

MAKER  
INNOVATIONS  
SERIES

# Digital Electronics for Musicians

Build Intuitive Electronics and  
Electroacoustic Music Interfaces

—  
*Second Edition*  
—

Alexandros Drymonitis

Apress®

# **Maker Innovations Series**

Jump start your path to discovery with the Apress Maker Innovations series! From the basics of electricity and components through to the most advanced options in robotics and Machine Learning, you'll forge a path to building ingenious hardware and controlling it with cutting-edge software. All while gaining new skills and experience with common toolsets you can take to new projects or even into a whole new career.

The Apress Maker Innovations series offers projects-based learning, while keeping theory and best processes front and center. So you get hands-on experience while also learning the terms of the trade and how entrepreneurs, inventors, and engineers think through creating and executing hardware projects. You can learn to design circuits, program AI, create IoT systems for your home or even city, and so much more!

Whether you're a beginning hobbyist or a seasoned entrepreneur working out of your basement or garage, you'll scale up your skillset to become a hardware design and engineering pro. And often using low-cost and open-source software such as the Raspberry Pi, Arduino, PIC microcontroller, and Robot Operating System (ROS). Programmers and software engineers have great opportunities to learn, too, as many projects and control environments are based in popular languages and operating systems, such as Python and Linux.

If you want to build a robot, set up a smart home, tackle assembling a weather-ready meteorology system, or create a brand-new circuit using breadboards and circuit design software, this series has all that and more! Written by creative and seasoned Makers, every book in the series tackles both tested and leading-edge approaches and technologies for bringing your visions and projects to life.

More information about this series at <https://link.springer.com/bookseries/17311>.

# **Digital Electronics for Musicians**

**Build Intuitive Electronics  
and Electroacoustic Music  
Interfaces**

**Second Edition**

**Alexandros Drymonitis**

**Apress®**

# ***Digital Electronics for Musicians: Build Intuitive Electronics and Electroacoustic Music Interfaces, Second Edition***

Alexandros Drymonitis  
Argyroupoli, Greece

ISBN-13 (pbk): 979-8-8688-0393-2

ISBN-13 (electronic): 979-8-8688-0394-9

<https://doi.org/10.1007/979-8-8688-0394-9>

Copyright © 2024 by Alexandros Drymonitis

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

Trademarked names, logos, and images may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, logo, or image we use the names, logos, and images only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

While the advice and information in this book are believed to be true and accurate at the date of publication, neither the authors nor the editors nor the publisher can accept any legal responsibility for any errors or omissions that may be made. The publisher makes no warranty, express or implied, with respect to the material contained herein.

Managing Director, Apress Media LLC: Welmoed Spahr  
Acquisitions Editor: Miriam Haidara  
Development Editor: James Markham  
Coordinating Editor: Jessica Vakili

Cover designed by eStudioCalamar

Distributed to the book trade worldwide by Apress Media, LLC, 1 New York Plaza, New York, NY 10004, U.S.A. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail [orders-ny@springer-sbm.com](mailto:orders-ny@springer-sbm.com), or visit [www.springeronline.com](http://www.springeronline.com). Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail [booktranslations@springernature.com](mailto:booktranslations@springernature.com); for reprint, paperback, or audio rights, please e-mail [bookpermissions@springernature.com](mailto:bookpermissions@springernature.com).

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at <http://www.apress.com/bulk-sales>.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub (<https://github.com/Apress>). For more detailed information, please visit <https://www.apress.com/gp/services/source-code>.

If disposing of this product, please recycle the paper

*To Katia and Alina.*

# Table of Contents

<b>About the Author .....</b>	<b>xvii</b>
<b>About the Technical Reviewer .....</b>	<b>xix</b>
<b>Acknowledgments .....</b>	<b>xxi</b>
<b>Introduction .....</b>	<b>xxiii</b>
<b>Chapter 1: Introduction to Pure Data .....</b>	<b>1</b>
Before We Start .....	3
Pd Basics: How It Works .....	8
Our First Patch .....	10
The Control Domain .....	14
Execution Order .....	15
Bang! .....	17
Comments .....	19
Getting Help .....	20
GUIs .....	22
Pd Patches Behave like Text Files .....	23
Making Oscillators in Pd .....	24
Making a Triangle Wave Oscillator .....	26
Making a Sawtooth Oscillator .....	28
Making a Square Wave Oscillator .....	29
Using Tables in Pd .....	30
Making Wireless Connections .....	34

