Physical Computing using Python

Grant Hutchison - grant.hutchison@gmail.com

https://bit.ly/CEMC2022_Python_Physical

Session Goals

Learning Goals

- Explore how physical computing can improve student learning in K-12 Computer Science.
  - **Why Physical Computing?**

  - Classroom Experiences introducing Physical Computing using Python
    - **Micro:bits** - introductory microcontroller (low floor) - explore communications
    - **Phidgets** - extend programming projects - Pygame Zero / CMU CS Academy
    - **Pi Pico** - Build low interactive control systems and robotics projects

Everyone Likes to Make Stuff!

- "**Physical computing** covers the design and realization of interactive objects and installations and allows students to develop concrete, tangible products of the real world that arise from the learners' imagination.

  This way, **constructionist learning** is raised to a level that enables students to gain haptic experience and thereby **concretizes the virtual**."

  [Przybylla/Romeike, 2014]

Why Physical Computing?

Excellent tool to implement **Universal Design for Learning** (UDL) in CS.

Engagement

Provide options for physical actions.

Provide options for expression and communication.

Provide options for **executive function**

Representation

Provide options for perception.

Provide options for language (build digital vocabulary)

Provide options for **comprehension.**

Action & Expression

Provide options for recruiting interests.

Provide options for sustaining effort and persistence.

Provide options for **self-regulation**.

Project Based Learning

Success Criteria

I am able to design an integrated hardware system that is controlled using Python. [Hardware first approach]

I am able to add hardware components as sensors and/or controllers to an existing software Python software project. [Software first approach]

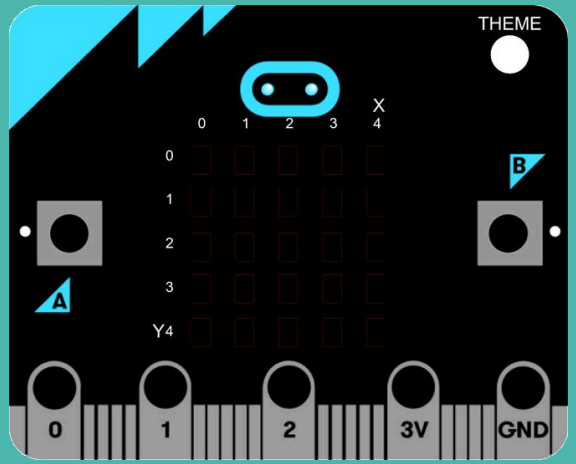
I can apply an engineering design process to design, build, and test integrated hardware and software projects. [Communication and Thinking]

Physical Computing - Which platforms and tools?

Category	Description	Examples
Packaged electronics. No programing.	Kits of packaged components and modules.	Snapcircuits
Packaged programmable products.	Robots.	Ozobot
	Construction kits.	LEGO® WeDo
Board level peripheral devices	Integrated I/O.	Makey Makey
	Modular I/O.	**Phidgets** - USB sensors
Board level embedded devices. Need PC to program but operate standalone.	Microcontroller boards with integrated I/O devices.	**Micro:bit** - MCU
	Microcontroller boards with low-level I/O.	**Raspberry Pi Pico** - MCU
Board level general-purpose devices	SBC - Single board computers.	Raspberry Pi family computers.

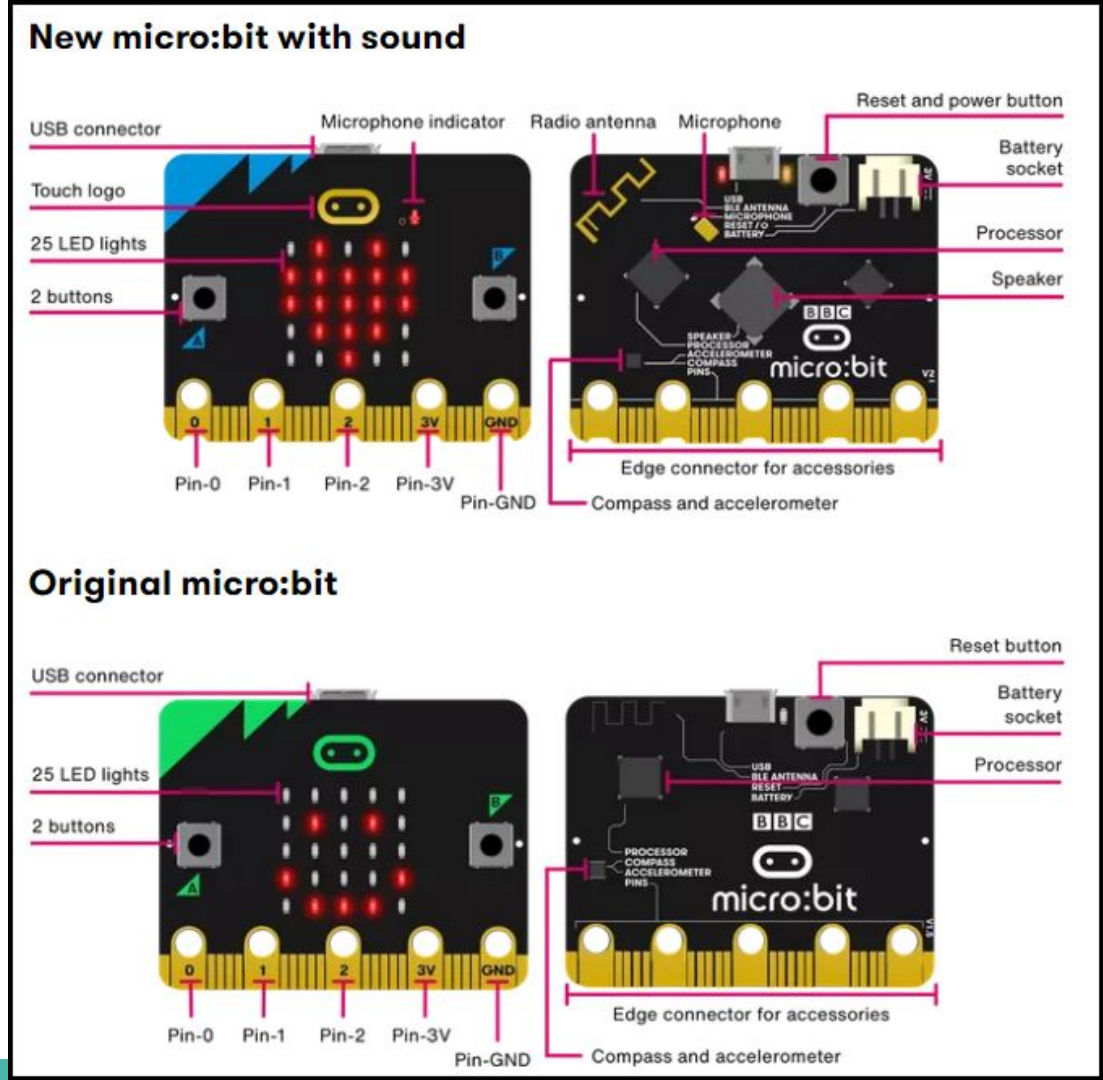
Adapted from: Hodges et al. - Physical computing: A key element of modern computer science education

Micro:bit - Coding With Python



Micro:bit Boards

- Microcontroller
  - v2 added
    - speaker
    - touch sensor (logo)
    - microphone
- Connect to any computer / chromebook
  - USB type A to micro USB
  - firmware can be updated
- Languages
  - MakeCode
    - Blocks, JavaScript, Python
  - Python



High Level Overview of Micro:bit Capabilities

Software programs can be simplified as an **Input**, **Processing**, and **Output** machine.

