Maker Advent Calendar Day #5: Hear my Code!

By The Pi Hut • Dec 5, 2022 • 9 comments

Welcome to day five of your 12 Projects of Codemas Advent Calendar. Today we'll be making sounds with our code using the custom buzzer you've just discovered in your box!

The buzzer is nice and easy to wire into our circuit, using just two pins with no other components required, yet can add SO much to a project! Adding audible feedback for errors, button presses and other actions gives your project a whole new dimension.

Let's make some noise!

Box #5 Contents

In this box you will find:

- **1x** Custom pre-wired buzzer with male jumper wire ends

Today's Project

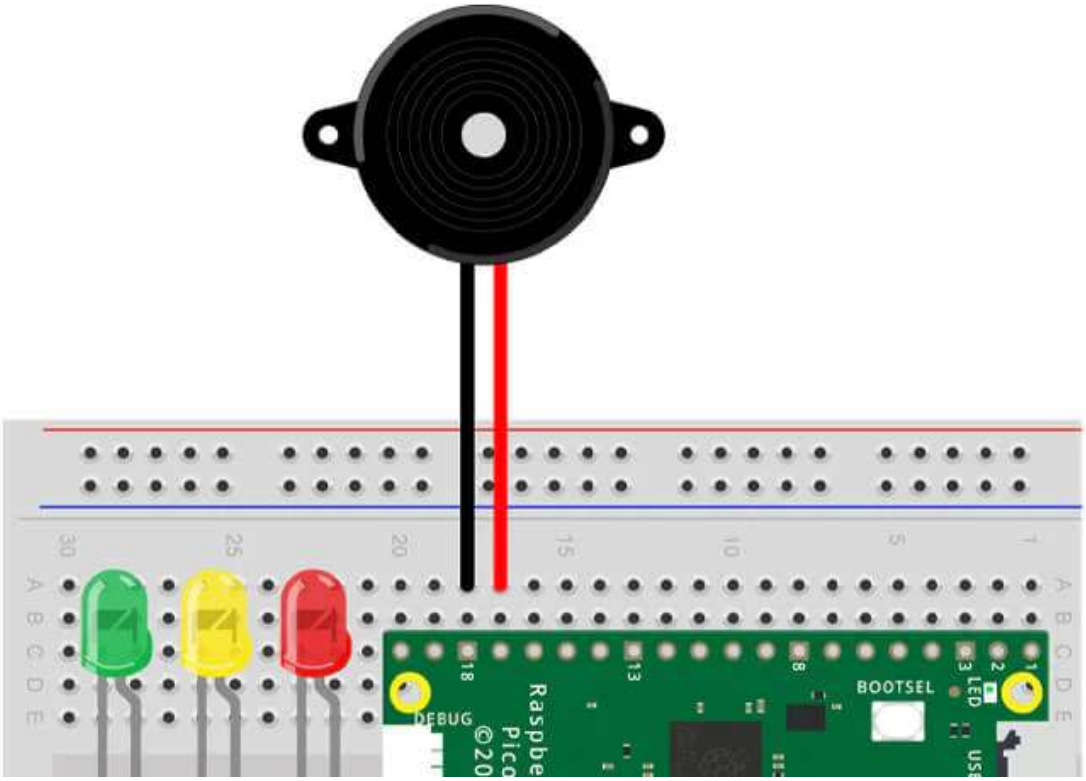
Today we'll be using our buzzer in a few different ways, first for simple beeps and tones, then combining with our potentiometer to control the volume, then finally something a little more advanced where we use different frequencies to generate festive tones!

This particular buzzer works with PWM signals to allow you to generate different tones, so we'll be tapping back into everything we learned about PWM in yesterday's box. *It's almost like we planned this...*

Construct the Circuit

This is one of the easier components to work with as we've included a custom pre-wired buzzer with male jumper wire ends.

Simply push the **red wire** into **GPIO 13** (***physical pin 17***) and the **black wire** into the **GND pin** next to it (***physical pin 18***). The Pico pin map is always on hand if you need to check it.



Activity 1: Let's buzz!

Our buzzer can generate different tones based on the PWM frequency and duty cycle we use in our code - this should sound familiar as we covered PWM yesterday.

PWM Frequency and Duty Cycle with Buzzers

The ***frequency*** here changes the ***tone*** of the buzzer - we find that our ears can detect changes in the tone when using values between 10 and 10000.

