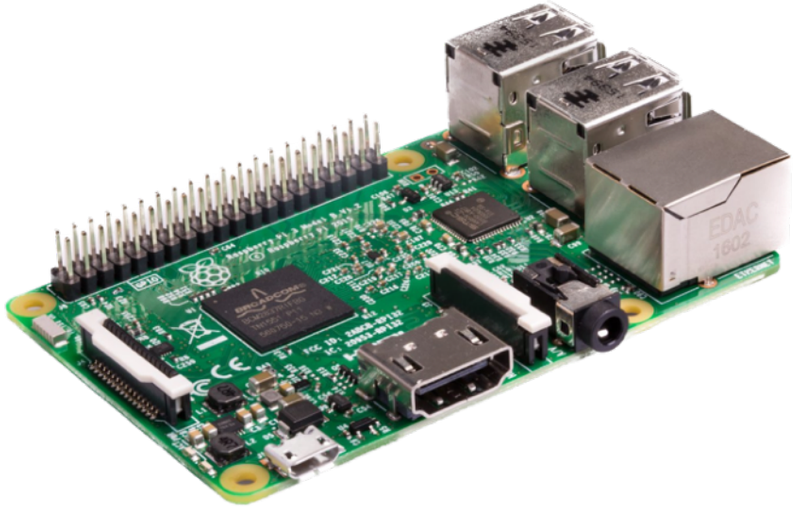
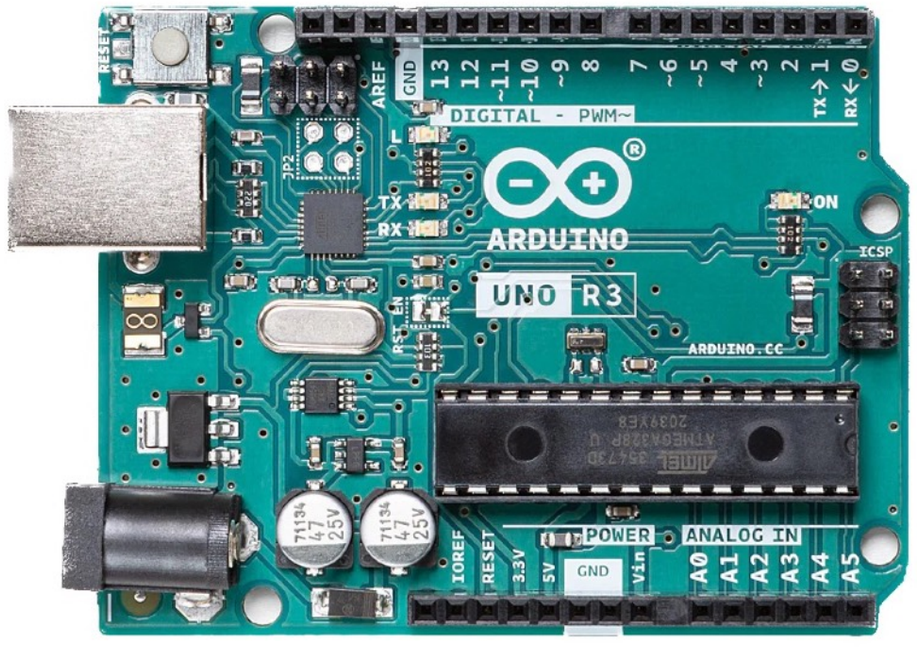
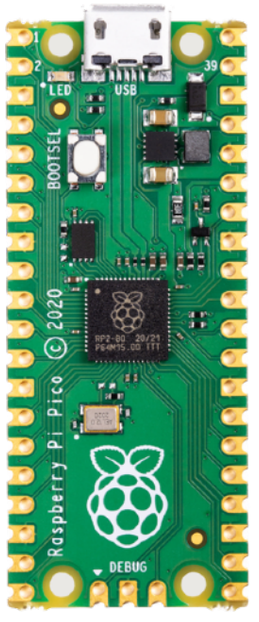
Embedded Controllers

Embedded Controllers

- Embedded control is the concept of using a microcontroller as the brains of your project







Each microcontroller choice has issues

- Processor complexity – number of bits for instruction set
- Development complexity – language used to program
- Documentation and Support – time in market and design goals
- Execution Speed – clock speed of processor
- Memory for storage and for execution – SRAM and Flash RAM
- Power consumption – battery issue?

