak-pulse* node to one of the *or* inputs. Pulse this twice to start and stop the sequence.

Sequences and Loops



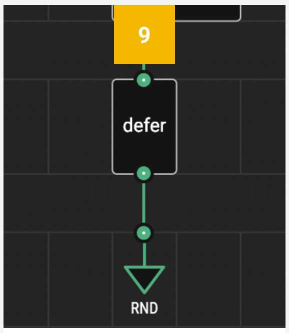
ADD NODES TO PLAY-TUNE

We will now set up a series of logic conditions to decide when to play each bar. Add the following nodes to the *play-tune* patch: *bar124*, *bar3*, *add*, *defer*, *equal* x4, *or*, *any*, *not* (*xod/core*), *button* (*xod/common-hardware*), *to-bus* x5, *from-bus* x9 (*xod/patch-nodes*).



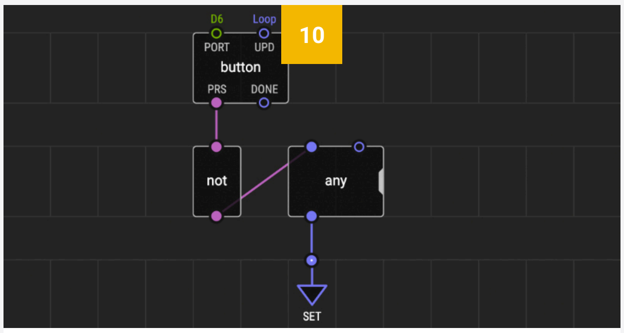
COUNTING THE ROUND NUMBER

To start we need to know the current round number. We added an *output-number* node (RND) to each of our nodes to count how many times it has been played. Connect both of these RND pins to the *add* node to sum these two values and find the total round number.



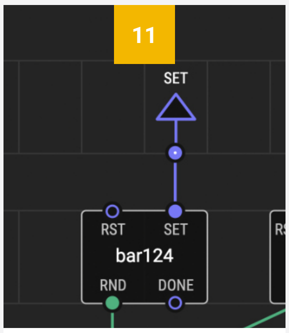
DEFER AND RND BUS

We will be creating a feedback loop again, so add a *defer* node to the *add* output. Then add a *to-bus* node to this and name it 'RND'.



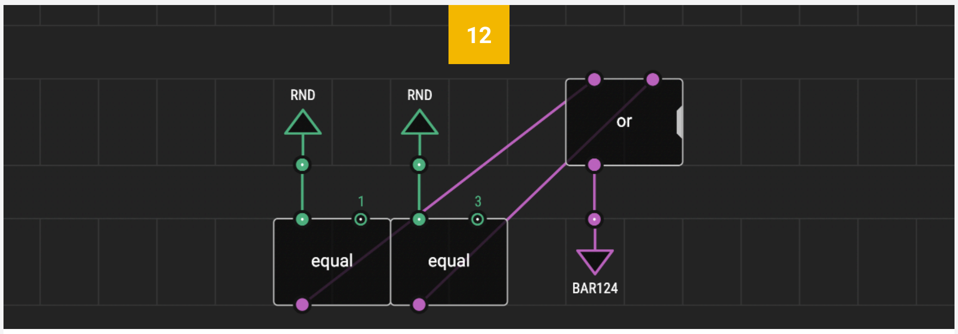
PLAYING BAR 1

We will use the button on the board to initiate round 1. Set the *button* PORT pin to 'D6' and PRS pin to 'True'. Then connect the PRS pin to the *not* node. Connect *not* output to one of the inputs of *any*. Leave the other *any* input unconnected for now. Add a *to-bus* node to the *not* output pin and name it 'SET'.



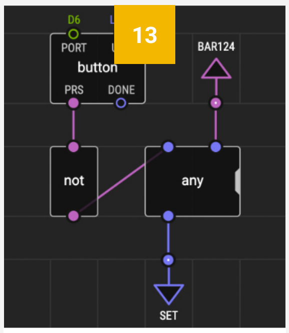
SET BUS

Add a *from-bus* node to the SET pin of *bar124* and name it 'SET'. This bus will initiate the *bar124* sequence.



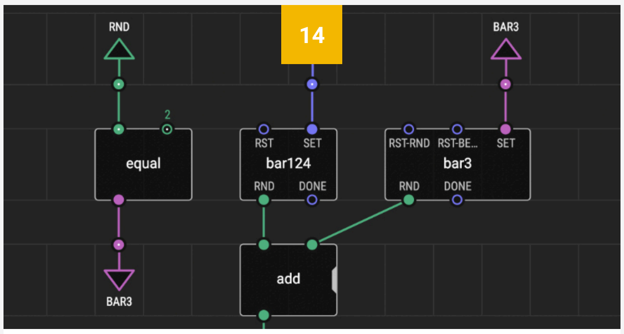
PLAYING BARS 2 AND 4

We also want to initiate *bar124* at the end of bars 1 and 3. Use two *equal* nodes. Connect a *from-bus* node to the first input of each and name them both ‘RND’ so that they receive the round number. Set the second input of one to ‘1’ and the other to ‘3’. Connect the outputs of both nodes to the input pins of the *or* node. Add a *to-bus* node to the output of *or* and name it ‘BAR124’.



BAR124 BUS

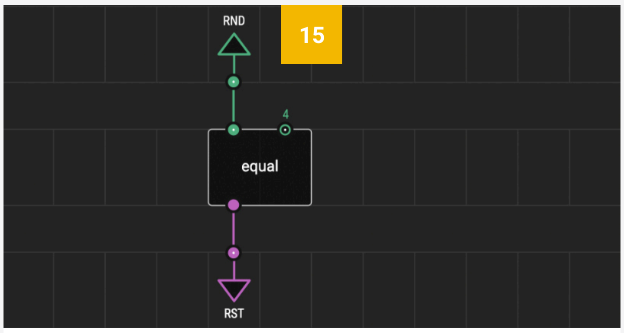
Add a *from-bus* node to the other *any* input (see Step 10). Name it ‘BAR124’. This will set the *bar124* sequence after bars 1 and 3.



PLAYING BAR 3 AND BAR3 BUS

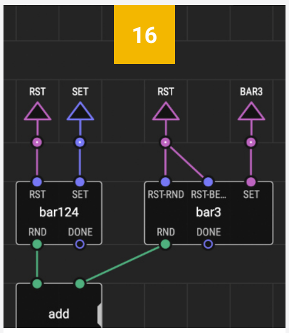
We want *bar3* to play at the end of bar 2. Add a *from-bus* node to the input of the third *equal* node and name it ‘RND’. Set the second input of *equal* to 2. Add a *to-bus* node to the output and name it ‘BAR3’. Add a *from-bus* node to the SET pin of *bar3* and name it ‘BAR3’. This bus will initiate *bar3* after bar 2.

Sequences and Loops



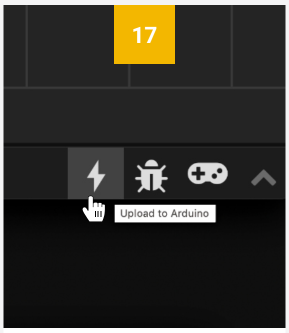
RESETTING THE SEQUENCE

After round 4 we want to reset the sequence so that the round count returns to 0. We added RST pins to our *bar124* and *bar3* nodes for this. Add a *from-bus* node to the first input of the final *equal* node and name it ‘RND’. Set the second input to ‘4’ so the reset happens after bar 4. Add a *to-bus* node to the output pin and name it ‘RST’.



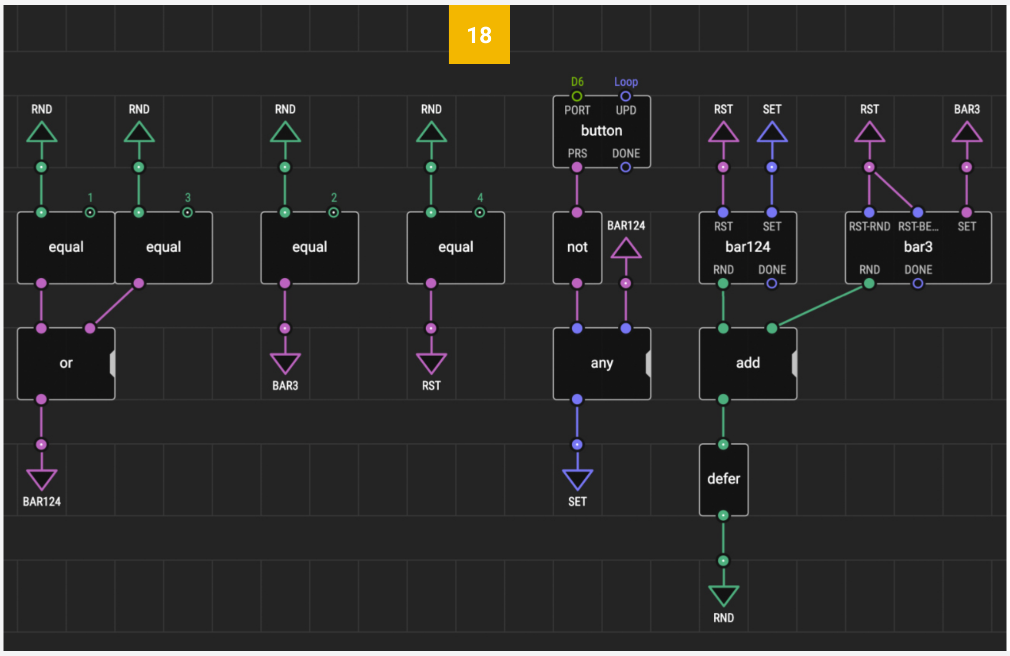
RST BUS

Name both *from-bus* nodes ‘RST’. Add one to the RST pin of *bar124* and one to the RST pins (RST-RND, RST-BEAT) of *bar3*.