The ***Duty Cycle*** sets the ***volume*** of the buzzer. The range is technically 0 to 65535 in MicroPython, however we find that values below 5000 are very difficult for our ears to hear. A duty of ***10000 appears to be the sweet spot*** for our buzzer.

If you want to silence the buzzer in your code, just set the duty to **0** - easy!

The Code

Now we know what the PWM values do here, let's try a quick example where we sound the buzzer for a few seconds.

The example below includes the same ingredients you're now becoming familiar with - ***imports***, ***pin setup*** (with PWM), ***PWM duty cycle*** and ***frequency*** - so there's nothing new or scary here, and our usual *slightly-excessive* code commentary tells you what each line is doing!

We set up the pin for the buzzer, then assign a frequency (1000) and finally set the PWM duty (volume) to 10000 to for a second, and then back down to 0 to mute it. Nice and easy!

Copy the example over to Thonny and give it a go:

```
# Imports
from machine import Pin, PWM
import time

# Set up the Buzzer pin as PWM
buzzer = PWM(Pin(13)) # Set the buzzer to PWM mode

# Set PWM frequency to 1000
buzzer.freq(1000)

# Set PWM duty
buzzer.duty_u16(10000)

time.sleep(1) # Wait 1 second

# Duty to 0 to turn the buzzer off
buzzer.duty_u16(0)
```

Now try changing that frequency number to 300 and see what happens. Did the tone go lower? You can try higher values too, but eventually you'll go too high (around 10000) and it won't work/your human ears won't hear it.

Activity 2: Changing tones

Here's another simple example just to show you how easy it is to change the tone on your buzzer from one to another.

Here we start with a frequency of ***1000*** for one second and then drop it down to ***500*** for one second.

Give it a try and hear it for yourself:

```
# Imports
from machine import Pin, PWM
import time

# Set up the Buzzer pin as PWM
buzzer = PWM(Pin(13)) # Set the buzzer to PWM mode

# Set PWM duty
buzzer.duty_u16(10000)

# Set PWM frequency to 1000 for one second
buzzer.freq(1000)
time.sleep(1)

# Set PWM frequency to 500 for one second
buzzer.freq(500)
time.sleep(1)

# Buzzer off
buzzer.duty_u16(0)
```

Activity 3: Potentiometer Volume Control

Let's use one of the previous components to control the volume of the buzzer. ***This one's a bit noisy*** and not perfect (*the sound can distort or cut out at high levels*) but it's a fun project to try out! If you find the buzzer a bit loud or harsh, you can always place some sticky tape or a blob of Blu-tack over the hole to calm it down.

The readings from the potentiometer are between ***0 and 65535***, and our buzzer duty (volume) uses the same range, so we can use the potentiometer readings to directly control the duty - handy!

In the example below, in each ***while loop*** we take a reading from the potentiometer and store it in a ***variable*** called 'reading' (just like we did yesterday in activity 3) and then we use this variable as the buzzer duty value (instead of adding a number there ourselves).

A little warning *- this example will never turn the buzzer off even if you stop the program as it's just a quick, basic example for us to get familiar with buzzer control. You can unplug the USB cable to stop the*

noise or just move on to the next activity.

```
# Imports
from machine import Pin, PWM, ADC
import time

# Set up the Buzzer and Potentiometer
potentiometer = ADC(Pin(27))
buzzer = PWM(Pin(13)) # Set the buzzer to PWM mode

reading = 0 # Create a variable called 'reading' and set it to 0

while True: # Run forever

    time.sleep(0.01) # Delay

    reading = potentiometer.read_u16() # Read the potentiometer value and s

    buzzer.freq(500) # Frequency to 500
    buzzer.duty_u16(reading) # Use the reading variable as the duty

    print(reading) # Print our reading variable
```

Activity 4: Festive Jingle!

How about a little festive tune with our buzzer? By changing the frequency between each pause we can create a tune that resembles a popular festive jingle!

A nice way of doing this is to define each musical note frequency as a ***variable*** using the note name, which will make our code easier to write, read and review. We've covered variables before so this should be familiar. We've also used a variable for the volume (duty cycle) as, again, it makes our code a little more *human*. Don't worry, we've included the tone values for the jingle below for you.

The example below is purposely inefficient and uses a LOT of lines, mostly because each section of the jingle includes lines to turn the volume on then off as well as delays. Try this first, then we'll move on to a nicer way of doing things in the next activity.

We've included commentary on the first jingle block of code to show you what we're doing here.