- Good news? Price is rarely an issue!

Simple Board Comparison

	Arduino Uno (2010)	Raspberry Pi Pico (2020)
Clock Speed	16MHz	133MHz
Instruction Width	8bit	32bit
SRAM (programs)	64kB	264kB
Flash RAM (data)	3kB	2MB
GPIO	23	26
PWM	6	16
Board Cost	$28	$4

So which one do we use?

- What is the purpose?
  - **Control**? – do you want something to blink, move? Does it need to do it quickly?
  - **Data Gathering?** – do you want to measure something over a period of time? How much analysis do you want to do on the microcontroller?
- For **control** - the **Uno** is the easiest to make the fastest, has a simple hardware model, making it easy to interface
- For **data gathering** – the **Pico** has the most memory, has a language which supports a file structure, making it easy to gather data

Blink – the first program

- Blink is invariably the first program as it is a fast way to see your results
- Blink can also indicate speed of execution
- Blink Iterations:
  - On/Off – indicate an operating state
  - Blink Faster – indicate an operating state efficiently
  - Blink Faster at a specified rate – communications protocol such as I2C or SPI
  - Blink Faster with different colors – video protocol such as VGA video
- The faster the processor can process commands, the more the processor can appear to do at once, **multi-tasking**

Activity – Blink!

On each board/development environment do the following:

1. Blink an LED
2. Determine the fastest speed the board can blink

Board/Development Environments

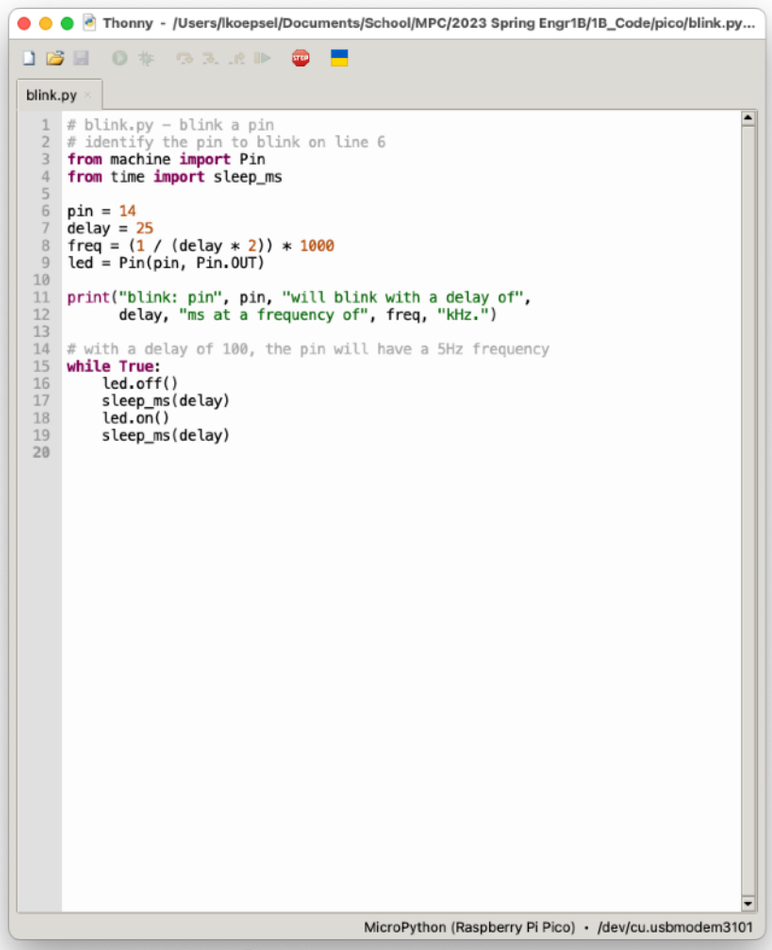
1. Arduino Uno using *C++* with the Arduino 1.8 IDE
2. Raspberry Pi Pico using *MicroPython* and Thonny IDE

