Generating Text with ChatGPT, Pico W & CircuitPython

Created by Jeff Epler

You can manipulate gravity to control the trajectory and speed of objects, making you a valuable addition to any sports team.

https://learn.adafruit.com/generating-text-with-chatgpt-pico-w-circuitpython

Last updated on 2024-03-15 05:24:27 PM EDT

Table of Contents

Overview

"As an AI language model, I am thrilled to be on the Raspberry Pi Pico W! This small yet powerful device enables seamless integration with a wide range of applications and systems, making it an ideal platform for AI and machine learning projects. The Raspberry Pi Pico W's versatility, simplicity and affordability make it a game-changer in the world of technology!" -- ChatGPT

Quote blocks like the one above and photos in this guide typically show text generated by ChatGPT.

In this guide, you will learn how to use OpenAI's ChatGPT (https://adafru.it/18BV) API to generate text from a prompt using CircuitPython on the Raspberry Pi Pico W.

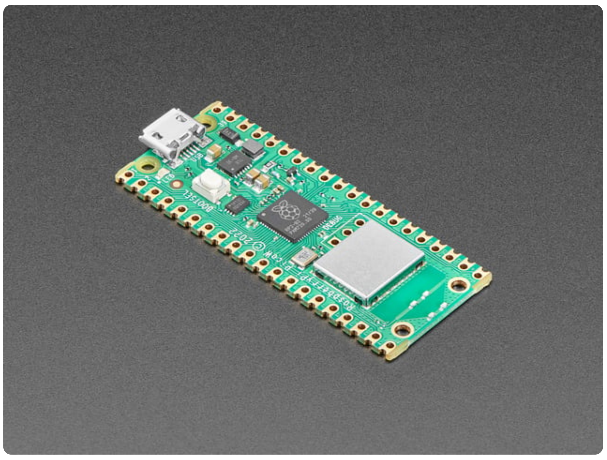
At startup, or when the arcade button is pressed, a new, original snippet of text will be generated on OpenAI's servers and shown on the OLED screen connected to your Pico W. Because of the random factor in the text ChatGPT generates, it's unlikely that two responses would ever be the same.

Since ChatGPT generates plausible text rather than making **true statements**, use this project only for situations where the truth is unimportant. For example, by default the request to ChatGPT asks for a description of an "unconventional but useful superpower".

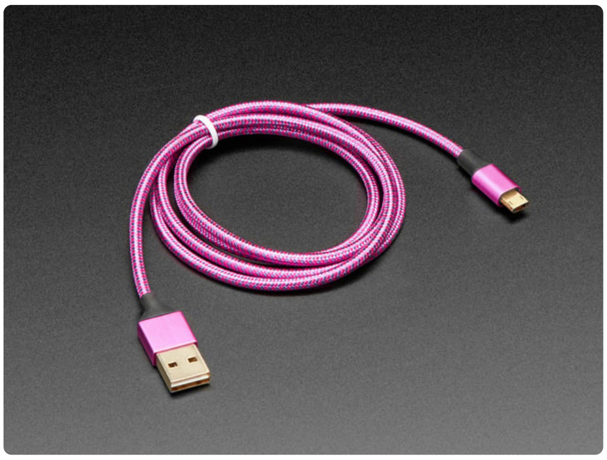
It's easy to customize the prompt using any text editor. This guide has some tips for creating prompts of your own. This works by writing a sentence or two in natural human language describing what you'd like ChatGPT to generate; no complicated coding is needed to get a description of an imaginary plant instead, or even to generate text in French instead of English!

The code in this guide does use a paid API at OpenAI, but based on the pricing in March 2023, the cost to access the API for this project is measured in fractions of a cent, not in dollars. During the whole development process of this guide, the author's costs on OpenAI were less than $0.25.

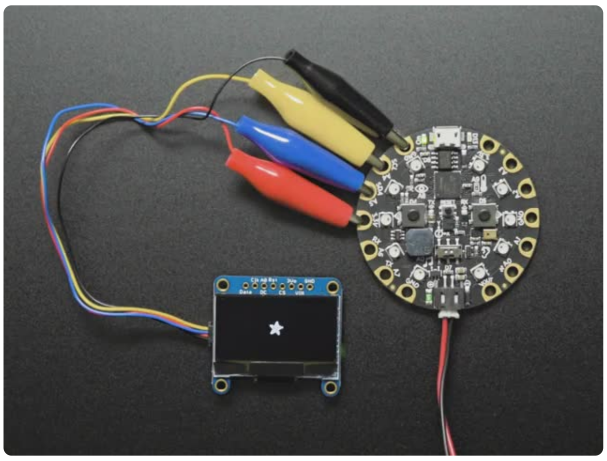
Parts



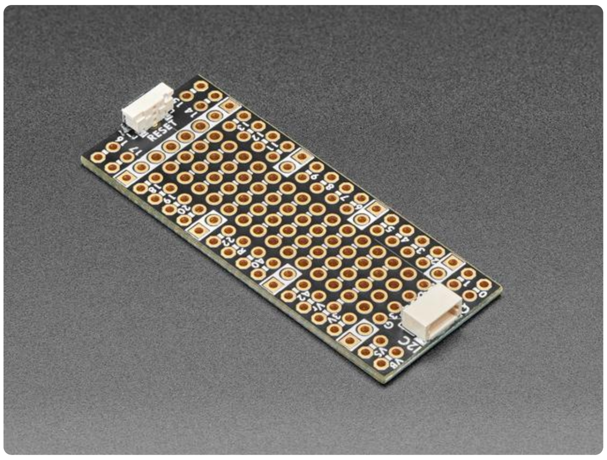
Raspberry Pi Pico W
The Raspberry Pi foundation changed single-board computing when they released the Raspberry Pi computer, now they're ready to...
https://www.adafruit.com/product/5526



Fully Reversible Pink/Purple USB A to micro B Cable - 1m long
This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also fully reversible! That's right, you will save seconds a day by...
https://www.adafruit.com/product/4111



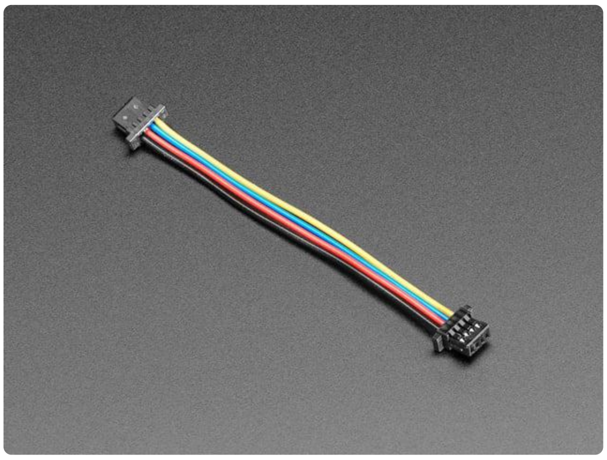
Monochrome 1.3" 128x64 OLED graphic display - STEMMA QT / Qwiic
These displays are small, only about 1.3" diagonal, but very readable due to the high contrast of an OLED display. This display is made of 128x64 individual white OLED pixels,...
https://www.adafruit.com/product/938



