

Error: Can't find stylesheet to import.

```
4 | @import "gist";  
   |         ^^^^^^
```

app/assets/stylesheets/application.pdf.scss 4:9 root stylesheet



Magic Storybook with ChatGPT

Created by Erin St Blaine



<https://learn.adafruit.com/magic-storybook-with-chatgpt>
Last updated on 2024-03-29 04:41:41 PM EDT

Table of Contents

[Overview](#)

- [Parts](#)

[Wiring Diagram](#)

[Create an Account with OpenAI](#)

[Software Setup](#)

- [Install Raspberry Pi OS 64-bit Desktop](#)
- [Setup Virtual Environment](#)
- [Blinka Installation](#)
- [Install OpenAI Library](#)
- [Additional Required Dependencies](#)
- [Enable Backlight Control](#)
- [Download the Files](#)

- [Move Prompt and Key Files](#)
- [Add Your OpenAI Key](#)
- [Update ALSA Config](#)
- [Create Desktop and Autostart Shortcuts](#)
- [Optional: Making the File System Read-only](#)

Code Overview

- [Story Code](#)
- [Configuration Options](#)
- [External Config Files](#)
- [UI Element Classes](#)
- [The Book Class](#)
- [Parse Arguments](#)
- [Main](#)
- [Listener Code](#)

Electronics Assembly

Book Assembly

Decoration

Usage

- [Running the Code](#)
- [Creating a New Story](#)
- [Navigation](#)
- [Putting the Book in a Sleep State](#)
- [Exiting the Script](#)
- [Changing the Prompt: bookprompt.txt](#)
- [Troubleshooting](#)

Overview

‘I wonder,’ he said to himself, ‘what’s in a book while it’s closed. Oh, I know it’s full of letters printed on paper, but all the same, something must be happening, because as soon as I open it, there’s a whole story with people I don’t know yet and all kinds of adventures, deeds and battles. And sometimes there are storms at sea, or it takes you to strange cities and countries. All those things are somehow shut in a book. Of course you have to read it to find out. But it’s already there, that’s the funny thing. I just wish I knew how it could be.’

Bastian, the Neverending Story

We live in an age of wonder. Every day, new innovations and inventions change the way we interact with the world around us and with each other. Many of the marvels we read about in the science fiction stories of our youth

are suddenly, magically possible with a little clever engineering and a healthy dose of imagination.

This guide will show you how to create your own never-ending story book using a Raspberry Pi and a touch screen display. Speak your request out loud, and the book will write you an original story based on your prompt. The software uses Open AI to leverage ChatGPT, a language-based AI, to fill an endless storybook with imaginative stories.

This code includes an editable prompt file called **bookprompt.txt**, so you can dictate the character names, story length, writing style, or any other parameters you'd like. Your daughter will love reading stories about herself and her cat as they adventure through an endless world of imagination, in the style of Dr. Seuss!

What will you imagine?



Parts

[Pi Found](#)
[Display](#)
[Touchsc](#)
[Display](#)
[Raspber](#)
The 7"
Touchsc
Display
Raspber
users th
create a

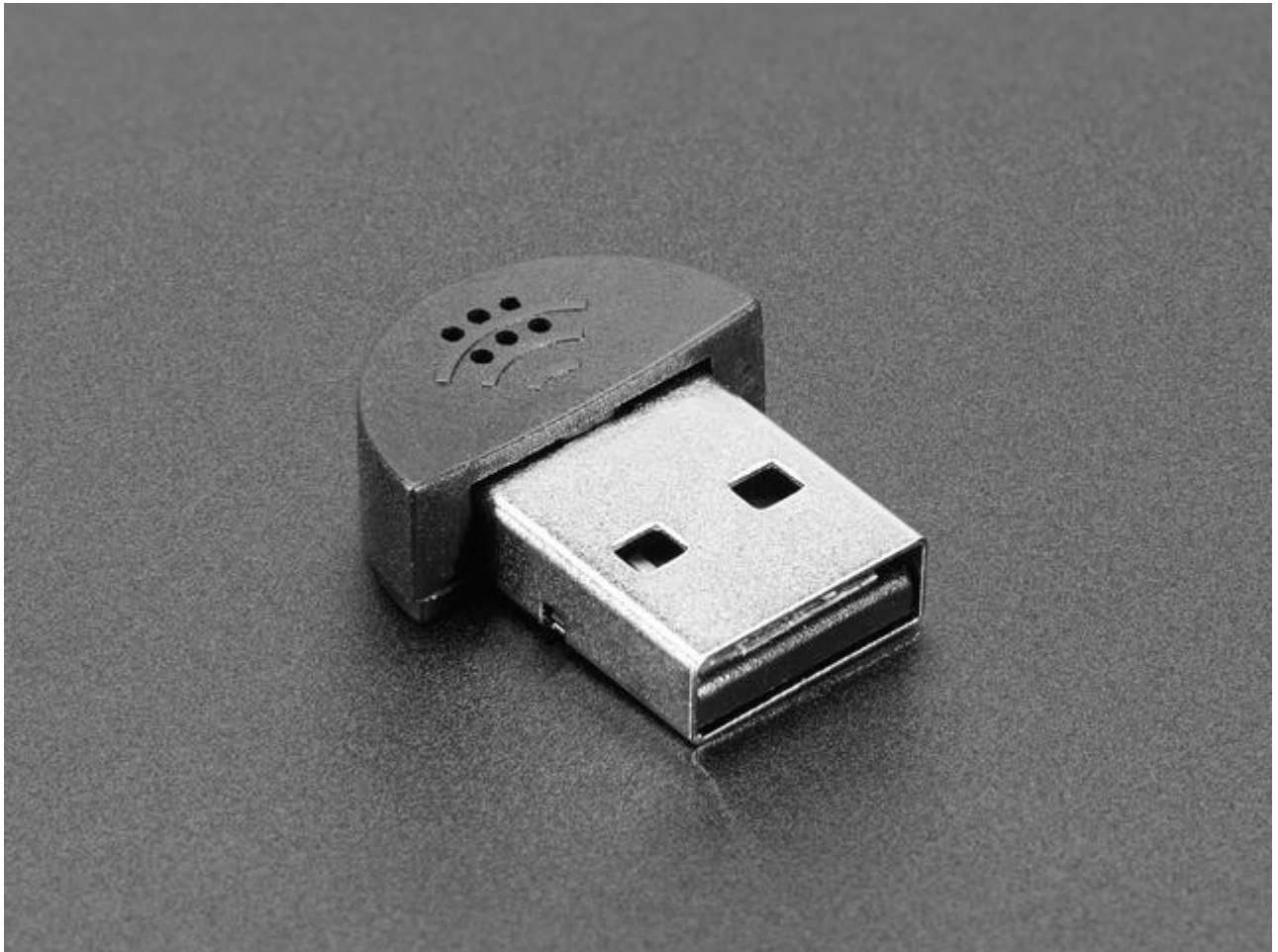


integrat
projects
tablets,
infotain
systems
embedd
[https://
www.ad
product](https://www.adproduct)

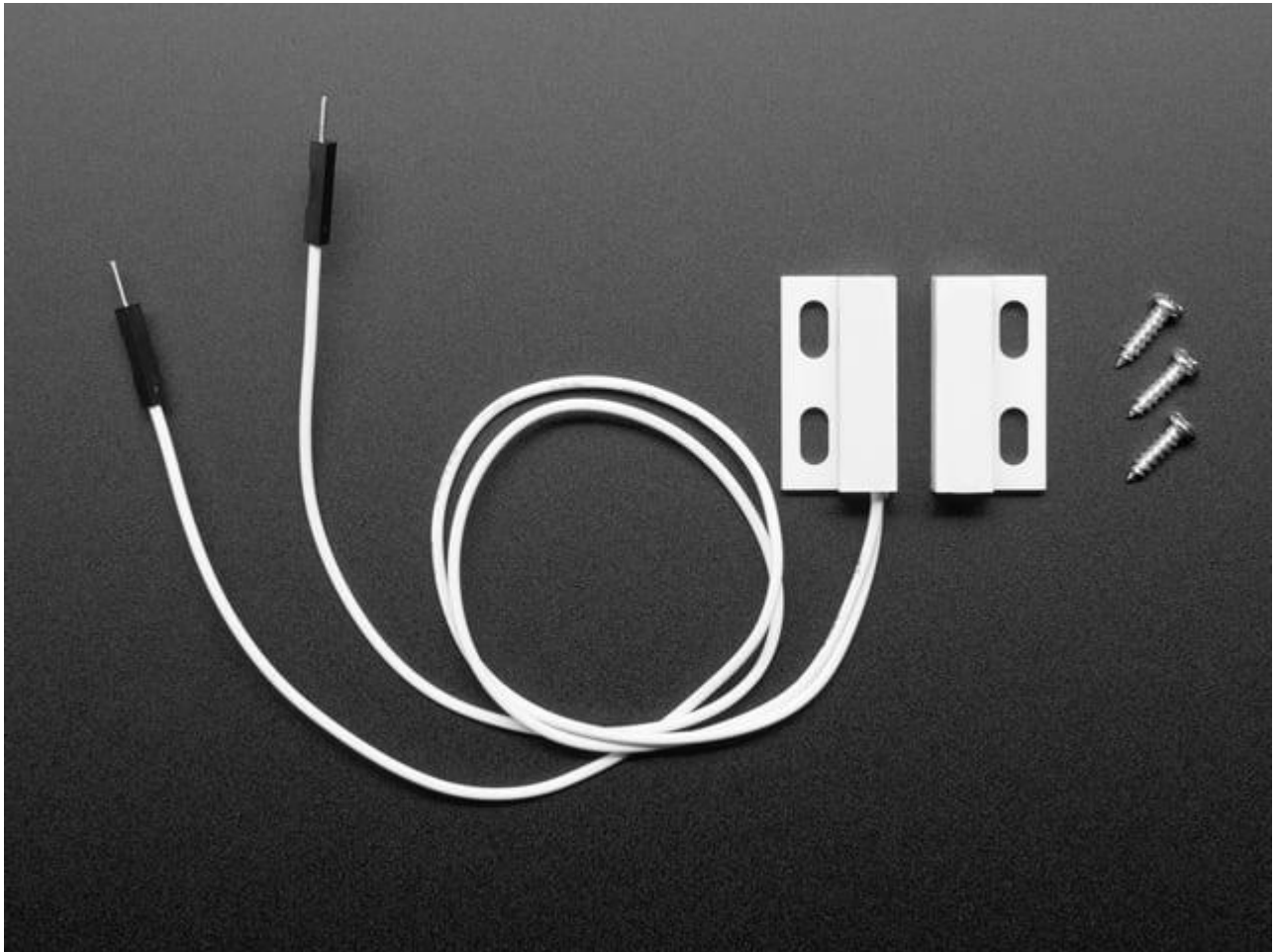
[Raspber](#)
[Model B](#)
[RAM](#)
The Ras
4 Model
newest
Pi comp
made, a
Foundat
you can
make a
better! A
could m
4 better
[https://
www.ad
product](https://www.adproduct)



[Mini USB](#)
[Microph](#)
Hey, list
the wor
smallest
microph
maybe i
the Guin
Record
it is pre
small! T
and-play
[https://](https://www.amazon.com/dp/B08N5M5S7R)
[www.ad](https://www.amazon.com/dp/B08N5M5S7R)
[product](https://www.amazon.com/dp/B08N5M5S7R)



[Magnet...](#)
[switch \(](#)
[sensor\)](#)
This sen
essentia
switch,
an ABS
shell. N
the reed
(no con
between
wires).
half is a
When...
[https://](https://www.ad...)
[www.ad](https://www.ad...)
[product](#)



1 x [Heat Shrink](#)

Heat Shrink Pack

<https://www.adafruit.com/product/344>

1 x [USB Panel Mount](#)

Panel Mount Extension USB Cable - Micro B Male to Micro B Female

<https://www.adafruit.com/product/3258>

1 x [On/Off Switch](#)

USB Cable with Switch

<https://www.adafruit.com/product/1620>

1 x [SD Card](#)

SD/MicroSD Memory Card - 16GB Class 10 - Adapter Included

<https://www.adafruit.com/product/2693>

1 x [Rare Earth Magnet](#)

High-strength 'rare earth' magnet

<https://www.adafruit.com/product/9>

1 x [NeoPixel](#)