## TABLE OF CONTENTS

Subpatches and Abstractions .....	37
The \$0 Number .....	41
Control Domain vs. Signal Domain.....	42
Basic Electronic Music Techniques.....	45
Additive Synthesis .....	45
Ring Modulation .....	47
Amplitude Modulation .....	48
Frequency Modulation.....	49
Envelopes .....	52
Audio Input in Pd.....	54
Delay Lines in Pd .....	55
Reverb .....	58
Filters .....	58
Audio and MIDI Settings .....	61
Additional Thoughts .....	63
Conclusion .....	64
<b>Chapter 2: Introduction to Arduino.....</b>	<b>67</b>
Arduino Jump Start.....	68
Parts List.....	70
The Blink Sketch .....	71
Digital Input.....	80
Defining Variables in Arduino.....	82
Further Explanation of the Code .....	82
Classes in Arduino and the Serial Communication.....	83
Further Explanation .....	83
Building Circuits on a Breadboard.....	85



Pull-Up vs. Pull-Down Resistors .....	88
Both Digital Input and Output.....	89
Analog Input.....	94
Analog Input and Output .....	97
Reading More Than One Pin, Arrays, and the for Loop.....	102
Explaining the for Loop.....	103
Using Arrays in Arduino .....	103
Analog and Digital Input.....	106
Communicating with Pd.....	112
Sending Data from Pd to Arduino.....	125
Conclusion .....	132
<b>Chapter 3: Embedded Computers .....</b>	<b>135</b>
Before You Begin.....	136
Parts List.....	137
Why Use Embedded Computers?.....	138
Getting Started with the Pi.....	139
Logging In to the Pi from macOS and Linux .....	141
Logging In from Windows .....	141
Navigating Through the Linux System.....	142
Editing Text Files in Linux .....	145
Installing Software.....	146
Launching Pd.....	148
Setting a Static IP on the Pi.....	150
Shutting Down or Rebooting the Pi .....	152
Setting a Static IP on Your Computer.....	152
Getting Started with the Bela.....	153

TABLE OF CONTENTS

Exchanging Files Between Your Computer and the Pi or Bela ..... 156

    Transfer Files from macOS and Linux to the Pi or Bela and Vice Versa ..... 156

    Transfer Files from Windows to the Pi or Bela and Vice Versa ..... 157

Conclusions..... 160

**Chapter 4: Going Wireless ..... 161**

    Before You Begin..... 162

    XBee vs. WiFi ..... 162

    Using the XBee with Arduino..... 163

        Configuring the XBee..... 166

        Writing AT Commands to Configure the XBee..... 168

        Testing the Wireless Connection..... 171

    Using the Arduino Nano ESP32 with WiFi..... 173

    Conclusion ..... 178

**Chapter 5: Getting Started with Musical Applications ..... 179**

    Parts List..... 180

    Frequency Modulation Interface ..... 180

        Making the Pd Patch ..... 180

        Arduino Code for Frequency Modulation Patch ..... 186

        Circuit for Arduino Code..... 188

    A Simple Drum Machine Interface ..... 190

        Building the Pd Patch ..... 190

        Arduino Code for Drum Machine Patch ..... 203

    Drum Machine and Frequency Modulation Combination ..... 207

        Arduino Code ..... 207

        Pd Patch for Drum Machine–FM Interface ..... 212

    Conclusion ..... 217

<b>Chapter 6: An Interactive Glove .....</b>	<b>219</b>
Parts List .....	220
Writing the Arduino Code .....	220
Building the Circuit of the Accelerometer .....	224
Building the Pd Patch.....	225
The Graph-On-Parent Subpatches.....	227
The [pd freq_modulation] Subpatch.....	230
The [pd pitch_shift] Subpatch .....	234
The Glove .....	235
Conclusion .....	239
<b>Chapter 7: An Interactive Drum Set.....</b>	<b>241</b>
Parts List.....	242
First Approach to Detecting Drum Hits.....	242
First Version of the Circuit .....	243
Read the Drum Hits in Pd .....	245
Getting the Maximum Value.....	246
Finalizing the Circuit and Arduino Code .....	250
Adding Switches and LEDs to the Circuit and Code .....	251
Building the Pd Patch.....	258
Building the Audio File Abstraction.....	258
Building the Abstraction to Receive Input from the Arduino.....	271
The Main Patch.....	272
Making the Circuit Enclosure .....	277
Conclusion .....	281