Input	Processing	Output
● buttons (2) ● accelerometer ● temperature ● compass ● radio ● pins (breakout) ● **microphone** ● **touch (logo)**	● Micropython ● MakeCode ○ blocks / JavaScript	● LED 5x5 matrix ● pins (breakout) ● **speaker**

Python Requirements

- Web IDEs
  - Micro:bit online editor - **https://python.microbit.org/v/2**
  - Makecode - **https://makecode.microbit.org/**

- Installable IDEs
  - Mu - **https://codewith.mu/**
  - Thonny

Preferred Python IDE for Integrated Projects for High School students

- Thonny - **https://thonny.org/**

Micro:bit Online Editor - https://python.microbit.org/v/2

micro:bit

Search

Reference

Ideas

API

Project

Variables
Keep track of data that changes

Display
The micro:bit's LED display output

Buttons
Use button inputs in your code

Loops
Count and repeat sets of instructions

Logic
Making decisions in code

Accelerometer
Detect gestures and movement

Comments

Beta release

More

Feedback

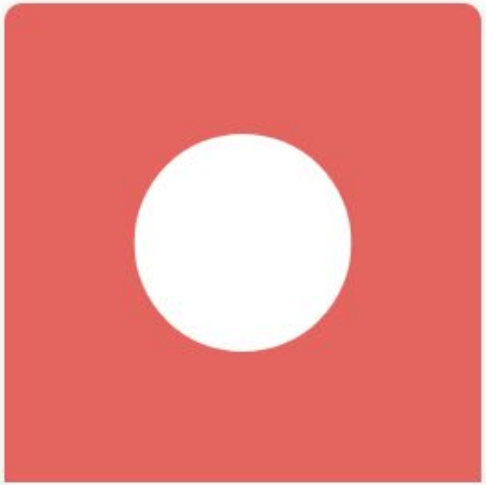
light_sensor_logging (2)

```python
# Import required libraries
from microbit import *
import log

log.set_labels('light')

# provide user with instructions
display.scroll("B TO STOP")

# set up variable to control when logging should happen
logging = True

# We will log light data until the button is pressed
while logging:
    if button_b.was_pressed():
        logging = False
    else:
        log.add({
          'light': display.read_light_level()
        })
        sleep(1000) # wait a second before logging again

display.show(Image.HAPPY)
```

Send to micro:bit

https://python.microbit.org/v/beta

Microbit - Lesson ideas - https://microbit.org/teach/for-teachers/#why-teach-computing?



Inputs and outputs

Use the micro:bit LEDs to explore the topic

Read more

Selection and sensors

Introduce students to the micro:bit sensors

Read more

Variables

Use micro:bit projects to teach variables

Read more

Micro:bit - Project opportunities

Opportunities

- Systems Monitoring and Data Analysis
  - built-in or external sensors to collect and analyze data
  - collected data can be visualized / analyzed
    - Python Digital Notebooks (Callysto.ca / Google Colab) or Google Sheets/MS Excel
- Integrated Systems
  - design and built a fully functional project
    - eg. Security System, Plant monitor and watering system, Electronic measuring device, etc..
- Robotics
  - Limited use - consider more extensible microcontrollers - Arduino or Pi Pico

Benefits

- Simple web programming interfaces - blocks, JavaScript, and Python
- Emulators available if you don't have access to Physical Devices
- Built-in LED matrix display and some sensors (speaker etc..)

Project Ideas for Micro:bits using Python

Rock, Paper, Scissors game - resource link

- Computing concepts: conditionals, randomness, state management using variables

- Extensions with Micro:bit
  - INPUT
    - Use buttons
      - selection (ROCK, PAPER, SCISSORS)
    - Use accelerometer
      - manage state change
  - OUTPUT
    - LED matrix and/or speakers

Project Ideas for Micro:bits using Python

Dot Chaser Game - resource link

- Computing concepts:
  - conditionals, randomness, state management using variables, 2 dimensional structures (rows/columns)

- Extensions using Micro:bit
  - INPUT
    - Use accelerometer to move player on matrix
      - manage movement of player and target (corners)
  - PROCESSING
    - Detect when target has been caught 5 times
    - Calculate elapsed playing time
  - OUTPUT
    - Pixel display
    - Scroll time to user

Project Ideas for Micro:bits using Python

Secret Messages

- Computing concepts:
  - data encoding, modulus arithmetic, communications (**built in radio on Micro:bit**)

- Extensions using Micro:bit
  - INPUT
    - Encode messages using Caesar cipher
  - PROCESSING
    - Send encoded messages
    - Receive encoded message and decode them properly
  - OUTPUT
    - Display decoded message

Phidgets - Coding With Python

Project Based Learning

Success Criteria

I am able to design an integrated hardware system that is controlled using Python. [Hardware first approach]

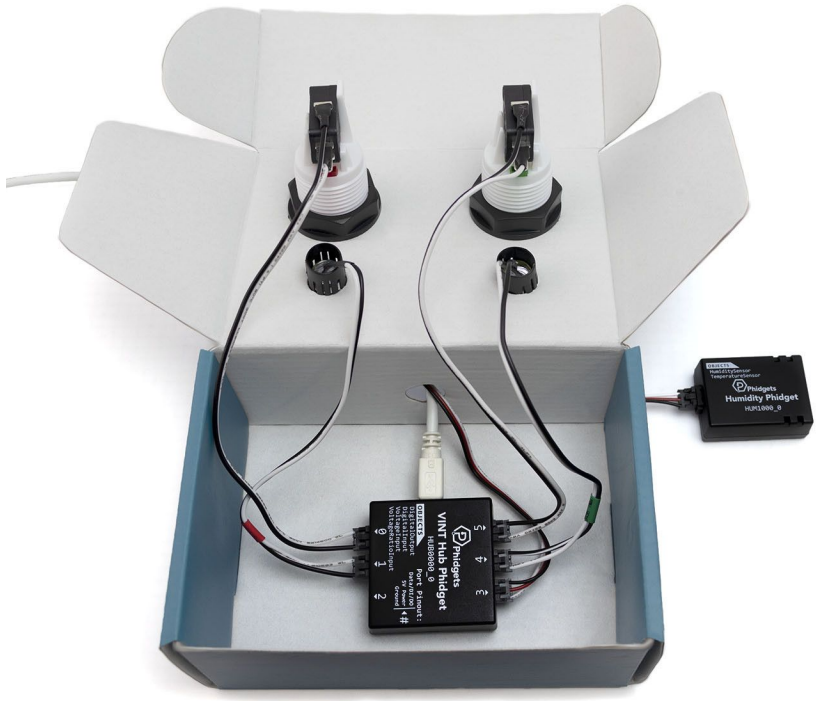
I am able to add hardware components as sensors and/or controllers to an existing software Python software project. [Software first approach]

I can apply an engineering design process to design, build, and test integrated hardware and software projects. [Communication and Thinking]

What is a Phidget?

Getting Started Kit

- VINT Hub Phidget (6 devices)
- Humidity Phidget
- Push buttons (2)
- LEDs (2)
- all required cables



Classroom use of Phidgets

2021/2022 School Year - Riverdale CI / TDSB

- 30 Getting Started Kits to be shared in Grade 11/12 Computer Science
- 15 Thumbstick Phidgets
- Servo Phidget / Sonar Phidget / Power (PSU) Phidget

Use Cases

- Pygame Zero
  - Thumbstick (Sprite/Actor motion) and button (firing)
  - PSU and Vibrating Motor for Haptic feedback in games
- CMU CS Academy
  - Dual thumbstick, buttons and LEDs for Agario style 2 player game

CMU - CS Academy

1. Python + Graphics
2. Integrated LMS
3. Controlled online environment

Go further ...