Adafruit PiCowbell Proto for Pico - Reset Button & STEMMA QT

Ding dong! Hear that? It's the PiCowbell ringing, letting you know that the new Adafruit PiCowbell Proto is finally in stock and ready to assist your

https://www.adafruit.com/product/5200



STEMMA QT / Qwiic JST SH 4-Pin Cable - 50mm Long

This 4-wire cable is 50mm / 1.9" long and fitted with JST SH female 4-pin connectors on both ends. Compared with the chunkier JST PH these are 1mm pitch instead of 2mm, but...

https://www.adafruit.com/product/4399

1 x Socket Headers for Raspberry Pi Pico
2 x 20 pin Female Headers
https://www.adafruit.com/product/5583

1 x 0.1" male header
Break-away 0.1" 36-pin strip male header
https://www.adafruit.com/product/392

1 x M2.5 Screws & Stand-offs
Black Nylon Machine Screw and Stand-off Set – M2.5 Thread
https://www.adafruit.com/product/3299

1 x Arcade Button w/LED
30mm arcade button, various colors
https://www.adafruit.com/product/3490

1 x Quick-Connect Wires
Arcade Button Quick-Connect Wire Pairs - 0.11" (10 pack)
https://www.adafruit.com/product/1152

1 x Right-angle headers
Break-away 0.1" 36-pin strip right-angle male header
https://www.adafruit.com/product/1540

1 x M2 Stand-offs
300pcs M2 Brass Standoff Kit
https://www.amazon.com/300pcs-Standoff-Column-Spacer-Assortment/dp/B07B9X1KY6/

A $2.00 budget suffices for multiple hours of play.

1 x OpenAI Account & API Key https://platform.openai.com/

A $2.00 budget suffices for multiple hours of play.

Installing CircuitPython

CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

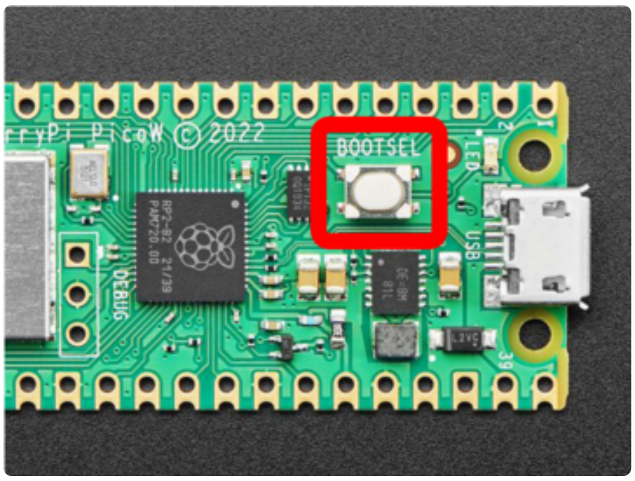
Follow this step-by-step to quickly get CircuitPython working on your board.

Download the latest version of CircuitPython for the Raspberry Pi Pico W from circuitpython.org

https://adafru.it/11xd

Click the link above and download the latest UF2 file.

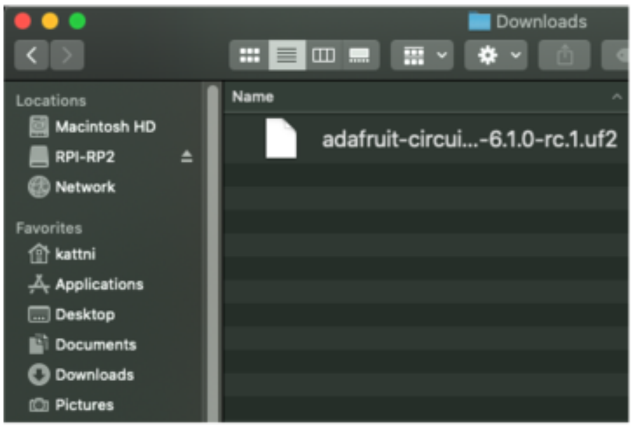
Download and save it to your desktop (or wherever is handy).



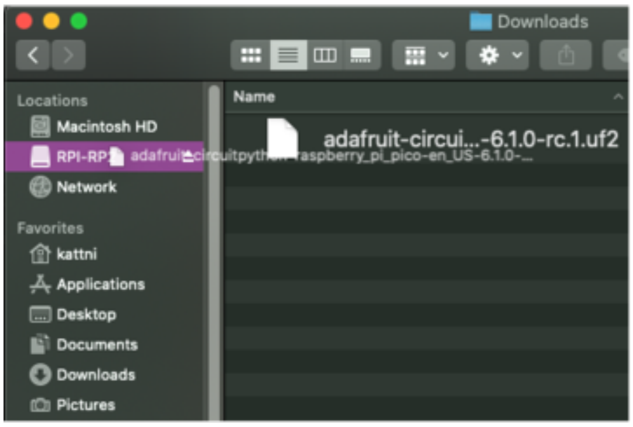
Start with your Pico W unplugged from USB. Hold down the **BOOTSEL** button, and while continuing to hold it (don't let go!), plug the Pico W into USB. **Continue to hold the BOOTSEL button until the RPI-RP2 drive appears!**

If the drive does not appear, unplug your Pico W and go through the above process again.

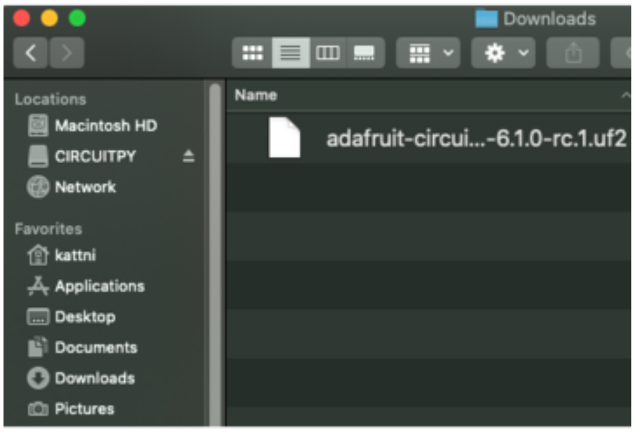
A lot of people end up using charge-only USB cables and it is very frustrating! **So make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **RPI-RP2**.



Drag the **adafruit_circuitpython_etc.uf2** file to **RPI-RP2.**



The **RPI-RP2** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it, you're done! :)

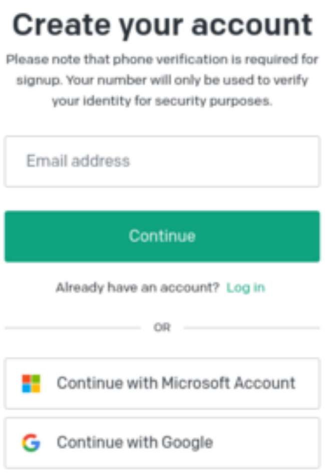
Flash Resetting UF2

If your Pico W ever gets into a really weird state and doesn't even show up as a disk drive when installing CircuitPython, try installing this 'nuke' UF2 which will do a 'deep clean' on your Flash Memory. You will lose all the files on the board, but at least you'll be able to revive it! After nuking, re-install CircuitPython

flash_nuke.uf2
https://adafru.it/QAJ

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.