Flora RGB Smart NeoPixel version 2 - Pack of 4

<https://www.adafruit.com/product/1260>

Additional Parts & Materials

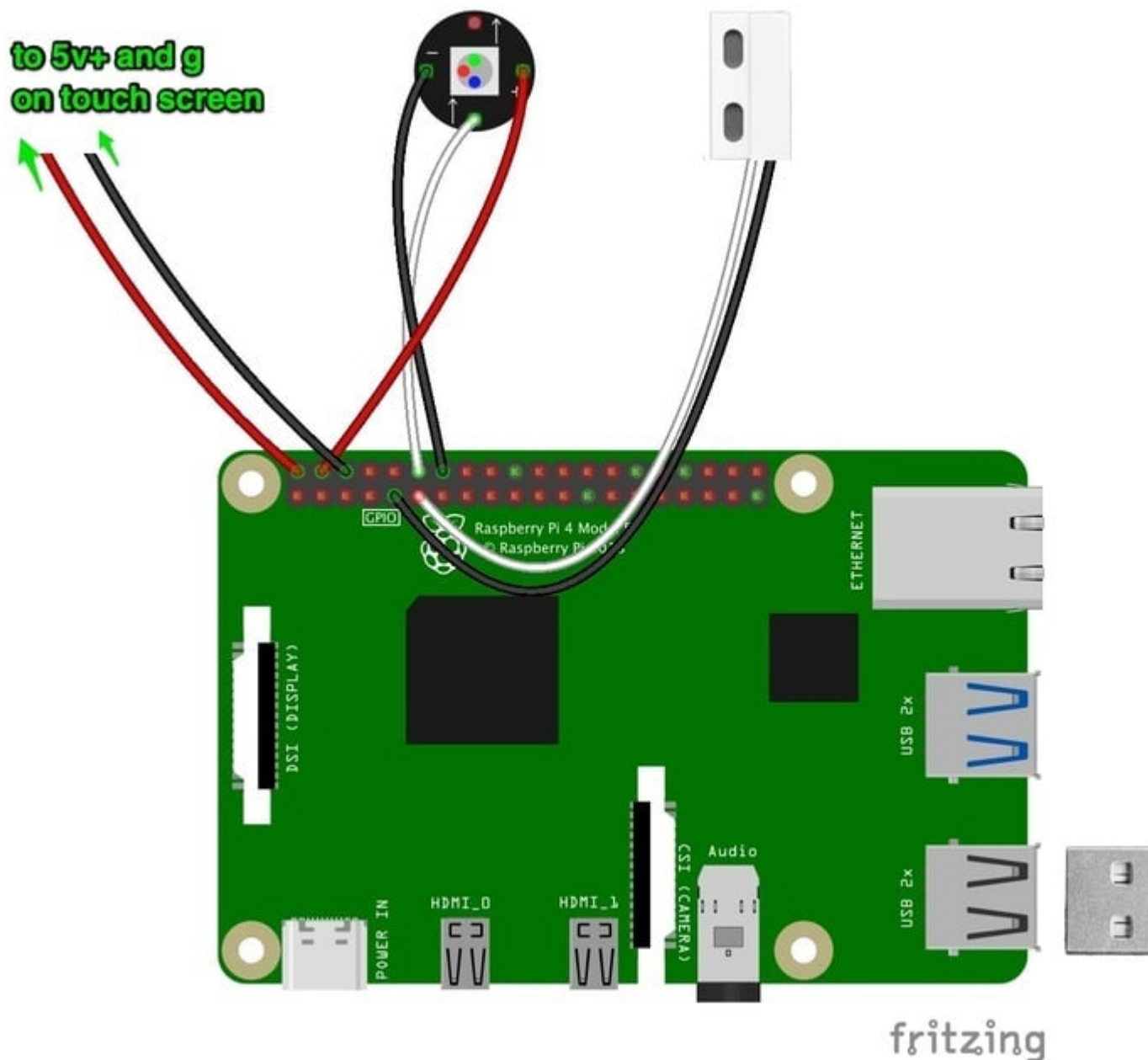
- A book that's at least 2 1/2" thick and larger than 8"x6"
- [A right-angle USB cable](https://adafru.it/18De) (<https://adafru.it/18De>) - this makes an easier fit inside the book

- [A slim USB battery](https://adafru.it/18Dg) (https://adafru.it/18Dg) - I tried a few and this one fit best
- 1/4" Craft foam to use as support inside the book
- Decor: [book corners](https://adafru.it/18Di) (https://adafru.it/18Di), metallic vinyl sticker paper, uv resin, scrapbook supplies and findings to match your aesthetic

Tools

- Jig saw & drill for hollowing out the book (or a very sharp knife and a lot of patience)
- [Very sharp knife](https://adafru.it/18Dj) (https://adafru.it/18Dj) - I like this one from OLFA
- [USB Keyboard](http://adafru.it/1736) (http://adafru.it/1736) for ease in programming the Pi
- E6000 glue, hot glue, white glue, mod podge. You're gonna need lots of glue.

Wiring Diagram



There are three elements to solder to the Raspberry Pi: a touchscreen display, a NeoPixel, and a magnetic reed switch.

<https://pinout.xyz/> (<https://adafru.it/18Dk>) is a great pinout diagram of the Raspberry Pi 4. If you're using a different model, be sure to check the correct pinout and adjust accordingly.

Touchscreen connections

- Pin 2 to Touchscreen 5V+
- Pin 6 to Touchscreen G

NeoPixel connections

- Pin 4 to NeoPixel +
- Pin 12 (GPIO 18) to NeoPixel IN
- Pin 14 to NeoPixel G

Magnetic Switch Connections

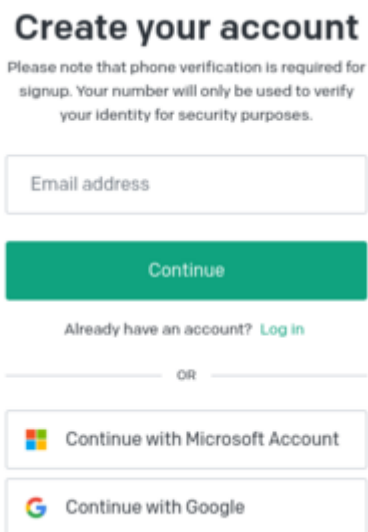
- Pin 9 to one switch wire
- Pin 11 (GPIO 17) to the other switch wire

USB Microphone

Plug the USB microphone into one of the USB ports to connect it.

Create an Account with OpenAI

The OpenAI platform is managed by OpenAI and changes at their discretion, and so the details may be slightly different from what is documented here.

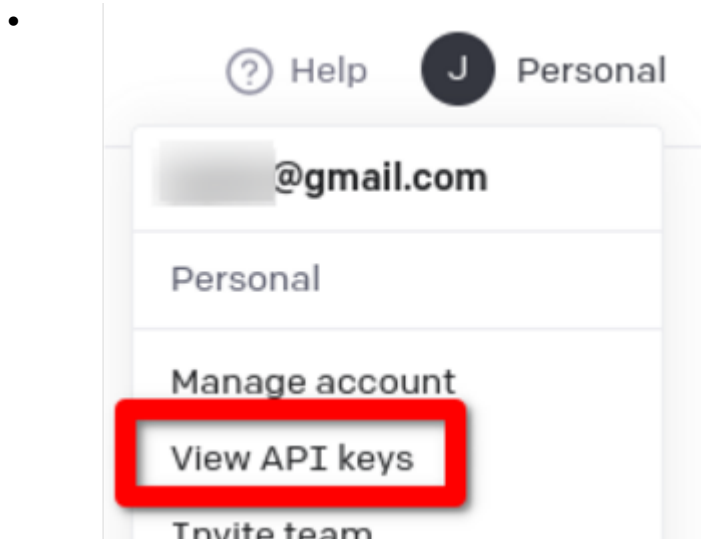
- 

In your web browser, visit <https://platform.openai.com/> (https://adafru.it/18Am)

Click the "sign up" link. Then, you can use your e-mail to sign up, or an existing Google or Microsoft account.

OpenAI may require additional steps such as e-mail or phone verification before you can log in to your account.

Once you have completed the verification process and logged in, you will next create an API key. Use the menu in the far upper right corner (probably labeled "Personal") and then select "View API Keys".

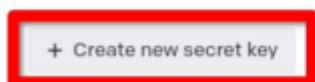


- ## API keys

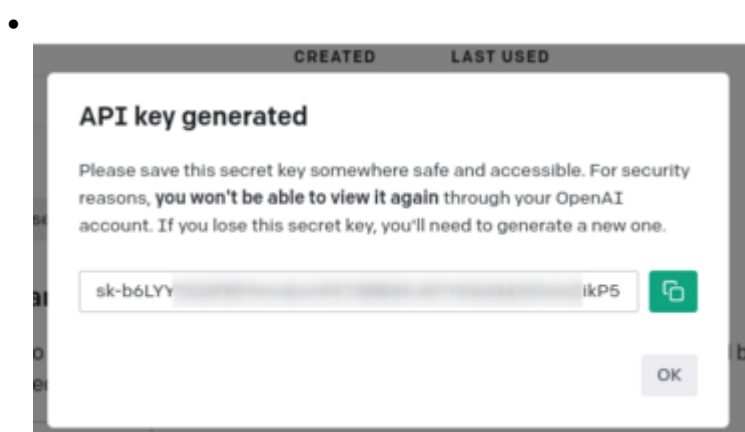
Your secret API keys are listed below. Please note that we do not display the full key after you generate them.

Do not share your API key with others, or expose it in the browser console. To protect the security of your account, OpenAI may also automatically revoke a key if it is found to have leaked publicly.

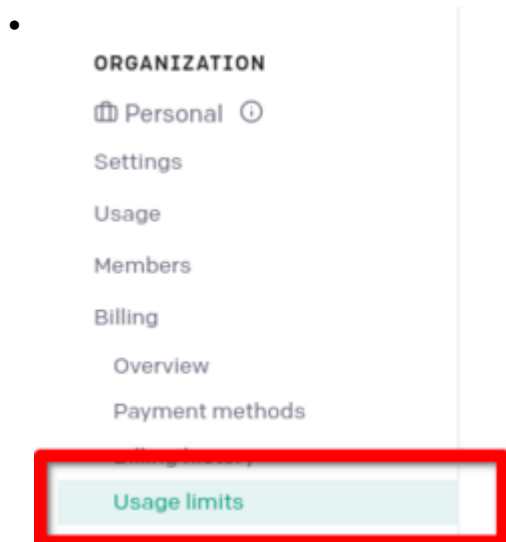
SECRET KEY	CREATED	LAST USED
sk-...9NpZ	Jul 13, 2022	Never
sk-...kWvZ	Feb 15, 2023	Mar 8, 2023



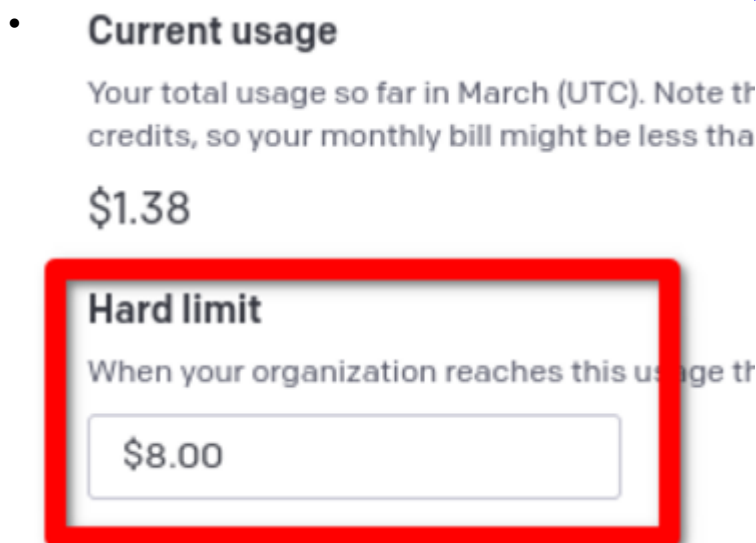
Then, create a fresh API key by clicking "Create new secret key".



Save this secret key to use later in the next step.



At the time of writing, OpenAI provides a free credit with new accounts. After the free credit is used or expires, you'll need to enter a credit card in your billing information to keep using the service.



Using the project tends to cost a few cents per session at most, and it's easy to limit your monthly bill to a pre-set amount such as \$8.00.

To set a hard usage limit per month, visit the "Usage Limits" section of the OpenAI website.

Software Setup

Install Raspberry Pi OS 64-bit Desktop

For the storybook, you will need the 64-bit version of the latest Raspberry Pi OS because the OpenAI libraries will only install on that version. The Desktop version makes displaying graphics much simpler. You can refer to the [CircuitPython Libraries on Linux and Raspberry Pi](https://adafru.it/BSN) (https://adafru.it/BSN) guide for more help setting it up.