Desktop CMU Graphics

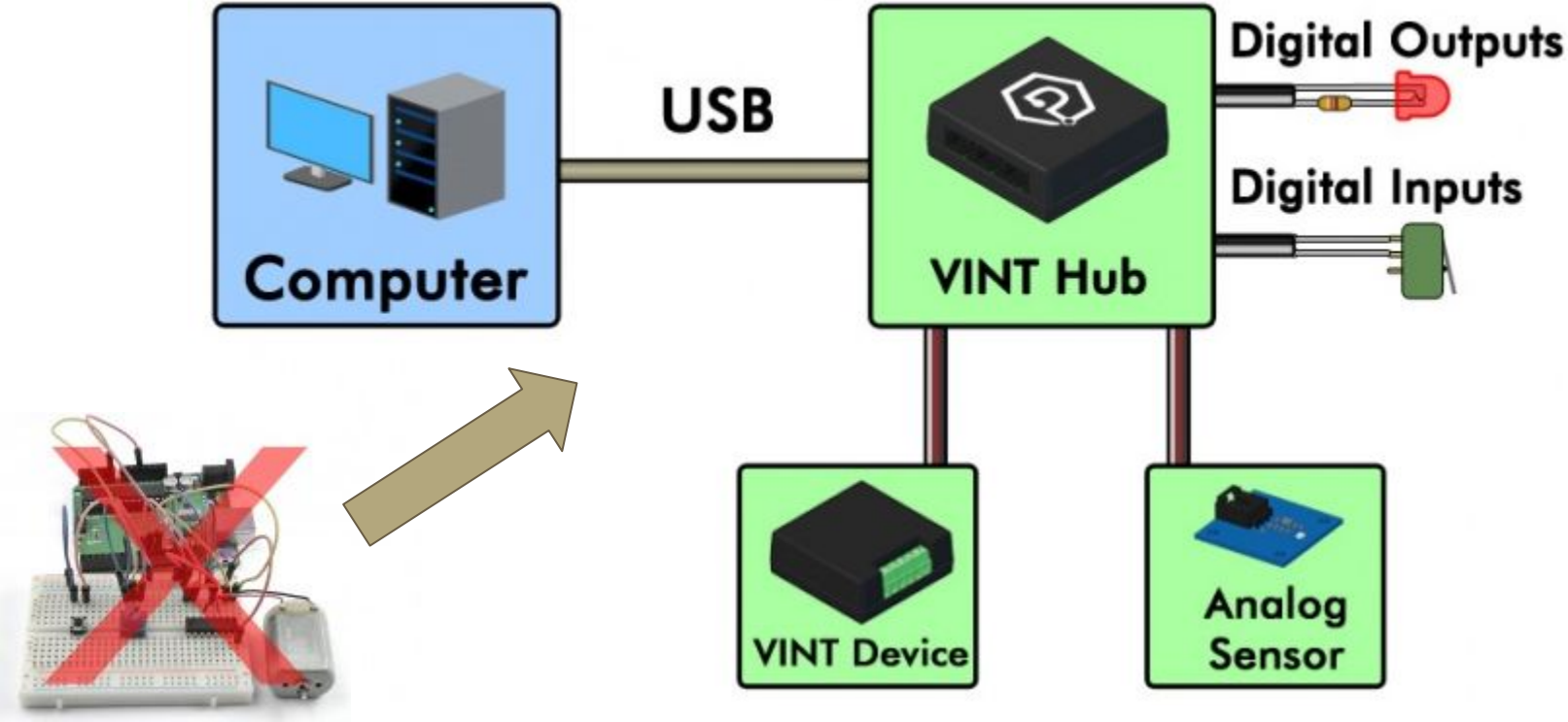
Physical Computing with Phidgets

Phidgets

1. Simple connectivity - USB
2. Plug and Fun
3. No limits

Creative and Interactive !!