UPLOAD AND TEST

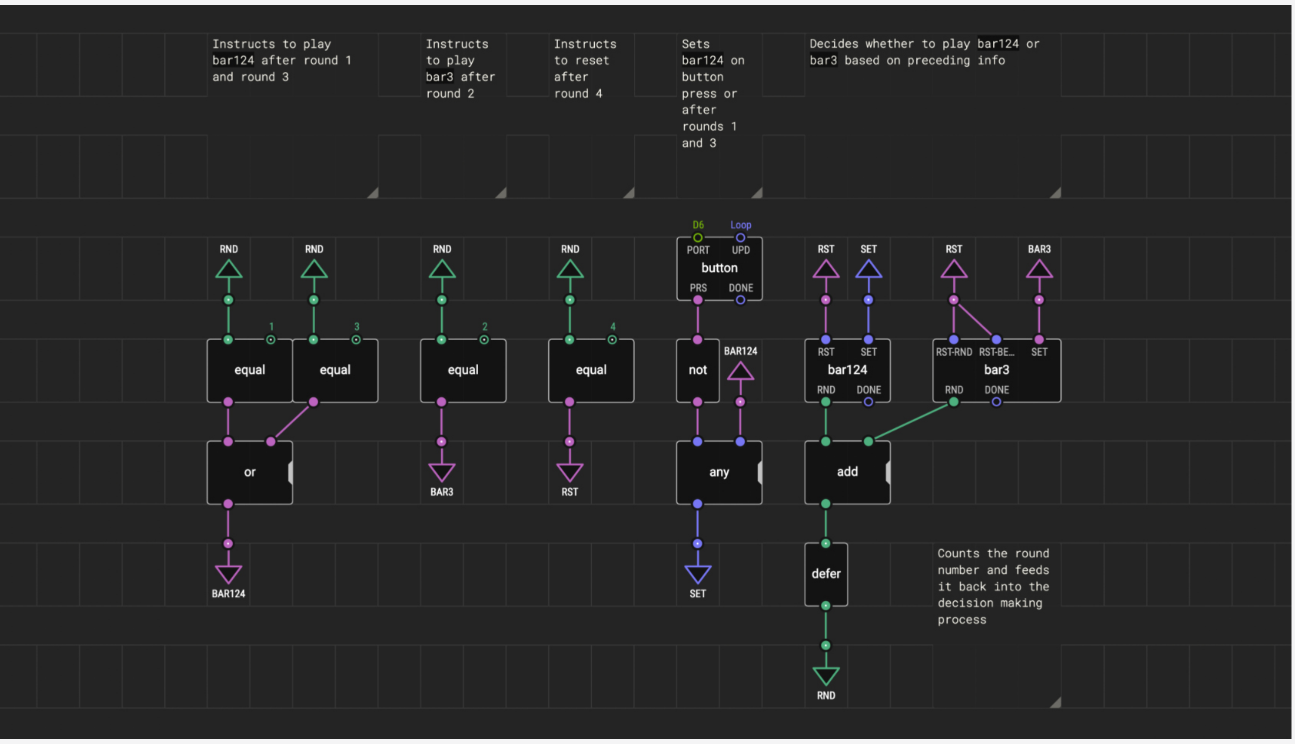
Finally, upload the programme and test it out by pressing the button on your board!



EXPERIMENT!

Congratulations, you’ve completed the final task! Your final patch should look something like this.

Play around with this patch and the nodes you have used. Try creating a different tune. Or try to make the LED flash along in time with the notes. The possibilities are endless now that you understand the core principles.



Instructs to play bar124 after round 1 and round 3

Instructs to play bar3 after round 2

Instructs to reset after round 4

Sets bar124 on button press or after rounds 1 and 3

Decides whether to play bar124 or bar3 based on preceding info

Counts the round number and feeds it back into the decision making process

Comment Boxes

When creating a more complex programme like this it is often useful to include comment boxes, both to help keep track of what you are doing, and to make it easier for others to follow your workflow. This is the XOD equivalent of ‘commenting out’ notes when writing code.

In the example above you can see that comments have been added above each section of the programme to describe what that part of the patch is doing. Try adding your comments to annotate your *play-tune* patch. You can add a comment box by navigating to ‘Edit > Insert Comment” in the menu bar.

You can also add formatting to your XOD comment boxes, for example:

- *Surround text with stars to add bold white text*
- **Surround text with two stars to add bold red text**
- - Use a dash before text to add a bullet list
- 1. Use a number and point before text to add a number list

You can read more about adding comments to document your nodes and XOD ‘markdown’ (formatting) on the XOD website at www.xod.io/docs/guide/documenting-nodes.

Lesson 5: Next Steps

Expanding Your Capability

Case Studies

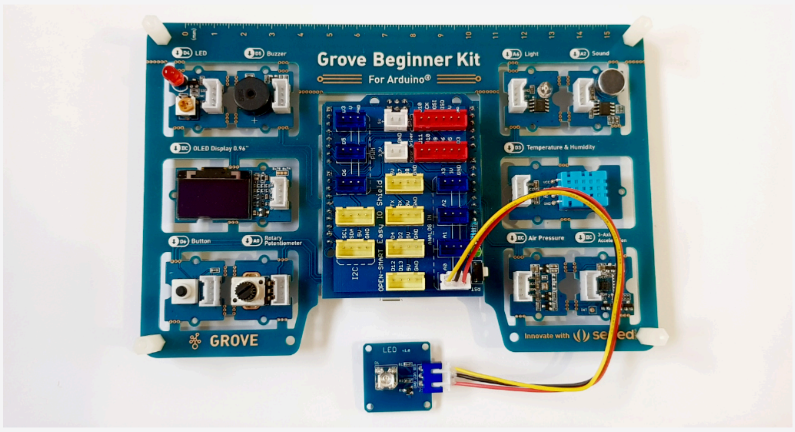
Additional Information

Next Steps

Congratulations, you have completed the No-Code Programming for Biology beginner’s course! Starting from understanding your board and how to programme it, through to building your own nodes and complex sequences, this guide has taken you through how to use each of the onboard devices, as well as how to preform a range of useful functions in XOD.

You should now be comfortable with your board and the XOD software, and should have gained a better understanding of how these tools can be used to create programmes and devices that respond to, and influence, their environment. Once you understand the basic principles of programming microcontrollers such as this, the possibilities for applications are endless.

In this chapter we will provide information about how you can build on these skills to start developing your own custom devices, as well as where to find components, information and help to guide you in your next steps. We will also provide some examples of how previous Biomaker participants have applied these skills to real-world applications to assist with biological research in the lab and field. Finally we provide some additional useful information including an overview of alternative development boards, a list of useful websites, a Grove board cheat sheet, a list of XOD nodes used in this guide, and a glossary of terms.



Open Smart easy-plug LED breakout board connected to the Grove board

OBJECTIVES

By the end of this chapter you should be able to:

- Recall where to find additional components compatible with your board.
- Recognise the different ways to connect new components to your board.
- Locate compatible XOD nodes for new devices.
- Outline some examples of how these skills can be applied to biological research.
- Recall where to find additional information and help with building your own devices.

Expanding Your Capability

Additional Components

The wide variety of low-cost components available for working with Arduino can be daunting, and understanding how to use and connect these components to your board can seem complex. Below we outline some of the simple systems available for connecting new hardware to your board, and on the next page we will discuss how to connect or wire up these components.

GROVE COMPONENTS	The Grove board is made by open hardware company Seeed Studio, that provides a whole series of Grove components that are compatible with the Grove board via simple 'plug-and-play' connectors. These components can be plugged directly into the board using the white plug sockets in the middle of the board, and the cables provided in the kit. You can browse Grove-compatible components on the Seeed Studio website (**www.seeedstudio.com > Shop > Grove**).
M5STACK	M5Stack is another hardware company that sells its own Grove-compatible components which they term 'units'. You can browse M5Stack Grove-compatible components on the M5Stack website (**www.m5stack.com > Store > Unit**).
OPEN SMART COMPONENTS	The Biomaker Expansion kit is comprised of components from an alternative supplier, Open Smart. Open Smart provide both an 'easy-plug' system similar to the Grove system, as well as a wider variety of components which require simple, no-solder wiring. Grove plugs are not compatible with Open Smart plugs, so an expansion shield is required to connect these components to the board (see p77). You can browse Open Smart components on the Open Smart Ali Express store (**www.open-smart.aliexpress.com**).
OTHER COMPONENTS	Grove and Open Smart provide easy-to-use systems for connecting components to your Grove (or any other Arduino board), however, there are also a staggering variety of other suppliers and components available if you introduce a small amount of wiring and soldering. Companies such as Adafruit, SparkFun and Seeed Studio all provide useful electronics modules that can be easily connected to your board by soldering, or using a prototyping shield (see p76).

Seeed Studio (Grove), Adafruit and SparkFun are all based in the USA, and you can buy components directly from their websites, although customs fees and taxes are likely to apply. Fortunately, many of these components are also available from UK suppliers such as Farnell (**www.uk.farnell.com**), Cool Components (**www.coolcomponents.co.uk**), RS Components (**www.uk.rs-online.com**) and Mouser Electronics (**www.mouser.co.uk**).

M5Stack and Open Smart are based in China and you can buy components from their storefronts on Ali Express (**www.open-smart.aliexpress.com** and **www.m5stack.aliexpress.com**).

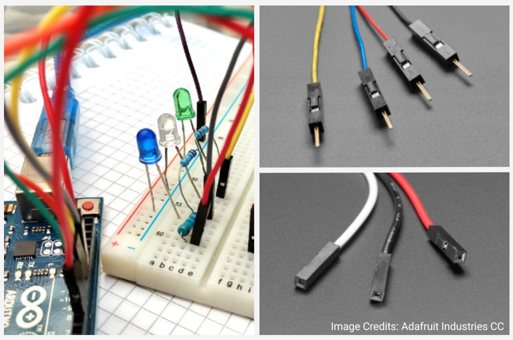
Expanding Your Capability

Connecting Components

There are a number of ways to connect additional components to your board, most of which make use of the board's 'header sockets'. These are the yellow plastic sockets arrayed around the left and right edges of the board's central module. Electronic components can be wired to these sockets (either directly or via a breadboard), added as part of a shield, or connected via a breakout board. Sometimes a combination of these methods is used. Below we explain each of these options.