Integrated Development Environment (IDE)

Arduino 1.8

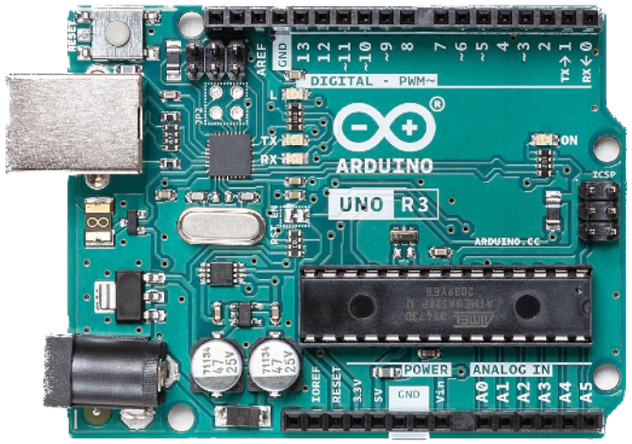
Arduino Uno

Thonny



Raspberry Pi Pico

Parts



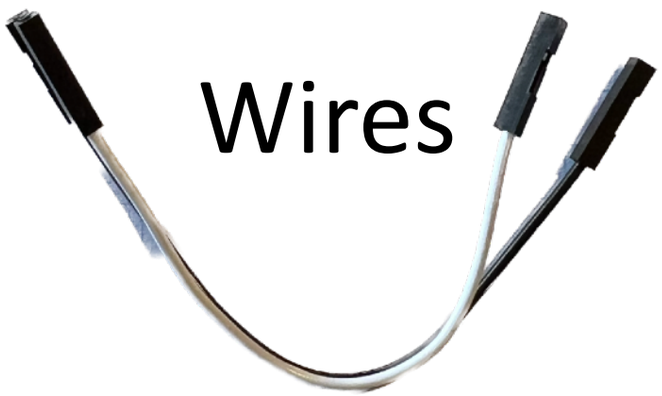
Arduino Uno

Uno USB cable

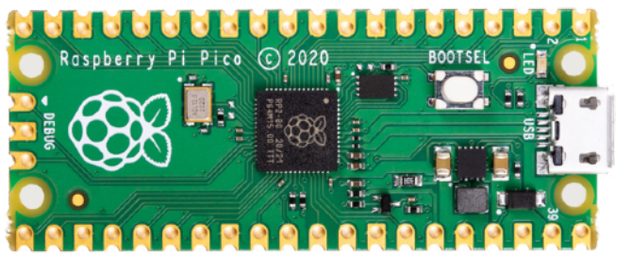
BlinkenLED

Other pin

Resistor pin



Wires



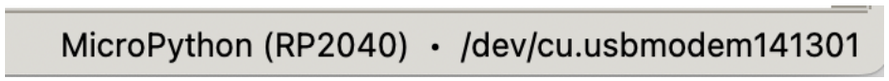
Pi Pico

microUSB cable

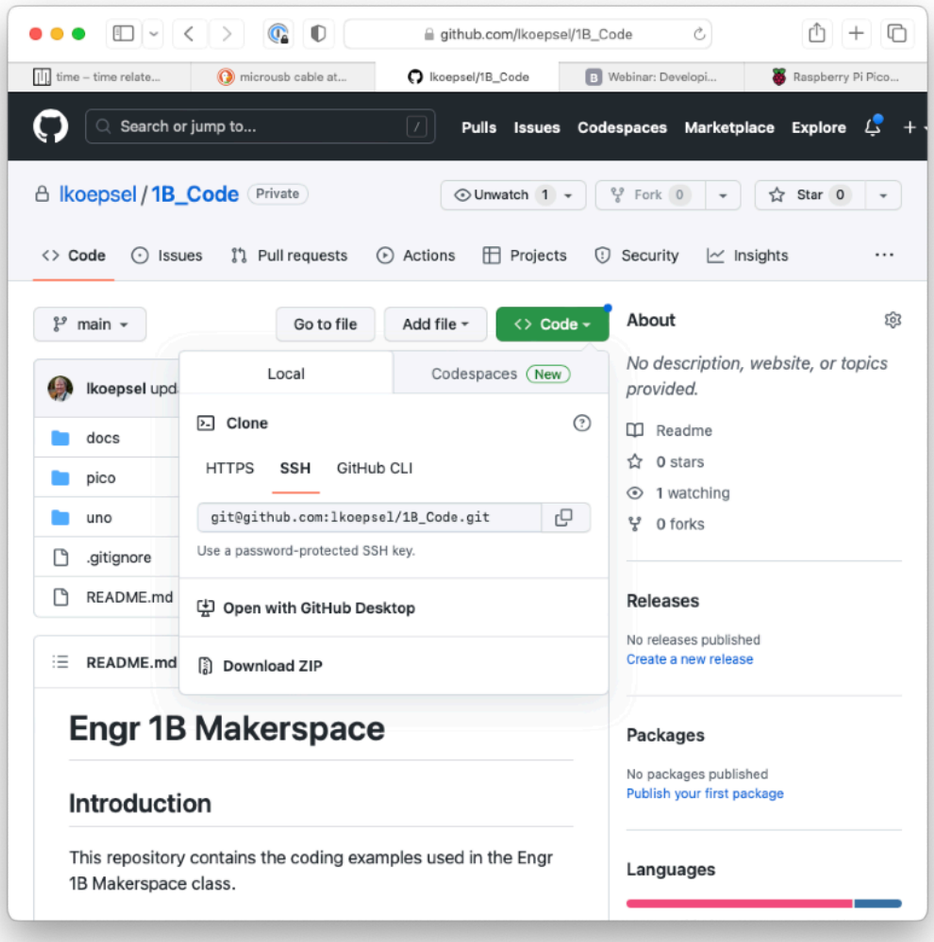
General Instructions for both boards

1. When making connections to the board pins, **DISCONNECT** the board from the computer
2. The **resistor-side** of the blinkenLED always goes to **ground**
3. Confirm connections **TWICE** before plugging board into USB, this reduces the chance of “frying” the board
4. Once plugged in, confirm the IDE can “see” the board before attempting to program
   - Arduino: Tools -> Port -> “board”

   - Thonny: Lower right-hand corner “board * port”



Download from https://github.com/lkoepsel/1B_Code



- Programs are pre-written to accelerate your understanding
- Click on green button then "Download ZIP"
- Extract to your desktop
- Pico files are in pico folder
- Uno files are uno folder