Phidget Architecture



Select your coding environment

LEVEL

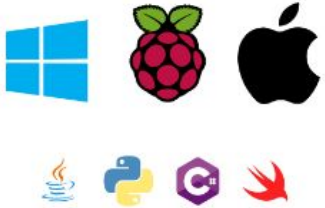
Scratch

Beginner-friendly, block-based coding through your browser

LEVEL

MakeCode

Intermediate, block-based and text-based coding through your browser



LEVEL

Desktop

Advanced, text-based coding on your desktop environment of choice

Phidgets / CMU

BEGINNER

Buttons

Learn how to use buttons with CMU Graphics

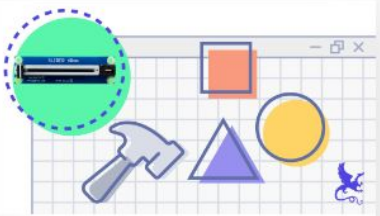
Learn more

BEGINNER

Thumbstick Phidget

Learn how to use the Thumbstick Phidget with CMU Graphics

Learn more

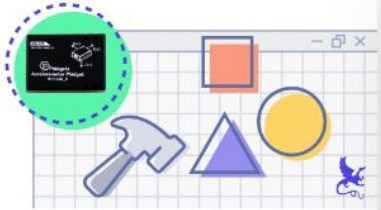


BEGINNER

Slider Phidget

Learn how to use the Slider Phidget with CMU Graphics

Learn more

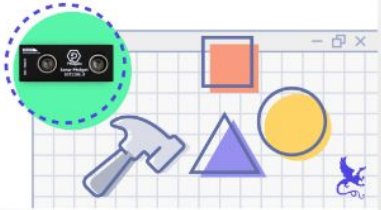


BEGINNER

Accelerometer Phidget

Learn how to use the Accelerometer Phidget with CMU Graphics

Learn more



BEGINNER

Distance Phidget

Learn how to use the Distance Phidget with CMU Graphics

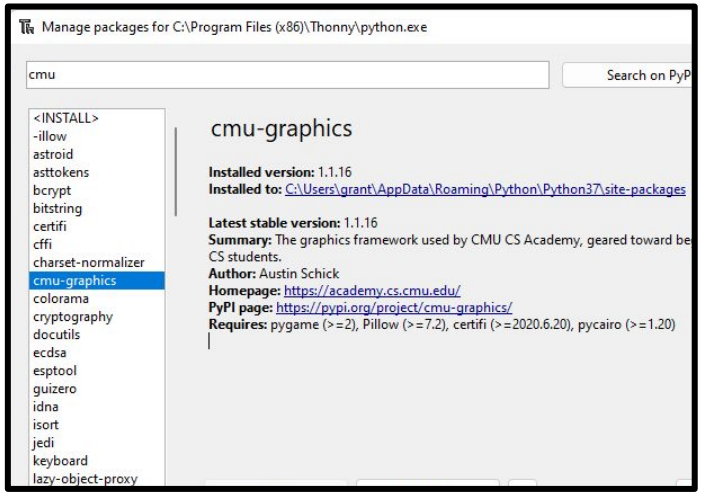
Learn more

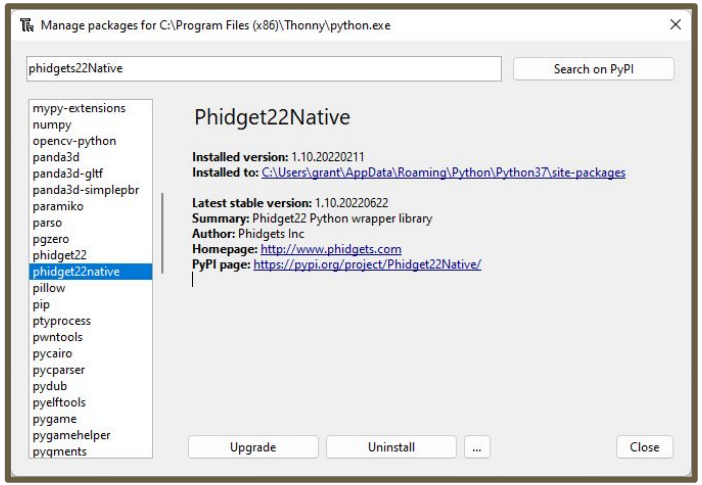
The Setup... Thonny IDE

1. Install Phidgets (platform)
   - USB Driver + Control Panel
     https://www.phidgets.com/docs/OS - Windows

2. Install CMU desktop graphics
   - PIP install
     https://pypi.org/project/cmu-graphics/

3. Install Phidgets - Python library





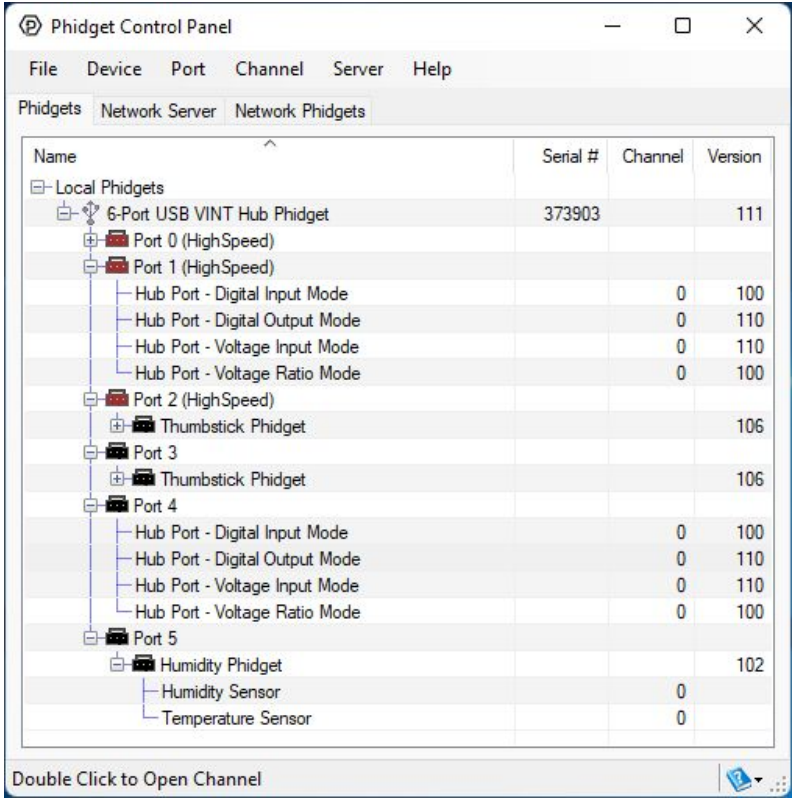
Agario Game using CMU and Phidgets

Components

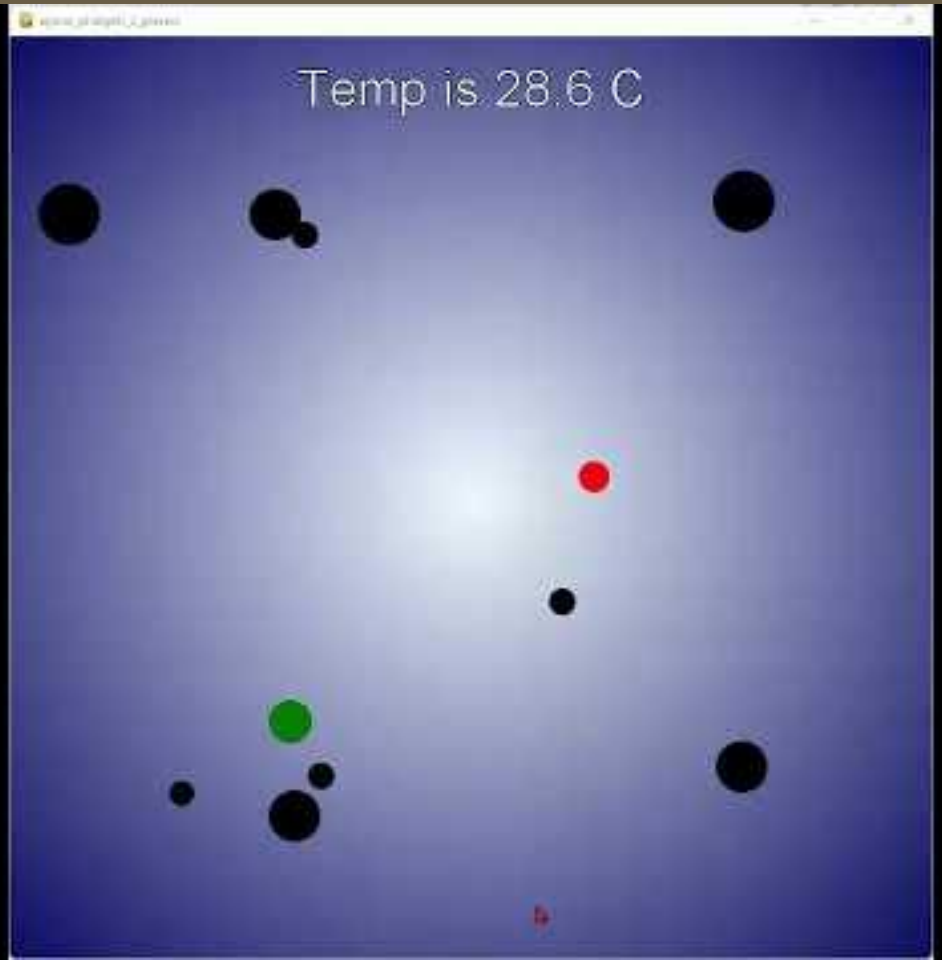
- Getting Started Kit
  - Red LED
  - Green LED
  - Humidity / Temp Sensor

- 2 Thumbstick Phidgets
  - integrated buttons

NEW WebUsb [Phidget Control Panel]



https://github.com/humberhutch/PhidgetFun



Phidgets - Project opportunities

Opportunities

- Gaming Extensions - Pygame Zero / CMU CS Academy
  - design and built a fully functional project using USB sensors and controllers

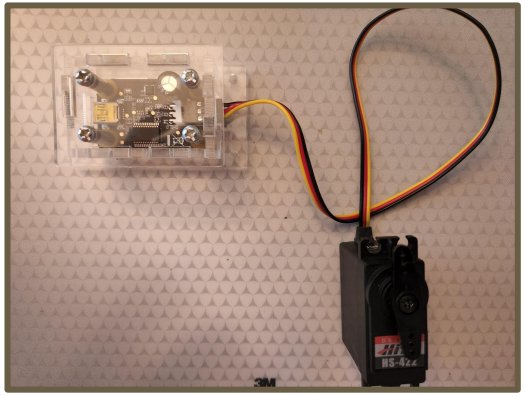
- Systems Monitoring and Data Analysis
  - 100s of modular USB sensors available to collect data
  - **collected data can be visualized / analyzed**
    - Python Digital Notebooks (Callysto.ca / Google Colab) or Google Sheets/MS Excel

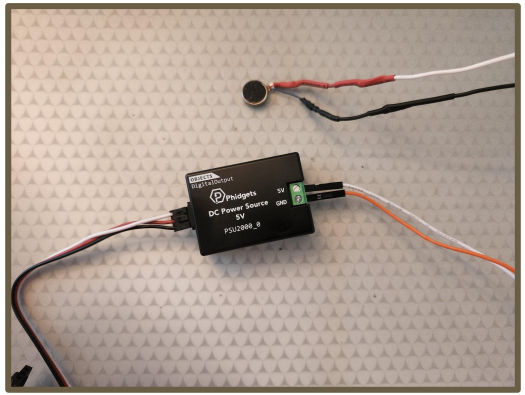
Benefits

- Minimal electrical engineering knowledge required
- Professional quality sensors
- Access to a full computing platform
  - Raspberry Pi would provide HDMI display and integrated camera capabilities

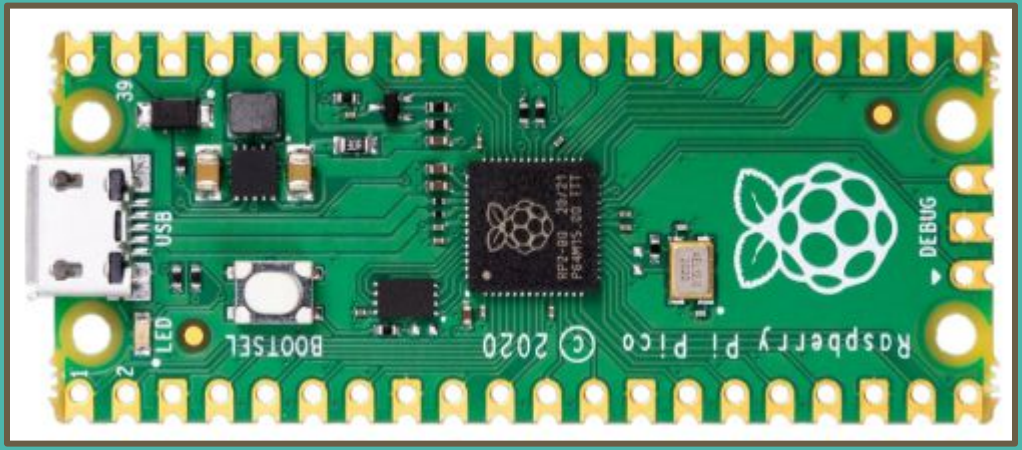
Advice - Phidgets in the Classroom

1. **CMU** - use **onStep()** function to **poll sensors**.
2. Consider using **Phidget event handlers**
3. Pygame Zero - overlay key handler functions with Thumbstick
4. Explore more phidgets
   - Distance Sensor
   - Light Sensor
   - Accelerometer
   - Motion Sensor
   - Servo Controller (single or 16x)
   - DC Power Source 5V - 500mA





Raspberry Pi Pico - Coding With Python



Project Based Learning

Success Criteria

I am able to design an integrated hardware system that is controlled using Python. [Hardware first approach]

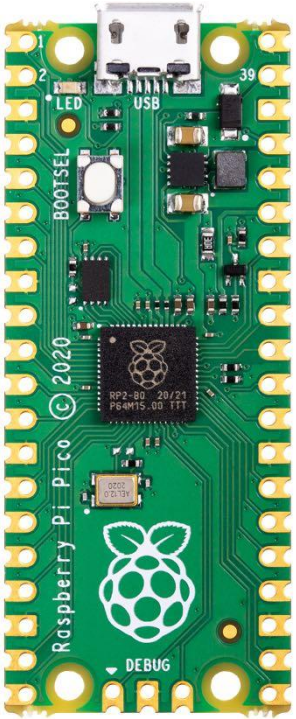
I am able to add hardware components as sensors and/or controllers to an existing software Python software project. [Software first approach]

I can apply an engineering design process to design, build, and test integrated hardware and software projects. [Communication and Thinking]

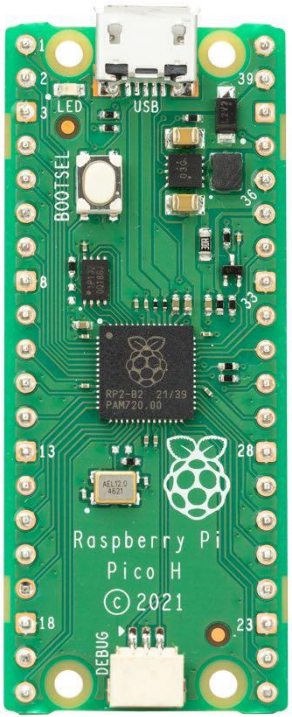
Specifications

- **RP2040** microcontroller chip designed by Raspberry Pi
- **Dual-core** Arm Cortex-M0+ processor - up to **133 MHz**
- 264KB on-chip SRAM
- 2MB on-board Flash storage
- **26 multifunction GPIO pins**, including **3 analogue inputs**
- 2 × UART, 2 × SPI controllers, **2 × I2C controllers**, **16 × PWM channels**
- 1 × USB 1.1 controller
- Supported input power **1.8–5.5V DC**
- Drag-and-drop programming using mass storage over USB
- Low-power sleep and dormant modes
- **Accurate on-chip clock**
- **Temperature sensor**