Create your account

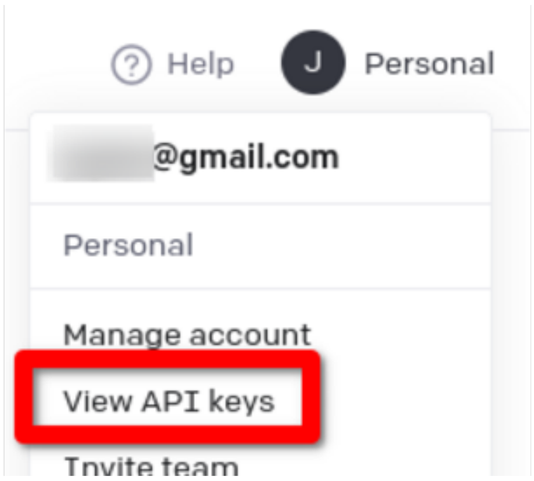
Please note that phone verification is required for signup. Your number will only be used to verify your identity for security purposes.

Email address
Continue
Already have an account? Log in
OR
Continue with Microsoft Account
Continue with Google

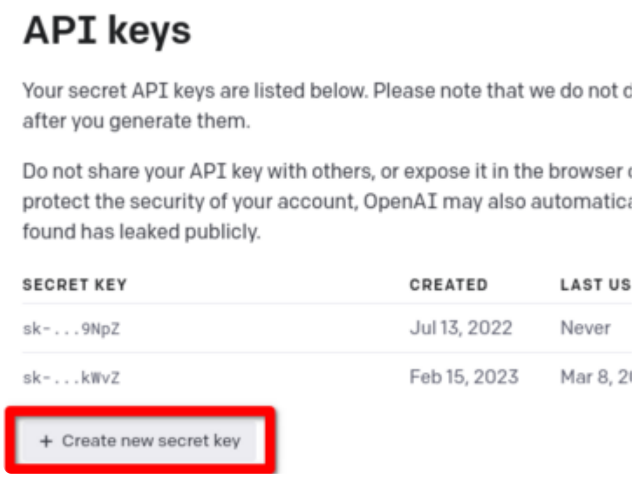
In your web browser, visit https://platform.openai.com/ (https://adafru.it/18Am)

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".



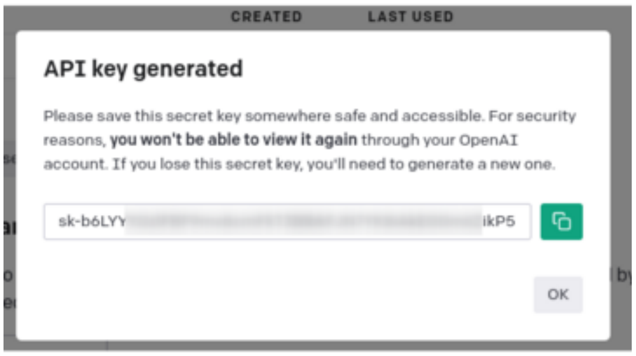
API keys

Your secret API keys are listed below. Please note that we do not d after you generate them.

Do not share your API key with others, or expose it in the browser o protect the security of your account, OpenAI may also automatica found has leaked publicly.

SECRET KEY	CREATED	LAST US
sk-...9NpZ	Jul 13, 2022	Never
sk-...kWvZ	Feb 15, 2023	Mar 8, 20

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



API key generated

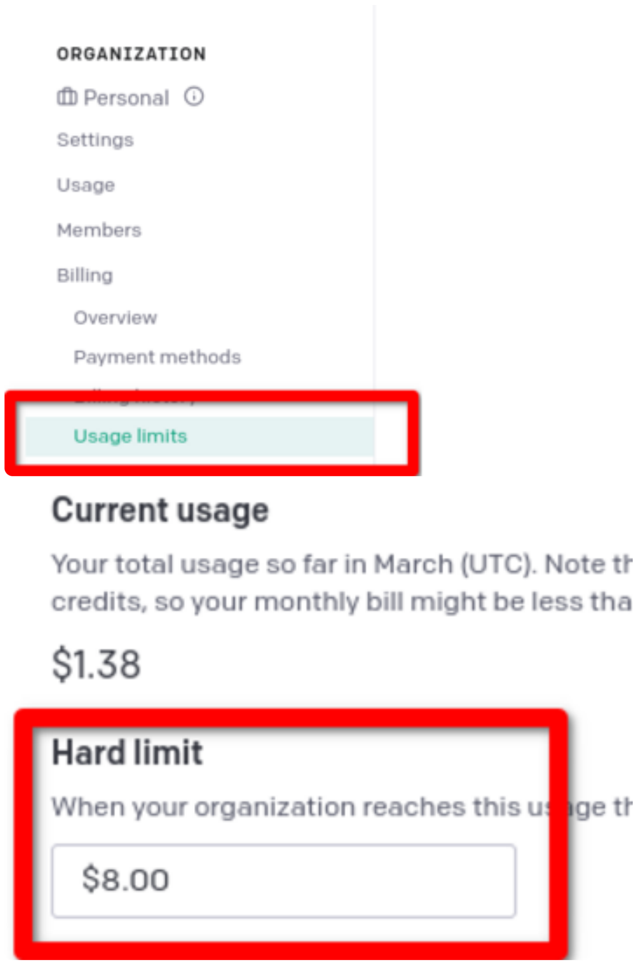
Please save this secret key somewhere safe and accessible. For security reasons, **you won't be able to view it again** through your OpenAI account. If you lose this secret key, you'll need to generate a new one.

sk-b6LYY[redacted]ikP5

Save this secret key in the file **settings.toml** on the **CIRCUITPY** drive in a line that looks like

```
OPENAI_API_KEY="sk-b6...kP5"
```

This file also requires your WiFI credentials, see the next page of the guide for the details.



Current usage

Your total usage so far in March (UTC). Note th
credits, so your monthly bill might be less tha

$1.38

Hard limit

When your organization reaches this usage th

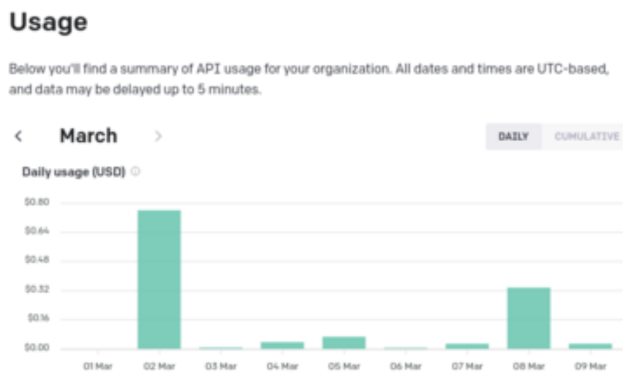
$8.00

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.

For projects that use more advanced features of the API, like vision, you'll need to enter a credit card so that your account is not on the free tier despite the free credit.