ELECTRONIC WIRING

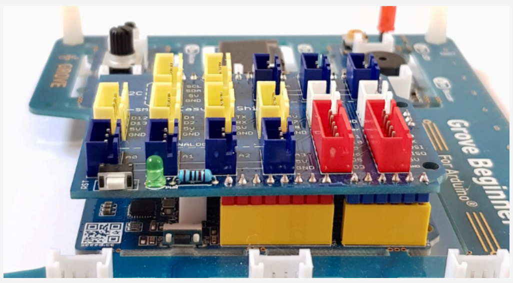
Each of the small header sockets on the board connects to one of the board's pins (see p11). You can wire components directly into these sockets, but more often a breadboard or shield is used. Breadboards are a way to easily make and test electronic circuits. You can learn more about them on the SparkFun website (**www.learn.sparkfun.com/tutorials/how-to-use-a-breadboard**). The hook-up wires used to connect components usually come with two types of ends: male and female. Female ends are like individual sockets (similar to the header sockets), whilst male ends are metal pins, which fit into header and female sockets.



Left:: Wiring to an Arduino using a breadboard
Right: Male (top) and Female (bottom) wires

SHIELDS

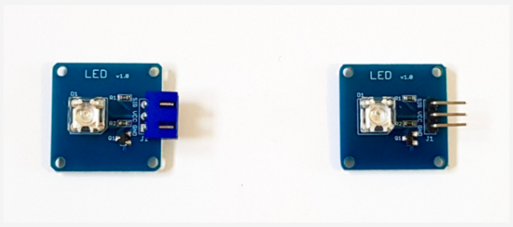
Shields are modular circuit boards that piggyback onto your Arduino to instil it with extra functionality. They use a series of header pins that fit directly into the header sockets. Shields can add a variety of functions, for example allowing the board to communicate via WiFi, adding extra storage capacity, or accessing GPS. Expansion and prototyping shields provide additional sockets or header pins that allow you to expand the capacity of your board and easily connect any number of custom components.



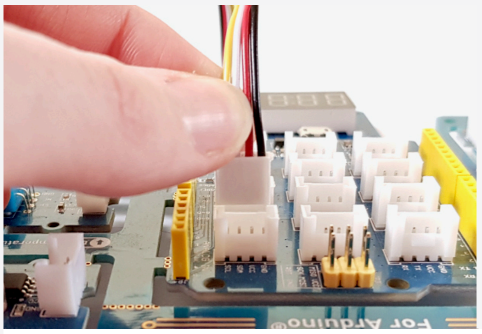
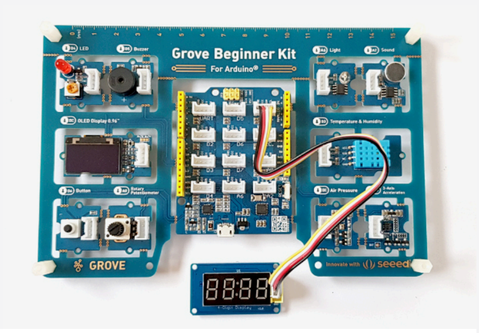
Open Smart shield connected to the Grove board

BREAKOUT BOARDS

A breakout board is similar to a shield but usually with a single or small number of electronic components included. They are used to make wiring of components easier. Each of the modules (LED, buzzer etc.) on your Grove board is essentially a built in breakout board. It is often easiest to purchase components as part of a breakout board and then use electronic wiring or a shield to connect the breakout board to the Arduino.

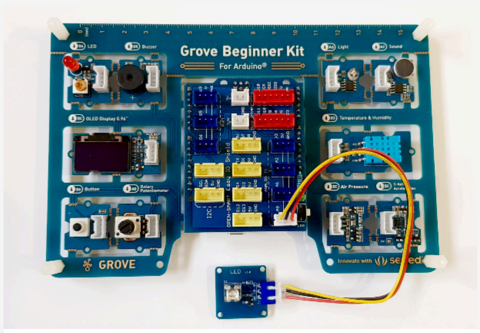
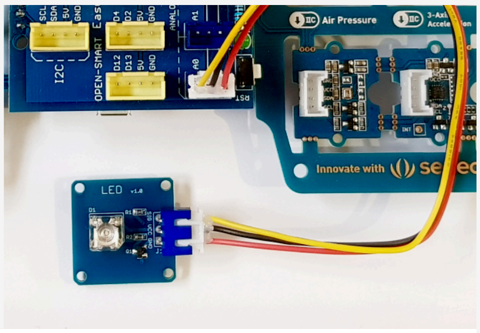


Open Smart easy-plug (left) and standard (right) LED breakout boards

CONNECTING GROVE COMPONENTS

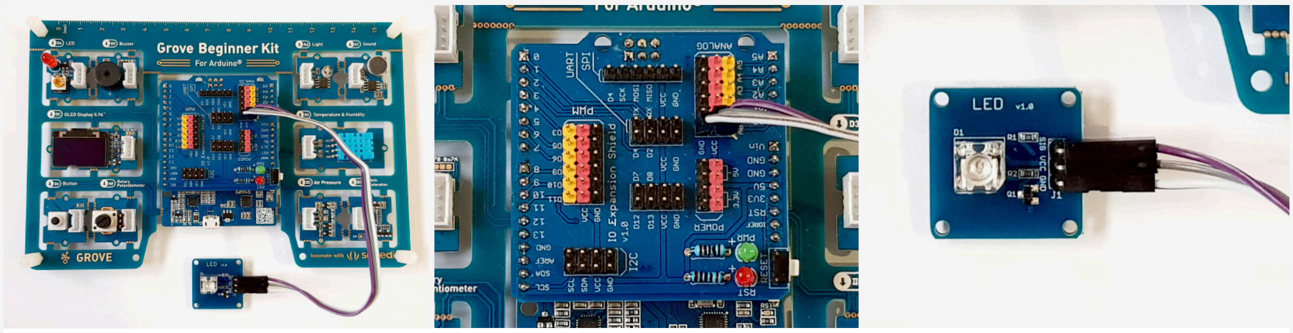
Grove components come as breakout boards with white Grove sockets included. You can use Grove cables (six are included in the Grove Beginner Kit) to connect them to the board. Simply plug one end into the breakout board, and one end into a white socket on the board. Note the name of the port you're using (A0, D5). This is written below the socket. Use A0, A2 and A6 for analog devices, D2-7 for digital devices and I2C for I2C devices.

CONNECTING OPEN SMART EASY-PLUG COMPONENTS

Open Smart 'easy-plug' components are similar to Grove components, but the plugs are not compatible. Adjust for this by plugging the easy-plug expansion shield into the header sockets, then use easy-plug cables to plug components in your breakout board. Again, note the name of the port you are using. Use A0-A3 for analog devices, D3, D5 and D6 for digital devices and I2C for I2C devices. For more information see the Biomaker website (**www.biomaker.org/s/Biomaker-Easy-Plug-Expansion-Kit-Information-Sheet.pdf**).

Expanding Your Capability



CONNECTING STANDARD OPEN SMART COMPONENTS

Standard Open Smart breakout boards come with male pins attached. You can use the Open Smart expansion shield and male-male wires to easily connect these components. Plug the Open Smart expansion shield into the header sockets, then use female-to-female wires to connect the pins on the breakout board to the pins on the expansion shield. Connect each pin to its corresponding pin on the expansion shield. E.g. connect VCC to VCC, GND to GND and SIG to a relevant pin (A0-5 for analog devices and D7,8,12 and 13 for digital devices). For I2C connect SDA to SDA and SCL to SCL. For more information see the Biomaker website (**www.biomaker.org/s/No-Code-Programming-for-Biology-Handbook.pdf**).

Connecting Other Types of Component

Breakout boards from other companies often come without connectors. You can add connectors by soldering header pins to them. Then you can use them like Open Smart components. For an excellent tutorial demonstrating this, see **www.rimstar.org/science_electronics_projects/pin_headers_soldering_cutting_male_female.htm**.

Component Clashes

Because your Grove board already has components connected to many of the pins you may encounter clashes if you try to connect a second device to the same pin. This may or many not disrupt your programme, depending on the devices involved. To avoid this, prioritise use of pins without devices already attached, such as D2. If clashes become an issue you can use a different Arduino, such as the Arduino Rich Uno R3 (provided in the Biomaker expansion kit), or you could detach the central module of the Grove board to use separately. Note that clashes are not an issue with I2C devices as they can be connected to the same pins and identified via their addresses.

Finding XOD Nodes

Once you have found a component that you would like to use you will need to find a XOD node to represent that hardware. The *xod/common-hardware* library provides nodes for a number of commonly used components and many more nodes have been created by the XOD community.

As a standard electronic components are assigned a 'reference designator'. This short combination of letters and numbers identifies specific components, for example, the Grove board uses the 'BMP280' barometer and the 'SSD1306' OLED screen. This system is useful as it is easy to compare components, understand what hardware others are using, and search for complimentary nodes and/or code for programming the hardware.

In XOD, many contributors have created specific libraries to deal with certain pieces of hardware, and you can search these libraries on the XOD website at **www.xod.io/libs**. It is usually easiest to search using the hardware's reference designator. Another useful way to find libraries is to search the XOD forum at **www.forum.xod.io**. This can help you to find relevant node and libraries, as well as identify any common issues others have had when using specific pieces of hardware.

Using Arduino IDE

XOD, and other 'no-code' programming options, are a really useful way to get started working with microcontrollers. They require less up-front learning and offer an alternative and intuitive way of thinking about programming. However, you may also be interested in learning to code - either as an extension to your XOD programming skills, or as an alternative. Arduino provides it own free software for programming: the Arduino IDE, which is available for download at **www.arduino.cc/en/software**. This software uses the C++ language for programming. Grove provides an excellent guide for programming your board using the Arduino IDE at **www.files.seeedstudio.com/wiki/Grove-Beginner-Kit-For-Arduino/res/Grove-Beginner-Kit-For-ArduinoPDF.pdf**.

One advantage of using the Arduino IDE is the vast amount of resources available for working with almost any piece of hardware. Whilst the XOD community is growing fast and new libraries are being added all the time, you may find that certain hardware and components do not yet have compatible XOD nodes. In this case, there is almost always an Arduino IDE library that can be used. Another option is to convert existing Arduino IDE libraries into XOD libraries. Matt Wayland has written an excellent guide detailing how to convert Arduino libraries for use in XOD, available at **www.biomaker.org/s/converting-arduino-to-xod_wayland.pdf**. Further guidance on creating libraries of XOD in C++ is available on the XOD website at **www.xod.io/docs/guide/nodes-for-xod-in-cpp** and **www.xod.io/docs/guide/analog-sensor-node**.