***Tip**: This is a simple buzzer, so it doesn't quite have the smooth sound that a normal speaker will create. The buzzer can sound a little more 'Nightmare Before Christmas' than 'Elf'!*

```
# Imports
from machine import Pin, PWM
import time

# Set up the Buzzer pin as PWM
buzzer = PWM(Pin(13)) # Set the buzzer to PWM mode

# Create our library of tone variables for "Jingle Bells"
C = 523
D = 587
E = 659
G = 784

# Create volume variable (Duty cycle)
volume = 10000

# Play the tune

# "Jin..."
buzzer.duty_u16(volume) # Volume up
buzzer.freq(E) # Set frequency to the E note
time.sleep(0.1) # Delay
buzzer.duty_u16(0) # Volume off
time.sleep(0.2) # Delay

# "...gle"
buzzer.duty_u16(volume)
buzzer.freq(E)
time.sleep(0.1)
buzzer.duty_u16(0)
time.sleep(0.2)

# "Bells"
buzzer.duty_u16(volume)
buzzer.freq(E)
time.sleep(0.1)
buzzer.duty_u16(0)
time.sleep(0.5) # longer delay

# "Jin..."
```

```
    buzzer.duty_u16(volume)
    buzzer.freq(E)
    time.sleep(0.1)
    buzzer.duty_u16(0)
    time.sleep(0.2)

    # "...gle"
    buzzer.duty_u16(volume)
    buzzer.freq(E)
    time.sleep(0.1)
    buzzer.duty_u16(0)
    time.sleep(0.2)

    # "Bells"
    buzzer.duty_u16(volume)
    buzzer.freq(E)
    time.sleep(0.1)
    buzzer.duty_u16(0)
    time.sleep(0.5) # longer delay

    # "Jin..."
    buzzer.duty_u16(volume)
    buzzer.freq(E)
    time.sleep(0.1)
    buzzer.duty_u16(0)
    time.sleep(0.2)

    # "...gle"
    buzzer.duty_u16(volume)
    buzzer.freq(G)
    time.sleep(0.1)
    buzzer.duty_u16(0)
    time.sleep(0.2)

    # "All"
    buzzer.duty_u16(volume)
    buzzer.freq(C)
    time.sleep(0.1)
    buzzer.duty_u16(0)
    time.sleep(0.2)

    # "The"
    buzzer.duty_u16(volume)
```

```
buzzer.freq(D)
time.sleep(0.1)
buzzer.duty_u16(0)
time.sleep(0.2)

# "Way"
buzzer.duty_u16(volume)
buzzer.freq(E)
time.sleep(0.1)
buzzer.duty_u16(0)
time.sleep(0.2)

# Duty to 0 to turn the buzzer off
buzzer.duty_u16(0)
```

Activity 5: Festive Jingle using Functions

Let's improve the *very long* code above by introducing something new - ***functions***!

Functions

Functions are blocks of code that you can create and then ***call*** (use) at any time in your program. They're ***great for repetitive blocks of code*** as it keeps your program short and efficient (which can be important if your microcontroller has limited storage).

Let's explain how functions work before we move on...

Writing a function

Below is an example of a simple function that prints 3 lines. We create a function by writing '***def***' followed by a space and then the name we want to give our function (no spaces in the function name) .

We've called ours '***myfunction***'. You then add brackets (***parentheses***) at the end which can include any ***arguments*** if required (*we're not using any in this example but we cover them in a moment*) followed by a colon (:).

Indentation is important again here - everything under the '***def***' line must be indented to indicate that it's part of the function:

```
def myfunction():
    print("This")
    print("is a")
    print("function")
```

Using Functions

So now that we have a function, how do we use it, and why is it better than just writing the code?

We can ***call*** (use) the function by simply using the function name followed by the brackets, like this:

```
myfunction()
```

So why is this better? Let's look at two examples below where we want to print a set of strings three times. Both of the following examples have the exact same outcome (try them yourself) however the code using functions is shorter.

It might not look like much in this example, however in large projects this can save a LOT of lines in your code. It also makes it a lot easier to update your code as you only have to amend the one function - not every part of your program.

Example without a function

```
print("This")
print("is a")
print("function")

print("This")
print("is a")
print("function")

print("This")
print("is a")
print("function")
```

Example with a function

```
def myfunction():
    print("This")
```

```
    print("is a")
    print("function")

myfunction()
myfunction()
myfunction()
```

Function Arguments

We mentioned ***arguments*** earlier so let's talk about them quickly before we get into our jingle example.