TABLE OF CONTENTS

<b>Chapter 8: A DIY Theremin.....</b>	<b>283</b>
Parts List.....	283
Using a Proximity Sensor with the Arduino.....	284
Smoothing Out the Sensor Values.....	286
Using Two Proximity Sensors and Four Potentiometers.....	289
The Arduino Code.....	291
The Circuit.....	294
The Pd Patch.....	295
Building the Oscillators for the Theremin.....	296
Creating Band-Limited Waveforms.....	297
Reading the Stored Band-Limited Waveforms.....	299
Finalizing the Interface.....	300
Adding Switches to the Arduino Code to Control the Waveforms of the Oscillators.....	300
Making the Circuit.....	308
Putting It All Together.....	310
Adding a Push Button for Switching Off the Embedded Computer.....	320
Reading the Extra Push Button in Pd and Shutting Down the Pi.....	321
Loading the Pd Patch on Boot.....	323
Some Tips.....	324
Conclusion.....	325
<b>Chapter 9: Making a Looper.....</b>	<b>327</b>
Parts List.....	327
What Is a Looper?.....	329
Recording Our First Sounds in Pd.....	330
Playing Back the Recorded Sound.....	331
Controlling the Speed and Direction of the Playback.....	334

Making Recordings of Arbitrary Lengths..... 336

    Retrieving the Length of the Recording..... 338

Overdubbing..... 339

    Getting Rid of Clipping..... 340

    Getting the Start Position of the Overdubbing ..... 341

    Knowing When to Stop Overdubbing ..... 342

Playing Back a Portion of the Recording..... 343

Synchronizing End of Recording with Beginning of Playback..... 344

Start Building the Interface..... 345

    Building the Circuit on a Breadboard..... 346

    Working Further on the Pd Patch..... 348

    Putting It All Together ..... 367

Making an Enclosure ..... 368

Running the Pd Patch on Boot ..... 371

Some Tips ..... 371

    Including the [tabwrite\_dir~] External Object..... 371

    Creating a Shield for the Circuit ..... 372

Conclusion ..... 374

**Chapter 10: A Patch-Bay Matrix Synthesizer .....375**

    Parts List..... 375

    What We Want to Achieve in This Chapter..... 378

    Extending the Arduino Analog Pins ..... 379

        How a Multiplexer Works..... 381

        Writing Code to Control a Multiplexer..... 383

        Wiring 16 Potentiometers to the Multiplexer..... 388

## TABLE OF CONTENTS

Extending the Arduino Digital Pins .....	390
Using the Input Shift Register .....	391
Using the Output Shift Register .....	398
Combining the Input and Output Shift Registers .....	407
Making a Patch-Bay Matrix.....	410
Implementing a Patch-Bay Matrix with the Arduino .....	410
Making the Pd Patch That Reads the Connections .....	417
Making the Patch-Bay Matrix Circuit.....	421
Start Building the Audio Part of the Pd Patch .....	423
A Signal Matrix in Pd .....	424
Building the Modules for the Synthesizer.....	426
The Final Patch .....	440
The <code>arduino_stuff</code> Subpatch .....	441
The Final Arduino Code .....	446
Using Binary Numbers for Masking.....	453
The <code>checkConnections()</code> Function .....	454
The <code>Main loop()</code> Function .....	454
Controlling the LEDs with the Switches of the Shift Registers.....	455
The Final Circuit .....	456
Making an Enclosure .....	457
Shutting Down the Embedded Computer .....	459
Conclusion .....	459
<b>Chapter 11: Interactive Projects with AI .....</b>	<b>461</b>
What Are Neural Networks? .....	462
Parts List.....	463
An Interactive Drum Synthesizer.....	465
The <code>[neuralnet]</code> External Object.....	466