After installing, make sure everything is up to date:

```
sudo apt update
sudo apt upgrade
sudo pip3 install --upgrade setuptools
```

Reboot after it completes with:

```
sudo reboot
```

Setup Virtual Environment

Starting with the Bookworm version of Raspberry Pi OS, you will need to install your python modules in a virtual environment. You can find more information in the [Python Virtual Environment Usage on Raspberry Pi](https://adafru.it/19a5) (https://adafru.it/19a5) guide. To Install and activate the virtual environment, use the following commands:

```
sudo apt install python3-venv
python3 -m venv story --system-site-packages
```

You will need to activate the virtual environment every time the Pi is rebooted. To activate it:

```
source story/bin/activate
```

To deactivate, you can use deactivate, but leave it active for now.

Blinka Installation

Once you have everything set up, you will need to open a terminal and install Blinka. Refer to the [Installing CircuitPython Libraries on Raspberry Pi](https://adafru.it/Deo) (https://adafru.it/Deo) page to quickly get up and running.

Run the raspi-blinka script, which will install Blinka and update some additional environment settings. You will be installing the NeoPixel library further down, which requires root access, so you will need to install everything with sudo:

```
cd ~
pip3 install --upgrade adafruit-python-shell
wget https://raw.githubusercontent.com/adafruit/Raspberry-Pi-Installer-Scripts/master/raspi-blinka.py
sudo -E env PATH=$PATH python3 raspi-blinka.py
```

Reboot after it completes.

```
pi@magicbook: ~  
l/lib/python3.9/dist-packages (from adafruit-blinka) (1.9.2)  
Blinka Requirement already satisfied: RPi.GPIO in /usr/local/lib/python3.9/dist-  
packages (from adafruit-blinka) (0.7.1)  
Blinka Requirement already satisfied: sysv-ipc>=1.1.0 in /usr/local/lib/python3.  
9/dist-packages (from adafruit-blinka) (1.1.0)  
Blinka Requirement already satisfied: pyusb!=1.2.0,>=1.0.0 in /usr/local/lib/pyt  
hon3.9/dist-packages (from pyftdi>=0.40.0->adafruit-blinka) (1.2.1)  
Blinka Requirement already satisfied: pyserial>=3.0 in /usr/lib/python3/dist-pac  
kages (from pyftdi>=0.40.0->adafruit-blinka) (3.5b0)  
Blinka Requirement already satisfied: typing-extensions~4.0 in /usr/local/lib/p  
ython3.9/dist-packages (from adafruit-circuitpython-typing->adafruit-blinka) (4.  
5.0)  
Blinka Requirement already satisfied: adafruit-circuitpython-busdevice in /usr/l  
ocal/lib/python3.9/dist-packages (from adafruit-circuitpython-typing->adafruit-b  
linka) (5.2.4)  
Blinka Requirement already satisfied: adafruit-circuitpython-requests in /usr/lo  
cal/lib/python3.9/dist-packages (from adafruit-circuitpython-typing->adafruit-bl  
inka) (1.13.2)  
DONE.  
  
Settings take effect on next boot.  
  
REBOOT NOW? [Y/n]  
Reboot started...
```

Install OpenAI Library

The main library that you will need for this project is the OpenAI library, which allows you to interact with the OpenAI:

```
pip3 install --upgrade openai
```

Additional Required Dependencies

A few more additional libraries and other dependencies are required including the NeoPixel and LED Animation libraries:

```
sudo apt install libpulse-dev pulseaudio apulse python3-pyaudio  
pip3 install --upgrade SpeechRecognition pygame adafruit-circuitpython-neo
```

Enable Backlight Control

The easiest way to enable backlight control is through the rpi-backlight library, which was installed in the previous section. In order for the library to work properly, you will need to add a rules file, which gives it access to the lower level hardware. You can create the rules file by running the following command:

```
echo 'SUBSYSTEM=="backlight",RUN+="/bin/chmod 666 /sys/class/backlight/%k/  
SUBSYSTEM=="backlight",RUN+="/bin/chmod 666 /sys/class/backlight/%k/bright
```

Download the Files

This project uses quite a few files. The easiest way to add the files to your Pi with the folders intact and without downloading the entirety of the Learn code is to use git with the sparse-checkout option. You can copy the relevant folder with the following commands:

```
cd ~  
git clone -n --depth=1 --filter=tree:0 https://github.com/adafruit/Adafruit  
cd Adafruit_Learning_System_Guides  
git sparse-checkout set --no-cone Magic_AI_Storybook  
git checkout  
mv Magic_AI_Storybook ..  
cd ../Magic_AI_Storybook  
rm -Rf ../Adafruit_Learning_System_Guides
```

Move Prompt and Key Files

There are a couple of files that will need to be moved outside of this folder. The first file is **bookprompt.txt**. You will be setting the file system to be read-only soon and moving this file to the **/boot** folder will allow you to easily change it on the SD Card plugged into a computer:

```
sudo mv bookprompt.txt /boot/ &gt; /dev/null 2>&1
```

The second file **keys.txt** is where you will provide your API keys. It is moved outside of the folder in order to prevent accidentally committing the code to the repository. It is placed in a text file instead of using environment variables in case you want to have the script start up automatically where you can't depend on the contents of the environment variables.

```
mv keys.txt ..
```

Add Your OpenAI Key

Add the secret key you created when you set up your OpenAI account to the **keys.txt** file in your home directory. From the command line, you can open it in your favorite editor such as nano:

```
nano ~/keys.txt
```

Change the value of the key so it reads something like:

```
OPENAI_API_KEY = sk-b6j4FFt78209dkifJhld783GtkP5
```

Update ALSA Config

With a default installation of the SpeechRecognition library and the ALSA config, a large number of warnings will appear whenever voice input is initiated. While the warnings are being printed to the console, there is a noticeable lag of a couple seconds before it actually starts listening. You can reduce this by making the following changes.

Open **/usr/share/alsa/alsa.conf** in a text editor. About halfway down you will find these lines:

```
# redirect to load-on-demand extended pcm definitions
pcm.cards cards.pcm

pcm.default cards.pcm.default
pcm.sysdefault cards.pcm.default
pcm.front cards.pcm.front
pcm.rear cards.pcm.rear
pcm.center_lfe cards.pcm.center_lfe
pcm.side cards.pcm.side
pcm.surround21 cards.pcm.surround21
pcm.surround40 cards.pcm.surround40
pcm.surround41 cards.pcm.surround41
pcm.surround50 cards.pcm.surround50
pcm.surround51 cards.pcm.surround51
pcm.surround71 cards.pcm.surround71
pcm.iec958 cards.pcm.iec958
pcm.spdif iec958
pcm.hdmi cards.pcm.hdmi
pcm.dmix cards.pcm.dmix
pcm.dsnoop cards.pcm.dsnoop
pcm.modem cards.pcm.modem
pcm.phoneline cards.pcm.phoneline
```

Comment out everything from `pcm.front cards.pcm.front` on down with a `#` at the beginning of the line. The result should look like this:

```
# redirect to load-on-demand extended pcm definitions
pcm.cards cards.pcm

pcm.default cards.pcm.default
pcm.sysdefault cards.pcm.default
#pcm.front cards.pcm.front
#pcm.rear cards.pcm.rear
#pcm.center_lfe cards.pcm.center_lfe
#pcm.side cards.pcm.side
#pcm.surround21 cards.pcm.surround21
#pcm.surround40 cards.pcm.surround40
#pcm.surround41 cards.pcm.surround41
#pcm.surround50 cards.pcm.surround50
#pcm.surround51 cards.pcm.surround51
#pcm.surround71 cards.pcm.surround71
```

```
#pcm.iec958 cards.pcm.iec958
#pcm.spdif iec958
#pcm.hdmi cards.pcm.hdmi
#pcm.dmix cards.pcm.dmix
#pcm.dsnoop cards.pcm.dsnoop
#pcm.modem cards.pcm.modem
#pcm.phoneline cards.pcm.phoneline
```

Create Desktop and Autostart Shortcuts

To create a desktop shortcut icon, there is a small script included that uses your current username to write the correct paths. While the script can correctly run using `sudo`, just running it without `sudo` is preferable, so that you can easily modify or delete the shortcuts in the future. To run the script, just use the following commands:

```
cd ~/Magic_AI_Storybook
python make_shortcut.py
```

Now the script should automatically run upon startup. If you don't want it to automatically run, you can simply delete the file at `~/.config/autostart/storybook.desktop`.

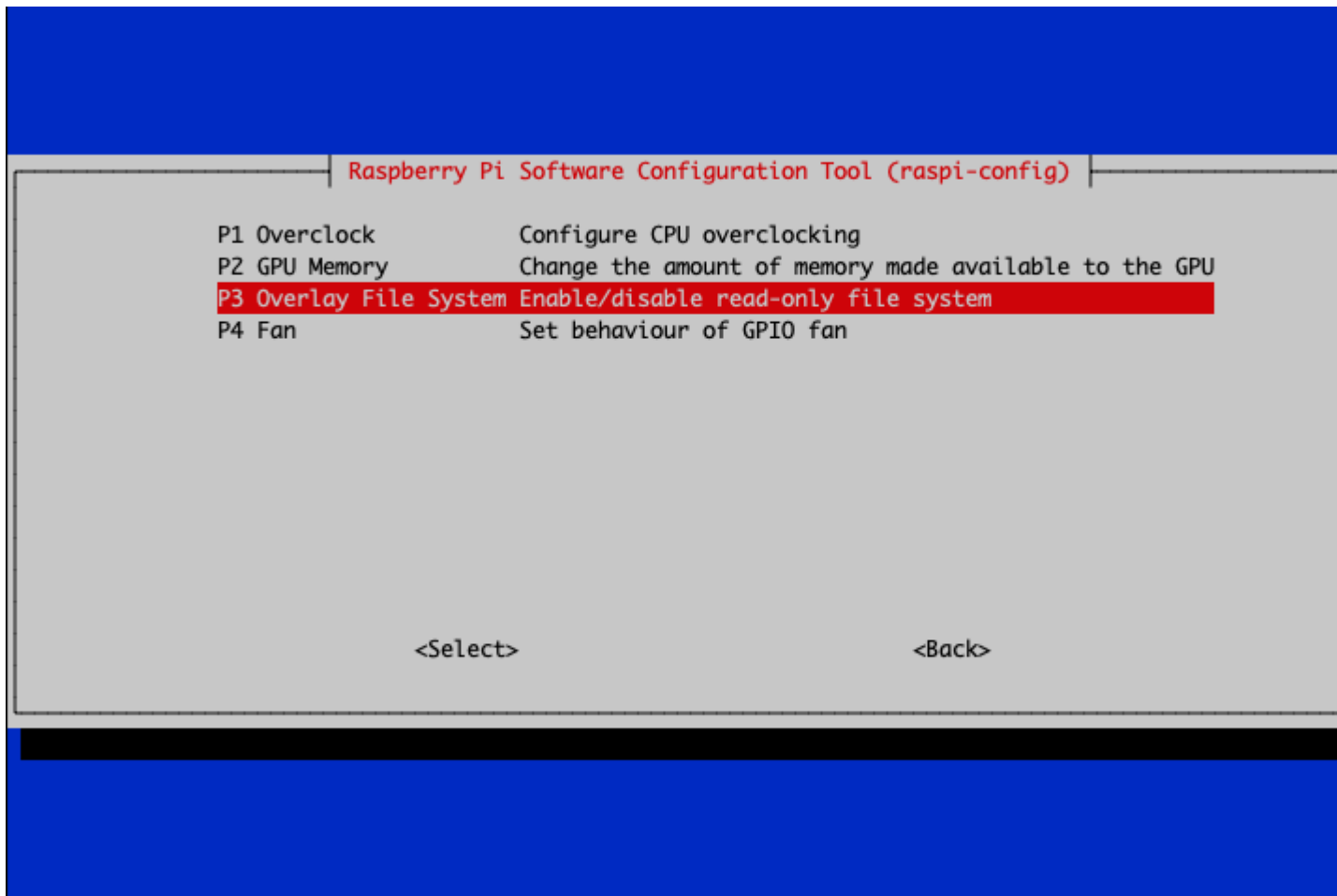
Optional: Making the File System Read-only

In order to turn the Raspberry Pi off without risking damaging the file system, you may wish to put it in read-only mode, which is called the "overlay". You can still make changes directly to files on the SD Card while it is plugged into another computer if needed or you can disable the Overlay File System to allow writing again.

If you would like to customize your code, you may want to wait on this step and come back after making changes.

To enable it, run `sudo raspi-config`, then go to **Performance Options** → **Overlay File System**. Choose **Yes** to enable the Overlay File System. This step may take a moment to finish. Then choose **Yes** again to write-protect the boot partition.