Usage

Below you'll find a summary of API usage for your organization. All dates and times are UTC-based, and data may be delayed up to 5 minutes.

This graph shows the author's usage costs while developing and playtesting an app, a total of $1.27 in API calls.

Configuring the settings.toml File

This project depends on you adding your WiFi settings and OpenAI API key in order to generate the text adventure.

Plug your CircuitPython board into your computer via a known good data + power USB cable. Your board should show up as a thumb drive in your File Explorer / Finder (depending on your operating system) named **CIRCUITPY**.

Create a file with the name **settings.toml** in the root directory of the **CIRCUITPY** drive.

Edit it to contain the keys `WIFI_SSID`, `WIFI_PASSWORD`, and `OPENAI_API_KEY`. (It's also OK for it to contain other keys)

Your file should look similar to the one shown below:

```
OPENAI_API_KEY="sk-b6...kP5"
WIFI_SSID="GuestAP"
WIFI_PASSWORD="i trust u"
```

3D Printing

> It's like watching a delicious pizza come out of the oven, but instead of pizza, it's a perfectly printed part. -- ChatGPT

This project uses a variant of the case from the guide Pico W HTTP Server with CircuitPython (https://adafru.it/18BX). The case body was remixed using the free and open-source OpenSCAD (https://adafru.it/XD4) (runs on Windows, Mac & Linux) to make it deeper and add a hole for the arcade button. Use the face plate from the original project, but grab the printable STL for the new case body using the link below, or grab the OpenSCAD source if you want to customize it further. Follow the printing instructions from the HTTP Server guide.

enlarged_picow.stl (printable case body)

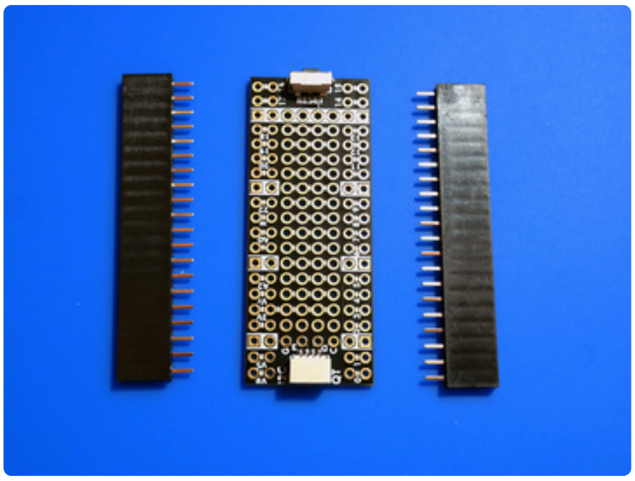
https://adafru.it/18BZ

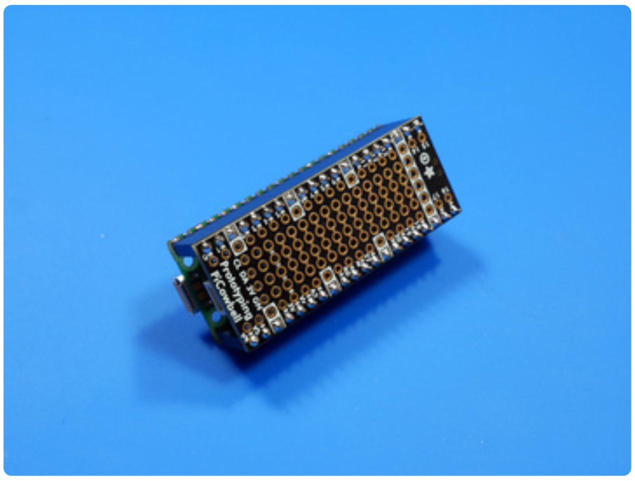
```
module base() { import("picowServerCase_mainCase_v1.stl", convexity=4); }
module clip_removed() {
    difference() {
        base();

        translate([-69,11,-40])
        cube(25);
    }
}

module z_slice(z0, dz, sc, extent=300) {
    render(convexity=4) intersection() {
        translate([0,0,-z0])
        children();

        translate([-extent/2, -extent/2, 0])
        cube([extent, extent, dz]);
    }
}

module elongated() {
    color("red")
    z_slice(-32, 4)
    clip_removed();

    color("green")
    translate([0,0,4])
    scale([1,1,8])
    z_slice(-28, 1)
    clip_removed();

    translate([0,0,12])
    z_slice(-27, 30)
    clip_removed();
}

module arcaded() {
    difference() {
        elongated();

        translate([-45, 53/2, 18])
        rotate([0,90,0])
        cylinder(d=29.5, h=10);
    }
}

arcaded();
```

Assembly

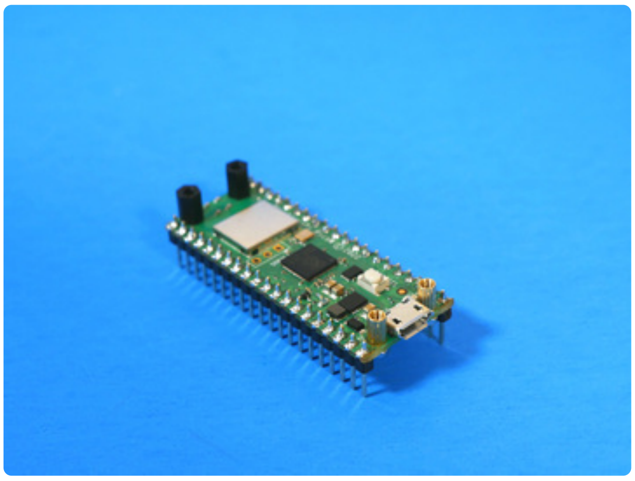
> As an AI language model, I don't have personal experiences to share, but I can generate a sentence for you: "Once, I accidentally soldered a piece of spaghetti onto my circuit board and it surprisingly still worked." -- ChatGPT

Some assembly photos are from the guide "Pico W HTTP Server with CircuitPython", so the parts shown don't match the rest of the project.

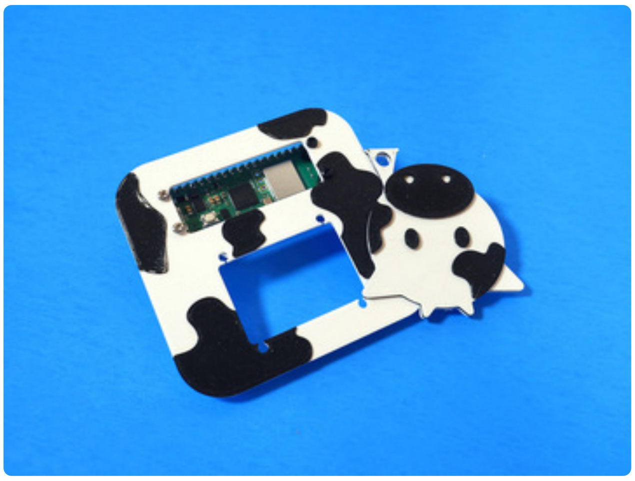


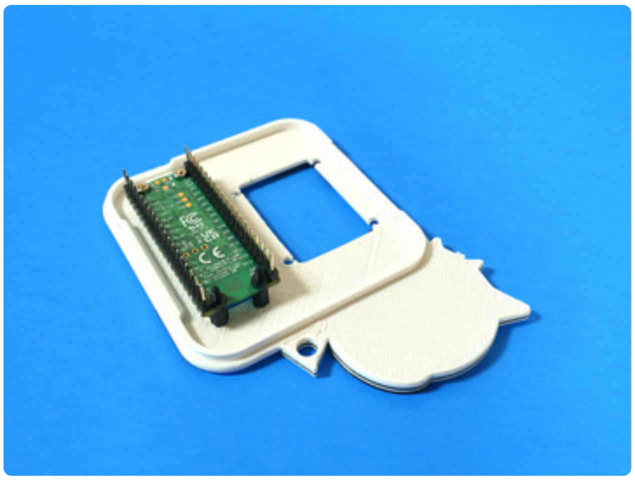


Solder socket headers to the PiCowbell. You can use the Pico W as a jig to keep the headers secure.