Summer 2022 - 3 Pico Options



Pico

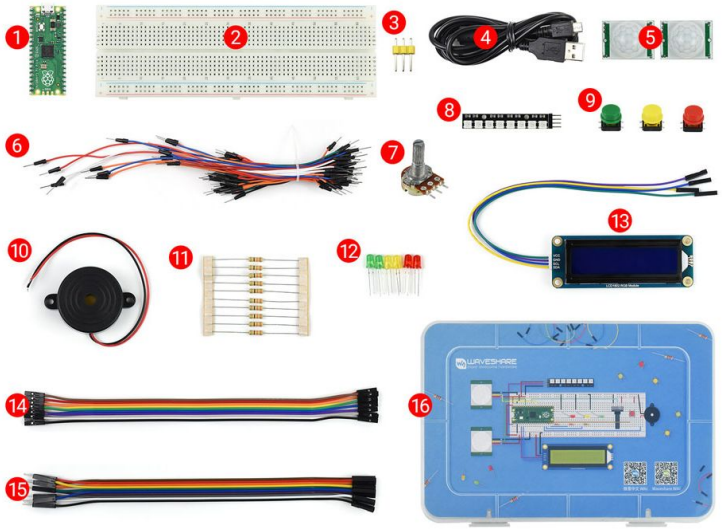


Pico H

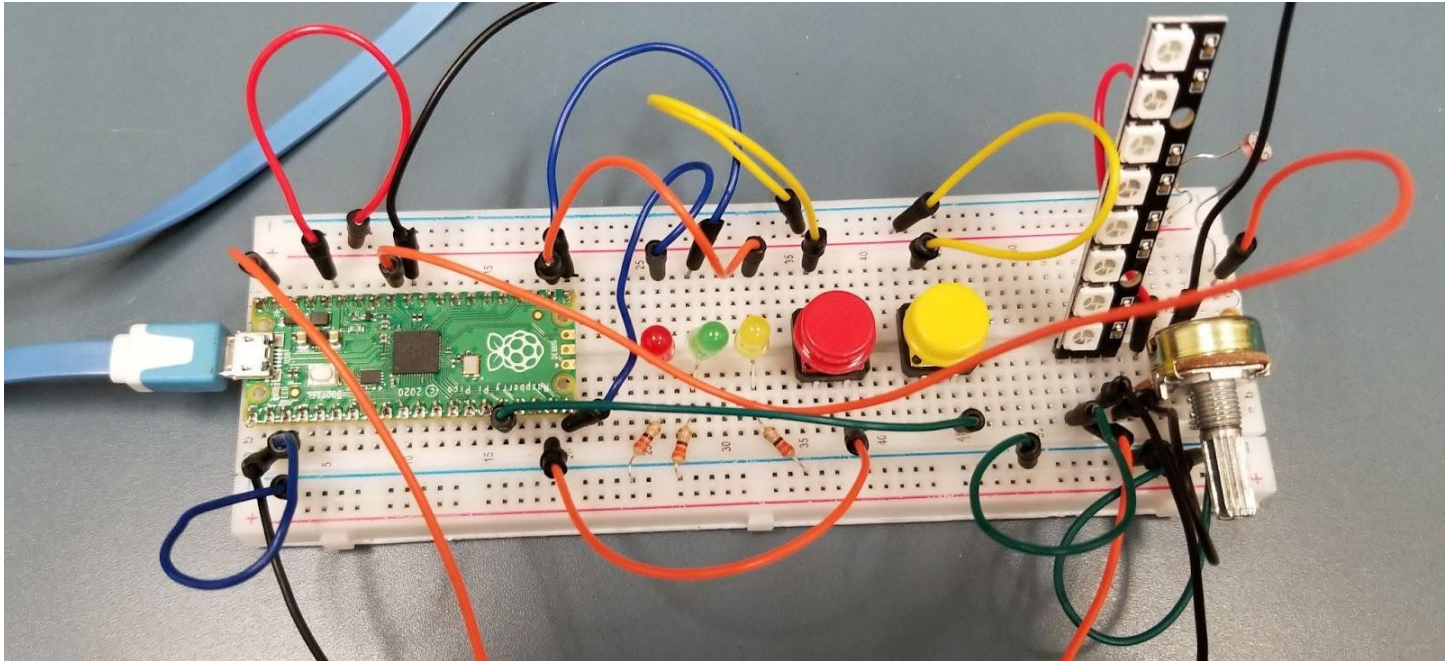
Pico W

Advice - Pi Pico in the Classroom

1. Get a kit with a box = $50 CDN (Abra)
2. Don't follow the Getting Started book
3. Teach Python basics early
4. Get good at using Thonny
   - alternate Python runtimes
   - be sure to connect I2C devices very well and use a test script to ensure connectivity
5. Don't assume that all sensors will work with Pico
   - Python library needed (supported)



Lots to work with in the classroom

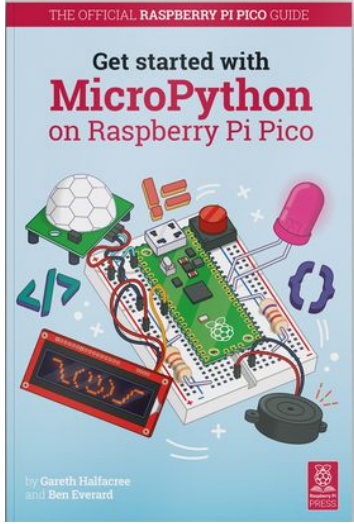


Raspberry Pi Pico - Overview

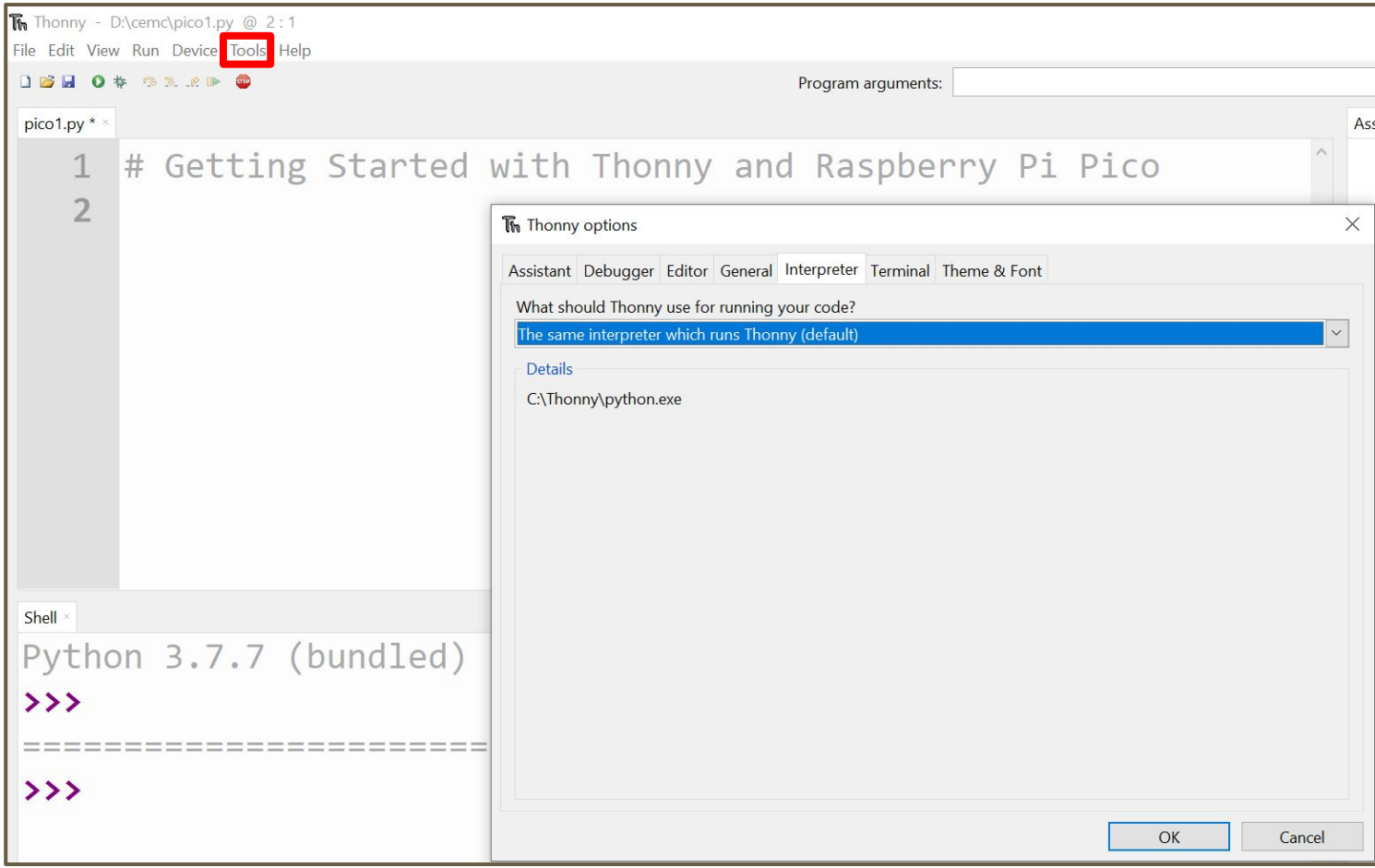
Pico is a Microcontroller and not a computer (SBC)

ARM based **dual core CPU** designed by the Raspberry Pi Foundation - **RP2040**.