Go ahead and **reboot**. The file system should now be read-only. Any new file changes will be discarded when the Pi is rebooted.



Code Overview

This is intended to be a high level overview of the code, since it is relatively complex. There are two files that handle the storybook code. These are the **story.py** file, which handles the majority of the code and the **listener.py** file, which handles the listening and speech recognition tasks.

Story Code

```
# SPDX-FileCopyrightText: 2023 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import threading
import sys
import os
import re
import time
import argparse
import math
import configparser
from enum import Enum
```

```

from collections import deque

import board
import digitalio
import neopixel
from openai import OpenAI
import pygame
from rpi_backlight import Backlight
from adafruit_led_animation.animation.pulse import Pulse

from listener import Listener

# Base Path is the folder the script resides in
BASE_PATH = os.path.dirname(sys.argv[0])
if BASE_PATH != "":
    BASE_PATH += "/"

# General Settings
STORY_WORD_LENGTH = 800
REED_SWITCH_PIN = board.D17
NEOPIXEL_PIN = board.D18
API_KEYS_FILE = "~/keys.txt"
PROMPT_FILE = "/boot/bookprompt.txt"

# Quit Settings (Close book QUIT_CLOSSES within QUIT_TIME_PERIOD to quit)
QUIT_CLOSSES = 3
QUIT_TIME_PERIOD = 5 # Time period in Seconds
QUIT_DEBOUNCE_DELAY = 0.25 # Time to wait before counting next closing

# Neopixel Settings
NEOPIXEL_COUNT = 1
NEOPIXEL_BRIGHTNESS = 0.2
NEOPIXEL_ORDER = neopixel.GRBW
NEOPIXEL_LOADING_COLOR = (0, 255, 0, 0) # Loading/Dreaming (Green)
NEOPIXEL_SLEEP_COLOR = (0, 0, 0, 0) # Sleeping (Off)
NEOPIXEL_WAITING_COLOR = (255, 255, 0, 0) # Waiting for Input (Yellow)
NEOPIXEL_READING_COLOR = (0, 0, 255, 0) # Reading (Blue)
NEOPIXEL_PULSE_SPEED = 0.1

# Image Settings
WELCOME_IMAGE = "welcome.png"
BACKGROUND_IMAGE = "paper_background.png"
LOADING_IMAGE = "loading.png"
BUTTON_BACK_IMAGE = "button_back.png"
BUTTON_NEXT_IMAGE = "button_next.png"
BUTTON_NEW_IMAGE = "button_new.png"

# Asset Paths
IMAGES_PATH = BASE_PATH + "images/"
FONTS_PATH = BASE_PATH + "fonts/"

# Font Path & Size

```

```

TITLE_FONT = (FONTS_PATH + "Desdemona Black Regular.otf", 48)
TITLE_COLOR = (0, 0, 0)
TEXT_FONT = (FONTS_PATH + "times new roman.ttf", 24)
TEXT_COLOR = (0, 0, 0)

# Delays Settings
# Used to control the speed of the text
WORD_DELAY = 0.1
TITLE_FADE_TIME = 0.05
TITLE_FADE_STEPS = 25
TEXT_FADE_TIME = 0.25
TEXT_FADE_STEPS = 51
ALSA_ERROR_DELAY = 0.5 # Delay to wait after an ALSA errors

# Whitespace Settings (in Pixels)
PAGE_TOP_MARGIN = 20
PAGE_SIDE_MARGIN = 20
PAGE_BOTTOM_MARGIN = 0
PAGE_NAV_HEIGHT = 100
EXTRA_LINE_SPACING = 0
PARAGRAPH_SPACING = 30

# ChatGPT Parameters
SYSTEM_ROLE = "You are a master AI Storyteller that can tell a story of an"
CHATGPT_MODEL = "gpt-3.5-turbo" # You can also use "gpt-4", which is slow
WHISPER_MODEL = "whisper-1"

# Speech Recognition Parameters
ENERGY_THRESHOLD = 300 # Energy level for mic to detect
RECORD_TIMEOUT = 30 # Maximum time in seconds to wait for speech

# Do some checks and Import API keys from API_KEYS_FILE
config = configparser.ConfigParser()

if os.geteuid() != 0:
    print("Please run this script as root.")
    sys.exit(1)
username = os.environ["SUDO_USER"]
user_homedir = os.path.expanduser(f"~{username}")
API_KEYS_FILE = API_KEYS_FILE.replace("~", user_homedir)

print(os.path.expanduser(API_KEYS_FILE))
config.read(os.path.expanduser(API_KEYS_FILE))
if not config.has_section("openai"):
    print("Please make sure API_KEYS_FILE points to a valid file.")
    sys.exit(1)
if "OPENAI_API_KEY" not in config["openai"]:
    print(
        "Please make sure your API keys file contains an OPENAI_API_KEY un
    )
    sys.exit(1)
if len(config["openai"]["OPENAI_API_KEY"]) < 10:

```

```

        print("Please set OPENAI_API_KEY in your API keys file with a valid key")
        sys.exit(1)
    openai = OpenAI(
        # This is the default and can be omitted
        api_key=config["openai"]["OPENAI_API_KEY"],
    )

# Check that the prompt file exists and load it
if not os.path.isfile(PROMPT_FILE):
    print("Please make sure PROMPT_FILE points to a valid file.")
    sys.exit(1)

def strip_fancy_quotes(text):
    text = re.sub(r"[\u2018\u2019]", "'", text)
    text = re.sub(r"[\u201C\u201D]", '"', text)
    return text

class Position(Enum):
    TOP = 0
    CENTER = 1
    BOTTOM = 2
    LEFT = 3
    RIGHT = 4

class Button:
    def __init__(self, x, y, image, action, draw_function):
        self.x = x
        self.y = y
        self.image = image
        self.action = action
        self._width = self.image.get_width()
        self._height = self.image.get_height()
        self._visible = False
        self._draw_function = draw_function

    def is_in_bounds(self, position):
        x, y = position
        return (
            self.x <= x <= self.x + self.width and self.y <= y <= self.y + self.height
        )

    def show(self):
        self._draw_function(self.image, self.x, self.y)
        self._visible = True

    @property
    def width(self):
        return self._width

```

```
@property
def height(self):
    return self._height
```

```
@property
def visible(self):
    return self._visible
```

```
class Textarea:
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height

    @property
    def size(self):
        return {"width": self.width, "height": self.height}
```

```
class Book:
    def __init__(self, rotation=0):
        self.paragraph_number = 0
        self.page = 0
        self.pages = []
        self.stories = []
        self.story = 0
        self.rotation = rotation
        self.images = {}
        self.fonts = {}
        self.buttons = {}
        self.width = 0
        self.height = 0
        self.textarea = None
        self.screen = None
        self.saved_screen = None
        self._sleeping = False
        self.sleep_check_delay = 0.1
        self._sleep_check_thread = None
        self._sleep_request = False
        self._running = True
        self._busy = False
        self._loading = False
        # Use a Double Ended Queue to handle the heavy lifting
        self._closing_times = deque(maxlen=QUIT_CLOSES)
        # Use a cursor to keep track of where we are in the text area
        self.cursor = {"x": 0, "y": 0}
        self.listener = None
        self.backlight = Backlight()
        self.pixels = neopixel.NeoPixel(
            NEOPIXEL_PIN,
```

```

        NEOPIXEL_COUNT,
        brightness=NEOPIXEL_BRIGHTNESS,
        pixel_order=NEOPIXEL_ORDER,
        auto_write=False,
    )
    self._prompt = ""
    self._load_thread = threading.Thread(target=self._handle_loading_s
    self._load_thread.start()

def start(self):
    # Output to the LCD instead of the console
    os.putenv("DISPLAY", ":0")

    self._set_status_color(NEOPIXEL_LOADING_COLOR)

    # Initialize the display
    pygame.init()
    self.screen = pygame.display.set_mode((0, 0), pygame.FULLSCREEN)
    pygame.mouse.set_visible(False)
    self.screen.fill((255, 255, 255))
    self.width = self.screen.get_height()
    self.height = self.screen.get_width()

    # Preload welcome image and display it
    self._load_image("welcome", WELCOME_IMAGE)
    self.display_welcome()

    # Load the prompt file
    with open(PROMPT_FILE, "r") as f:
        self._prompt = f.read()

    # Initialize the Listener
    self.listener = Listener(
        openai.api_key, ENERGY_THRESHOLD, RECORD_TIMEOUT
    )

    # Preload remaining images
    self._load_image("background", BACKGROUND_IMAGE)
    self._load_image("loading", LOADING_IMAGE)

    # Preload fonts
    self._load_font("title", TITLE_FONT)
    self._load_font("text", TEXT_FONT)

    # Add buttons
    back_button_image = pygame.image.load(IMAGES_PATH + BUTTON_BACK_IM
    next_button_image = pygame.image.load(IMAGES_PATH + BUTTON_NEXT_IM
    new_button_image = pygame.image.load(IMAGES_PATH + BUTTON_NEW_IMAG
    button_spacing = (
        self.width
        - (
            back_button_image.get_width()

```



```

        + next_button_image.get_width()
        + new_button_image.get_width()
    )
) // 4
button_ypos = (
    self.height
    - PAGE_NAV_HEIGHT
    + (PAGE_NAV_HEIGHT - next_button_image.get_height()) // 2
)

self._load_button(
    "back",
    button_spacing,
    button_ypos,
    back_button_image,
    self.previous_page,
    self._display_surface,
)

self._load_button(
    "new",
    button_spacing * 2 + back_button_image.get_width(),
    button_ypos,
    new_button_image,
    self.new_story,
    self._display_surface,
)

self._load_button(
    "next",
    button_spacing * 3
    + back_button_image.get_width()
    + new_button_image.get_width(),
    button_ypos,
    next_button_image,
    self.next_page,
    self._display_surface,
)

# Add Text Area
self.textarea = Textarea(
    PAGE_SIDE_MARGIN,
    PAGE_TOP_MARGIN,
    self.width - PAGE_SIDE_MARGIN * 2,
    self.height - PAGE_NAV_HEIGHT - PAGE_TOP_MARGIN - PAGE_BOTTOM_MARGIN,
)

# Start the sleep check thread after everything is initialized
self._sleep_check_thread = threading.Thread(target=self._handle_sleep_check)
self._sleep_check_thread.start()

self._set_status_color(NEOPIXEL_READING_COLOR)

```

```

def deinit(self):
    self._running = False
    self._sleep_check_thread.join()
    self._load_thread.join()
    self.backlight.power = True

def _handle_sleep(self):
    reed_switch = digitalio.DigitalInOut(REED_SWITCH_PIN)
    reed_switch.direction = digitalio.Direction.INPUT
    reed_switch.pull = digitalio.Pull.UP

    while self._running:
        if self._sleeping and reed_switch.value: # Book Open
            self._wake()
        elif not self._sleeping and not reed_switch.value:
            self._sleep()
        time.sleep(self.sleep_check_delay)

def _handle_loading_status(self):
    pulse = Pulse(
        self.pixels,
        speed=NEOPIXEL_PULSE_SPEED,
        color=NEOPIXEL_LOADING_COLOR,
        period=3,
    )

    while self._running:
        if self._loading:
            pulse.animate()
            time.sleep(0.1)

    # Turn off the Neopixels
    self.pixels.fill(0)
    self.pixels.show()

def _set_status_color(self, status_color):
    if status_color not in [
        NEOPIXEL_READING_COLOR,
        NEOPIXEL_WAITING_COLOR,
        NEOPIXEL_SLEEP_COLOR,
        NEOPIXEL_LOADING_COLOR,
    ]:
        raise ValueError(f"Invalid status color {status_color}.")

    # Handle loading color by setting the loading flag
    self._loading = status_color == NEOPIXEL_LOADING_COLOR

    # Handle other status colors by setting the neopixels
    if status_color != NEOPIXEL_LOADING_COLOR:
        self.pixels.fill(status_color)
        self.pixels.show()

