Names ____

INTRODUCTION

About 15 years ago a new term was coined, *physical computing*. Let me give you the Wikipedia introduction:

> **Physical computing** involves interactive systems that can sense and respond to the world around them. While this definition is broad enough to encompass systems such as smart automotive traffic control systems or factory automation processes, it is not commonly used to describe them. In a broader sense, physical computing is a creative framework for understanding human beings' relationship to the digital world. In practical use, the term most often describes handmade art, design or DIY hobby projects that use sensors and microcontrollers to translate analog input to a software system, and/or control electro-mechanical devices such as motors, servos, lighting or other hardware.

The scientific terminology for this is *interfacing* or *instrumenting* – connecting a computing device to sensors (input) and controllers (output.) If you think about doing experiments you can break the process into changing parameters, and measuring what happens.

Today’s lab starts you down the road to designing and building experiments.

GETTING STARTED WITH INTERFACING

Start working through Ch. 3 *Physical computing*. Here are some notes. Please read them as you go.

- **Importing**. Python, like most programming languages, is a general purpose language that people use for a wide variety of purposes. So the basic language leaves out commands that are specialized. Like most languages, there is a way to extend the languages for specific purposes. In python this is done by *importing modules*. All other languages have a way to do this. For example, in c++ libraries are add by the **`#include`** statement.

  It is crucial for you to learn how importing works in python, and what modules are available for you to import. For example, to use common mathematical functions you import the **`math`** module. How do you find out what is in a module? After you import a module you can use the **`help`** function. For the **`math`** example, if you type **`help(math)`** you will get a list of what is in the math module: things like **`pi`**, **`sqrt`**, **`sin`**, **`cos`**, **`exp`**, and more. I use this all of the time You should learn to do it too!

  To get a list of all of the available builtin modules type **`help(‘modules’)`**. You will learn how to use many of the modules in this course.
- **Pin Numbering**. There is a difference between the *physical* pin numbers and the *logical* pin numbers.
  - The physical numbers start at 1 for the top left pin, and run to 20 on the lower left. Pin 21 is on the lower right corner and the pin numbers run up to 40 in the upper right corner.
  - When programming you always use the logical pin numbers. You have to look at the Pico W pinout chart to find these because they are not consequtive. Pin GP0 is at the top left and GP15 is at the lower left. There are four ground, GND, pins scattered along the side.
- **Challenge 1** (loc. 660) *Resistor color code*.
  - Give the color codes for 100 Ω, 5.6 kΩ, 330 kΩ, and 1.5 MΩ resistors.

Names ____

- ► Find the value and precision of the following resistors: brn-blk-yel, ora-ora-red-gold, gry-whi-brn-silver.

Chapter 4 – Physical Computing with Rapsberry Pi Pico

Move on to this chapter and work through it.

- **Challenge 2** (loc. 741) *Longer Light-Up*. Use the **`led_onboard.value()`** function to make the LED stay on longer, then shorter. Finally change your code so the LED is on half as long as it is off.
- **Challenge 3** (loc. 825) *Multiple LED's*. Wire up three external LED's of different colors. Program them to turn on and off successively and continue that pattern.
  - ► **Note**: In the real world, when a switch opens or closes, it actually might jump 4 or 5 times before it settles on the new state. This is because two metal surfaces aer rubbing against each other. The solution in software is to wait for a brief amount of time before declaring that the button changed state. Here is one solution:

```
def wait_pin_change(pin):
    # wait for pin to change value
    # it needs to be stable for a continuous 20ms
    cur_value = pin.value()
    active = 0
    while active < 20:
        if pin.value() != cur_value:
            active += 1
        else:
            active = 0
        sleep(0.001)
```

  You might use this code like below:

```
while True:
    wait_pin_change(button_pin)
    LED.toggle()
```

  - ► Write your program using the **`wait_pin_change`** function for key presses (and releases!)
- **Challenge 4** (loc 908) *Building It Up*. Work through the challenge, then wire one external button and two LED's, and program things so you can turn each LED or both at the same time, or turn both off by pushing the button.

Chapter 5 – Traffic Light Controller

Note: The topics of threads, introduced on loc. 1000, is actually an advanced topic. You have to be careful using threads because getting them to communicate with each other is a tricky and subtle issue.

- **Challenge 4** (loc. 1099) Add a second button that operates like the first so a pedestrian on either side of the road can trigger the walk.

Chapter 6 – Reaction Game

This chapter introduces interrupt routines, a very important topic in interfacing a microcontroller to the real world. You will use interrupts in other projects later in the course.

Work through and build the Reaction Time games in this chapter. This is a challenging activity! Follow directions carefully.

- **Challenge 5** (loc. 1268) Make the modifications suggested in the *Challenge* box.

Names ____

- **Challenge 6** (loc. 1337) Make the three person modification and print out how fast the first person’s reaction time was.

Names ____

MATERIALS

- Red, yellow, and green LED’s
- Three 220 Ω resistors
- Three four-leg button
- Buzzer
- Several male-male jumper wires