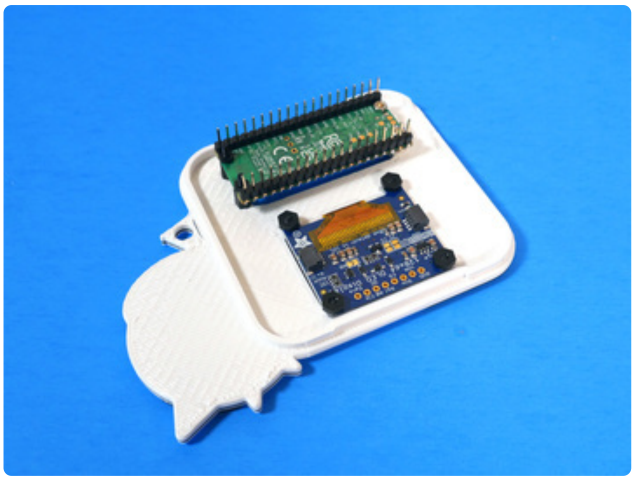
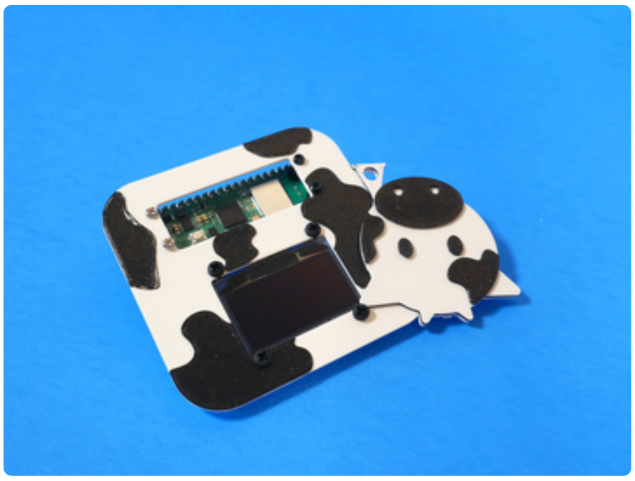


Attach four M2 standoffs to the Pico W's four mounting holes with M2 nuts.

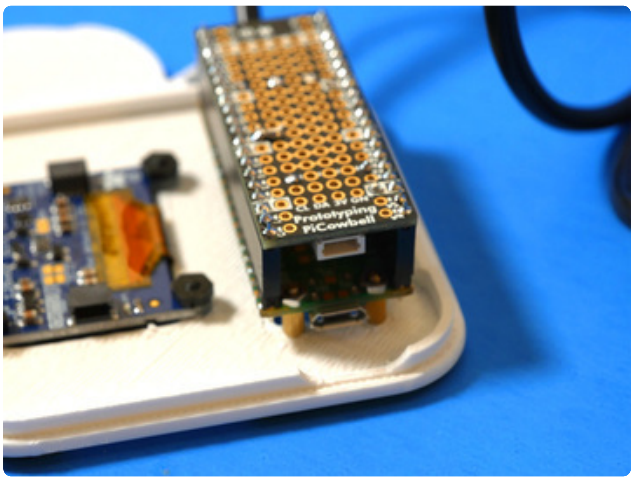




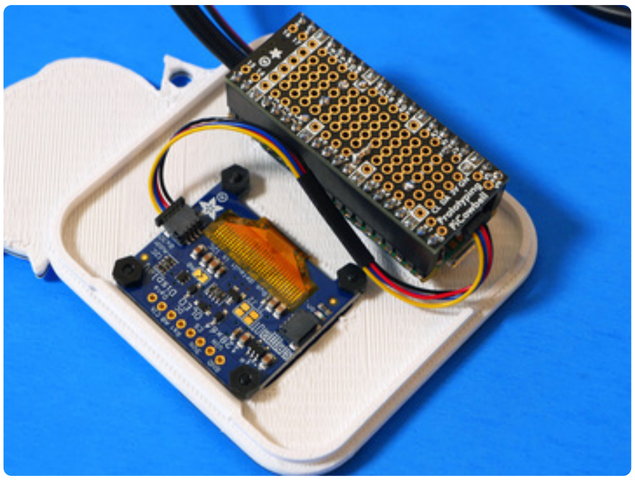
Attach the Pico W to the case lid with four M2 screws.

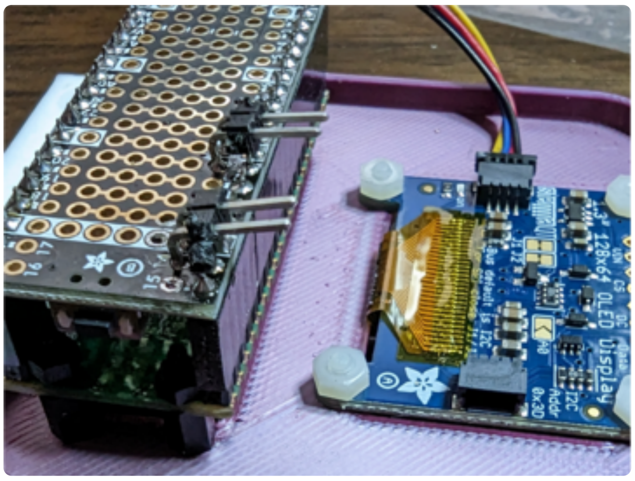
Attach the OLED screen with four M2.5 screws and nuts.



Plug the PiCowbell into the Pico W. The STEMMA port on the PiCowbell should be below the USB port on the Pico W.

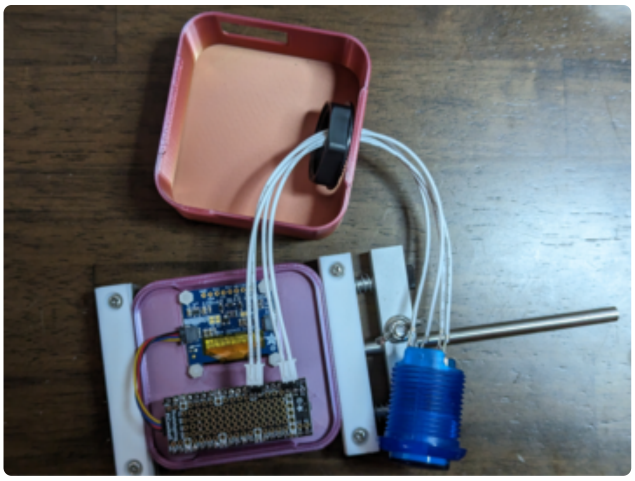


Connect the OLED to the STEMMA port on the PiCowbell with a STEMMA QT cable.