```

```

def handle_events(self):
    if not self._sleeping:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                raise SystemExit
            if event.type == pygame.MOUSEBUTTONDOWN:
                self._handle_mousedown_event(event)
        time.sleep(0.1)

def _handle_mousedown_event(self, event):
    if event.button == 1:
        # If button pressed while visible, trigger action
        coords = self._rotate_mouse_pos(event.pos)
        for button in self.buttons.values():
            if button.visible and button.is_in_bounds(coords):
                button.action()

def _rotate_mouse_pos(self, point):
    # Recalculate the mouse position based on the rotation of the screen
    # So that we have the coordinates relative to the upper left corner
    angle = 360 - self.rotation
    y, x = point
    x -= self.width // 2
    y -= self.height // 2
    x, y = x * math.sin(math.radians(angle)) + y * math.cos(
        math.radians(angle)
    ), x * math.cos(math.radians(angle)) - y * math.sin(math.radians(a
    x += self.width // 2
    y += self.height // 2
    return (round(x), round(y))

def _load_image(self, name, filename):
    try:
        image = pygame.image.load(IMAGES_PATH + filename)
        self.images[name] = image
    except pygame.error:
        pass

def _load_button(self, name, x, y, image, action, display_surface):
    self.buttons[name] = Button(x, y, image, action, display_surface)

def _load_font(self, name, details):
    self.fonts[name] = pygame.font.Font(details[0], details[1])

def _display_surface(self, surface, x=0, y=0, target_surface=None):
    # Display a surface either positionally or with a specific x,y coo
    buffer = self._create_transparent_buffer((self.width, self.height))
    buffer.blit(surface, (x, y))
    if target_surface is None:
        buffer = pygame.transform.rotate(buffer, self.rotation)
        self.screen.blit(buffer, (0, 0))

```

```

else:
    target_surface.blit(buffer, (0, 0))

def _fade_in_surface(self, surface, x, y, fade_time, fade_steps=50):
    background = self._create_transparent_buffer((self.width, self.height))
    self._display_surface(self.images["background"], 0, 0, background)

    buffer = self._create_transparent_buffer(surface.get_size())
    fade_delay = round(
        fade_time / fade_steps * 1000
    ) # Time to delay in ms between each fade step

    def draw_alpha(alpha):
        buffer.blit(background, (-x, -y))
        surface.set_alpha(alpha)
        buffer.blit(surface, (0, 0))
        self._display_surface(buffer, x, y)
        pygame.display.update()

    for alpha in range(0, 255, round(255 / fade_steps)):
        draw_alpha(alpha)
        pygame.time.wait(fade_delay)
        if self._sleep_request:
            draw_alpha(255) # Finish up quickly
            return

def display_current_page(self):
    self._busy = True
    self._display_surface(self.images["background"], 0, 0)
    pygame.display.update()

    print(f"Loading page {self.page} of {len(self.pages)}")
    page_data = self.pages[self.page]

    # Display the title
    if page_data["title"]:
        self._display_title_text(page_data["title"])

    self._fade_in_surface(
        page_data["buffer"],
        self.textarea.x,
        self.textarea.y + page_data["text_position"],
        TEXT_FADE_TIME,
        TEXT_FADE_STEPS,
    )

    # Display the navigation buttons
    if self.page > 0 or self.story > 0:
        self.buttons["back"].show()
    self.buttons["next"].show()
    self.buttons["new"].show()
    pygame.display.update()

```

```

        self._busy = False

    @staticmethod
    def _create_transparent_buffer(size):
        if isinstance(size, (tuple, list)):
            (width, height) = size
        elif isinstance(size, dict):
            width = size["width"]
            height = size["height"]
        else:
            raise ValueError(f"Invalid size {size}. Should be tuple, list,
buffer = pygame.Surface((width, height), pygame.SRCALPHA, 32)
buffer = buffer.convert_alpha()
return buffer

    def _display_title_text(self, text, y=0):
        # Render the title as multiple lines if too big
        lines = self._wrap_text(text, self.fonts["title"], self.textarea.w
self.cursor["y"] = y
delay_value = WORD_DELAY
for line in lines:
    words = line.split(" ")
    self.cursor["x"] = (
        self.textarea.width // 2 - self.fonts["title"].size(line)[
    )
    for word in words:
        text = self.fonts["title"].render(word + " ", True, TITLE_
        if self._sleep_request:
            delay_value = 0
            self._display_surface(
                text,
                self.cursor["x"] + self.textarea.x,
                self.cursor["y"] + self.textarea.y,
            )
        else:
            self._fade_in_surface(
                text,
                self.cursor["x"] + self.textarea.x,
                self.cursor["y"] + self.textarea.y,
                TITLE_FADE_TIME,
                TITLE_FADE_STEPS,
            )

        pygame.display.update()
        self.cursor["x"] += text.get_width()
        time.sleep(delay_value)
        self.cursor["y"] += self.fonts["title"].size(line)[1]

    def _title_text_height(self, text):
        lines = self._wrap_text(text, self.fonts["title"], self.textarea.w
        height = 0
        for line in lines:

```

```

        height += self.fonts["title"].size(line)[1]
    return height

    @staticmethod
    def _wrap_text(text, font, width):
        lines = []
        line = ""
        for word in text.split(" "):
            if font.size(line + word)[0] < width:
                line += word + " "
            else:
                lines.append(line)
                line = word + " "
        lines.append(line)
        return lines

    def previous_page(self):
        if self.page > 0 or self.story > 0:
            self.page -= 1
            if self.page < 0:
                self.story -= 1
                self.load_story(self.stories[self.story])
                self.page = len(self.pages) - 1
            self.display_current_page()

    def next_page(self):
        self.page += 1
        if self.page >= len(self.pages):
            if self.story < len(self.stories) - 1:
                self.story += 1
                self.load_story(self.stories[self.story])
                self.page = 0
            else:
                self.generate_new_story()
        self.display_current_page()

    def new_story(self):
        self.generate_new_story()
        self.display_current_page()

    def display_loading(self):
        self._display_surface(self.images["loading"], 0, 0)
        pygame.display.update()
        self._set_status_color(NEOPIXEL_LOADING_COLOR)

    def display_welcome(self):
        self._display_surface(self.images["welcome"], 0, 0)
        pygame.display.update()

    def display_message(self, message):
        self._busy = True
        self._display_surface(self.images["background"], 0, 0)

```



```

        height = self._title_text_height(message)
        self._display_title_text(message, self.height // 2 - height // 2)
        self._busy = False

def load_story(self, story):
    # Parse out the title and story and render into pages
    self._busy = True
    self.pages = []
    if not story.startswith("Title: "):
        print("Unexpected story format from ChatGPT. Missing Title.")
        title = "A Story"
    else:
        title = story.split("Title: ")[1].split("\n\n")[0]
    page = self._add_page(title)
    paragraphs = story.split("\n\n")[1:]
    for paragraph in paragraphs:
        lines = self._wrap_text(paragraph, self.fonts["text"], self.textarea.width)
        for line in lines:
            self.cursor["x"] = 0
            text = self.fonts["text"].render(line, True, TEXT_COLOR)
            if (
                self.cursor["y"] + self.fonts["text"].get_height()
                > page["buffer"].get_height()
            ):
                page = self._add_page()

            self._display_surface(
                text, self.cursor["x"], self.cursor["y"], page["buffer"]
            )
            self.cursor["y"] += self.fonts["text"].size(line)[1]

        if self.cursor["y"] > 0:
            self.cursor["y"] += PARAGRAPH_SPACING
    print(f"Loaded story at index {self.story} with {len(self.pages)}")
    self._set_status_color(NEOPIXEL_READING_COLOR)
    self._busy = False

def _add_page(self, title=None):
    page = {
        "title": title,
        "text_position": 0,
    }
    if title:
        page["text_position"] = self._title_text_height(title) + PARAGRAPH_SPACING
    page["buffer"] = self._create_transparent_buffer(
        (self.textarea.width, self.textarea.height - page["text_position"])
    )
    self.cursor["y"] = 0
    self.pages.append(page)
    return page

def generate_new_story(self):

```

```

self._busy = True
self.display_message("Speak aloud the story you wish to read.")

if self._sleep_request:
    self._busy = False
    time.sleep(0.2)
    return

def show_listening():
    # Pause for a beat because the listener doesn't
    # immediately start listening sometimes
    time.sleep(ALSA_ERROR_DELAY)
    self.pixels.fill(NEOPIXEL_WAITING_COLOR)
    self.pixels.show()

self.listener.listen(ready_callback=show_listening)

if self._sleep_request:
    self._busy = False
    return

if not self.listener.speech_waiting():
    # No response from user, so return
    print("No response from user.")
    return

story_request = self.listener.recognize()
print(f"Whisper heard: {story_request}")
story_prompt = self._make_story_prompt(story_request)
self.display_loading()
response = self._sendchat(story_prompt)
if self._sleep_request:
    self._busy = False
    return
print(response)

self._busy = True
self.stories.append(response)
self.story = len(self.stories) - 1
self.page = 0
self._busy = False

self.load_story(response)

def _sleep(self):
    # Set a sleep request flag so that any busy threads know to finish
    self._sleep_request = True
    if self.listener.is_listening():
        self.listener.stop_listening()
    while self._busy:
        time.sleep(0.1)
    self._sleep_request = False

```

```

if (
    len(self._closing_times) == 0
    or (time.monotonic() - self._closing_times[-1]) > QUIT_DEBOUNCE
):
    self._closing_times.append(time.monotonic())

# Check if we've closed the book a certain number of times
# within a certain number of seconds
if (
    len(self._closing_times) == QUIT_CLOSSES
    and self._closing_times[-1] - self._closing_times[0] < QUIT_TIMEOUT
):
    self._running = False
    return

self._sleeping = True
self._set_status_color(NEOPIXEL_SLEEP_COLOR)
self.sleep_check_delay = 0
self.backlight.power = False

def _wake(self):
    # Turn on the screen
    self.backlight.power = True
    self.sleep_check_delay = 0.1
    self._set_status_color(NEOPIXEL_READING_COLOR)
    self._sleeping = False

def _make_story_prompt(self, request):
    return self._prompt.format(
        STORY_WORD_LENGTH=STORY_WORD_LENGTH, STORY_REQUEST=request
    )

def _sendchat(self, prompt):
    response = ""
    print("Sending to chatGPT")
    print("Prompt: ", prompt)
    # Package up the text to send to ChatGPT
    stream = openai.chat.completions.create(
        model=CHATGPT_MODEL,
        messages=[
            {"role": "system", "content": SYSTEM_ROLE},
            {"role": "user", "content": prompt},
        ],
        stream=True,
    )

    for chunk in stream:
        if chunk.choices[0].delta.content is not None:
            response += chunk.choices[0].delta.content
        if self._sleep_request:
            return None

```

```

        # Send the heard text to ChatGPT and return the result
        return strip_fancy_quotes(response)

    @property
    def running(self):
        return self._running

    @property
    def sleeping(self):
        return self._sleeping

def parse_args():
    parser = argparse.ArgumentParser()
    # Book will only be rendered vertically for the sake of simplicity
    parser.add_argument(
        "--rotation",
        type=int,
        choices=[90, 270],
        dest="rotation",
        action="store",
        default=90,
        help="Rotate everything on the display by this amount",
    )
    return parser.parse_args()

def main(args):
    book = Book(args.rotation)
    try:
        book.start()
        while len(book.pages) == 0:
            if not book.sleeping:
                book.generate_new_story()
            book.display_current_page()

        while book.running:
            book.handle_events()
    except KeyboardInterrupt:
        pass
    finally:
        book.deinit()
        pygame.quit()

if __name__ == "__main__":
    main(parse_args())

```

Configuration Options

This project has a lot of configuration options to get it to run exactly as you would like. In order to get through them, they will be discussed in groups.

General Settings

- **STORY_WORD_LENGTH**: The approximate number of words used in the generated stories.
- **REED_SWITCH_PIN**: The pin that your reed switch is wired to in case your wiring varies from the wiring in this guide.
- **NEOPIXEL_PIN**: The pin your NeoPixels are wired to. There are very few PWM pins on the Pi, so this shouldn't change.
- **API_KEYS_FILE**: The location of your **keys.txt** file. By default, it points to your home directory.
- **PROMPT_FILE**: The location of your **bookprompt.txt** file. This was placed in the **/boot** folder for your convenience if you enabled the read-only file system.

Other Setting Groups

- **Quit Settings**: Gesture settings related to quitting the app
- **NeoPixel Settings**: NeoPixel parameters and Colors
- **Image Settings**: The filenames of the images
- **Asset Paths**: The file locations of the images and fonts. By default they are relative to the main script.
- **Font Path & Size**: The font files and font sizes for the Title and Text
- **Delay Settings**: Used to control the animations of the text
- **Whitespace Settings**: Settings to control the amount of whitespace around the text.
- **ChatGPT Parameters**: The Basic ChatGPT settings
- **Speech Recognition Parameters**: Parameters to control the voice input values

External Config Files

Next the script pulls in external configuration files including **keys.txt** and **bookprompt.txt**. The keys are read in as a config file, which is a built-in Python mechanism that makes it easy to parse and the prompt is read as a plain text file.

UI Element Classes

The Button and Textarea classes are intended to make handling the buttons and text areas easier. The buttons are the clickable elements and the text area is the area on the page that displays text. They mostly store all the relevant information related to their particular special purpose, though the button also handles displaying and checking if you clicked within its boundaries.