You can also use XOD in combination with Arduino IDE. For example, you could write a programme in XOD, then navigate to 'Deploy > Show Code for Arduino' in the menu bar to export this programme in code. You could then add additional code in Arduino. For example, code to control a device not supported in XOD.

Case Studies

eCO-SENSE: Soil Sensors Powered by Plant Photosynthesis

This project aims to prototype a low-cost soil sensor powered by biophotovoltaics. The device uses an Arduino Uno, temperature sensor, moisture sensor and gas sensor to take measurements of soil conditions, and adds a bluetooth module to send the data wirelessly to a phone or computer. Working together with Dr Paolo Bombelli from the University of Cambridge, the team aims to use biophotovoltaic cells to power their device, allowing it to be used in-situ and in low-resource environments.

The project used a DHT22 temperature and humidity sensor (similar to the DHT11 sensor on the Grove board), an FC-28 moisture sensor, an SGP30 gas sensor and an nRF8001 bluetooth breakout board. Grove compatible alternatives include the Grove DHT11 or DHT22 sensors (use node *xod/dev/dht2x-hygrometer* and socket D2), the Grove Soil Moisture sensor (use node *xod/common-hardware/analog-sensor* and sockets A0/A2/A6 - beware of clashes), the Grove VOC and eCO2 sensor (convert SparkFun or Adafruit Arduino libraries and use I2C socket, address 58h) and the Grove Blueseeed module (use UART socket). Wireless communication is not yet fully supported in XOD, but for a useful tutorial exploring how to send sensor data to your Android phone via bluetooth using Arduino IDE see **www.instructables.com/How-to-Receive-Arduino-Sensor-Data-on-Your-Android**.

You can read more about the eCO-SENSE project on their Hackster page:
www.hackster.io/glen-chua/eco-sense-soil-sensors-powered-by-plant-photosynthesis-be80a2

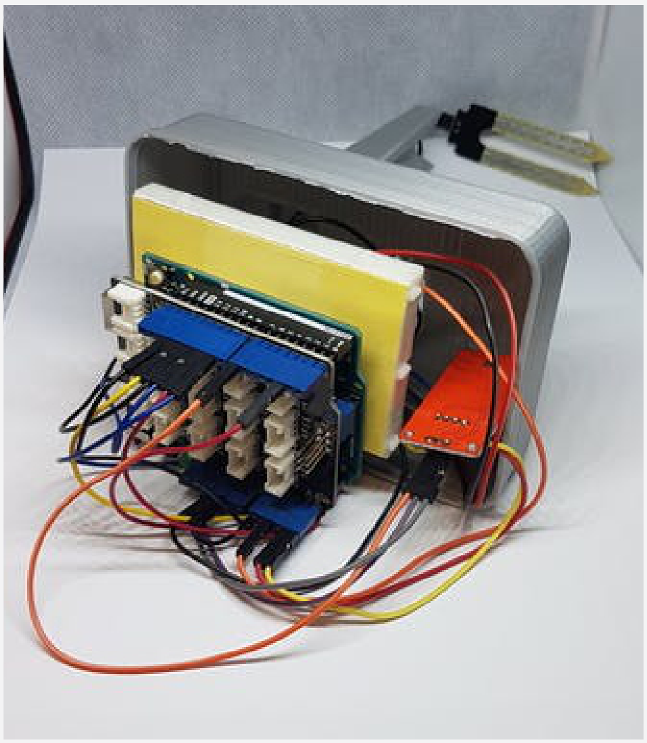
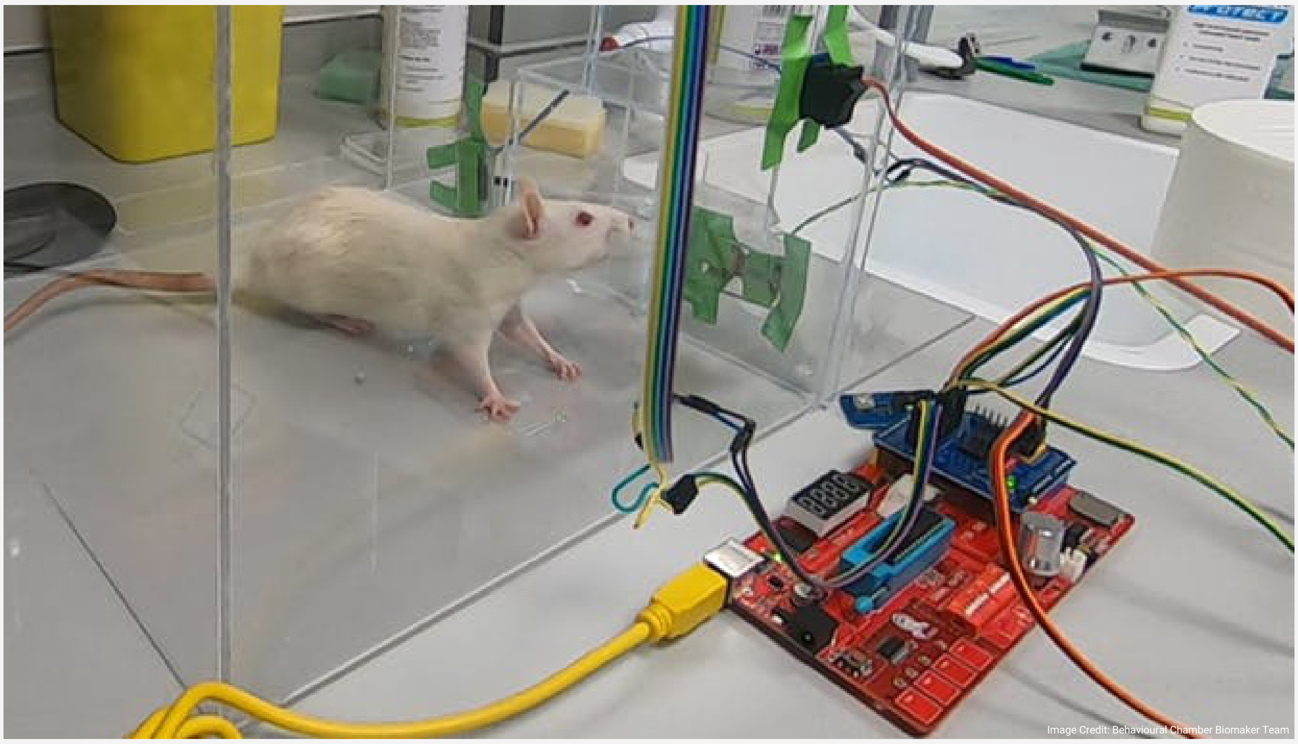


Image Credits: eCO-SENSE Biomaker Team



Behavioural Chamber to Evaluate Rodent Forelimb Grasping

This project uses a light emitter and light sensor to monitor when a rodent moves past a certain threshold, and triggers release of a sugar pellet when it does.

The project uses a red laser pointer and GL5528 light sensor to create a trip sensor that notifies the programme when a rodent has crossed a boundary. This then instructs a ULN2003 motor driver to initiate the custom built pellet dispenser. A count of how many times a rodent has completed this task is shown on an LCD screen. Grove compatible alternatives include the Grove Light Sensor (included on the board, use node *xod/common-hardware/analog-sensor* and sockets A0/A2/A6 - beware of clashes), the Grove I2C Motor Driver (use node *gweimer/h-bridge/h-bridge-2dir* and I2C socket, variable address) and the Grove 16 x 2 LCD (use node *xod-dev/text-lcd/text-lcd-i2c-16x2* and I2C socket, address 3Eh).

You can read more about the behavioural chamber project on their Hackster page:
www.hackster.io/alejandrocarn/a-behavioural-chamber-to-evaluate-rodent-forelimb-grasping-bedb1a

Case Studies

Camera for Monitoring Plant Pollination Events

This project developed a video and time-lapse monitor to record pollinators interacting with plants. The set up included an environmental sensor to measure temperature, humidity and barometric pressure and wrote this data into the image file names on a microSD card for later analysis.

The project used a Raspberry Pi (see p84), a 160° variable focus camera, a BME280 temperature pressure and humidity sensor and a 128x64 OLED screen. Grove compatible alternatives include the Grove BME280 Barometer sensor (use node *emiliosancheza/bme280-sensor/sensor-bme280* and I2C socket, address 76h), the Grove OLED Display 0.96 inch (included on the board, use library *wayland/ssd1306-oled-i2c* and I2C socket, address 3Ch) and the Grove Serial Camera Kit with a Grove SD Card Shield. Camera modules are not yet supported in XOD, but information of how to use the Grove Serial Camera is available at **www.wiki.seeedstudio.com/Grove-Serial_Camera_Kit**. Note that for a high resolution auto-focussing camera, like the one used in this project, Raspberry Pi is a better option than Arduino, as these tasks require high processing power.

