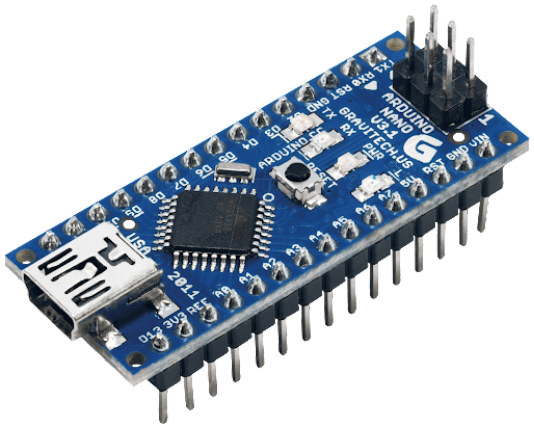
IoT Lab -1

A big movement over the past few years has been to integrate all of our things which are not computers to those things which are computers for ease of use and control on more aspects of our lives than just the traditional keyboard, mouse, and screen. Known as Internet of Things (IoT) the first step of successfully controlling and communicating with household appliances from a computer is the ability to toggle them on and off and take sensory readings about the real world to know when to toggle the appliance on and off.

The two most common types of tools used to achieve this are either by using a microcontroller or a single board computer (SBC). You could think of a microcontroller as essentially a single purpose computer, they don't have an operating system to react to numerous input stimuli, they have a single script that will run over and over again until the power is cut off (most often written in C++ or a related language). An SBC on the other hand is a small computer like we would be used to, though it's essentially just the motherboard that we get, everything else would still need to be connected and set up as needed, an SBC runs an operating system like the computers we are used to and process instructions as a subroutine that only gets a fraction of the available resources. What these two type of devices have in common though, and what makes them useful for IoT projects is the inclusion of General Purpose In Out (GPIO) connections. Along with some of the regular connections like we may be used to: USB, Wi-Fi, Bluetooth, etc. Microcontrollers and SBC's typically include either pins (or pads that pins and wires can be soldered to on the printed circuit board) which can be controlled from the processor to send or receive electrical signals as needed to monitor and control other appliances in the real world.

Pictured below are a couple examples of such devices (a Raspberry Pi 4 SBC on the left and a Arduino Nano Microcontroller on the right), note the silicon block which you may already know is the component used for processing, and the many small GPIO connection pins.



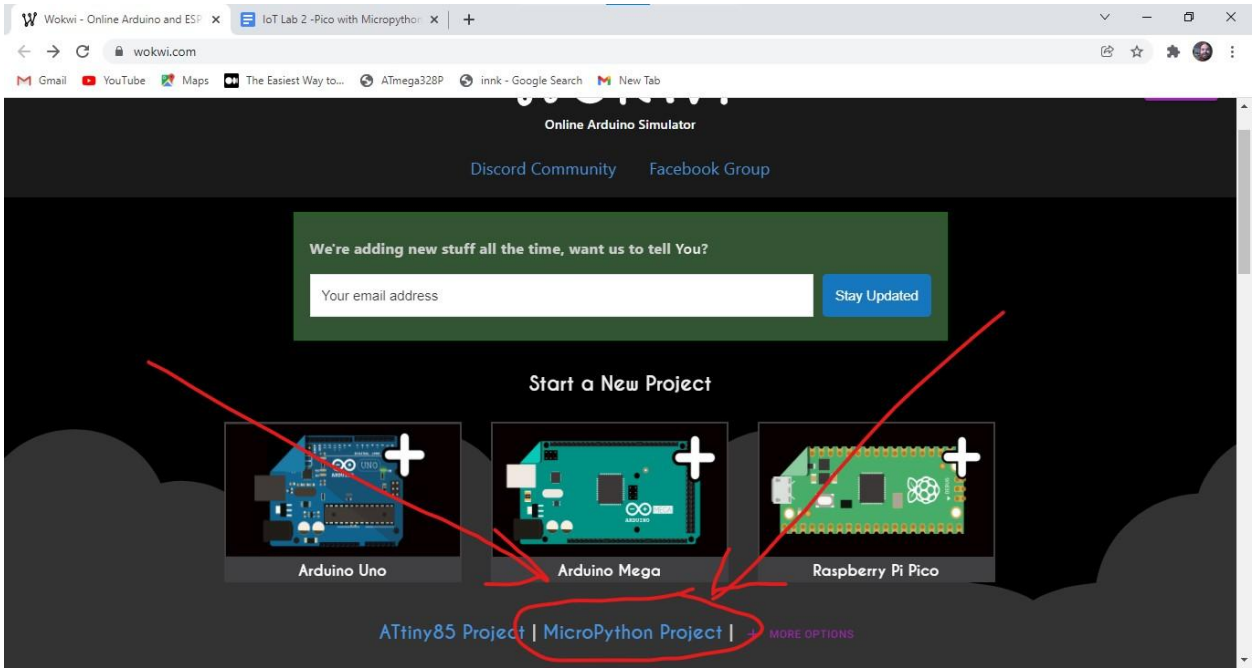
By connecting those GPIO pins to other electrical circuits around our homes we can control things on those electrical circuits from computer scripts. The SBC pictured there runs a Linux operating system, so we would just develop a program and run it just as we would for our desktop or laptop computers with a Linux operating system, on the microcontroller pictured above we simply write the program we want the device to continuously run and load it on the chip so that it just runs over and over again whenever power is applied to the device (so stopping the script similarly only involves removing power to the microcontroller).