The Book Class

The book class is the main class that handles everything related to the book and handles the majority of the logic. Here are some of the notable functions:

- **Start:** Handles the stating and initialization of the code. This is what runs while it initially displays the welcome screen. It sets up Pygame. Then it loads the images, buttons, and fonts and starts some subthreads.
- **Handle Functions:** These run in the background and handle various tasks such as putting the book to sleep when the reed switch is closed, displaying the pulsing green NeoPixels while loading, and handling button presses.
- **Display Surface:** Handles displaying and rotating the image
- **Fade In Surface:** Handles the fade in animation of the title text
- **Display Current Page:** Displays the page of the story we are currently set to as well as the buttons.
- **Display Title Text:** Handles slicing up and centering the title text
- **Previous Page, Next Page, New Story:** Button handlers for navigation or making a new story.
- **Display Loading, Welcome, Message:** Display the appropriate special page
- **Load Story:** Parse the story output from ChatGPT into paragraphs and a title
- **Add Page:** Creates a new page in memory which includes a title and the height at which the text should display.
- **Generate New Story:** Prompts the user for a story. Display a message and use the listener class to listen and recognize the speech.
- **Sleep and Wake:** Put the book in sleep or wake modes. This turns off the backlight and coordinates with other threads to put the book into a sleep state or wake from it.
- **SendChat:** Formats the prompt into a data structure that is passed along to ChatGPT. Special characters that may be returned are stripped.

Parse Arguments

The app only takes a single argument, which is rotation. By default it is set to 90 degrees, but can also be set to 270 if the display is installed upside down.

Main

This is the entry point for the application and handles initializing the book and calling handle events in a loop while the book is running.

Listener Code

```
# SPDX-FileCopyrightText: 2023 Melissa LeBlanc-Williams for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import time

import speech_recognition as sr

class Listener:
    def __init__(
        self, api_key, energy_threshold=300, record_timeout=30
    ):
        self.listener_handle = None
        self.microphone = sr.Microphone()
        self.recognizer = sr.Recognizer()
        self.recognizer.energy_threshold = energy_threshold
        self.recognizer.dynamic_energy_threshold = False
        self.recognizer.pause_threshold = 1
        self.phrase_time = time.monotonic()
        with self.microphone as source:
            self.recognizer.adjust_for_ambient_noise(
                source
            ) # we only need to calibrate once, before we start listening
        self.record_timeout = record_timeout
        self._audio = None
        self.listener_handle = None
        self.api_key = api_key

    def listen(self, ready_callback=None):
        print("Start listening...")
        self._start_listening()
        if ready_callback:
            ready_callback()

        while (
            self.listener_handle and not self.speech_waiting()
        ):
            time.sleep(0.1)
        self.stop_listening()

    def _save_audio_callback(self, _, audio):
        print("Saving audio")
        self._audio = audio

    def _start_listening(self):
        if not self.listener_handle:
            self.listener_handle = self.recognizer.listen_in_background(
                self.microphone,
                self._save_audio_callback,
```

```

        phrase_time_limit=self.record_timeout,
    )

def stop_listening(self, wait_for_stop=False):
    if self.listener_handle:
        self.listener_handle(wait_for_stop=wait_for_stop)
        self.listener_handle = None
    print("Stop listening...")

def is_listening(self):
    return self.listener_handle is not None

def speech_waiting(self):
    return self._audio is not None

def recognize(self):
    if self._audio:
        # Transcribe the audio data to text using Whisper
        print("Recognizing...")
        attempts = 0
        while attempts < 3:
            try:
                result = self.recognizer.recognize_whisper_api(
                    self._audio, api_key=self.api_key
                )
                self._audio = None
                return result.strip()
            except sr.RequestError as e:
                print(f"Error: {e}")
                time.sleep(3)
            attempts += 1
            print("Retry attempt: ", attempts)
        print("Failed to recognize")
        return None
    return None

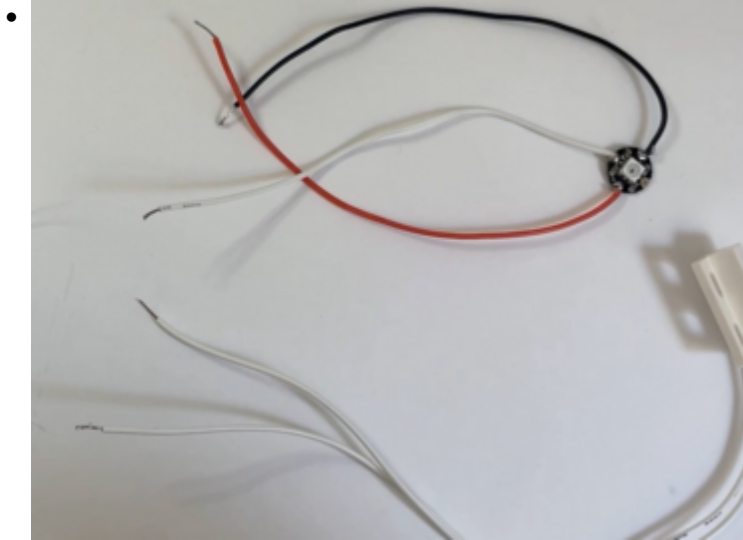
```

The Listener Class is fairly simple. It is mostly a wrapper for the SpeechRecognition library that listens in the background when the listening has started and is able to be stopped early in case the book needs to go to sleep.

The recognize function simply makes use of some recently added functionality in SpeechRecognition that interfaces with OpenAI's whisper API. If there is any communication issue, it will make up to 3 attempts before failing.

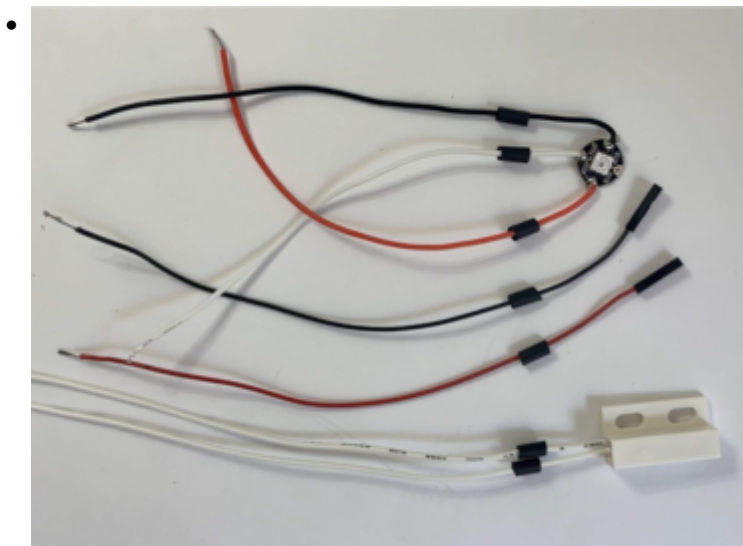
Electronics Assembly

Solder a power, ground,
and data IN wire to your



NeoPixel. Make the wires about 6-8" long.

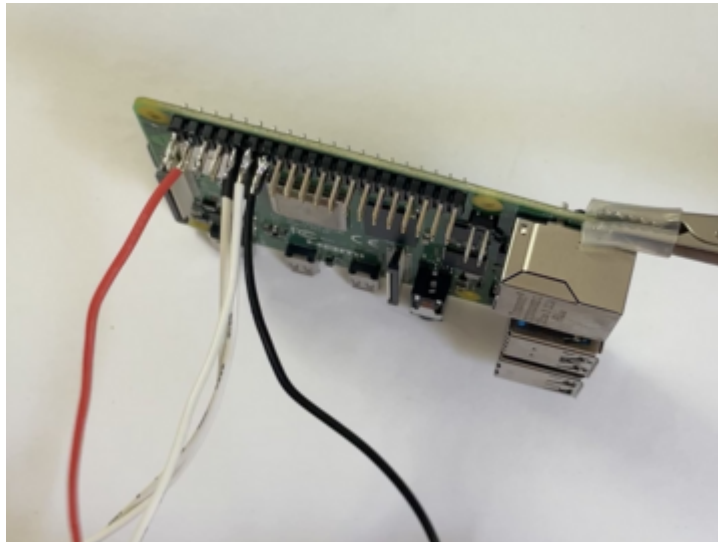
Trim the female jumper connectors off the end of your magnetic switch and off one end of the black and red jumper cables that came with the touch screen. They're a little too tall for this build so we'll solder these wires to the Pi instead.



Add a 1/4" piece of heat shrink to each of your wires.

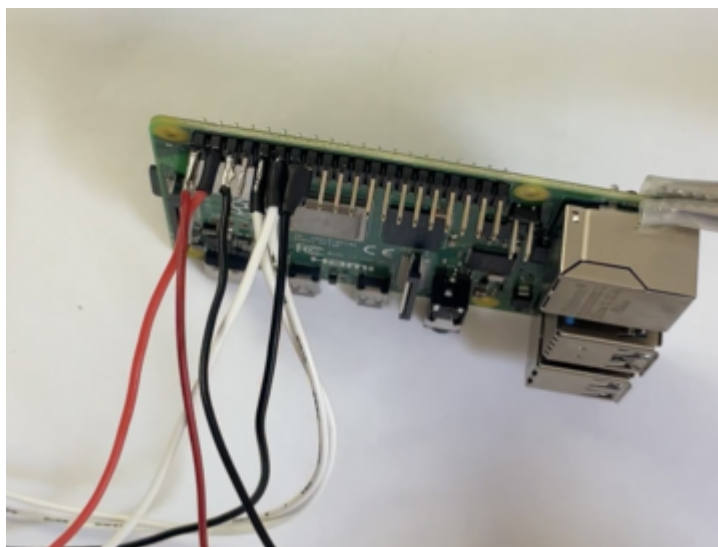


[Here's a great pinout diagram for the Raspberry Pi 4.](https://adafru.it/18Dk) (<https://adafru.it/18Dk>) Solder your switch wires to **G** and **GPIO 17**: the 5th and 6th pin on the inside row of pins.

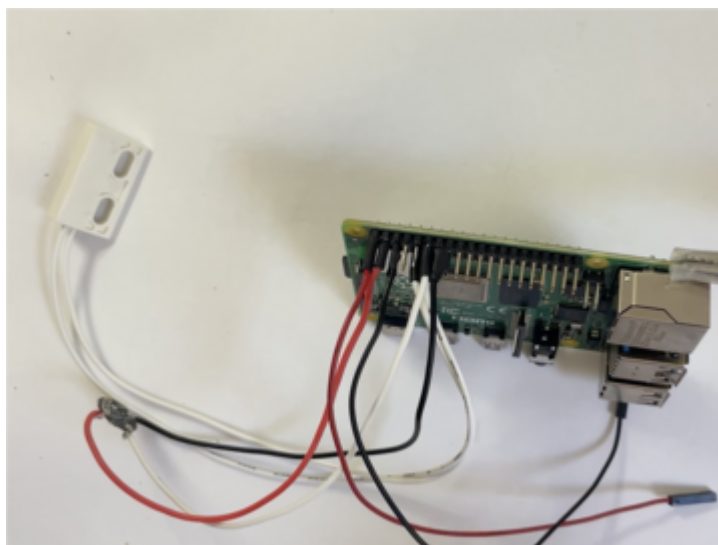


Solder your NeoPixel wires to the outside row of pins as shown:

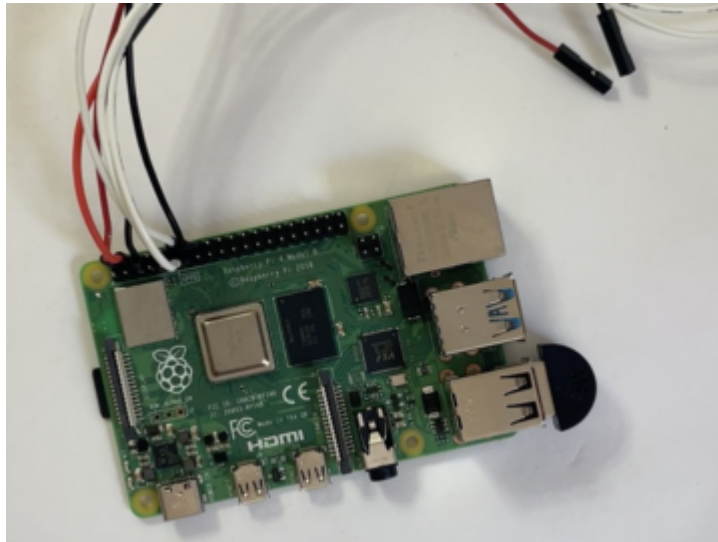
- **5v** (red) to **5v** power (2nd pin on the Pi)
- **Data IN** (white) to **GPIO 18** (6th pin on the Pi)
- **G** (black) to **G** (7th pin on the Pi)



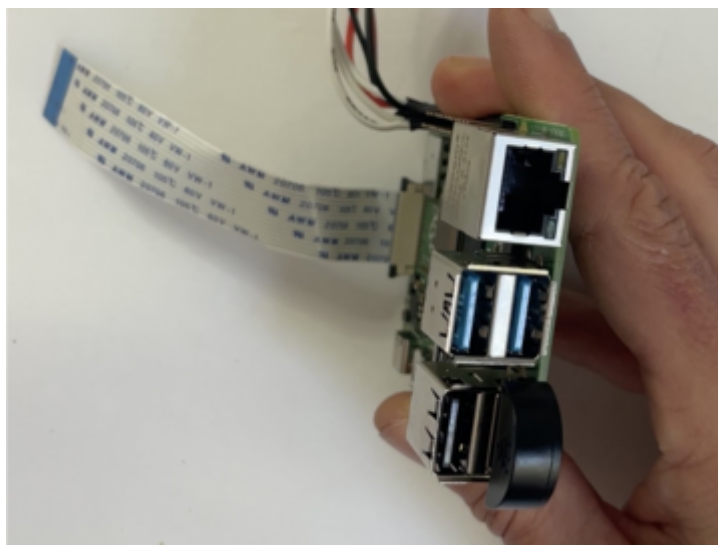
Solder the red and black jumper cables for the touch screen to 5v Power (red), the first pin on the Pi on the outside row, and G (black) to the 3rd pin on the Pi also on the outside row.