## TABLE OF CONTENTS

The Main Pd Patch.....	467
The NN Settings Subpatches .....	470
The [pd percussion] Subpatch.....	474
The [pd sliders] Subpatch.....	475
How to Train the Networks .....	476
Some Tips .....	478
An Augmented MIDI Keyboard with AI-Waveshaping.....	480
Creating the Circuit.....	482
Controlling the LCD.....	483
Configuring the Pi to Receive MIDI Through Its GPIO Pins.....	494
Making the Pd Patch .....	495
The ai_synth_voice.pd Abstraction.....	517
Training the Network.....	519
Writing the Scripts to Launch the Project.....	519
Some Tips .....	522
Conclusion .....	523
<b>Index.....</b>	<b>525</b>

# About the Author



**Alexandros Drymonitis** is a sound and new media artist. He has a PhD from the Royal Birmingham Conservatoire, Birmingham City University, on the creation of musical works with the Python programming language, while his previous studies were on the classical guitar at the Conservatorium van Amsterdam, where he got his first stimuli on music technology.

Ever since, he has been making electronic music using open source software and hardware, like Pure Data, Arduino, and Python. Besides this book, he has used Pure Data and Arduino to develop the modular synthesizer system 3dPdModular. He is currently doing postdoc research at the Cyprus University of Technology on instant synthesis for computer-controlled acoustic instruments with live coding and AI.



# About the Technical Reviewer



**Sai Yamanoor** is an embedded engineer based in Oakland, CA. He has over ten years of experience as an embedded systems expert, working on hardware and software design. He is a coauthor of three books on using Raspberry Pi to execute DIY projects, and he has also presented a Personal Health Dashboard at Maker Faires across the country. Sai is also working on projects to improve quality of life (QoL) for people with chronic health conditions. Check out his projects at <https://saiyamanoor.com>.

# Acknowledgments

I would like to thank Miriam Haidara for proposing this second edition to me, as well as the whole Apress team for being helpful and responsive throughout the writing and proofreading of this second edition. The Pure Data community and its developers have contributed by developing this great open source software that I daily use, as well as the Arduino team. These two software have opened up a whole universe of exploration and creativity.

# Introduction

This book aims at giving insight on a few of the most widely used tools in the fields of creative coding and DIY digital electronic musical interfaces. It is a result of personal exploration in these fields and an attempt to gather information about the combination of the popular prototyping platform, the Arduino, with the also popular visual programming language for multimedia, Pure Data (Pd).

The main focus of the book is interactivity with the physical world and how to make this musical. It is split among several projects where each project brings a fresh idea on how to combine musical instruments with computers, whereas the use of programming builds up gradually. Also, this book uses only open source software, because of the great advantages one can have from an open source community, but also in order to bring the cost of every project to its minimum.

The first edition of this book, which was published in late 2015, used Pd-extended, a version of Pd that included a number of external packages, in addition to the ones that come with Pd's core – called Pd-vanilla. Since then, a lot has changed, and Pd-extended is now no longer maintained. Still, at the time of writing (early 2024), it is easier than ever to install Pd-vanilla and any external packages in any operating system (OS). Additionally, the version of the Arduino Integrated Development Environment (IDE) has also moved up by a significant number, and it is now much more feature-rich and easy to use. All this, together with the advent of artificial intelligence (AI) in the field of creative coding, rendered this second edition necessary.

## INTRODUCTION

Besides software versions, a significant addition to this second edition is the Bela, a prototyping platform for interactive musical projects. This is introduced in Chapter 3, together with the Raspberry Pi, and a project is built with it in Chapter 9. An extra chapter has been written, focusing on AI. This is Chapter 11. The initial Chapter 5 has been completely removed, as I felt it didn't really add to this book. This chapter covered MIDI, which has been taken up by the second project of Chapter 11.

Chapter 3 of the first edition has been split in two, and now these are Chapters 3 and 4, where Chapter 3 focuses on embedded computers – namely, the Raspberry Pi and the Bela – and Chapter 4 focuses on wireless communication. In addition to the XBee radios used in the first edition, wireless communication over WiFi has been added, while the Bluetooth communication has been removed, as it seems to not be very popular and requires a few steps in its configuration, both in software and hardware, rendering it not easy enough to include it in this edition.

I hope the reader finds this book useful for their own projects and that it will kick-start their own endeavors in the fields of creative/interactive coding and DIY electronics.