In this lab we will take a look at using a microcontroller to turn on and off an LED at the press of a button to study the concepts of how a microcontroller controls the real world. The only difference between toggling an LED with a button here and running our coffee maker from voice commands set up on our phones across the internet is how we connect the devices together, we're getting some sort of input (push the button here, or talk to the phone) and controlling some sort of output (toggle power to the LED or toggle power to the coffee maker –this actually involves stepping up the voltage: using the low power GPIO pin to flip a switch handling much larger voltages: 3.3 or 5 volts on the SBC or microcontroller and 110 volts in our kitchen).

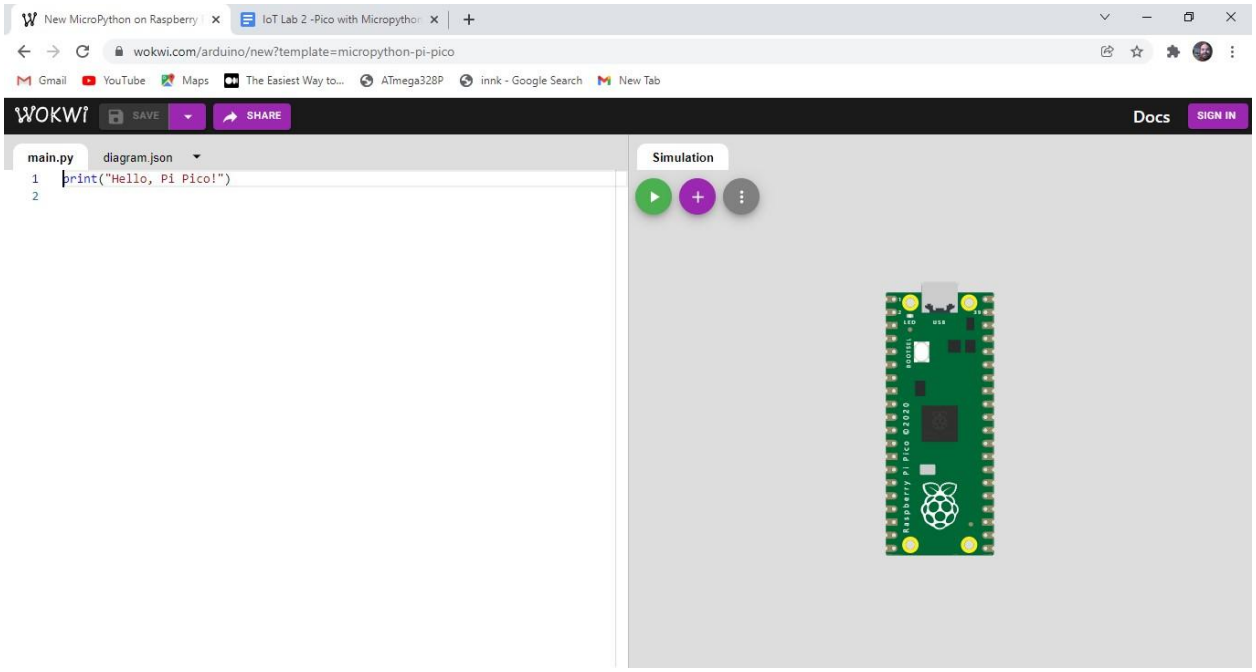
We'll be using an online simulator for microcontrollers in this lab, but just before the pandemic I purchased a number of microcontrollers and components (along with already having a huge assortment of electrical components personally) to do labs like this in the classroom, or the school has a number of Raspberry Pi kits, so if you live near campus, or Clarksville, we can try to find a time and place to meet up and perform this lab with real physical components instead, just let me know if you'd like to try that.