Arguments are optional in functions. They allow you to pass information ***into*** your function, if needed. We pass the information to the function when we ***call*** it - that might be a bit confusing so let's jump to an example so you can see it working.

In the example below our function prints three lines again, but this time we include ***arguments*** asking for the first name, middle name and surname - as we might want to change the names each time we call this function. Those arguments are used in the print lines as you can see below:

```
def myfunction(first,middle,last):
    print(first)
    print(middle)
    print(last)
```

Now, to call the function AND pass the names on to it, we add the names in the same places inside the brackets (making sure we enter them inside inverted commas as we're dealing with strings here):

```
myfunction("Joe","David","Bloggs")
```

Try this yourself, then try adding a fourth argument and printing a fourth line to go with it.

The Code

Now we know how functions work, let's apply them to our jingle example to see if we can reduce the length of the code.

The example below uses a function called ***playtone*** to handle all of the volume changes, delays and notes. All you have to do is add the arguments for each.

Our arguments are:

- **note** - the note to use
- **vol** - the volume
- **delay1** - the first ***time.sleep*** period
- **delay2** - the second ***time.sleep*** period

The code is roughly half the number of lines and does the same thing - much better!

```
# Imports
from machine import Pin, PWM
import time

# Set up the Buzzer pin as PWM
buzzer = PWM(Pin(13)) # Set the buzzer to PWM mode

# Create our library of tone variables for "Jingle Bells"
C = 523
D = 587
E = 659
G = 784

# Create volume variable (Duty cycle)
volume = 32768

# Create our function with arguments
def playtone(note,vol,delay1,delay2):
    buzzer.duty_u16(vol)
    buzzer.freq(note)
    time.sleep(delay1)
    buzzer.duty_u16(0)
    time.sleep(delay2)

# Play the tune
playtone(E,volume,0.1,0.2)
playtone(E,volume,0.1,0.2)
playtone(E,volume,0.1,0.5) #Longer second delay

playtone(E,volume,0.1,0.2)
playtone(E,volume,0.1,0.2)
playtone(E,volume,0.1,0.5) #Longer second delay
```

```
playtone(E,volume,0.1,0.2)
playtone(G,volume,0.1,0.2)
playtone(C,volume,0.1,0.2)
playtone(D,volume,0.1,0.2)
playtone(E,volume,0.1,0.2)

# Duty to 0 to turn the buzzer off
buzzer.duty_u16(0)
```

Bonus - More notes!

Chris E kindly added a comment below providing a nice full list of notes (including sharps). Look up your favourite songs, use these notes and get creative. Thanks Chris!

- **A** = 440
- **As** = 466
- **B** = 494
- **C** = 523
- **Cs** = 554
- **D** = 587
- **Ds** = 622
- **E** = 659
- **F** = 698
- **Fs** = 740
- **G** = 784
- **Gs** = 830

Day #5 Complete!

We've covered lots today fellow makers, pat yourselves on the back!

As we cover more advanced topics such as functions, you may find that you need to refer back to these days to refresh your memory later on - and that's *absolutely normal*! Even the most seasoned professional programmers have to use search engines regularly (some even admit it!).

So what did we cover on day #5? Today you have:

- Built a circuit with a buzzer, your first audio component
- Learnt how to use the buzzer with MicroPython and the Pico
- Learnt about PWM frequencies and duty cycle with buzzers
- Used analogue inputs to control audio volume using PWM
- Created a festive jingle with MicroPython
- Learnt how to use functions to make your code easier to manage and more efficient

Keep your circuit safe somewhere until tomorrow (**don't take anything apart just yet**).

We used Fritzing to create the breadboard wiring diagram images for this page.

Featured Products

Maker Advent Calendar (includes Raspberry Pi Pico H)

£40 incl. VAT

NOTIFY ME

9 comments

Jonothan Hunt

December 10, 2022 at 10:00 pm

This calendar is so much fun!