## CHAPTER 1

# Introduction to Pure Data

Pure Data (Pd) is a visual programming language and environment for audio, visuals, and multimedia. It is open source, and it was developed by Miller Puckette during the 1990s. Pd is a very powerful and flexible programming language, still actively developed, used by professionals and hobbyists alike around the world.

Visual programming means that instead of writing code (a series of keywords and symbols that have a specific meaning in the context of a programming language), you use a graphical interface to create programs, where in the most usual cases, a “box” represents a certain function, and you connect these “boxes” with lines. This kind of programming is also called *data flow programming* because of the way the parts of a program are connected, which indicates how its data flows from one part of the program to another.

Visual programming can have various advantages compared to textual programming. One advantage is that a visual programming language can be very flexible and quick for prototyping, where in many textual programming cases, you need to write a good number of lines of code before you can achieve even something simple. Another advantage is that visual programming can be considered more intuitive than textual

programming. There seems to be a consensus among nonprogrammers where textual code is considered as something coming out of a horror movie, making visual programming languages more attractive.

Throughout this book, we will use Pd for all of our audio and sequencing programming, most of the times in combination with the Arduino. The Arduino is a prototyping platform used for physical computing, which enables us to connect the physical world with the world of computers. A thorough introduction to Arduino is given in Chapter 2. This chapter is an introduction to Pd, where we will go through its basics, its philosophy, as well as some general electronic music techniques. If you are already using Pd and know its basics, you can skip this chapter and go straight to the next one. Still, if you are using Pd but have a fuzzy understanding on some of its concepts, you might want to read this chapter. Mind that the introduction to Pd made in this chapter is centralized around the chapters that follow, and even though some generic concepts will be covered, it is focused on the techniques that will be used in this book's projects.

In order to follow this chapter and the rest of this book, you will need to install Pd on your computer. Luckily, Pd runs on all three major operating systems (OS), macOS, Linux, and Windows. At the time of writing, there are a few different versions of Pd. The basic one is called Pd-vanilla, and it is the core on which all other versions are based. This is the version developed by Miller Puckette. Other versions include Pd-L2Ork (standing for Linux Laptop Orchestra), Purr Data (a Pd-L2Ork version for all OSes), and PlugData. All these extra versions have a different Graphical User Interface (GUI) than Pd. Pd-L2Ork and Purr Data are monolithic versions that ship with a large collection of external objects (packages) in addition to the core set of native Pd objects, following the paradigm of the now discontinued Pd-extended. PlugData ships with a smaller collection of externals, but it adds the capability of using it as a plug-in inside other software (like Digital Audio Workstations (DAWs)) or to compile your Pd programs as C/C++ code. PlugData also offers flexibility on the configuration of its GUI.

Throughout the entire book, we will be using Pd-vanilla. Even though Pd-vanilla comes with the native Pd objects only, it is very easy to add external objects, which we will see next. You can download it for free from its website: <https://puredata.info/>. You will find two versions of Pd, vanilla (Pure Data) and Purr Data. Navigate to the download page of Pure Data and choose the version for your OS. At the time of writing, the latest Pd version is 0-54.1.

By the end of this chapter, you'll be able to

- Understand how a Pd program works
- Create small and simple programs in Pd
- Find help in the Pd environment
- Create oscillators in Pd
- Make use of existing abstractions in Pd and create your own
- Realize standard electronic music techniques in Pd