Cover each connection with heat shrink so there's no chance they'll short with each other.



Plug your USB microphone into one of the USB ports on the Pi.



Plug your USB microphone into one of the USB ports on the Pi. Plug the ribbon cable in as shown with the silver side facing in to the Pi.

This connector is tricky... pull upwards gently on the plastic corners to release and push them back toward the Pi to tighten.



Plug the other end of the cable into your touch screen as shown. Plug the jumper cables into 5v (red) and G (black).

Book Assembly

Give a little thought to the first and last pages that you cut. I left the portrait of Sir Thomas Moore intact, so that he can enjoy the stories along with the reader.



We will attach one page to the screen after cutting a window in it, so figure out which pages you want to keep and start cutting below that. I also didn't cut all the way to the back of the book - my book was thick enough that I could keep some pages.

I started by tracing the touch screen onto a sheet of paper, which I could then use as a template to line up my cutout. Give yourself a little extra room so you can get the electronics in and out easily.



To stabilize the pages, I built a box out of some wood scraps and screwed it all together around the pages I wanted to cut. I put a screw right through the center as well -- we're cutting that part out so it's okay to put a hole in it.

For drilling and cutting, use eye protection and consider work gloves. If you are younger, ask an adult for assistance. Care should be used and plenty of time to do it right.

-

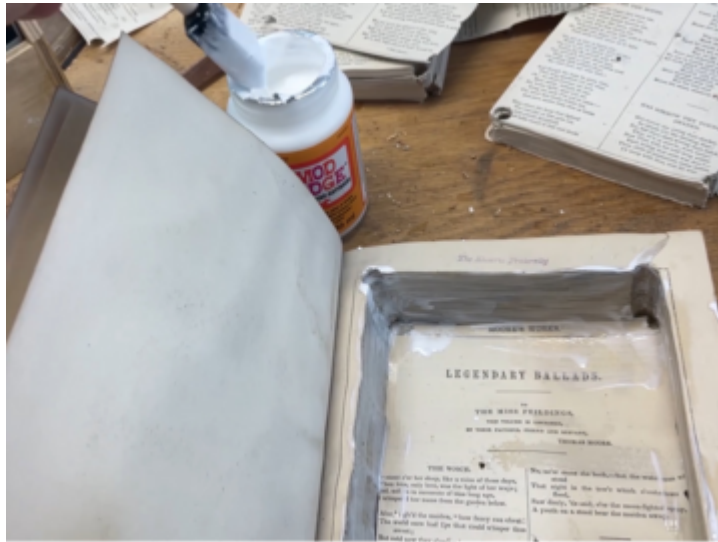


Next I drilled a hole right in the corner so I could get my jig saw blade through. The corners are the hardest part with the saw, and I wanted them rounded, so using the drill made this easier. I could just cut straight from hole to hole and the drill holes made the corners rounded for me.

-



Next I cut all around the edges with a jig saw. If you don't have access to a jig saw, it's perfectly possible to cut the pages out a few at a time with a sharp knife and a lot of patience. The inside of the box won't show in the finished project so it doesn't need to be perfect.



To secure the pages together and keep them from tearing, I coated the outside and inside of the pages with mod podge. This dried perfectly clear and kept the book looking just like a book, but made the pages into a solid block.



I used my dremel to hollow out a divot in the cover for my rare earth magnet, and glued it securely in place with E6000. Then I carved out a hollow for my NeoPixel and a hollow for the reed switch, which I made sure to line up with the magnet. The magnet will activate the switch when the book is closed and cause the project to sleep.



I used a sharp utility knife to carve a hole for the panel mount USB cable.



We'll use this for charging the battery. I cut another hole on the side of the book that's big enough to poke my finger in to activate the USB on/off switch.



Your layout may vary.



I glued some thick craft foam supports inside, and used them to securely wedge the on/off switch and the battery in place behind the holes.

I glued the NeoPixel and reed switch in place, making sure they were



flush with the top. I spent some time getting the foam inserts just right so there is room inside for all the cables and the touch screen sits as securely and flush as possible with the top of the cut pages.

Decoration



I cleaned the outside of the book and then rubbed it with a black wash by diluting some black acrylic paint in water until it was pretty thin and runny, then painting it all over the book. Before it dried, I wiped the whole book with a paper towel, removing the black from the higher sections but leaving it in the low points and corners. This accents all the fabulous detail on the book cover and really makes it pop.

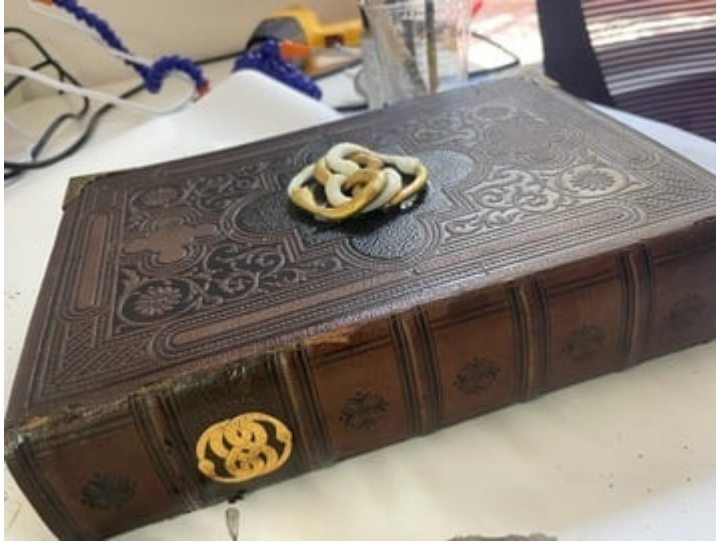




I covered up the old gold lettering on the front and the spine with a thicker layer of black acrylic paint.



I added some book corners I found on [Amazon](https://adafru.it/18Dl) (<https://adafru.it/18Dl>). I used some strong pliers and also some E6000 to secure the corners.



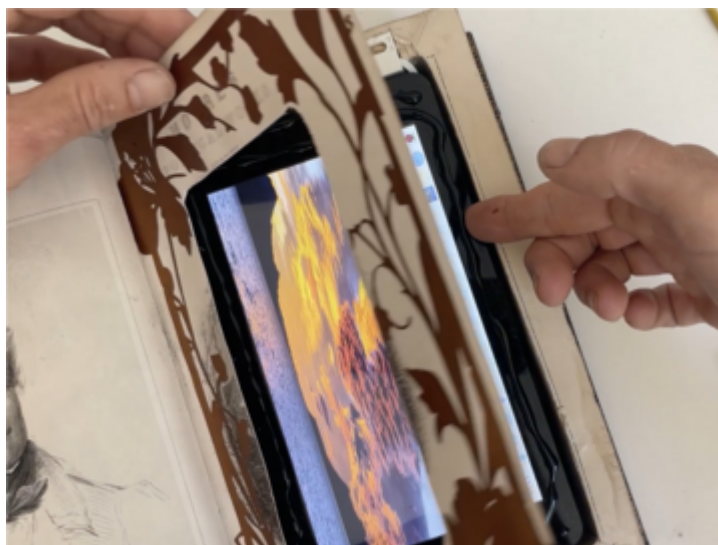
To finish off the cover, I glued on a [3D printed Auryn symbol](https://adafru.it/CRM) (<https://adafru.it/CRM>), from the Neverending Story movie. This feels so appropriate. I also cut a vinyl auryn sticker on my Cricut vinyl cutter and fixed it to the spine, using some extra glue to be sure it stays stuck.



I wanted to soften the transition between the book and the screen, so I decided to add a metallic leafy filigree sticker. This artwork was made in Midjourney and I converted it to a cuttable file in Photoshop, then uploaded it to the Cricut Design Space to cut it out on my vinyl cutter.

[Here's a link to my project](https://adafru.it/18Dm) (<https://adafru.it/18Dm>) in case you want to use this artwork with your Cricut. The project contains the leaf border, a small Auryn for the spine, and a template of the screen size cutout.

I traced out the screen dimensions using a paper template, then carefully cut through the page so the screen shows through.



I made sure everything lined up to my satisfaction, then glued this title page to the edge of the touch screen using E6000 glue. This paper is a little bit brittle, so take some time and be sure to get this right the first time if you can.

The screen is held in place by this page. I didn't secure it down inside the book, because I will probably need to access the USB ports or SD card at some point, so I didn't want to make it permanent. If you're

making this for a kid, it might be a good idea to glue this top page down to the book once you're sure you're happy with the setup.



To highlight the NeoPixel indicator light, I used a small ring from my scrapbooking drawer filled with UV resin. The resin cures in minutes when you shine a blacklight on it and does a great job of pulling the light up into a dome so it diffuses and refracts in a magical way.