Primary programing language is **MicroPython**.



Thonny


Thonny - D:\cemc\pico1.py @ 2 : 1
File Edit View Run Device Tools Help
Program arguments:
pico1.py *
Getting Started with Thonny and Raspberry Pi Pico
Shell
Python 3.7.7 (bundled)
>>>
==========
>>>
Thonny options
Assistant Debugger Editor General Interpreter Terminal Theme & Font
What should Thonny use for running your code?
The same interpreter which runs Thonny (default)
Details
C:\Thonny\python.exe
OK
Cancel

Thonny - Validate connection to Pico

Thonny - D:\cemc\pico1.py @ 2 : 1

File Edit View Run Device Tools Help

Program arguments:

pico1.py *

```
1  # Getting Started with Thonny and Raspberry Pi Pico
2
```

Shell

```
>>>
========================== RESTART ==========================
>>>
========================== RESTART ==========================
MicroPython v1.16 on 2021-06-18; Raspberry Pi Pico with RP2040
Type "help()" for more information. [backend=GenericMicroPython]
>>>
```

Thonny

Thonny - D:\cemc\10_neopixel_demo.py @ 8 : 9

File Edit View Run Device Tools Help

Soft reboot Ctrl+D
Upload current script as main script Ctrl+U
Upload current script as boot script
Upload current script with current name
Show device's main script
Show device's boot script
Close serial connection

Program arguments:

1-blink.py 2-ADC 26.py 4-digital-input.py 5_threads.py 6_interrupts.py 7_pwm_flash.py 8_pwm_potentiometer.py 9_rtc.py 10_neopixel_demo.py

```
1  #  opixel.py file on the Pi Pico
2  #  laz-r/pi_pico_neopixel
3  fr  eopixel
4  import utime
5  num_pixels = 8
6  pixels = Neopixel(num_pixels, 0, 0, "GRB")
7
8  while True:
9      for i in range(num_pixels):
10         pixels.set_pixel(i,(255,0,0))
11         pixels.show()
12         utime.sleep(.1)
```

Shell

```
Connection closed.
Select Run → Stop/Restart or press Ctrl+F2 to connect again.

========================= RESTART =========================
>>>
```

CS Pedagogy

1. Sensing Loop
    - Blocking code for Sensors / Actuators

2. Event based programming

3. Concurrent programming
    - Threads or Interrupt handling

Blinky - (onboard LED)

```
# Getting Started with Thonny and Raspberry Pi Pico

import machine
import utime

led = machine.Pin(25, machine.Pin.OUT)

while True:
    led.value(1)
    print("ON")
    utime.sleep(1)
    led.value(0)
    print("OFF")
    utime.sleep(1)
```

utime.sleep (n)	# n is seconds
utime.sleep_ms (m)	# m is milliseconds
utime.sleep_us (p)	# p is microseconds

Blinky - Event Based (GPIO 15 - external LED)

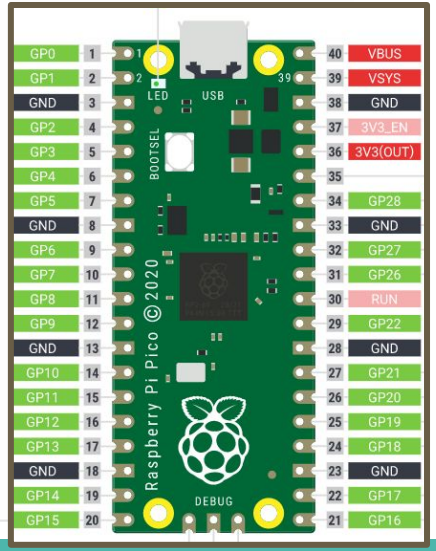
```
from machine import Pin, Timer
led = Pin(15, Pin.OUT)
timer = Timer()

def blink(timer):
    led.toggle()

timer.init(freq=2.5, mode=Timer.PERIODIC, callback=blink)
```

Digital Input and Output

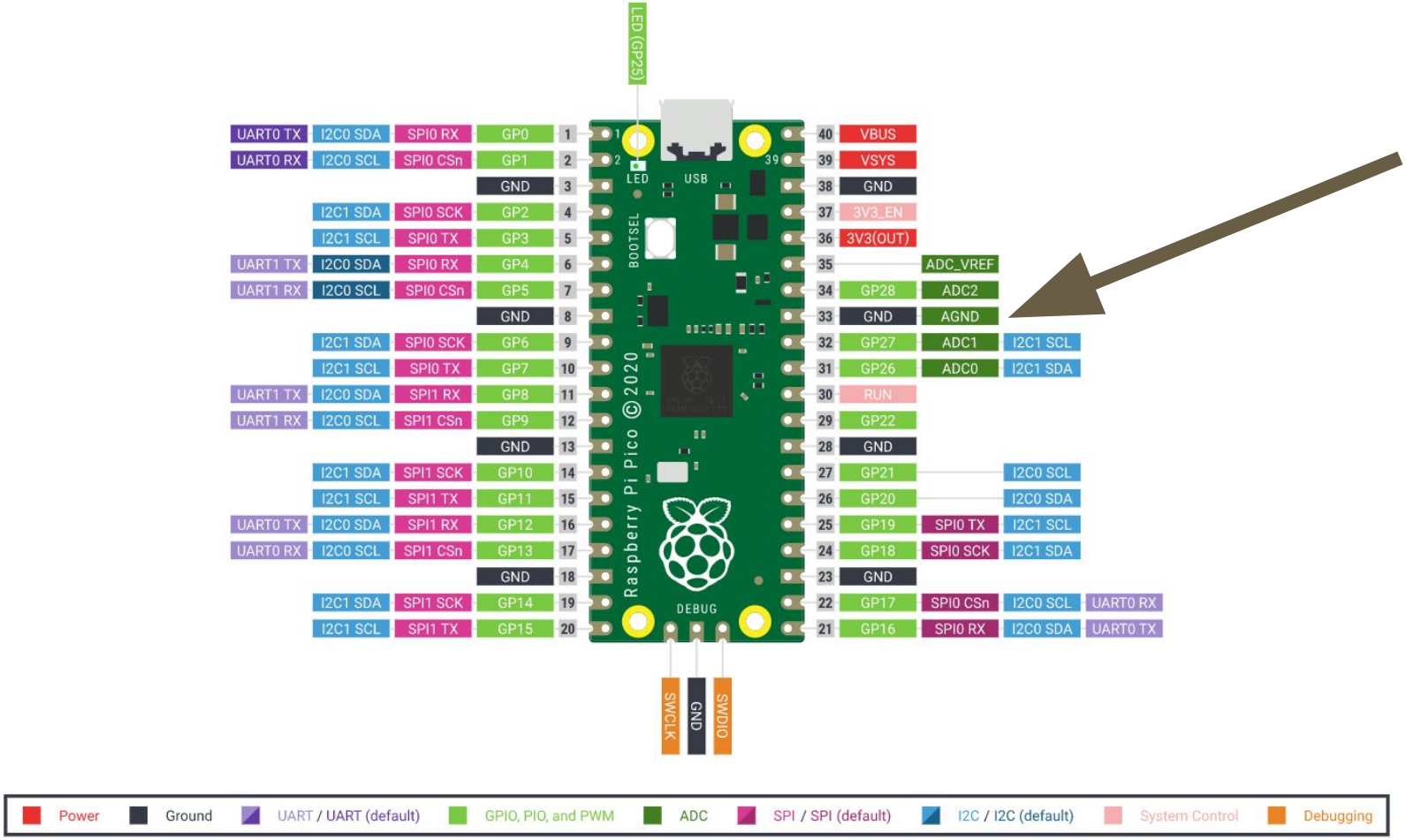
Behaviour - wiring

5-digital-input.py

```python
# Digital input using buttons
import utime
import machine

button = machine.Pin(14,machine.Pin.IN,machine.Pin.PULL_DOWN)
out = machine.Pin(15,machine.Pin.OUT)

while True:
    if (button.value()==1):
        print("pressed")
        out.value(1) # turn on LED attached to pin 15
    else:
        print("not pressed")
        out.value(0) # turn off LED attached to pin 15
    utime.sleep(.01)
```