## Before We Start

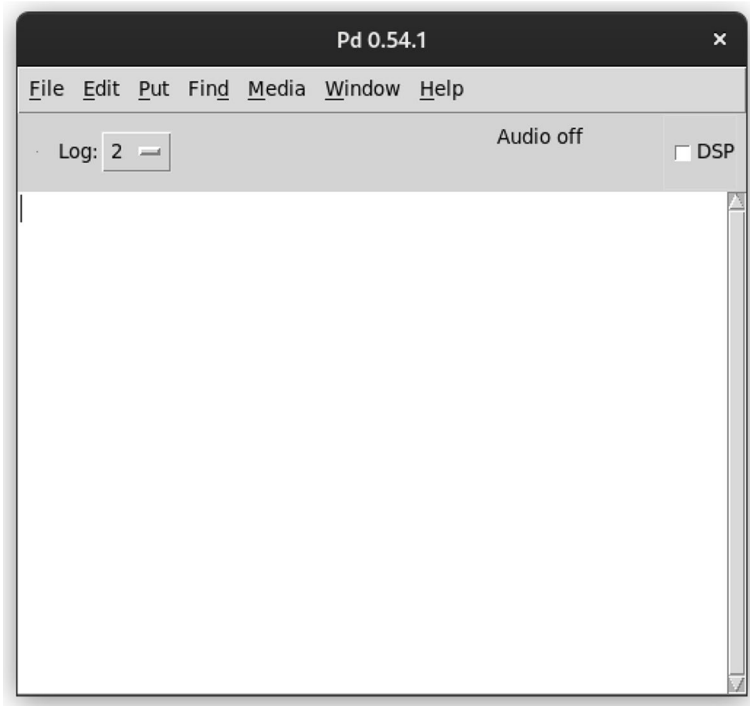
The first thing we need to do is launch Pd and test its audio output. When launching Pd for the first time, it will ask whether you would like it to create a folder called Pd inside your Documents folder. Click “OK” as this is where we will be saving our patches throughout this book. Once this is done, go to **Media ► Test Audio and MIDI...** and a window will open (in Pd jargon, a program we write in Pd is called a patch, and this is what we will refer to Pd programs from now on). On the left-hand side of this patch, there is an area labeled “OUTPUT MONITOR” and a radio button with three options: 80, 60, and off. 80 and 60 refer to decibels (dB), where 100 is full amplitude. Click one of these two dB values to determine if Pd’s audio output works properly. You should hear a sine wave at 440Hz.

If no sound is coming out of your computer, go to **Media ► Audio Settings....** Click the button in the **Output Devices** field, and check the list of available output devices. Choose either your computer's internal speakers, or, if you are using an external audio interface and it shows up in the list, choose that. Click the **Apply** and **OK** buttons and see if that works. If you still don't get any sound, make sure you do get sound from some other software, and try different choices in the **Output Devices** list.

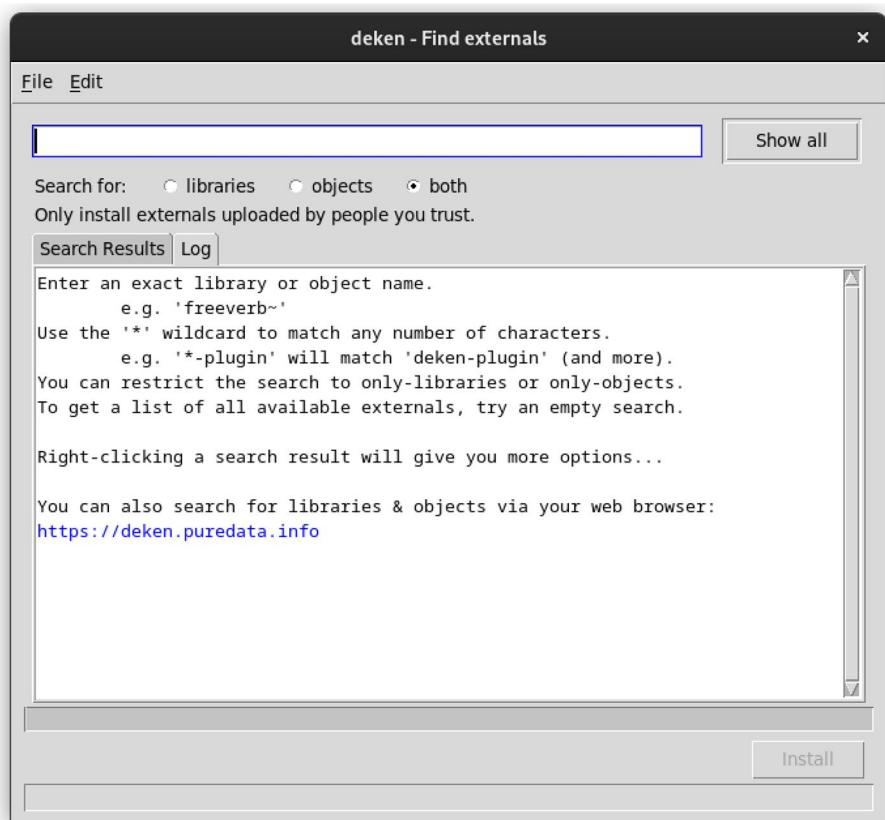
We also need to install two externals, one library (a set of external objects) and a single external. When you launch Pd, all you will see is the Pd console, shown in Figure 1-1. A window will also pop up, asking you if you want Pd to create a folder called "Pd" inside your "Documents" folder and inside there to create a folder called "externals" (from now on, a folder will be referred to as a directory). Go ahead and click OK. The Pd console is where certain information is printed while Pd runs, like error and warning messages. To install the externals we want, we have to go to **Help ► Find Externals**, and the "deken" plug-in will open, shown in Figure 1-2.