Solder two 2-pin sections of 90 degree header:

Pin 14 and the GND pin next to it
Pin 10 and the GND pin next to it
Point the long side of the header outwards.



Take two of the quick-connect wire harnesses. Press the ends gently but firmly onto the arcade button. Then pass the wire harnesses through the hole in the case, through the threaded nut, and attach them to the right angle headers.

Plug the button in on Pin 14 and the LED on Pin 10. Check the polarity of the LED, so that side marked "-" on the Arcade Button is connected to GND.

At this point, you can load the code and make sure that the button and LED work. If the LED is not working, try reversing the connection by simply plugging the connector in the other way. If nothing's working, try swapping the two connectors.

As the spell went awry, the professor's hat transformed into a teapot and poured steaming hot tea on all the nearby broomsticks, causing them to sprout rainbow colored wings and take flight into the night sky.

Now, secure the arcade button in place with the nut, then carefully curl the wires around the interior of the box until you can snap the lid into place.

Coding the Text Generator

> Learning to program is like having a superpower that enables you to create something out of nothing with just a few lines of code. -- ChatGPT

Text Editor

Adafruit recommends using the **Mu** editor for editing your CircuitPython code. You can get more info in this guide (https://adafru.it/ANO).

Alternatively, you can use any text editor that saves simple text files.

Download the Project Bundle

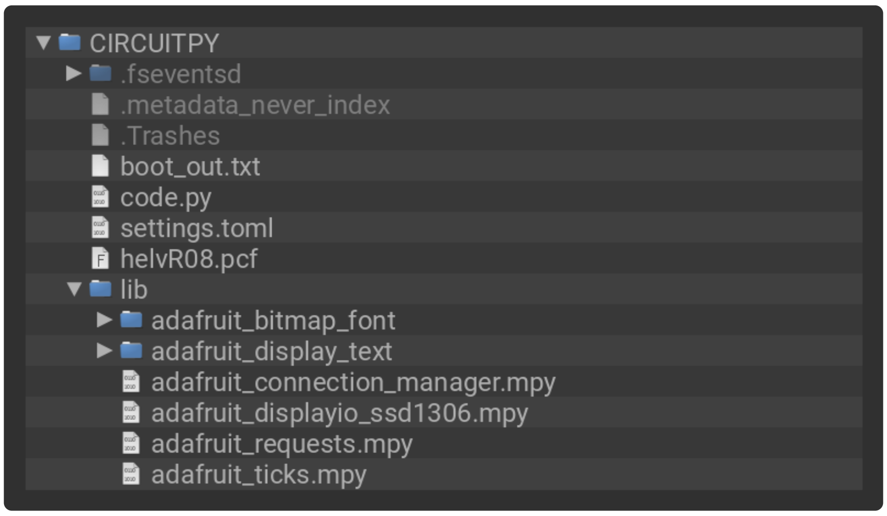
Your project will use a specific set of CircuitPython libraries and the **code.py** file. To get everything you need, click on the **Download Project Bundle** link below, and uncompress the .zip file.

Hook your Pico W to your computer via a known good USB data+power cable. It should show up as a thumb drive named **CIRCUITPY**.

Using File Explorer/Finder (depending on your Operating System), drag the contents of the uncompressed bundle directory onto your board's **CIRCUITPY** drive, replacing any existing files or directories with the same names, and adding any new ones that are necessary.

Once the code restarts, it will connect to WiFi and start using OpenAI to generate text according to the default prompt, "Write 1 sentence starting "you can" about an unconventional but useful superpower". The code uses OpenAI's "streaming" mode, so the response appears by chunks, also known as tokens.