ADC


LED (GP25)
UART0 TX I2C0 SDA SPI0 RX GP0 1
UART0 RX I2C0 SCL SPI0 CSn GP1 2
GND 3
I2C1 SDA SPI0 SCK GP2 4
I2C1 SCL SPI0 TX GP3 5
UART1 TX I2C0 SDA SPI0 RX GP4 6
UART1 RX I2C0 SCL SPI0 CSn GP5 7
GND 8
I2C1 SDA SPI0 SCK GP6 9
I2C1 SCL SPI0 TX GP7 10
UART1 TX I2C0 SDA SPI1 RX GP8 11
UART1 RX I2C0 SCL SPI1 CSn GP9 12
GND 13
I2C1 SDA SPI1 SCK GP10 14
I2C1 SCL SPI1 TX GP11 15
UART0 TX I2C0 SDA SPI1 RX GP12 16
UART0 RX I2C0 SCL SPI1 CSn GP13 17
GND 18
I2C1 SDA SPI1 SCK GP14 19
I2C1 SCL SPI1 TX GP15 20
40 VBUS
39 VSYS
38 GND
37 3V3_EN
36 3V3(OUT)
35 ADC_VREF
34 GP28 ADC2
33 GND AGND
32 GP27 ADC1 I2C1 SCL
31 GP26 ADC0 I2C1 SDA
30 RUN
29 GP22
28 GND
27 GP21 I2C0 SCL
26 GP20 I2C0 SDA
25 GP19 SPI0 TX I2C1 SCL
24 GP18 SPI0 SCK I2C1 SDA
23 GND
22 GP17 SPI0 CSn I2C0 SCL UART0 RX
21 GP16 SPI0 RX I2C0 SDA UART0 TX
LED
USB
BOOTSEL
Raspberry Pi Pico ©2020
DEBUG
SWCLK
GND
SWDIO
Power
Ground
UART / UART (default)
GPIO, PIO, and PWM
ADC
SPI / SPI (default)
I2C / I2C (default)
System Control
Debugging

Analog Input

ADC Module	GPIO Pins
ADC0	GP26
ADC1	GP27
ADC2	GP28

ADC - measures voltage on Pin

- resolution - **12 bits 0-4095**
  - transformed into 16-bit number (0-65535) in Python

- 4 Channel with 3.3V internal reference
  - General use ADC Channels are 0, 1, 2, 3
  - Actual pins are GPIOs - **GP26, GP27, GP28**

- ADC Channel 4 is an internal **temperature sensor**
  - Reference as ADC(4) object - see next example

Temperature Sensing (Lower Level than Micro:bit)

2-ADC-Temperature.py

```python
# Pico - measure channel 1 voltage in and temperature
import machine
import utime

adc = machine.ADC(26) # GP26 or channel 0
adc_temp = machine.ADC(4) # Channel 4 or temperature
factor = 3.3 / 65535

while True:
    reading = adc_temp.read_u16() * factor
    print ("Voltage is :", reading, "Volts", end = " : ")

    temperature = 27 - (reading - 0.706)/0.001721 # slope
    print("Temperature is", round(temperature,2), "degrees Celsius.")
    utime.sleep(1)
```

Shell

```
Voltage is : 0.7067292 Volts : Temperature is 26.58 degrees Celsius.
```

Analog Output

How to generate analog output using Pico?

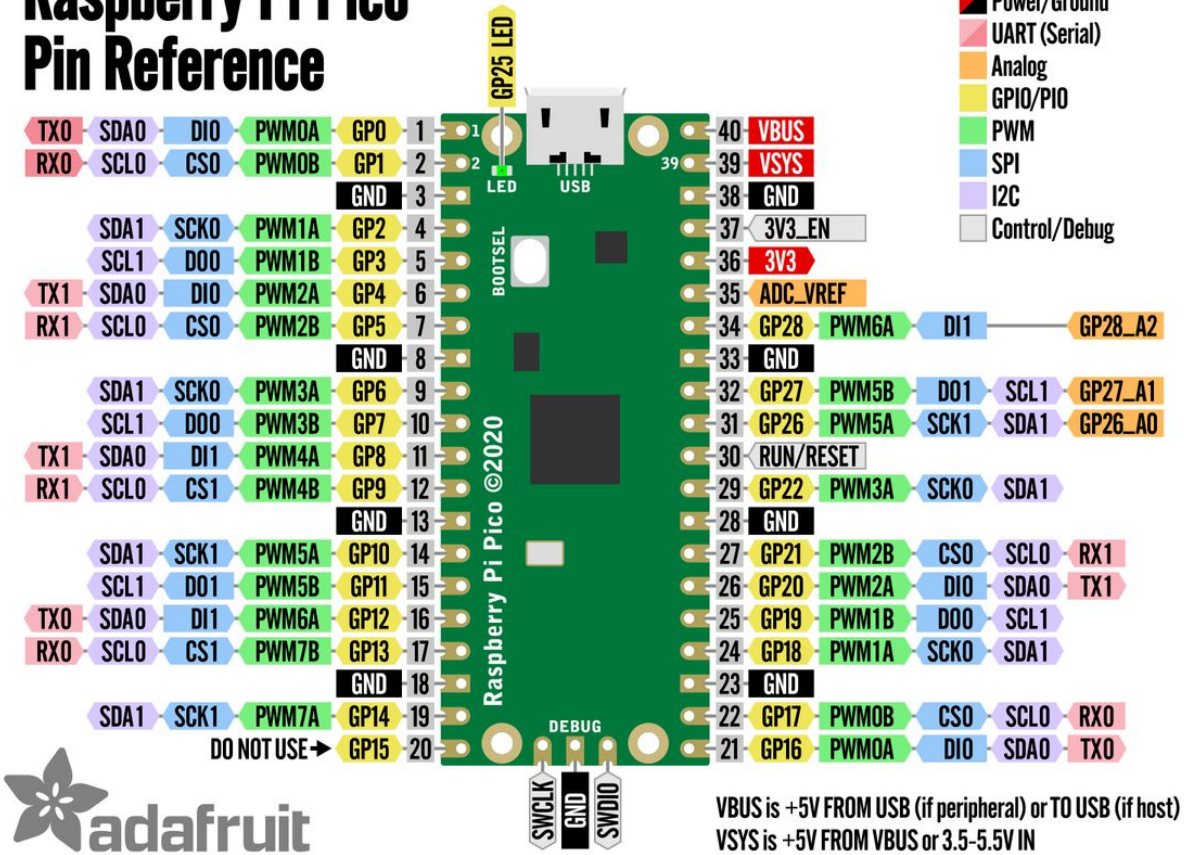
Pulse Width Modulation (PWM) Pin object

Frequency can be set between **7 Hz and 125 MHz**

Uses:

- Control **motor** rotation speed
- Control the frequency (tone) of a **speaker** output
- Control the rotation of a **servo**

Raspberry Pi Pico Pin Reference



```python
# Interrupt handling version
import utime
import machine

button = machine.Pin(14,machine.Pin.IN,machine.Pin.PULL_DOWN)
yellow_led = machine.Pin(15,machine.Pin.OUT)
green_led = machine.Pin(16,machine.Pin.OUT)

global pressed
pressed = False

def irq_handler(pin): # single parameter is required
    global pressed
    if button.value() ==1:
        if pressed == False:
            pressed = True
        else:
            pressed = False
    print (pin, end= ":")
    print (pressed)
button.irq (trigger=machine.Pin.IRQ_RISING, handler=irq_handler)
```

```python
while True: # main thread
    yellow_led.value(1)
    utime.sleep(.5) # half second delay
    yellow_led.value(0)
    utime.sleep(.5) # half second delay

    global pressed
    if pressed:
        green_led.value(1)
        utime.sleep(.5) # half second delay
        green_led.value(0)
        utime.sleep(.5) # half second delay
```