You can read more about the plant pollination monitor project on their Hackster page:
www.hackster.io/team-ppi/variable-time-camera-for-monitoring-plant-pollination-events-ad21e7

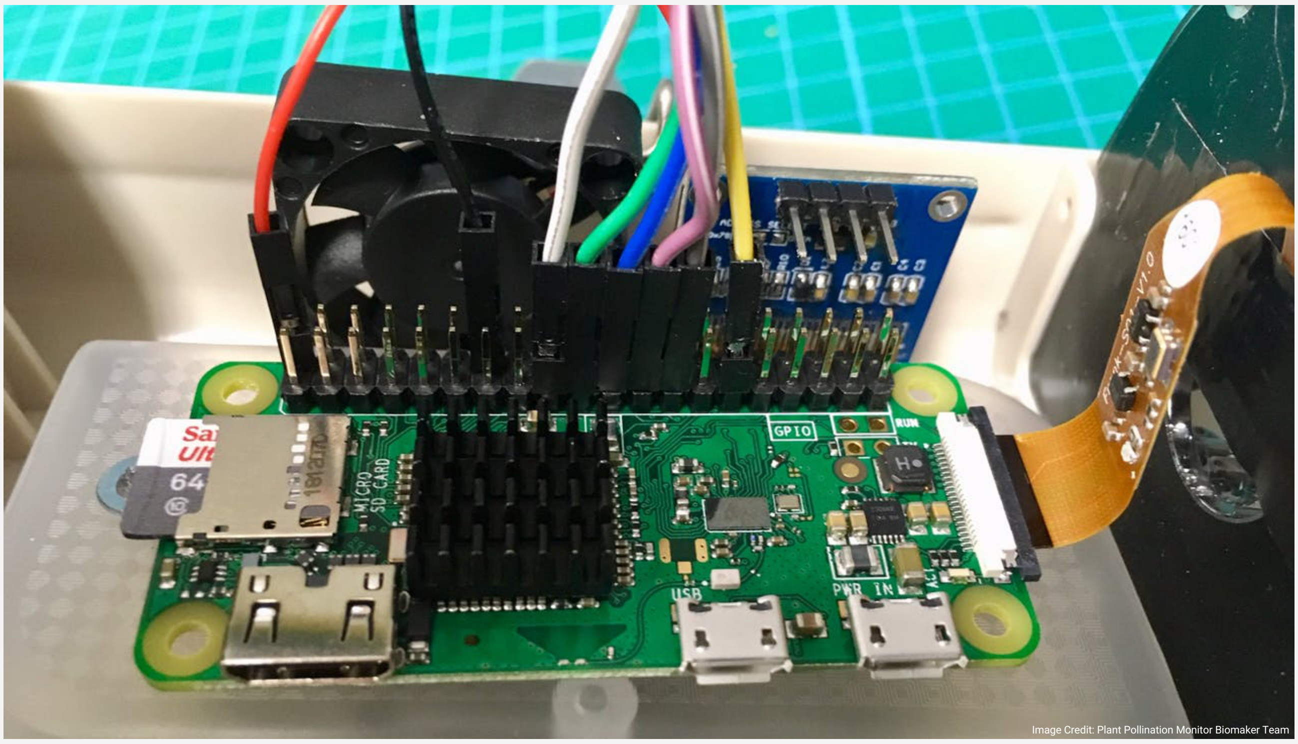


Image Credit: Plant Pollination Monitor Biomaker Team

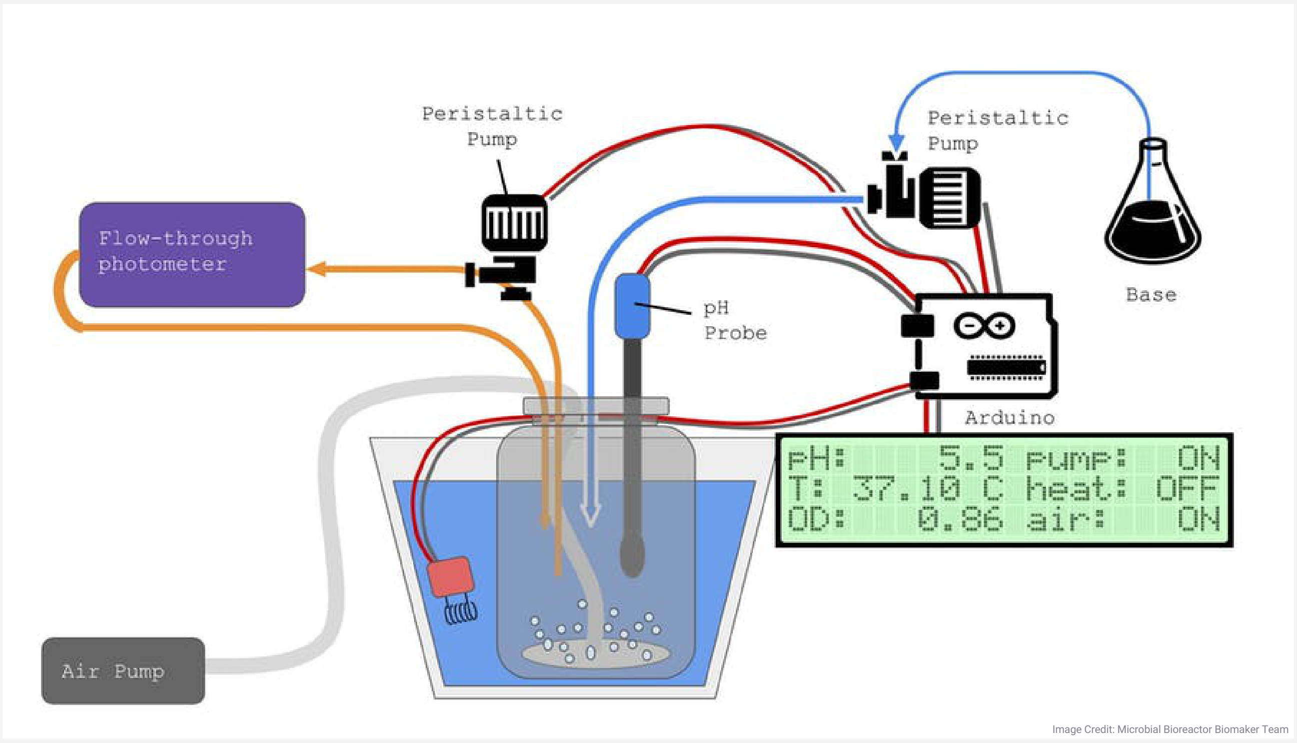


Image Credit: Microbial Bioreactor Biomaker Team

Open Source Microbial Bioreactor

This project aims to develop an open source bioreactor to optimise yield of enzymes producing recombinant proteins for molecular biology. The bioreactor measures optical density of the culture and monitors and regulates the pH, temperature and aeration.

The project uses an LED and photodiode to measure optical density, a pH probe and peristaltic pump to maintain pH and an LCD screen to display the reactor conditions. The team also aims to add a temperature sensor and heating pad to maintain temperature, and an oxygen sensor and agitation device to maintain aeration. Grove compatible alternatives include the Grove Red LED (included on the board, use node *xod/common-hardware/led* and socket D4), the Grove Light sensor (included on the board, use node *xod/common-hardware/analog-sensor* and sockets A0/A2/A6 - beware of clashes), the Grove pH sensor (use node *xod/common-hardware/analog-sensor* and sockets A0/A2/A6 - beware of clashes), the Grove I2C Motor Driver to drive a peristaltic pump (use node *gweimer/h-bridge/h-bridge-2dir* and I2C socket, variable address) and the Grove 16 x 2 LCD (use node *xod-dev/text-lcd/text-lcd-i2c-16x2* and I2C socket, address 3Eh).

You can read more about the microbial bioreactor project on their Hackster page:
www.hackster.io/open-bioeconomy-lab/microbial-bioreactor-d7f61b

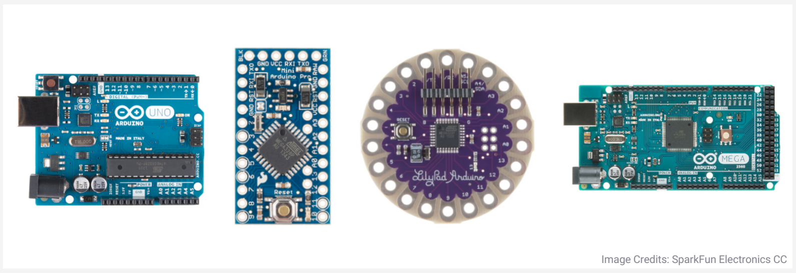
Additional Information

Alternative Development Boards

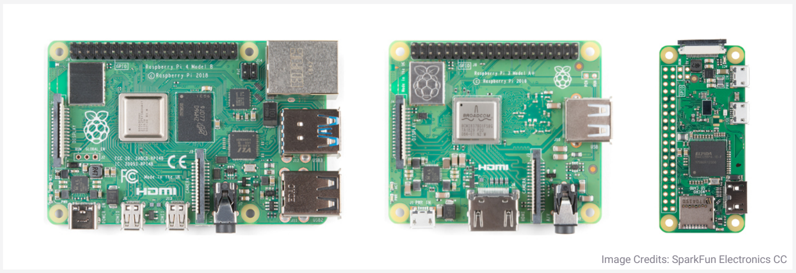
The Grove board is a great place to start with building your own devices, as it is simple to use, low-cost, easily accessible, and comes with a range of useful inbuilt components. However, there are a wide variety of other boards available for getting started with projects like this.

The two most commonly used types of development boards are Arduino and Raspberry Pi, and each of these companies provide a range of boards for different uses. Whilst Arduino boards are microcontrollers that can perform one programme at a time, Raspberry Pi boards are fully-operational computers that can perform multiple tasks at once. A Raspberry Pi may be better for more complex projects, but Arduino boards are easier to use and suited for most simple projects. Note that XOD does not yet support programming of Raspberry Pi boards.

You can find a comparison of Arduino models on the SparkFun website (**www.learn.sparkfun.com/tutorials/arduino-comparison-guide**) and a comparison of Raspberry Pi models on the PiHut website (**www.thepihut.com/blogs/raspberry-pi-roundup/raspberry-pi-comparison-table**).