In the top entry, type "zexy" and hit Return (the Enter key). In the "Search Results" area, you should see an expandable entry. Click the little arrow on the left of the library name (in our case, "zexy") to expand its contents, and choose the topmost item, which should be the most up-to-date version of the library for your OS. Once you have done that, in the search entry, at the top of this window, type "comport". Follow the same procedure to install this one as well. The zexy library is, as itself states, the Swiss army knife of Pd. We will be using a few objects from this library. The comport object enables Pd to communicate with the serial ports of your computer, thus enabling us to communicate with the Arduino.





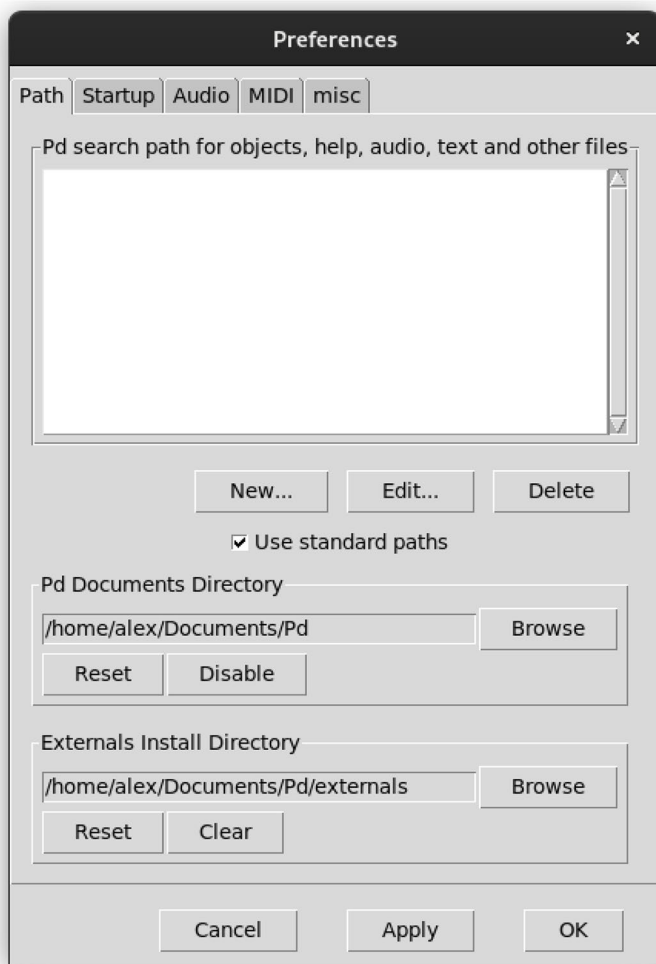
**Figure 1-1.** *The Pd console*



**Figure 1-2.** *The deken plug-in*

There's one last thing left to do before we start learning Pd. All external objects are installed in the “externals” directory, when downloaded from deken, but Pd doesn't know it has to look for objects there. In addition to that, zexy is a single-binary library, which means that all objects of this library are compiled from one source code file into a single executable file (called a binary file). Such cases of external libraries need to be treated differently. Close the deken window, and in the Pd console, go to **File ► Preferences**, and a submenu will appear, in which you should select “Edit Preferences.” The window shown in Figure 1-3 will open. Click the “Path” tab, on the top of this window, and then click “New.” Another window will

open, where you should navigate to the “zexy” and “comport” directories, one at a time. Go to the “zexy” directory, and once selected, click OK. A new entry should appear in the main area of the Preferences window. Do the same for comport. The directories that appear in this area are now added to Pd’s search paths, which means that Pd will search there, in addition to its standard paths, when you try to create an object.