Head on to the next pages for advice on how to modify it with your own original prompts as well as explanation of key parts of the code.


```
CIRCUITPY
  .fseventsd
  .metadata_never_index
  .Trashes
  boot_out.txt
  code.py
  settings.toml
  helvR08.pcf
  lib
    adafruit_bitmap_font
    adafruit_display_text
    adafruit_connection_manager.mpy
    adafruit_displayio_ssd1306.mpy
    adafruit_requests.mpy
    adafruit_ticks.mpy
```

```
# SPDX-FileCopyrightText: 2023 Jeff Epler for Adafruit Industries
# SPDX-License-Identifier: MIT
import json
import os
import ssl
import traceback

import board
import displayio
import digitalio
import keypad
import socketpool
import supervisor
from wifi import radio

import adafruit_requests
import adafruit_displayio_ssd1306
from adafruit_bitmap_font.bitmap_font import load_font
from adafruit_display_text import wrap_text_to_pixels
from adafruit_display_text.bitmap_label import Label
from adafruit_ticks import ticks_add, ticks_less, ticks_ms


# Choose your own prompt and wait messages, either by changing it below inside
# the """triple quoted""" string, or by putting it in your settings.toml file,
# like so:
#
# MY_PROMPT="Give me an idea for a gluten free, keto dinner. Write one sentence"
# PLEASE_WAIT="Cooking something up just for you"
#
# Experimentation is best to figure out what works. Usually you'll want to ask
# for just one sentence or paragraph, since the 128x32 pixel screen can't hold
# much text!

# Here are some that the author found worked reasonably well:

# Give me an idea for a plant-based dinner. Write one sentence
#
# Give jepler (they/them) a cliched and flowery description as a comic book
# supervillain. write one sentence.
#
# Invent and describe an alien species. write one sentence
#
# Invent a zany 'as seen on' product that can't possibly work. One sentence
#
# Tell a 1-sentence story about a kitten and a funny mishap
#
# Make up a 1-sentence fortune for me
#
# In first person, write a 1-sentence story about an AI avoiding boredom in a
creative way.
#
# Pick an everyday object (don't say what it is) and describe it using only the
# ten hundred most common words.
#
# Invent an alien animal or plant, name it, and vividly describe it in 1
# sentence
#
# Invent and vividly describe an alien species. write one paragraph

prompt=os.getenv("MY_PROMPT", """
Write 1 sentence starting "you can" about an unconventional but useful superpower
""").strip()
please_wait=os.getenv("PLEASE_WAIT", """
Finding superpower
""").strip()

openai_api_key = os.getenv("OPENAI_API_KEY")

nice_font = load_font("helvR08.pcf")
line_spacing = 9 # in pixels

#  i2c display setup
displayio.release_displays()
oled_reset = board.GP9

# STEMMA I2C on picowbell
i2c = board.STEMMA_I2C()
display_bus = displayio.I2CDisplay(i2c, device_address=0x3D, reset=oled_reset)

WIDTH = 128
HEIGHT = 64

display = adafruit_displayio_ssd1306.SSD1306(
    display_bus, width=WIDTH, height=HEIGHT
)
if openai_api_key is None:
    input("Place your\nOPENAI_API_KEY\nin settings.toml")
display.auto_refresh = False

class WrappedTextDisplay(displayio.Group):
    def __init__(self):
        super().__init__()
        self.offset = 0
        self.max_lines = display.height // line_spacing
        for i in range(self.max_lines):
            self.make_label("", i * line_spacing)
        self.lines = [""]
        self.text = ""

    def make_label(self, text, y):
        result = Label(
            font=nice_font,
            color=0xFFFFFF,
            background_color=0,
            line_spacing=line_spacing,
            anchor_point=(0, 0),
            anchored_position=(0, y),
            text=text)
        self.append(result)

    def add_text(self, new_text):
        print(end=new_text)
        if self.lines:
            text = self.lines[-1] + new_text
        else:
            text = new_text
        self.lines[-1:] = wrap_text_to_pixels(text, display.width, nice_font)
        self.scroll_to_end()

    def set_text(self, text):
        print("\n\n", end=text)
        self.text = text
        self.lines = wrap_text_to_pixels(text, display.width, nice_font)
        self.offset = 0

    def show(self, text):
        self.set_text(text)
        self.refresh()

    def add_show(self, new_text):
        self.add_text(new_text)
        self.refresh()

    def scroll_to_end(self):
        self.offset = self.max_offset()

    def scroll_next_line(self):
        max_offset = self.max_offset()
        self.offset = (self.offset + 1) % (max_offset + 1)

    def max_offset(self):
        return max(0, len(self.lines) - self.max_lines)

    def on_last_line(self):
        return self.offset == self.max_offset()

    def refresh(self):
        lines = self.lines
        # update labels from wrapped text, accounting for scroll offset
        for i in range(len(self)):
            offset_i = i + self.offset
            if offset_i >= len(lines):
                text = ""
            else:
                text = lines[offset_i]
            if text != self[i].text:
                self[i].text = text

        # Actually update the display all at once
        display.refresh()

display.root_group = wrapped_text = WrappedTextDisplay()

def wait_button_scroll_text():
    led.switch_to_output(True)
    keys.events.clear()
    deadline = ticks_add(ticks_ms(),
            5000 if wrapped_text.on_last_line() else 1000)
    while True:
        if (event := keys.events.get()) and event.pressed:
            break
        if wrapped_text.max_offset() > 0 and ticks_less(deadline, ticks_ms()):
            wrapped_text.scroll_next_line()
            wrapped_text.refresh()
            deadline = ticks_add(deadline,
                    5000 if wrapped_text.on_last_line() else 1000)
    led.value = False

if radio.ipv4_address is None:
    wrapped_text.show(f"connecting to {os.getenv('WIFI_SSID')}")
    radio.connect(os.getenv('WIFI_SSID'), os.getenv('WIFI_PASSWORD'))
requests = adafruit_requests.Session(socketpool.SocketPool(radio),
ssl.create_default_context())

def iter_lines(resp):
    partial_line = []
    for c in resp.iter_content():
        if c == b'\n':
            yield (b"".join(partial_line)).decode('utf-8')
            del partial_line[:]
        else:
            partial_line.append(c)
    if partial_line:
        yield (b"".join(partial_line)).decode('utf-8')

full_prompt = [
    {"role": "user", "content": prompt},
]

keys = keypad.Keys((board.GP14,), value_when_pressed=False)
led = digitalio.DigitalInOut(board.GP10)
led.switch_to_output(False)

try:
    while True:
        wrapped_text.show(please_wait)

        with requests.post("https://api.openai.com/v1/chat/completions",
            json={"model": "gpt-3.5-turbo", "messages": full_prompt, "stream":
True},
            headers={
                "Authorization": f"Bearer {openai_api_key}",
            },
            ) as response:

            wrapped_text.set_text("")
            if response.status_code != 200:
                wrapped_text.show(f"Uh oh! {response.status_code}:
{response.reason}")
            else:
                wrapped_text.show("")
                for line in iter_lines(response):
                    led.switch_to_output(True)
                    if line.startswith("data: [DONE]"):
                        break
                    if line.startswith("data:"):
                        content = json.loads(line[5:])
                        try:
                            token = content['choices'][0]['delta'].get('content',
'')
                        except (KeyError, IndexError) as e:
                            token = None
                        led.value = False
                        if token:
                            wrapped_text.add_show(token)
                wait_button_scroll_text()
except Exception as e: # pylint: disable=broad-except
    traceback.print_exception(e) # pylint: disable=no-value-for-parameter
    print(end="\n\n\nAn error occurred\n\nPress button\nto reload")
    display.root_group = displayio.CIRCUITPYTHON_TERMINAL
    display.auto_refresh = True
    while True:
        if (event1 := keys.events.get()) and event1.pressed:
            break
    supervisor.reload()
```

Customizing the Text Generator

> Creating a chatbot prompt is like crafting a key that unlocks a door to productive conversation with your audience. -- ChatGPT

With this project, it's easy to customize the "prompt", meaning the text that is sent to ChatGPT and used to generate the response. You can do this by adding or editing two lines in the **settings.toml** file on your **CIRCUITPY** drive. This isn't coding per se (because you write in plain english), though there are a few simple rules you have to follow:

- You can't embed newlines or blank lines, so keep each item on its own single line
- Double quote characters (") have special meaning, so if you want to put quotes inside your prompt or the "please wait" text, the simplest solution is to use single quotes (') instead.
- If your editor replaces double quote characters (**"..."**) with smart quote characters (**“...”**) it won't work, so disable this or use a simple text editor that doesn't do it.

Add your lines to the **settings.toml** but don't remove the lines for WiFi and API keys you made earlier!

For example, here are the **settings.toml** lines for a 'magical university' prompt, together with other settings required for the project:

```
PLEASE_WAIT="Somewhere in the halls of Jynx University..."
MY_PROMPT="Write a vivid description of a magical mishap at the Jynx University for
Witches and Warlocks (1 sentence; no frogs; no injuries; don't say the school's
name)"
OPENAI_API_KEY="sk-b6...kP5"
WIFI_SSID="GuestAP"
WIFI_PASSWORD="i trust u"
```

A few seconds after saving the file, CircuitPython will re-load and show text based on the new prompt.

It takes trial and error to transform a mediocre prompt into a good one. So feel free to try slight re-wordings of your prompt to see what variations you like best! Here are some of the things I try to do when composing one:

- It seems silly but asking for a "vivid description" actually does tend to make ChatGPT produce more descriptive language
- Ask for a short length that's more likely to fit on the screen ("1 sentence")
- You can tell it things you don't want ("no frogs; no injuries")

- You can ask it to return the answer in a certain format
- If the prompt doesn't end with a sentence-ending period sometimes ChatGPT's response starts with just a "." followed by a blank line, so always include a "."

Of course, these things are only guidelines and the text returned by ChatGPT may not precisely match what you've asked for.