Arduino boards, left to right: Arduino Uno, Arduino Pro Mini, Lilypad Arduino, Arduino Mega 2560



Raspberry Pi boards, left to right: Raspberry Pi 4 Model B, Raspberry Pi 3 A+, Raspberry Pi Zero W

Useful Links

BIOMAKER	**www.biomaker.org** Collection of technical information, pointers to tutorials and software resources, information about the Biomaker Challenge
NO-CODE PROGRAMMING	**www.biomaker.org/nocode-programming-for-biology-handbook** Information on the No-Code Programming for Biology programme, handbook downloads, tutorials and videos.
HACKSTER	**www.hackster.io/biomaker** Biomaker community hub used for open documentation of Biomaker projects and tutorials.
XOD	**www.xod.io** Download XOD software, libraries, documentation and forum advice.
ARDUINO	**www.arduino.cc** Official repository of Arduino information.
ARDUINO CREATE	**www.create.arduino.cc** Integrated resource for code and project-sharing.
SEEED STUDIO	**www.seeedstudio.com** Hardware supplier for the Biomaker Starter Kit and Grove components.
OPEN SMART	**www.open-smart.aliexpress.com/** Source of hardware for Biomaker expansion kit.
SPARKFUN	**www.sparkfun.com** Good source of practical information about microcontrollers and devices.
ADAFRUIT	**www.adafruit.com** Good source of practical information about microcontrollers and devices.
INSTRUCTABLES	**www.instructables.com/classes/** Classes in many maker skills, including electronics and 3D printing.
FRITZING	**www.fritzing.org** Open source circuit layout and illustration.
PROCESSING	**www.processing.org** Software sketchbook for dynamic graphics and visual arts.
SYNTHETIC BIOLOGY IRC	**www.synbio.cam.ac.uk** Information, news and events from the Synthetic Biology Interdisciplinary Research Centre at the University of Cambridge
OPENPLANT	**www.openplant.org** Information, news and events from the BBSRC-EPSRC Synthetic Biology Research Centre

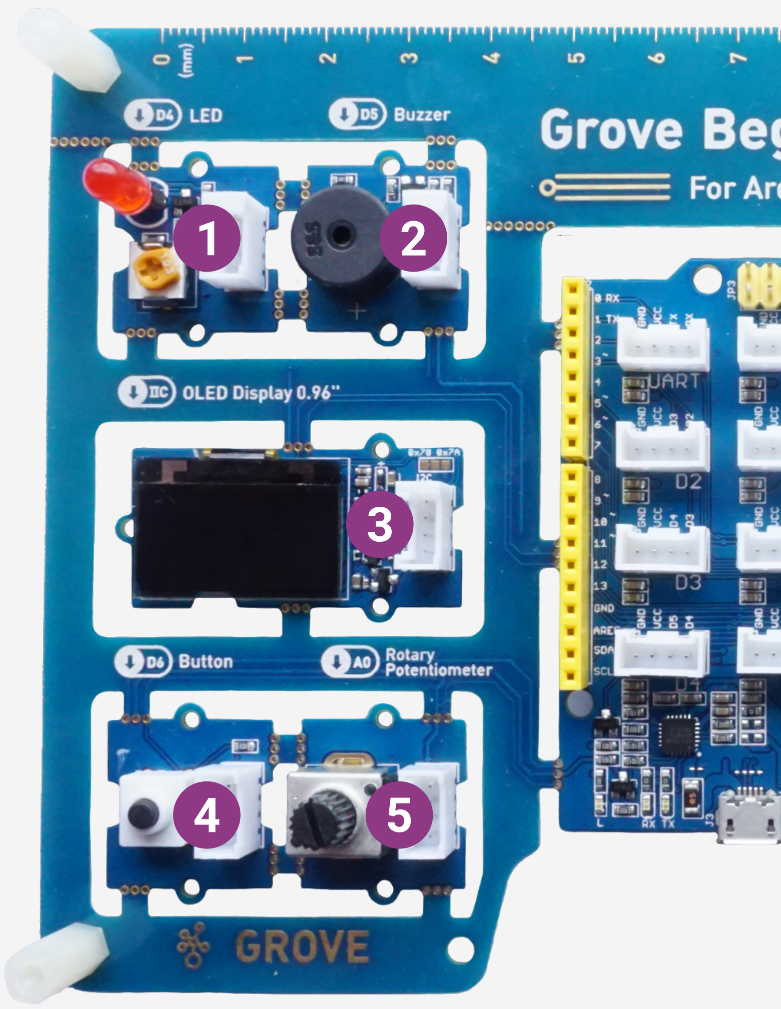
Contacts

synbio@hermes.cam.ac.uk
Dr. Steph Norwood: **san43@cam.ac.uk**
Prof. Jim Haseloff: **jh295@cam.ac.uk**

Additional Information

Grove Board Cheat-Sheet

This cheat-sheet provides a quick guide to which XOD node to use for each of the inbuilt components on the Grove All-In-One Beginner Kit for Arduino.



1 **LED**

Node:	*xod/common-hardware/led*
Settings:	PORT = D4
	LUM = luminance (brightness) between 0-1
	ACT = True
Used in:	**Task 1** (p20-25), **Task 4** (p36-41)

2 **BUZZER**

Node:	*marcoaita/malibrary/buzzer*
Settings:	PORT = D5
	EN = True
	FREQ = 440
Used in:	**Task 2** (p26-29), **Task 9** (p64-71)

3 **OLED SCREEN (SSD1306)**

Nodes:	*wayland/ssd1306-oled-i2c/ssd1306-oled-i2c-device*
	wayland/ssd1306-oled-i2c/clear-display
	wayland/ssd1306-oled-i2c/send-buffer-to-display
Settings:	ADDRESS = 3Ch
	WEIGHT = 128
	HEIGHT = 64
	RESET = -1
Notes:	Add any other nodes from *wayland/ssd1306-oled-i2c* between *clear-display* and *send-buffer-to-display*. Connect *ssd1306-oled-i2c-device* DEV to all DEV pins. Connect *clock* node to *clear-display* UPD then connect each UPD pin in turn.
Used in:	**Task 6** (p49-55), **Task 7** (p56-59), **Task 8** (p60-63)

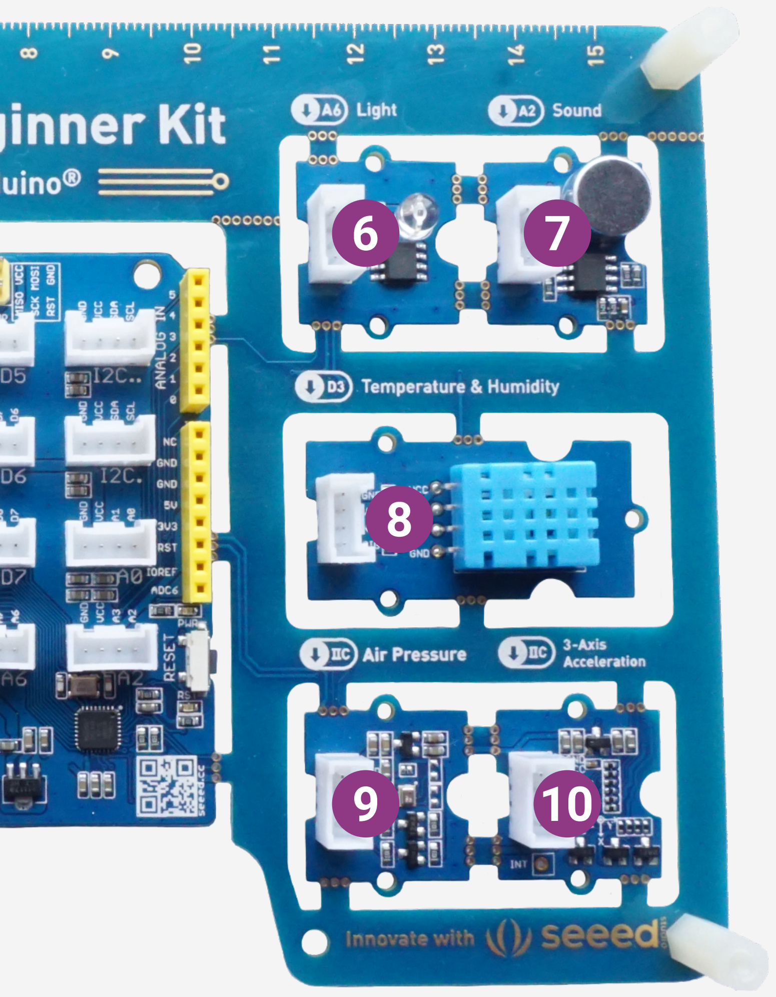
4 **BUTTON**

Node:	*xod/common-hardware/button*
Settings:	PORT = D6
	UPD = Loop
Notes:	*button* is automatically on and turns off with a press. Use a *not* node to invert this.
Used in:	**Task 2** (p26-29)

5 **ROTARY POTENTIOMETER**

Node:	*xod/common-hardware/pot*
Settings:	PORT = A0
	UPD = Loop
Used in:	**Task 2** (p26-29)

The guide suggests a node for each component as well as some standard settings. Other nodes and settings can also be used, and we highly encourage playing around with node settings and searching for new nodes.



6

LIGHT SENSOR

Node:	*wayland/analog-read-no-port-check/ analog-read-no-port-check*
Settings:	PORT = A6
	UPD = Loop
Used in:	**Task 8** (p60-63)

7

SOUND SENSOR

Node:	*xod/common-hardware/analog-sensor*
Settings:	PORT = A2
	UPD = Loop
Used in:	**Task 6** (p49-55)

8

TEMPERATURE AND HUMIDITY SENSOR (DHT11)

Node:	*xod-dev/dht/dht11-hygrometer*
Settings:	PORT = D3
	UPD = Connect *clock* node
Notes:	Setting UPD to 'Loop' can cause errors.
Used in:	**Task 3** (p33-35)

9

AIR PRESSURE SENSOR (BMP280)

Node:	*wayland/bmp280-barometer/barometer-thermometer*
Settings:	MODE = 03h
	OST = 02h
	OSP = 05h
	FILT = 04h
	STDBY = 04h
	UPD = Loop
Used in:	**Task 5** (p42-45)

10

3-AXIS ACCELERATION SENSOR (LIS3DH)

Node:	*wayland/lis3dh-accelerometer/accelerometer*
Settings:	ADDR = 19h
	RATE = 07h
	RANGE = 00h
Used in:	**Task 7** (p56-59)