**Figure 1-3.** Pd’s Preferences window

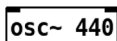
Zexy, being a single-binary library, needs to be added as a library when Pd starts. In the Preferences window, click the “Startup” tab, click “New,” and an editable entry will appear. In there, type “zexy” and hit Return. The word “zexy” should appear in the main area of the window. For all these settings to become effective, you will need to click the “Apply” button, at the bottom of the deken window, and then “OK.” Now restart Pd to see if your changes have been correctly applied. In Pd’s console, you should see a multiline message about the zexy library, like the one shown in Listing 1-1. This means that zexy has been correctly installed and imported. You are now ready to start learning Pd!

**Listing 1-1.** Message on Pd’s Console When the zexy Library Is Loaded

```
♥♥♥
♥ the zexy external 2.4.2
♥ (c) 1999-2023 IOhannes m zmölnig
♥ forum::für::umläute
♥ iem @ kug
♥ compiled Dec 6 2023
♥ send me a 'help' message
♥♥♥
```

## Pd Basics: How It Works

Pd consists of several elements that work together to create programs. The most basic elements are the *object* and the *message*. An object is a function that receives input and gives output. Figure 1-4 shows the *osc~* Pd object.

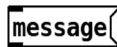


The image shows a small rectangular window with a black border. Inside the window, the text "osc~ 440" is displayed in a monospaced font. The "osc~" part is in a lighter color, and "440" is in a darker color.

**Figure 1-4.** A Pd object

This specific object is a sine wave oscillator with a 440-hertz (Hz) frequency. There is no need to understand what this object does; we will go through that in a bit. There are a few things we need to note. First of all, there is specific text inside the object box, in this case “osc~ 440”. “osc” stands for *oscillator*, and the ~ (called the *tilde*) means that this object is a signal object. In Pd, there are two types of objects: signal and control. A *signal object* is a function that deals with signals (a digital form of an electric signal). A signal object will run its function for as long as the audio is on (the audio is also called the DSP, which stands for *Digital Signal Processing*). A *control object* is independent of audio and runs its function only when it is told to. We will get a better picture of the difference between the two as we go. The last part of the text, “440,” is called an *argument*. This is the data that a function receives, and we provide it as an argument when we want to initialize an object with it. It is not necessary to provide an argument; when there’s no argument in an object, the object is initialized with the value(s) of zero (0).

The second main element in Pd is the message, which is shown in Figure 1-5.



**Figure 1-5.** A Pd message

It is a little bit different from the object, because on its right side, it is indented, and it looks a bit like a flag. The message delivers data. There’s no function here, only the data we write in the message (sometimes referred to as a *message box*). One thing the object and the message have in common is the inlets and the outlets. These are the little rectangles on the top and the bottom, respectively, of the object and the message. All messages have the same form, no matter what we type in them. They all have one inlet to receive data and one outlet to provide the data typed in them. The objects differ, in the sense that each object has as many inlets as it needs to receive data for its function and as many outlets as it needs to

give the output(s) of the function. With the `osc~` object, we can see that it has two inlets and one outlet. The left inlet and the outlet are different than the right inlet, even though they look alike. The left inlet and the outlet are signal inlets/outlets, and the right inlet is a control inlet. Their differences are the same as the signal and control objects. Note that a signal object can have control inlets/outlets, but a control object cannot have signal inlets/outlets.

Objects and messages in Pd are connected with lines, which we also simply call *connections*. A message connected to the `osc~` object is shown in Figure 1-6.



**Figure 1-6.** A message connected to an object

A connection comes out of the outlet of the message and goes to the inlet of the object. This way, we connect parts of our programs in Pd.

## Our First Patch

Now let's try to make the little program. Launch Pd to see its console. It is very important to always have this window open and visible, because we get important information there, like various messages printed from objects, error messages, and so forth.

Go to **File** ► **New** to create a new window. You will get another window that is totally empty (don't make it full screen because you won't be able to see the Pd console anymore). Note that the mouse cursor is a little hand instead of an arrow. This means that you are in Edit Mode, so you can edit your patch. In this window, we will put our objects and messages. In this