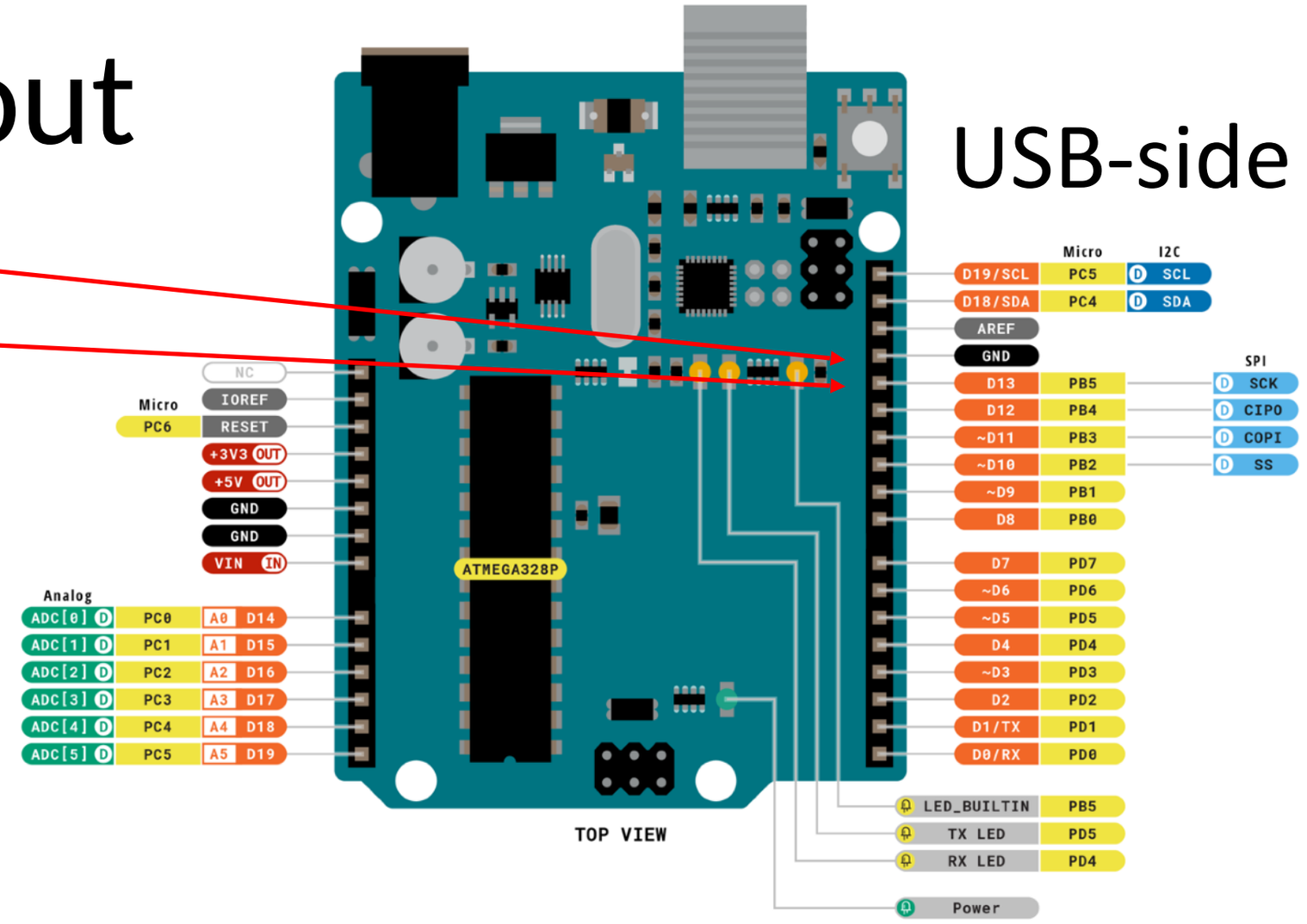
Arduino Uno - Blink

1. Use Arduino to edit your program and send it to the Uno
2. With the Uno **unconnected**:
   A. Connect , resistor-side pin to GND (4th from top, USB-side)
   B. Connect other pin to D13 (5th from top, USB-side)
3. Plugin Uno
4. Confirm connection in Arduino
5. Open blink.ino from uno folder and upload to Uno board
6. Is light blinking correctly?
7. Now alter the program to run as fast as possible, still blinking
8. At what freq does it blink? (*Ask for digital multimeter*)

Uno R3 Pinout

GND

D13



Legend:

Power

Ground

Digital

Analog

Main Part

I2C

SPI

Analog

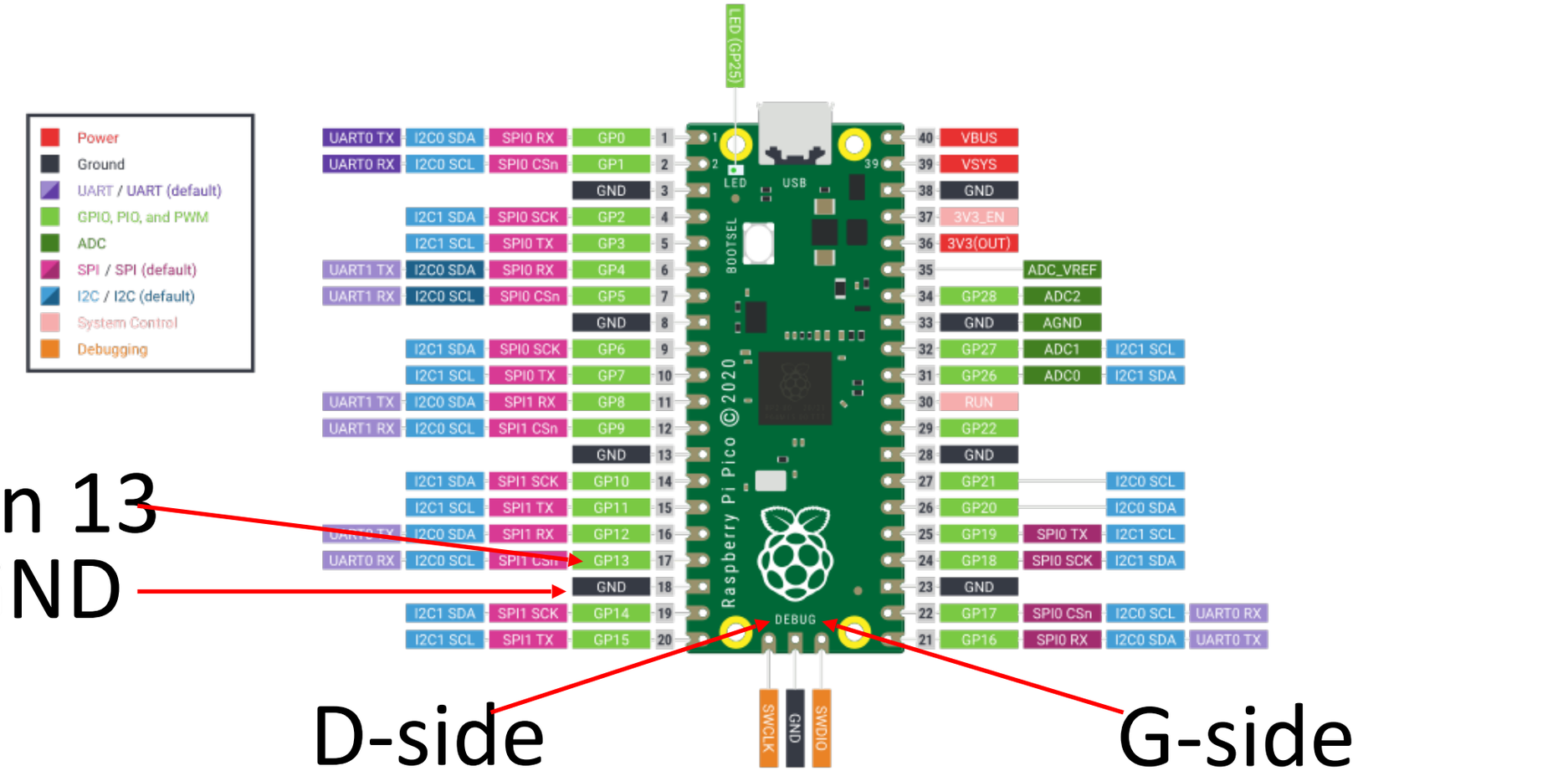
ARDUINO

ARDUINO UNO REV3
SKU code: A000066
Pinout
Last update: 6 Oct, 2022

Raspberry Pi Pico - Blink

1. Use Thonny to edit your program and send it to the Pico
2. With the Pico **unconnected**:
   A. Connect , resistor-side pin to GND (3rd from bottom, D-side)
   B. Connect other pin to GP13 (4th from bottom, D-side)
3. Plugin Pico
4. Confirm connection in Thonny
5. Open blink.py from pico folder and Save As to pico board
6. Is light blinking correctly?
7. Now alter the program to run as fast as possible, still blinking
8. At what freq does it blink? (*Ask for digital multimeter*)