Additional Information

List of Nodes Used

accelerometer	wayland/lis3dh-accelerometer/accelerometer	**Task 7** p56-59
add	xod/core/add	**Task 9** p64-71
analog-read-no-port-check	wayland/analog-read-no-port-check/ analog-read-no-port-check	**Task 8** p60-63
analog-sensor	xod/common-hardware/analog-sensor	**Task 6** p49-55
		Case Studies p80-83
and	xod/core/and	**Task 8** p60-63
any	xod/core/any	**Task 8** p60-63
		Task 9 p64-71
bar124	custom node created in...	**Task 9** p64-71
bar3	custom node created in ..	**Task 9** p64-71
barometer-thermometer	wayland/bmp280-barometer/barometer-thermometer	**Task 5** p42-45
between	e/comparison/between	**Task 9** p64-71
button	xod/common-hardware/button	**Task 1** p20-25
		Task 2 p26-29
		Task 9 p64-71
buzzer	marcoaita/malibrary/buzzer	**Task 2** p26-29
buzzer-timed	marcoaita/malibrary/buzzer-timed	**Task 9** p64-71
clear-display	wayland/ssd1306-oled-i2c/clear-display	**Task 6** p49-55
		Task 7 p56-59
click-detector	wayland/lis3dh-accelerometer/click-detector	**Task 7** p56-59
clock	xod/core/clock	**Task 1** p20-25
		Task 4 p36-41
		Task 6 p49-55
		Task 7 p56-59
		Task 9 p64-71
concat	xod/core/concat	**Task 5** p42-45
		Task 8 p60-63
count	xod/core/count	**Task 4** p36-41
		Task 6 p49-55
		Task 9 p64-71
dec-to-2digits	cesars/utils/dec-to-2digits	**Task 5** p42-45
dec-to-4digits	cesars/utils/dec-to-4digits	**Task 5** p42-45
defer	xod/core/defer	**Task 9** p64-71
delay	xod/core/delay	**Task 9** p64-71
dht11-hygrometer	xod-dev/dht/dht11-hygrometer	**Task 3** p33-35
dht2x-hygrometer	xod/dev/dht2x-hygrometer	**Case Studies** p80-83
draw-circle	wayland/ssd1306-oled-i2c/draw-circle	**Task 6** p49-55
		Task 7 p56-59
draw-line	wayland/ssd1306-oled-i2c/draw-line	**Task 6** p49-55
draw-pixel	wayland/ssd1306-oled-i2c/draw-pixel	**Task 6** p49-55
draw-rectangle	wayland/ssd1306-oled-i2c/draw-rectangle	**Task 6** p49-55
draw-rounded-rectangle	wayland/ssd1306-oled-i2c/draw-rounded-rectangle	**Task 6** p49-55
draw-text	wayland/ssd1306-oled-i2c/draw-text	**Task 6** p49-55
draw-triangle	wayland/ssd1306-oled-i2c/draw-triangle	**Task 6** p49-55
equal	xod/core/equal	**Task 9** p64-71
flip-flop	xod/core/flip-flop	**Task 4** p36-41
flip-n-times	xod/core/flip-n-times	**Task 4** p36-41
format-number	xod/core/format-number	**Task 5** p42-45
from-bus	xod/patch-nodes/from-bus	**Task 7** p56-59
		Task 8 p60-63
		Task 9 p64-71
greater	xod/core/greater	**Task 8** p60-63
h-bridge-2dir	gweimer/h-bridge/h-bridge-2dir	**Case Studies** p80-83
if-else	xod/core/if-else	**Task 8** p60-63
		Task 9 p64-71

input-boolean	xod/patch-nodes/input-boolean	Task 6 p49-55	Task 9 p64-71
input-byte	xod/patch-nodes/input-byte		Task 6 p49-55
input-number	xod/patch-nodes/input-number	Task 6 p49-55	Task 7 p56-59
			Task 8 p60-63
input-port	xod/patch-nodes/input-port	Task 6 p49-55	Task 8 p60-63
input-pulse	xod/patch-nodes/input-pulse	Task 6 p49-55	Task 8 p60-63
			Task 9 p64-71
input-string	xod/patch-nodes/input-string		Task 6 p49-55
input-t1	xod/patch-nodes/input-t1		Task 6 p49-55
invert-display	wayland/ssd1306-oled-i2c/invert-display		Task 6 p49-55
join	xod/core/join		Task 5 p42-45
led	xod/common-hardware/led	Task 1 p20-25	Task 4 p36-41
			Case Studies p80-83
less	xod/core/less		Task 8 p60-63
light-sensor	custom node created in...		Task 8 p60-63
map	xod/math/map	Task 2 p26-29	Task 7 p56-59
multiply	xod/core/multiply		Task 8 p60-63
nand	xod/core/nand		Task 8 p60-63
nor	xod/core/nor		Task 8 p60-63
not	xod/core/not	Task 1 p20-25	Task 2 p26-29
			Task 9 p64-71
number-split-to-digit	gst/number-split-to-digit/number-split-to-digit		Task 5 p42-45
or	xod/core/or	Task 8 p60-63	Task 9 p64-71
output-boolean	xod/patch-nodes/output-boolean		Task 6 p49-55
output-byte	xod/patch-nodes/output-byte		Task 6 p49-55
output-number	xod/patch-nodes/output-number	Task 6 p49-55	Task 9 p64-71
output-port	xod/patch-nodes/output-port		Task 6 p49-55
output-pulse	xod/patch-nodes/output-pulse	Task 6 p49-55	Task 9 p64-71
output-string	xod/patch-nodes/output-string	Task 6 p49-55	Task 8 p60-63
output-t1	xod/patch-nodes/output-t1		Task 6 p49-55
pot	xod/common-hardware/pot		Task 2 p26-29
pulse-on-true	xod/core/pulse-on-true	Task 8 p60-63	Task 9 p64-71
rotate-display	wayland/ssd1306-oled-i2c/rotate-display	Task 6 p49-55	Task 7 p56-59
round	xod/math/round		Task 7 p56-59
send-buffer-to-display	wayland/ssd1306-oled-i2c/ send-buffer-to-display	Task 6 p49-55	Task 7 p56-59
sensor-bme280	emiliosancheza/bme280-sensor/sensor-bme280		Case Studies p80-83
square-wave	xod/waves/square-wave		Task 4 p36-41
ssd1306-oled-i2c-device	wayland/ssd1306-oled-i2c/ssd1306-oled-i2c-device	Task 6 p49-55	Task 7 p56-59
text-lcd-i2c-16x2	xod-dev/text-lcd/text-lcd-i2c-16x2		Case Studies p80-83
to-bus	xod/patch-nodes/to-bus	Task 7 p56-59	Task 8 p60-63
			Task 9 p64-71
tweak-boolean	xod/debug/tweak-boolean		Task 3 p33-35
tweak-byte	xod/debug/tweak-byte		Task 3 p33-35
tweak-color	xod/debug/tweak-color		Task 3 p33-35
tweak-number	xod/debug/tweak-number	Task 3 p33-35	Task 7 p56-59
			Task 8 p60-63
tweak-pulse	xod/debug/tweak-pulse	Task 3 p33-35	Task 4 p36-41
tweak-string-16/32/64/128	xod/debug/tweak-string-16/32/64/128		Task 3 p33-35
watch	xod/debug/watch	Task 3 p33-35	Task 4 p36-41
		Task 5 p42-45	Task 6 p49-55
			Task 8 p60-63
write-dot-to-oled	custom node created in...		Task 7 p56-59
write-text-to-oled	custom node created in...	Task 6 p49-55	Task 8 p60-63
xor	xod/core/xor		Task 8 p60-63

Note: for items in grey examples of their use are given, but a full demonstration is not provided in this guide.

Glossary

ACCELEROMETER	Accelerometers measure the acceleration of an object, i.e. any change in velocity (speed and direction). A 3-axis accelerometer, like the one included in the Grove board, can sense when the board is moved or tilted in any direction (X, Y and Z axes).
ACTUATOR	Actuators are output devices that convert electronic signals into mechanical movement. For example motors, belts or pumps.
ADAFRUIT	Adafruit Industries is an open-source hardware company the provides electronic components, tools, accessories and learning resources. Their components are compatible with Arduino and Raspberry Pi hardware.
ANALOG	Analog signals, unlike digital signals, are continuous and and can take an infinite number of values. Analog devices measure continuous variables, such as sound or light intensity. Many environmental sensors are analog devices. Computers use digital, rather than analog signals, so analog signals must first be converted to digital signals by the microcontroller.
ARDUINO	Arduino is an open-source electronics company. They make openly available programming software and low-cost hardware to allow anyone to get started making their own interactive electronics projects.
ARDUINO IDE	The Arduino Integrated Development Environment (IDE) is Arduino’s free software for programming Arduino boards using the C++ programming language. It is an alternative to the XOD IDE, and can be used alongside XOD (see p79).
ARDUINO UNO	The Arduino UNO was the first USB-based Arduino board, consisting of a microcontroller chip, printed circuit board (PCB) and a series of digital and analog input-output pins to connect shields and external hardware. The Arduino UNO R3 is the third revision of this board, and is what the Seeduino and Grove board are based upon.
ATMEGA328P	The Atmega328P is the microcontroller chip used in the latest versions of the Arduino board, including the Grove board.
BAROMETER	A barometer device measures air pressure, and can therefore be used to monitor or forecast weather, or to measure altitude.
BIOMAKER	Biomaker is an initiative run by the University of Cambridge Synthetic Biology IRC and OpenPlant that focuses on training and providing funding and resources for researchers interested in the intersection of biology, engineering and computing. Biomaker activities include the annual Biomaker Challenge and No-Code Programming for Biology training.
BREADBOARD	Breadboards are simple devices for developing and prototyping electrical circuits, without the need for soldering. They consist of rows of sockets that are connected via electrical wiring. Components can be plugged directly into these sockets using wires or metal pins. Any devices plugged into the same row of sockets will be connected together.
BREAKOUT BOARDS	Breakout boards are used to make wiring of electronic components easier. They usually consist of a small PCB board with a single, or small number of electronic components attached. For example, an LED or OLED Screen breakout board. They can be easily attached to a development board via wires, pins and sockets, or plugs.