If anyone wants a sc-fi lights and sound effect, I've made this script. When you turn the potentiometer, the speaker plays a random sound with pitch from the reading and each light lights randomly too!

```
Imports
from machine import Pin, PWM, ADC
import time
```

import random Set up the Buzzer pin as PWM
buzzer = PWM) # Set the buzzer to PWM mode
led = Pin(18, Pin.OUT) # LED 1
led2 = Pin(19, Pin.OUT) # LED 2
led3 = Pin(20, Pin.OUT) # LED 3
potentiometer = ADC) # Potentiometer
reading = 0
newReading = 0

delay= 0.05
played = 0
delay = 0.03

while played < 100:
newReading = potentiometer.read_u16() # new reading from potentiometer
note = random.randint(100, 500) + int(newReading / 40) # noise chosen randomly + added pitch from the new reading

Print old reading and new reading
print("old: “, reading)
print(”new: ", newReading) If new reading is > 500 > last reading Or new reading is > 500 < last reading Then play the random sound and activate the lights randomly
if newReading > (reading + 500) or newReading < (reading – 500):
buzzer.duty_u16(200) buzzer.freq(note) time.sleep(delay) buzzer.duty_u16(0)
led.value(1 if random.random() > 0.5 else 0) led2.value(1 if random.random() > 0.5 else 0) led3.value(1 if random.random() > 0.5 else 0) reading = newReading played += 1 Sleep for 0.05 before checking a new reading
time.sleep(0.05) Turn lights and buzzer off
led.value(0)
led2.value(0)
led3.value(0)
buzzer.duty_u16(0)

This calendar is so much fun!

If anyone wants a sc-fi lights and sound effect, I’ve made this script. When you turn the potentiometer, the speaker plays a random sound with pitch from the reading and each light lights randomly too!

Imports
from machine import Pin, PWM, ADC
import time

import random Set up the Buzzer pin as PWM
buzzer = PWM) # Set the buzzer to PWM mode
led = Pin(18, Pin.OUT) # LED 1
led2 = Pin(19, Pin.OUT) # LED 2
led3 = Pin(20, Pin.OUT) # LED 3
potentiometer = ADC) # Potentiometer
reading = 0
newReading = 0
delay= 0.05
played = 0
delay = 0.03

while played < 100:
newReading = potentiometer.read_u16() # new reading from potentiometer
note = random.randint(100, 500) + int(newReading / 40) # noise chosen randomly + added pitch from the new reading

Print old reading and new reading
print("old: “, reading)
print(”new: ", newReading) If new reading is > 500 > last reading Or new reading is > 500 < last reading Then play the random sound and activate the lights randomly
if newReading > (reading + 500) or newReading < (reading – 500): buzzer.duty_u16(200) buzzer.freq(note) time.sleep(delay) buzzer.duty_u16(0) led.value(1 if random.random() > 0.5 else 0) led2.value(1 if random.random() > 0.5 else 0) led3.value(1 if random.random() > 0.5 else 0) reading = newReading played += 1 Sleep for 0.05 before checking a new reading time.sleep(0.05) Turn lights and buzzer off
led.value(0)
led2.value(0)
led3.value(0)
buzzer.duty_u16(0)

HEATH

December 10, 2022 at 10:00 pm

so so good very cool

so so good very cool

Rob Box

December 9, 2022 at 4:04 pm

“Jin…”
play_note(volume, E, 0.15, 0.15) “…gle”
play_note(volume, E, 0.15, 0.15) “Bells”
play_note(volume, E, 0.3, 0.3) “Jin…”
play_note(volume, E, 0.15, 0.15) “…gle”
play_note(volume, E, 0.15, 0.15) “Bells”
play_note(volume, E, 0.3, 0.3) “Jin…”
play_note(volume, E, 0.3, 0) “…gle”
play_note(volume, G, 0.3, 0) “All”
play_note(volume, C, 0.3, 0.15)
Try the following timings; it should make things sounds a bit less like an electronic card! ;) “The”
play_note(volume, D, 0.15, 0) “Way”
play_note(volume, E, 0.3, 0.15)

“Jin…”
play_note(volume, E, 0.15, 0.15) “…gle”
play_note(volume, E, 0.15, 0.15) “Bells”
play_note(volume, E, 0.3, 0.3) “Jin…”
play_note(volume, E, 0.15, 0.15) “…gle”
play_note(volume, E, 0.15, 0.15) “Bells”
play_note(volume, E, 0.3, 0.3) “Jin…”
play_note(volume, E, 0.3, 0) “…gle”
play_note(volume, G, 0.3, 0) “All”
play_note(volume, C, 0.3, 0.15)
Try the following timings; it should make things sounds a bit less like an electronic card! ;) “The”
play_note(volume, D, 0.15, 0) “Way”
play_note(volume, E, 0.3, 0.15)

giles

December 6, 2022 at 3:17 pm

Thanks for the remaining notes including sharps @Chris Ellis.