Raspberry Pi Pico Pinout



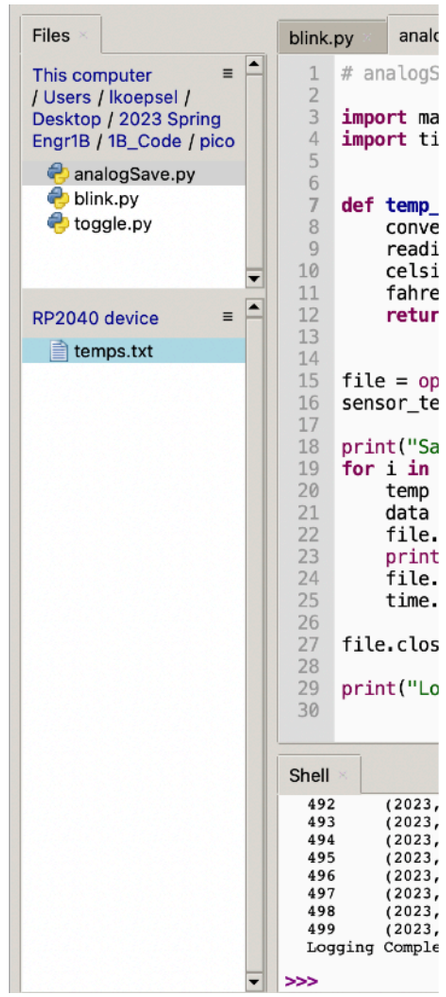
Raspberry Pi

Arduino Uno - Data Logger

1. Use Arduino to edit your program and send it to the Uno
2. For the data logging, we'll leave the pin A0 unconnected and measure air temperature
3. Plugin Uno
4. Confirm connection in Arduino
5. Open AnalogSave.ino from uno folder and upload to Uno board
6. Open eepromRead from the uno folder and upload to Uno Board
7. Is the data read the same as what you wrote?
8. How many samples are you able to capture?

Raspberry Pi Pico - Data Logger

1. Use Thonny to edit your program and send it to the Pico
2. For the data logging, we'll use the processor core temperature
3. Plugin Pico
4. Confirm connection in Thonny
5. Open AnalogRead.py from pico folder and upload to Pico board
6. Open *temps.txt* from the pico
    A. Use View -> Files
    B. Look in the RP2040 window for temps.txt, double click to open
7. Is the data read the same as what you wrote?
8. How many samples are you able to capture?
9. What else do you notice about the data?



Appendix - Extra Slides

Simplifying the tradeoffs

- **Speed of development:**
  - How hard is the processor, language and framework to learn?
  - How complex is the development environment?
  - How quickly can I have a minimum viable project?

- **Ability of execution:**
  - What is the speed of the controller, compared to the task?
  - How much data can the controller handle, compared to the task?
  - How can the internal hardware help?

Why is Speed of Development important?

- You want to accomplish something!
- You want to be efficient.
- You want to enjoy what you are doing.
- You want to search less and develop more.
- You don’t want to have to jump through hoops.

Speed of Development Components

- **Language – Python, C Language, Assembly Language**
- Framework – Libraries, Headers, Functions, Library Management
- Tool Chain – Compilers, Linkers, Editors
- GUI vs. CLI – Ease of view vs. Ease of Automation

- Ultimately, all decisions come down the **language followed by the framework**

Speed of Execution

- Board and Language Compatibility
  - **Raspberry Pi** – all languages, however, OS has significant overhead
  - **Raspberry Pi Pico** – all languages
  - **Arduino Uno** – C/C++, Assembly Language, best supported processor
  - ESP32 – all languages
  - STM32 – all languages

Language vs. Speed