BUS (XOD)	In computing a bus is a communication system that transfers information between computers, or between different parts of the same computer. In XOD, we can use buses to transfer information between one part of our patch and another, without having to connect them via links. This is done using the *to-bus* and *from-bus* nodes to send information to, and receive information from a particular bus.
COMMUNICATION PROTOCOL	A communication protocol is the method by which two or more electronic components exchange data. Ethernet, wi-fi and bluetooth are all examples of communication protocols. Different components use different communication protocols (e.g. analog, digital, I2C) and so will need to be connected to the Arduino board in different ways.
DEBUGGER (XOD)	The Debugger is XOD's simulator function. It can be used to simulate your patch, or to edit and 'debug' your patch after upload to the board. You can start the debugger using the 'Simulate' button, or using the 'Upload and Debug' button to upload the patch at the same time. In debugger mode, you can use tweak and watch nodes to edit and monitor your patch in real-time (see **Task 3** p33-35).
DEVELOPMENT BOARD	A microcontroller development board, like the Grove Arduino board, houses a microcontroller chip on a small PCB board along side some additional parts and connections making it easy for anyone to programme and connect components to a microcontroller at home. Development boards are intended to be cheap and easily accessible, and are often used for developing prototypes and custom instruments.
DEVELOPMENT HOST	A development host is the device you will use to write and develop the programme you want to install on the development board. To programme the Grove board you will use a laptop or PC as the development host.
DIGITAL	Digital signals, unlike analog signals, can only take finite and discrete values. For example, an LED can be 'on' or 'off'. Digital signals can be made to behave in a similar way to analog signals, for example, you can change the brightness of an LED, but ultimately there are a finite and discrete number of values that the brightness of an LED can take. Computers use digital signals, and most electronic components are digital. For example, screens, buttons, and some types of sensor.
GROVE	Grove is toolkit of easy-to-use Arduino-compatible electronics. It uses a 'plug-and-play' system of modules that can be easily fitted together to build custom devices. It is developed by the company Seeed Studio.
HACKSTER	Hackster is an online platform for recording and sharing electronics projects. It provides a simple way to document and browse projects, and has a large community of contributors, including companies such as Arduino and Seeed Studio.
HEADER SOCKETS	The header sockets (also known as female headers) are connectors that are wired to the PCB board and provide "female" sockets. They give us a way to easily connect external components to the board, either via male-to-male hook-up wires, or via an expansion shield.
HOOK-UP WIRES	Hook-up wires (also known as jumper wires or jumper cables) are used to connect components to the Arduino board and come in several different types. Female-to-female hook-up wires have connector sockets at each end that plug into metal pins on components, on the Arduino board, and on shields. Male-to-male wires, which have metal pins on each end that fit into female sockets, header sockets or breadboards. Male-to-female wires that have a female socket at one end and a male pin at the other end. Hook-up wires can also come pre-fitted with plugs to fit into compatible sockets. For example Grove plugs or Open Smart (JST-XH) plugs.
HYGROMETER	Hygrometer devices are used to measure humidity, i.e. the amount of water vapour present in the air or in soil.
I2C	Inter-integrated circuits (I2C) are a digital communication protocol used to communicate with multiple devices at once. With I2C communication several devices can be connected to the same pin of the microcontroller, and each device is given a "name" digitally (known as an address). Addresses are written as XXh, with XX being a two digit code of numbers and letters. For example 19h or 3Ch.
INSPECTOR (XOD)	In XOD, the Inspector pane is the place where a nodes can be edited. For example, you can change the parameters of a node's pins, change the name of a node, or add a description. You must click on a node in the patch for these options to appear. The Inspector pane appears on the left hand side of the screen below the Project Browser pane, and can be toggled on and off using the slider bar button in the top left, or by navigating to 'View > Toggle Inspector' in the menu bar.
LED	A light-emitting diode (LED) is a bright, low-power light source that generates light by passing a current through a semiconductor diode.

Glossary

LIBRARY (XOD)	In XOD (and in other coding software such as Arduino), libraries are collections of ready-to-use nodes (or code). They are often designed to help you use a specific piece of hardware (e.g. wayland/bmp280-barometer) or as a collection of nodes with similar functions (e.g. *xod/math*). The XOD IDE has several libraries pre-installed, but you can add more libraries using the ‘Add Library’ button (books with a + symbol, in the Project Browser) or by navigating to ‘File > Add Library...’ in the menu bar.
M5STACK	M5Stack is a hardware company which provides it’s own wi-fi and bluetooth enabled development system, as well as a series of Grove-compatible components called ‘units’.
MICROCONTROLLER	A microcontroller is a small low-power computer embedded into a device. In contrast to a general purpose computer like a laptop or PC, microcontrollers are often designed to complete one task and run one specific programme. The Grove Arduino board contains a reprogrammable microcontroller so you can upload your own programme on to the board and create your own devices.
NO-CODE PROGRAMMING	Node-code programming is an increasingly popular mechanism allowing people to programme hardware and software using a graphical user interface, rather than requiring them to learn and write text-based code. The Biomaker No-Code Programming for Biology initiative has adopted no-code and low-code programming as a way to train biologists, and others with little or no coding experience, to build their own custom devices.
NODE (XOD)	In XOD nodes are used as "a visual representation of a physical device or function". They can represent an electronic component, a mathematical function or any number of other functions that a computer can perform. They appear on a patch as a dark grey box outlined in white, with the name of the node printed in the middle. They may have small coloured circles (pins) on the top and bottom which represent inputs and outputs.
OLED SCREEN	Organic light emitting diode (OLED) screens are an alternative to LCD screens used mainly for TVs. Instead of having a backlight to illuminate pixels, each pixel can produce its own light. This can improve contrast.
OPEN SMART	OpenSmart is a group of technology companies interested in the production and development of open-source hardware. They are based in Shenzhen, China. Open Smart components are used in the Biomaker Expansion Kit.
PATCH (XOD)	In XOD a patch is the working area in which a programme is built. It is similar to a document or source file in other systems, but instead of text code the patch is built with nodes.
PCB	A printed circuit board (PCB) is composed of a thin fibreglass board with conductive tracks of copper etched on the surface or between the layers. They are used to connect electrical components, which are usually soldered onto the board.
PHOTORESISTOR	A photoresistor or light dependent resistor (LDR) is a component that is sensitive to light. When light falls upon it then the resistance changes, and this change is used as an electronic signal.
PIEZOELECTRIC BUZZER	Piezo buzzers are simple devices that can generate basic beeps and tones. They work by using a piezo crystal, a special material that changes shape when voltage is applied to it. If the crystal pushes against a diaphragm, it can generate a pressure wave which the human ear picks up as sound.

PIN (ARDUINO)	In electronics, "pin" is used to refer to the electrical contacts on a component, i.e. the parts of a component that are used to connect to other components. On the Grove board, the microcontroller chip has a number of pins that are connected to both the components on the board, and to the Grove sockets and header sockets of the central module, which allows them to communicate with additional components. In XOD the Arduino pins are referred to as "Ports" to avoid confusion with XOD pins.
PIN (XOD)	In XOD "pin" is used to refer to the inputs and outputs associated with a specific node. They appear as small round circles on the top (input pins) and bottom (output pins) of a node. Pins are coloured according to their data type.
POTENTIOMETER	Potentiometers (often shortened to "pot") are variable resistors that allow you to alter the resistance, and therefore the current flowing through a circuit, without the need to reprogram the device. They are often found in the form of a knob, slider or screw.
PROJECT BROWSER (XOD)	The XOD Project Browser is where you will find the current project you are working and libraries you have installed. Under the 'My Project/[name of your project]' dropdown you will find all of the patches (or files) in your project. Below that is a list of libraries. Clicking on the dropdown button of a library will allow you to browse the nodes in that library. Dragging a node from the Project Browser into the patch will add that node to your patch. At the top of the Project Browser are the 'New Patch' and 'Add Library' buttons. The Project Browser pane appears on the top left of the screen, and can be toggled on and off using the hub button in the top left, or by navigating to 'View > Toggle Project Browser' in the menu bar.
QUICK HELP (XOD)	The Quick Help pane in XOD is where you can find information about a node and it's pins. When you click on a node information about that node and it's pins will appear in the Quick Help pane. The Quick Help pane appears on the top right of the screen, and can be toggled on and off using the question mark button in the top right, or by navigating to 'View > Toggle Quick Help' in the menu bar.
SEEED STUDIO	Seeed Studio is an open-source hardware company. They developed the Seeduino Lotus development board (based on the Arduino Uno R3 development board) and the Grove system of components which use plugs to easily connect modules.
SHIELD	Shields are modular circuit boards that piggyback onto your Arduino to instil it with extra functionality. Shields can have specific functions, such as a wifi shield that will allow your board to transfer information via wifi, or can have more general functions, like an expansion or prototyping shield. These allow you to easily connect any number of custom components. Some shields can be stacked on top of one another to create combinations of modules and functions.
SPARKFUN	SparkFun is an open-source hardware company that provide a range of development boards and components, as well as tutorials and learning resources on programming, electronics and working with hardware.
TERMINALS (XOD)	XOD terminal nodes (input and output nodes) are used to allow a patch to communicate with 'the-outside-world'. Adding terminal nodes to a patch will allow that patch to be used as a node in other patches, with the names and types of the terminals corresponding to the names and types of the pins on your new node.
USB DRIVER	A USB driver is a piece of software that allows your computer's operating system to communicate with external hardware, such as a hard drive or a microcontroller development board like the Grove board. The Grove board uses the CP210 driver from Silicon Labs, and you may need to download this driver in order to use your board.
XOD	XOD is an open-source software company that provides the the XOD Integrated Development Environment (IDE). The XOD IDE is free software that allows you to programme Arduino-based development boards using visual programming rather than text-based coding. XOD software uses graphical nodes to represent functions, and nodes are connected together to visualise data flow and programme hardware.

Index

Acknowledgements

AUTHORS

Stephanie Norwood
Jim Haseloff

IMAGES

Stephanie Norwood
SparkFun Electronics, Inc.
Adafruit Industries, LLC
Biomaker Challenge Participants

DESIGN

Stephanie Norwood

HARDWARE

Seeed Technology Co.,Ltd.
Arduino S.r.l.
Adafruit Industries, LLC
M5stack-store
OpenSmart Tech.
Spark Fun Electronics, Inc.

SOFTWARE

XOD, Inc.
Arduino S.r.l.
Hackster, Inc.

XOD LIBRARY CREATORS

Matt Wayland
Marco Aita
Cesar Sosa
XOD user: gst
XOD user: e
XOD user: gweimer

FUNDERS AND SPONSORS

Biotechnology and Biological Sciences Research Council (BBSRC)
Engineering and Physical Sciences Research Council (EPSRC)
Natural Environment Research Council (NERC)
OpenPlant
Synthetic Biology Interdisciplinary Research Centre (SynBio IRC), University of Cambridge
National Science Foundation (NSF)

CONTACTS

Stephanie Norwood
synbio@hermes.cam.ac.uk

Jim Haseloff
jh295@cam.ac.uk

Designed for those with little to no experience working with coding or hardware, this guide makes use of free open-source software and low-cost hardware to introduce you the principles behind making your own instruments.

Learn how to:

- Understand and control an Arduino board
- Programme without using code
- Use simple electronic devices such as screens and sensors
- Build your own devices for use in biological research

UKRI Biotechnology and Biological Sciences Research Council
UKRI Engineering and Physical Sciences Research Council
UKRI Natural Environment Research Council
NSF
OpenPlant
SynBio IRC
UNIVERSITY OF CAMBRIDGE