I was rather surprised to discover that Github copilot understood that we are writing music code (it is an AI code auto-completer from Microsoft that controversially learns how to code from all the open source software on Github)

I found it was able to fill in the extra note's frequencies (but not sharps!) for me and write a tune with correct frequencies and delays. However, it seemed fixated on star wars music and I've not yet convinced it to auto-complete jingle bells for me!

I'll post a demo soon.

Thanks for the remaining notes including sharps @Chris Ellis.

I was rather surprised to discover that Github copilot understood that we are writing music code (it is an AI code auto-completer from Microsoft that controversially learns how to code from all the open source software on Github)

I found it was able to fill in the extra note's frequencies (but not sharps!) for me and write a tune with correct frequencies and delays. However, it seemed fixated on star wars music and I've not yet convinced it to auto-complete jingle bells for me!

I'll post a demo soon.

Angelo

December 12, 2022 at 11:45 am

How could you set the volume of the final tune using the PWM value of the potentiometer? I have tried and keep getting stuck in a while loop or end up having it only set the volume at the beginning of each play through.

How could you set the volume of the final tune using the PWM value of the potentiometer? I have tried and keep getting stuck in a while loop or end up having it only set the volume at the beginning of each play through.

Alex

December 12, 2022 at 11:38 am

Nice project! Any idea if you can combine multiple PWM outputs to create more complex sounds / chords?

Nice project! Any idea if you can combine multiple PWM outputs to create more complex sounds / chords?

Chris Ellis

December 6, 2022 at 1:51 am

Really enjoying this, if anyone else is interested here's a scale of notes to play with if you want to make more tunes:

A = 440
As = 466
B = 494
C = 523
Cs = 554
D = 587
Ds = 622
E = 659
F = 698
Fs = 740
G = 784
Gs = 830

Really enjoying this, if anyone else is interested here's a scale of notes to play with if you want to make more tunes:

A = 440
As = 466
B = 494
C = 523
Cs = 554
D = 587
Ds = 622
E = 659
F = 698
Fs = 740
G = 784
Gs = 830

The Pi Hut

December 5, 2022 at 1:49 pm

@Phil C – You're absolutely right – good spot! One of those moments where you don't realise the code is technically wrong because the error still allows it to work as expected. Thanks for letting us know.

@Phil C – You're absolutely right – good spot! One of those moments where you don't realise the code is technically wrong because the error still allows it to work as expected. Thanks for letting us know.

Phil C

December 5, 2022 at 1:47 pm

Enjoying this so far. Just a quick note on the last piece of code:
the variable volume is fed into the function as vol, but then inside the function it is still using the global volume variable

Enjoying this so far. Just a quick note on the last piece of code:
the variable volume is fed into the function as vol, but then inside the function it is still using the global volume variable

Leave a comment

All comments are moderated before being published.

This site is protected by reCAPTCHA and the Google Privacy Policy and Terms of Service apply.

Name

E-mail

Message

SUBMIT

Related Posts

Maker Advent Calendar Day #6: Looking for Light!

The Pi Hut • Dec 6, 2022

Welcome to day six of your 12 Projects of Codemas Advent Calendar. Today we'll be using a light sensor which - you guessed it - can detect the amount of light...

Read more

Maker Advent Calendar Day #4: Amazing Analogue!

The Pi Hut • Dec 4, 2022

Welcome to day four of your 12 Projects of Codemas Advent Calendar. Today we'll be playing with analogue inputs using a potentiometer and combining this with some of the components we've...

Read more

Maker Advent Calendar Day #3: Bashing Buttons!

The Pi Hut • Dec 2, 2022

Welcome to day three of your 12 Projects of Codemas Advent Calendar. Today we'll be adding physical inputs to your program using buttons! So far we've relied purely on code...

Read more

Handy Links

All Products

FAQs

Popular Searches

Search

Site Reviews

Got any questions?

Contact Us / Support Portal

Can I Cancel My Order?

Has My Order Shipped Yet?

Where Is My Order?

Do You Ship To {insert country name}

How Much Is Shipping?

Terms & Conditions

Delivery

Lithium Shipping

Pre-Orders

Privacy Statement

Policies

Terms of Service

Company Info

FAQ

Klarna FAQ

Quick Start Guide

Search

Support Portal

Our Store Sections

Raspberry Pi

Maker Store

micro:bit

Arduino

Gifts

Follow us

We accept