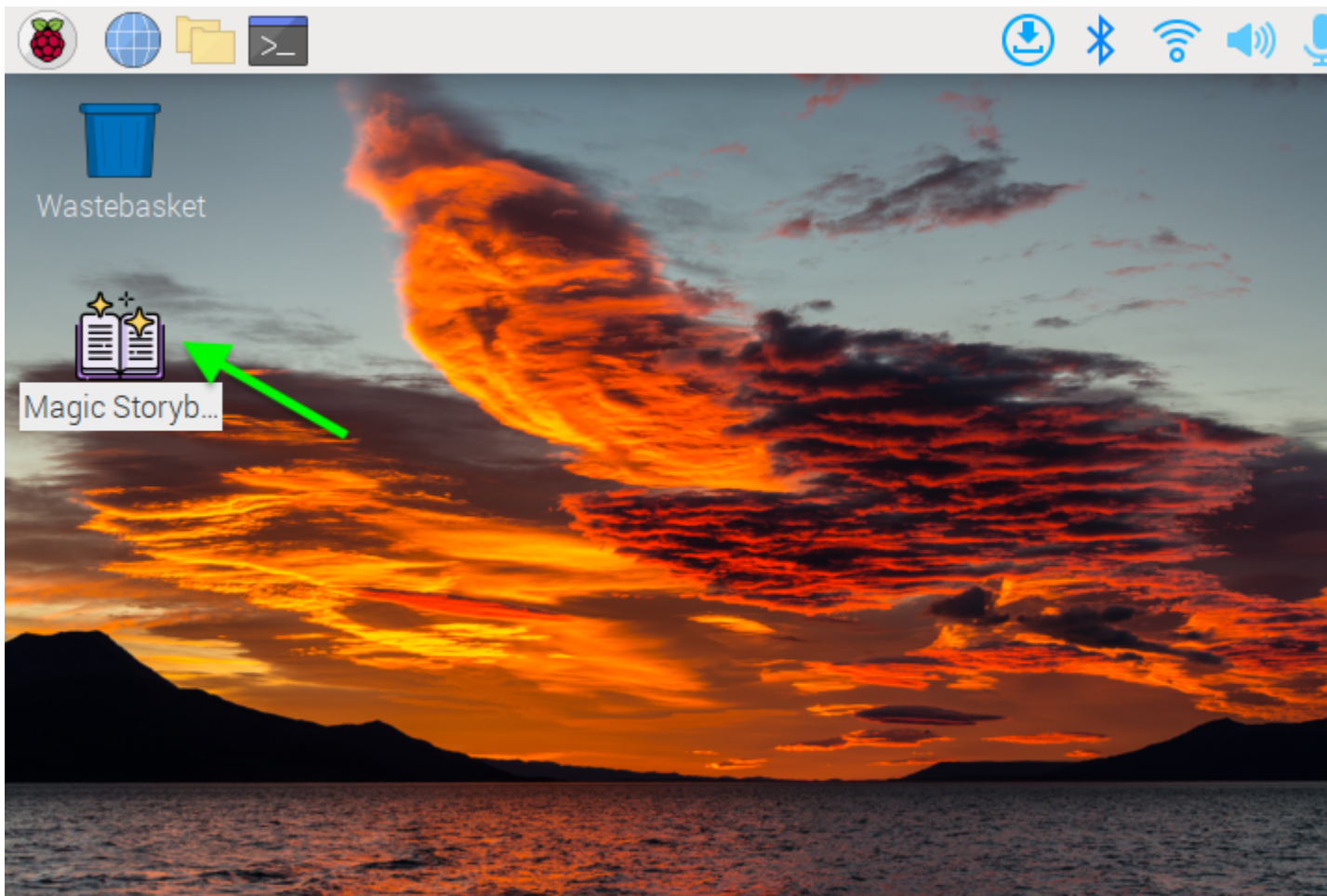


Usage

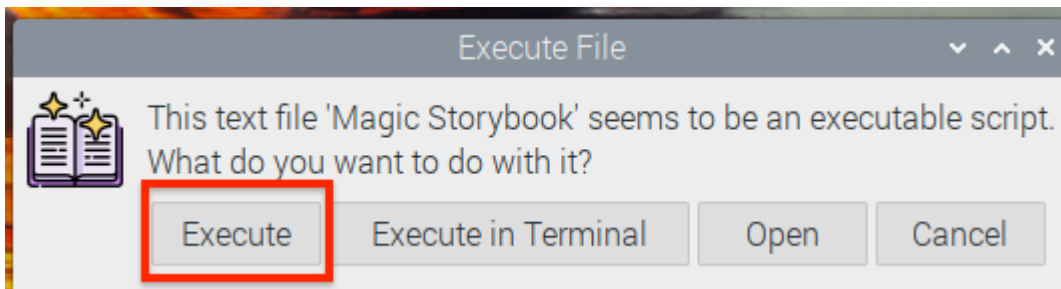
Running the Code

There are a few different ways to run the code. If you followed the setup instructions and created the icons, it should automatically run upon starting the Pi.

If you exited the code and you would like to run the code again, you can do so simply by double-tapping the icon on the desktop.



If it asks you what you want to do with the file, just choose **execute**.



Another option is to use **ssh** to connect remotely to your Raspberry Pi from your desktop computer. You can then type commands into the Pi without having to disassemble your book and plug in a keyboard, or remove the SD card in order to enter commands. This is also a great way to troubleshoot your project: when you're running the script from an ssh terminal you can see any error codes to help figure out why your project isn't working.

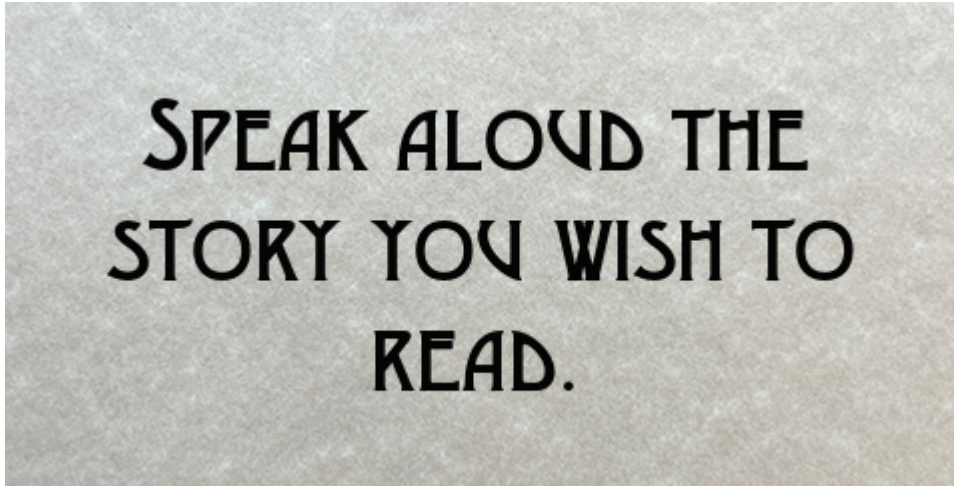
[Adafruit's Raspberry Pi Lesson 6. Using SSH \(https://adafru.it/jsE\)](https://adafru.it/jsE)

From either your ssh window or from a keyboard plugged into your Pi, you can run the code manually from the command line / terminal window using the following commands:

```
source ~/story/bin/activate  
cd ~/Magic_AI_Storybook  
sudo -E PATH=$PATH python story.py
```

Creating a New Story

When you first start the script, you will be prompted about what kind of story you would like.



Wait for the NeoPixels to turn yellow and speak into the microphone using a loud clear voice. If it doesn't hear you correctly, it may generate a story different than requested. Once it has detected speech, it will display a Dreaming screen (created with Midjourney AI) while it generates the story:



Once the story has been generated, it will display the story for you to read:

THE LOST TREASURE OF EL DORADO

In the heart of the Amazon rainforest lies a mystery that has intrigued adventurers for centuries. A legend speaks of a hidden treasure, known as the Lost Treasure of El Dorado. Many have attempted to find it, but none have succeeded. That is until now.

A young adventurer named Maya had always dreamed of uncovering the treasure. She spent years studying ancient maps, deciphering hidden codes, and mastering the art of survival in the jungle. Finally, she believed she had cracked the code and found the location of the treasure.

With a small team, Maya embarked on a perilous journey deep into the Amazon.

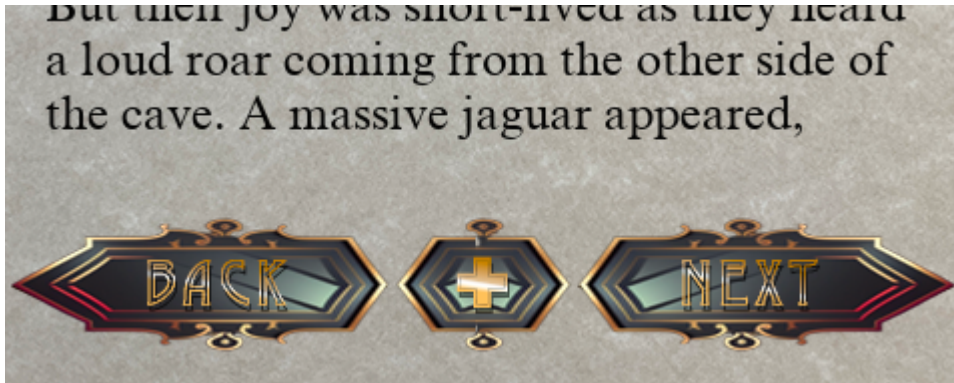


Navigation

At the bottom of each page, there will be two or three buttons. **Back** and **Next** are used to navigate between pages and stories. If you are at the end of the last story, pressing the Next button will generate a new story.

If there are multiple stories, they will be displayed in the order they were generated and pressing Back at the beginning of a story will load the previous one and pressing Next at the end of a story which has another story after it will load that story.

Pressing the **New Story** button, which looks like a plus sign, at any time will prompt you for a new story and then display that story.



Putting the Book in a Sleep State

Closing the reed switch by closing the cover should put the book in a sleep state. If it is in the middle of a more complex task such as creating a new story, it attempts to stop those tasks and then places the book in a sleep state.

Exiting the Script

To exit the script, you can close the book three times within five seconds. If you have changed the Quit settings from the defaults, the behavior could be a bit different.

Changing the Prompt: bookprompt.txt

To change the ChatGPT prompt, you will want edit the **/boot/bookprompt.txt** file. If you have changed the file system to read-only, you will need to shut down the Pi, remove the SD Card, place it in a computer and you can just edit the file from the boot partition. Place it back into the Pi when finished and start it back up again.

The file is at **/boot/bookprompt.txt**. By default, it reads:

Write a complete story. It must begin with a title and have a body of approximately {STORY_WORD_LENGTH} words long and a happy ending. The specific story request is "{STORY_REQUEST}". The title should each be specified and the body should start on the next line.

To change the prompt, ssh into your Pi or plug in your keyboard and open a terminal window. Type these commands:

```
cd /boot
sudo nano bookprompt.txt
```

This will open **bookprompt.txt** in the nano text editor.

This is a powerful bit of code that lets you set whatever parameters you'd like for your stories. Example text you could add:

```
"Make every story about a girl named Sarah and her teddy bear Sugarbear"
"Write in the style of Neil Gaiman"
"The story should be in rhyming couplets and iambic pentameter"
"The stories should be magical and strange, with wild animal spirit magic
based on ancient Norse and Finnish mythology and should end with a
confusing moral"
"Every story should contain a pun"
```

The possibilities are endless. What kind of stories will your book imagine?

The prompts in **bookprompt.txt** will affect every story you ask for, so if you want just one rhyming story, use the voice control feature to ask for it. If you want EVERY story to rhyme, change it here in **bookprompt.txt**.

The variables `STORY_WORD_LENGTH` and `STORY_REQUEST` can be found in **story.py**. By default the stories are 800 words long. Change `STORY_WORD_LENGTH` in **story.py** to edit this.

Troubleshooting

This is a complicated project with a lot of steps. Here are a few things to try if you're running into trouble.

Story.py Doesn't Run

1. Use ssh to connect to your Pi using your computer. This is a powerful little tool that will allow you to control your Pi without having to plug in a keyboard and monitor. Here is a full tutorial that will help you get started:

[Adafruit's Raspberry Pi Lesson 6. Using SSH](https://adafru.it/jsE) (<https://adafru.it/jsE>)

Once you can ssh into your Pi, type `sudo -E PATH=$PATH ~/story/bin/python ~/Magic_AI_Storybook/story.py` into your terminal window to start the story script. The terminal window will give you a verbose output about what's going on with your project, which is VERY helpful in the debugging process.

2. Start Over. Head back to the Software Install page and run through every step again. It's easy to miss a command, or not to notice if you've mistyped something. This is where ssh comes in handy again: you can copy/paste the commands easily from your computer's web browser into your terminal window and that lessens the chance of a typo.

The Pi Keeps Rebooting / Crashing

Is your battery sufficiently charged? And is it plugged in to the charger? Try unplugging it. I had trouble with the Pi rebooting over and over when I started it up while the charging cord was plugged in. Be sure the battery is charged and unplug before you start.

If the **story.py** script keeps quitting on you, perhaps you missed our little Easter Egg: if you open and close the book, activating the magnetic reed switch 3x within 5 seconds, then **story.py** will exit so you can get to the Pi desktop. This is a feature, not a bug! Relaunch the script by double-clicking the desktop icon.

The App is Still Running in Fullscreen, but not Responding

If you set up the Pi with the Read-Only File system, you can simply restart it. You can also plug a keyboard in and hit Alt+F4. After about 5 seconds, a popup should appear asking if you want to **Wait** or **Force Quit**. Choose **Force Quit**.

The Story pages through by itself

This can happen if you have something obscuring or stuck to part of the touch screen. I initially stuck a pretty vinyl sticker on the edges of the screen to cover the paper transition, but the vinyl activated the capacitive touch response on the screen and made it act like I was touching it. Be sure your screen is completely clear of glue or stickers or anything that will make it react.

It Doesn't Listen / Can't Seem to Hear You

Try changing the sensitivity of the microphone by editing the `ENERGY_THRESHOLD` variable in **story.py**.

If that doesn't help, try running this script. Save it as **microphone_recognition.py** in the same folder as your **listener.py** file. You'll have to paste your API key into it.

```
from listener import Listener

OPENAI_API_KEY = ""

listener = Listener(OPENAI_API_KEY)

print("Say Something")
listener.listen()

if listener.speech_waiting():
    print(f"Whisper API thinks you said {listener.recognize()}")
else:
    print("No speech detected")
```

Authentication Errors

If you get stuck on the "Dreaming" page, you may have an authentication error. Did you remember to add your Open API key to **keys.txt**? Try editing **keys.txt** and make sure your key is in there and looks correct.

```
cd ~  
sudo nano keys.txt
```

If it still doesn't work, try going back to Open API and generating a new key, and use that one.