If you want to keep a prompt around for later, you can put a comment character (#) in front of each line, then write your new prompt above and below it. Then, to switch prompts, simply put # in front of the current prompt and remove them from the next prompt you want to use. If all the lines are marked with # at the beginning of the line, then the prompt that is built into the program will be used. Here are some other prompts I tested, but commented out:

```
#PLEASE_WAIT="pip install --random"
#MY_PROMPT="Make up a humorous, obviously fictional module on pypi (give it a
name). Use the following format:\npip install <modulename>\n\n<1 sentence blurb>"

#PLEASE_WAIT="This Person Does Not Exist"
#MY_PROMPT="Invent a character and describe them vividly in 1 sentence"
```

You can write prompts in other languages as well (the font included in this project supports many European languages):

```
PLEASE_WAIT="Trouver votre superpuissance"
MY_PROMPT="Ecrivez 1 phrase commençant par 'vous pouvez' à propos d'une
superpuissance non conventionnelle mais utile."
```

While using Large Language Models like ChatGPT for "factual things" is not the best idea, and this guide has concentrated on fictional items, you may also find that there are practical things you could do with it, such as get meal ideas:

```
MY_PROMPT="Give me an idea for a gluten free, keto dinner. Write one sentence"
PLEASE_WAIT="Cooking something up just for you"
```

Here are some more prompts the author thought were amusing or useful:

- Invent a zany 'as seen on' product that can't possibly work. One sentence.
- Tell a 1-sentence story about a kitten and a funny mishap.
- Make up a 1-sentence fortune for me.
- In first person, write a 1-sentence story about an AI avoiding boredom in a creative way.
- Pick an everyday object (don't say what it is) and describe it using only the ten hundred most common words.
- Invent an alien animal or plant, name it, and vividly describe it in 1 sentence.

- Invent and vividly describe an alien species. write one paragraph.
- What's one possible synergy between CircuitPython & ChatGPT? (1 sentence, practical)

Code Walkthrough

> CircuitPython could potentially be used to create a chatbot that can interact with users through hardware devices like sensors and LEDs -- ChatGPT

This program is large and complex. This guide will gloss over a lot of the details; if you'd like to learn more about using WiFi (https://adafru.it/18C0) or displayio (https://adafru.it/EGh) on CircuitPython there are dedicated guides for those topics. Below you'll find explanations of some key parts of the program functionality.

Fetching optional items from settings.toml

`os.getenv` can be used to fetch string values from the **settings.toml** file. Providing a second argument gives a default value when the key is not present. Calling `strip()` removes any whitespace from the start or end of the string, which can trip up ChatGPT. This is an easy way to add "no-code" customizations to your own CircuitPython programs:

```
prompt=os.getenv("MY_PROMPT", """
Write 1 sentence starting "you can" about an unconventional but useful superpower
""").strip()
please_wait=os.getenv("PLEASE_WAIT", """
Finding superpower
""").strip()
```

Vertically scrolling wrapped text

There aren't yet any CircuitPython libraries for dealing with larger amounts of text. This project includes a class called **WrappedTextDisplay** which can help.

It allows showing a screen full of text which can be part of a larger document. The text can be scrolled by lines, and new words can be added at the end of the text incrementally, with relatively good performance.

This functionality is very closely matched to the needs of this program but it could provide some ideas for a future library.

The text is parsed into lines as it arrives. A certain number of lines can be visible on the screen at one time, an each one of those visible lines of text gets its own bitmap label object. Scrolling consists of changing which logical line in the document is the first line visible on the display. This approach performs relatively well in terms of both repaint time—especially while streaming content from ChatGPT—and memory used.

```
class WrappedTextDisplay(displayio.Group):
    ...
```

Once a response from ChatGPT is complete, this function repeatedly scrolls through the full response if it's more than one screenful, then returns when the button is pressed:

```
def wait_button_scroll_text():
    led.switch_to_output(True)
    deadline = ticks_add(ticks_ms(),
            5000 if wrapped_text.on_last_line() else 1000)
    while True:
        if (event := keys.events.get()) and event.pressed:
            break
        if wrapped_text.max_offset() > 0 and ticks_less(deadline, ticks_ms()):
            wrapped_text.scroll_next_line()
            wrapped_text.refresh()
            deadline = ticks_add(deadline,
                    5000 if wrapped_text.on_last_line() else 1000)
    led.value = False
```

Streaming an HTTP response by lines

When the OpenAI API is used with `"stream": True`, the text is returned as it is generated. You can check the OpenAI API documentation for more details, but in short each line starts "data:" followed by a JSON document all on one line.

By using the `iter_lines` function you can handle the HTTP response one line at a time:

```
def iter_lines(resp):
    partial_line = []
    for c in resp.iter_content():
        if c == b'\n':
            yield (b"".join(partial_line)).decode('utf-8')
            del partial_line[:]
        else:
            partial_line.append(c)
    if partial_line:
        yield (b"".join(partial_line)).decode('utf-8')
```

Prompt and Request

In this program, the Prompt is simple, it consists of just a single "user" message. The content of the message is the prompt, either from the top of the code or from **settings.toml**:

```
full_prompt = [
    {"role": "user", "content": prompt},
]
```

When there is a dialogue between ChatGPT and the user that takes place over multiple exchanges, then the `full_prompt` can consist of multiple messages with various roles (system, user, and assistant). But in this project there's just one prompt and one response.

Within the program's forever loop, the full prompt is put together with other information in the json payload of the request (See OpenAI's dedicated documentation pages (https://adafru.it/18Ao) for more details on the API):

```
while True:
        wrapped_text.show(please_wait)

        with requests.post("https://api.openai.com/v1/chat/completions",
            json={"model": "gpt-3.5-turbo", "messages": full_prompt, "stream":
True},
            headers={
                "Authorization": f"Bearer {openai_api_key}",
            },
            ) as response:
```

If the response is successful, then it can be read line by line; each line may contain some additional part of the response (called the "delta"). This text is added to the display, which is refreshed. Just to emphasize that something is happening, the LED blinks during this process. At the end, the program waits for the button to be pressed before it repeats the process again.

```
#
            if response.status_code != 200:
                wrapped_text.show(f"Uh oh! {response.status_code}:
{response.reason}")
            else:
                wrapped_text.show("")
                for line in iter_lines(response):
                    led.switch_to_output(True)
                    if line.startswith("data: [DONE]"):
                        break
                    if line.startswith("data:"):
                        content = json.loads(line[5:])
                        try:
                            token = content['choices'][0]['delta'].get('content',
'')
                        except (KeyError, IndexError) as e:
                            token = None
                        led.value = False
                        if token:
                            wrapped_text.add_show(token)
                wait_button_scroll_text()
```

Error handling

In the case of most errors, the program catches the error so that a press of the button can re-start from scratch. The original error is shown on the REPL for troubleshooting purposes.

```
except Exception as e: # pylint: disable=broad-except
    traceback.print_exception(e) # pylint: disable=no-value-for-parameter
    print(end="\n\n\nAn error occurred\n\nPress button\nto reload")
    display.root_group = displayio.CIRCUITPYTHON_TERMINAL
    display.auto_refresh = True
    while True:
        if (event1 := keys.events.get()) and event1.pressed:
            break
    supervisor.reload()
```