Python Coding with Interrupts

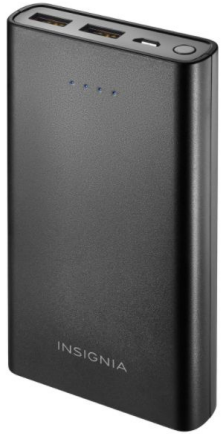
```python
# Multithreaded application
import utime
import machine
import _thread

button = machine.Pin(14,machine.Pin.IN,machine.Pin.PULL_DOWN)
yellow_led = machine.Pin(15,machine.Pin.OUT)
green_led = machine.Pin(16,machine.Pin.OUT)

global pressed
pressed = False

def button_handler():
    global pressed
    while True:
        if button.value() ==1:
            pressed = True
        utime.sleep(.01)

_thread.start_new_thread(button_handler,())
```

```python

while True: # main thread
    yellow_led.value(1)
    utime.sleep(.5) # half second delay
    yellow_led.value(0)
    utime.sleep(.5) # half second delay
    global pressed
    if pressed:
        green_led.value(1)
        utime.sleep(.5) # half second delay
        green_led.value(0)
        utime.sleep(.5) # half second delay
```

Python Coding with Threads - Using Second Core of CPU

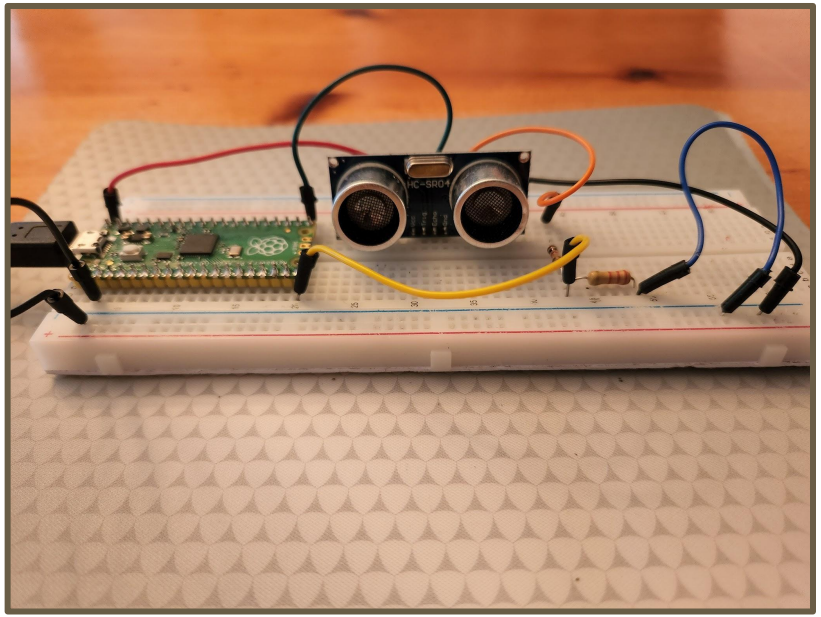
Pico - Data Logging - Onboard RTC

```
# Pico - measure channel 1 voltage in and temperature
import machine
import utime
adc_temp = machine.ADC(4) # Channel 4 or temperature
factor = 3.3 / 65535
rtc = machine.RTC() # create RTC object
rtc.datetime((2021,08,19,3,15,30,0,0)) # August 19, 3:30pm - Thursday
file = open("temps.csv","w")

for i in range(60):
    reading = adc_temp.read_u16() * factor
    temperature = 27 - (reading - 0.706)/0.001721 # slope
    dateTime = rtc.datetime() # current time

    CSV_FMT = "{:04d}-{:02d}-{:02d},{:02d}:{:02d}:{:02d},{:.2f}"
    file.write((CSV_FMT.format(dateTime[0],dateTime[1],dateTime[2],
                    dateTime[4],dateTime[5],dateTime[6],temperature)))
    file.write("\n")
    file.flush()
    utime.sleep(1)
file.close()
```



Electronic Measuring Device

```
from machine import Pin
import utime
trigger = Pin(16, Pin.OUT)
echo = Pin(15, Pin.IN)

def ultra():
   trigger.low()
   utime.sleep_us(2)
   trigger.high()
   utime.sleep_us(5)
   trigger.low()
   while echo.value() == 0:
       signaloff = utime.ticks_us()
   while echo.value() == 1:
       signalon = utime.ticks_us()
   timepassed = signalon - signaloff
   distance = (timepassed * 0.0343) / 2
   print("The distance from object is ",distance,"cm")

while True:
   ultra()
   utime.sleep(1)
```



Pi Pico - Project opportunities

Opportunities

- Systems Monitoring / Data Logging and Analysis
  - **limited standalone usage compared to a full Raspberry Pi with Phidgets attached**

- Integrated Systems
  - design and build engaging projects using **discrete** or **integrated sensors (i2C)** - Python libraries
    - eg. Alarms, Electronic measuring device
- **Robotics**
  - **pair sensors with motors and motor controllers to design and build robots**
    - **eg. line following robot**

Benefits

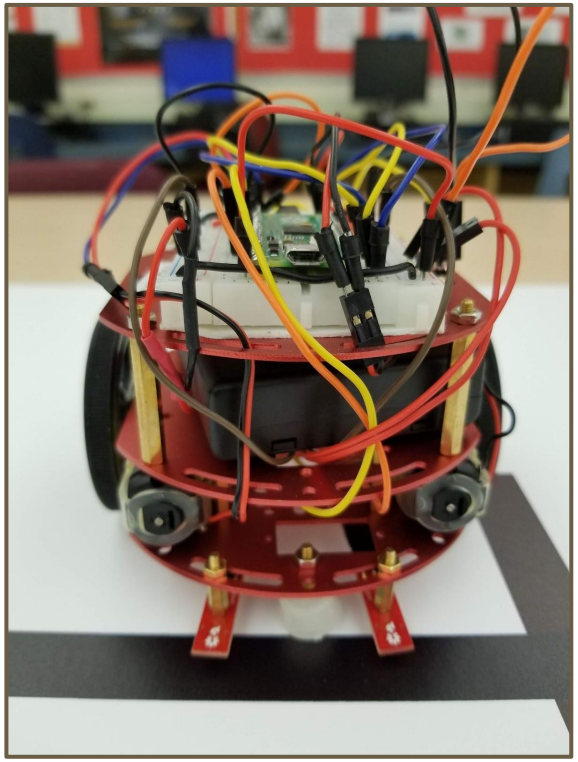
- Ideal platform to learn digital and analog electronics
- Python is a core language used for programming (instead of C for Arduino)
  - More powerful device than most standard Arduino boards like the Uno R3
- Low power requirements for battery operated projects - Power with 5V source

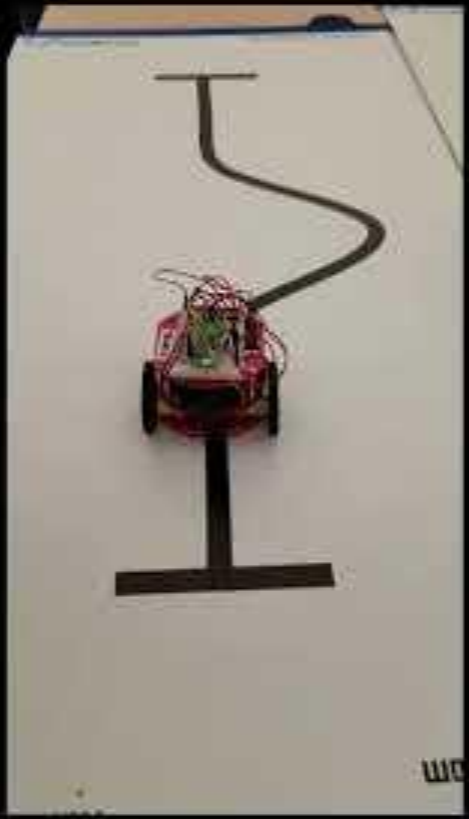
Project Ideas for Pi Pico using Python

Line Following Robot

Computing concepts:

- modularity
  - use of functions for driving and turning
- sensing
- state management





Robot Report - Student Assessment

List of Materials

Locate the links to the materials used to build your robot - you will be able to find these from **Abra Electronics**

- Pi Pico with headers
- Motor Controller
- Analog Redbot Line Sensor (SparkFun)
- Battery pack
- Low Drop Out (LDO) 5V Voltage Regulator
- Robot platform kit with wheels
- Additional Robot platform plate

Software Program

Include a copy of your final software program used to control the robot on the race course.

Reflections

Complete the following sentences

1. The most difficult challenge while **building the robot** was…..
2. The most difficult challenge while writing the **software** to control the robot was…..

Answer the following questions

1. Describe how you modified the hardware or software to ensure the robot followed the line **consistently**.
2. Describe how you modified the hardware or software to **optimize the speed** of the robot.

Thank you

Will you be adding a Physical Computing Unit to your Computer Science Courses?