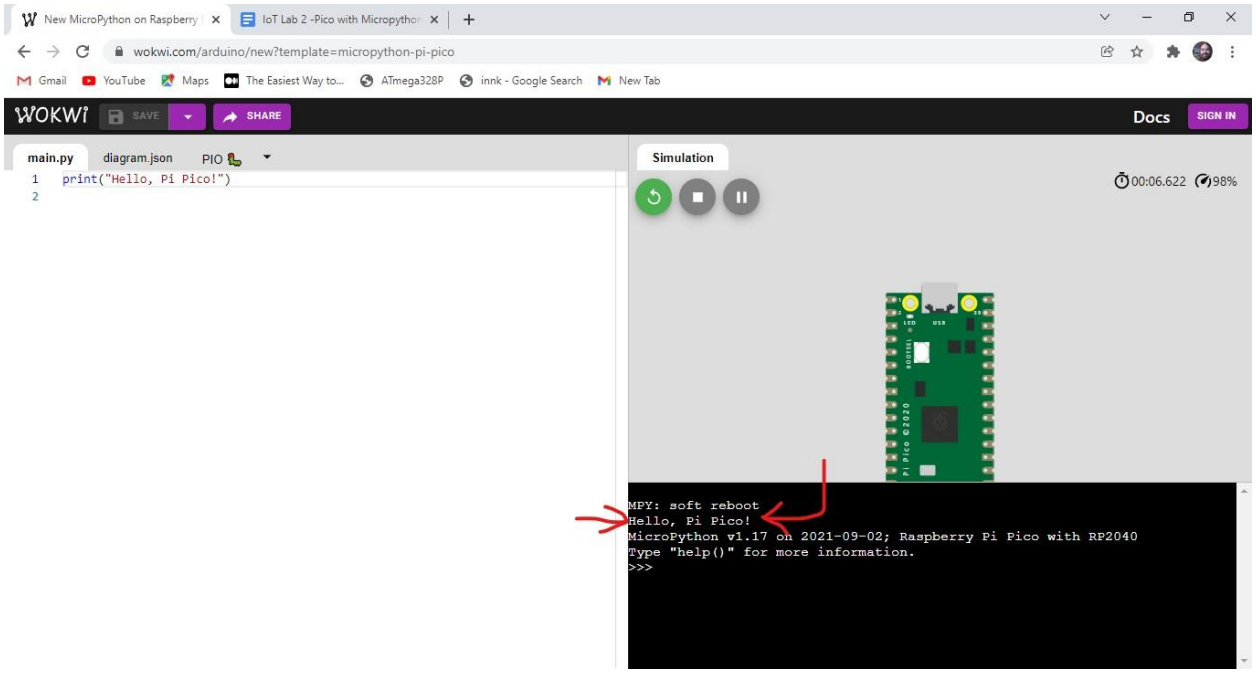
While Raspberry Pi is well known for SBC devices, early in the year 2021 the Raspberry Pi foundation released a microcontroller known as the Pico. They developed their own chip to run it, the RP2040 (which is open source like everything they do so the chip can be found on boards of much higher quality Arduino, Adafruit, and ESP just to name a few –check out the Arduino RP2040 Connect if you find yourself interested in these things). The numbers are impressive by microcontroller standards, but what concerns us here is that it is programmed with the python programming language, which makes it much easier to work with than microcontrollers typically have been (i.e. coded in C++, though you would need to code a Pico with C++ or assembly language to take full advantage of its powers python is still a slower language here just like it is on our regular computers).

To start this lab navigate to https://wokwi.com/ and select the "MicroPython Project" option (see image below).



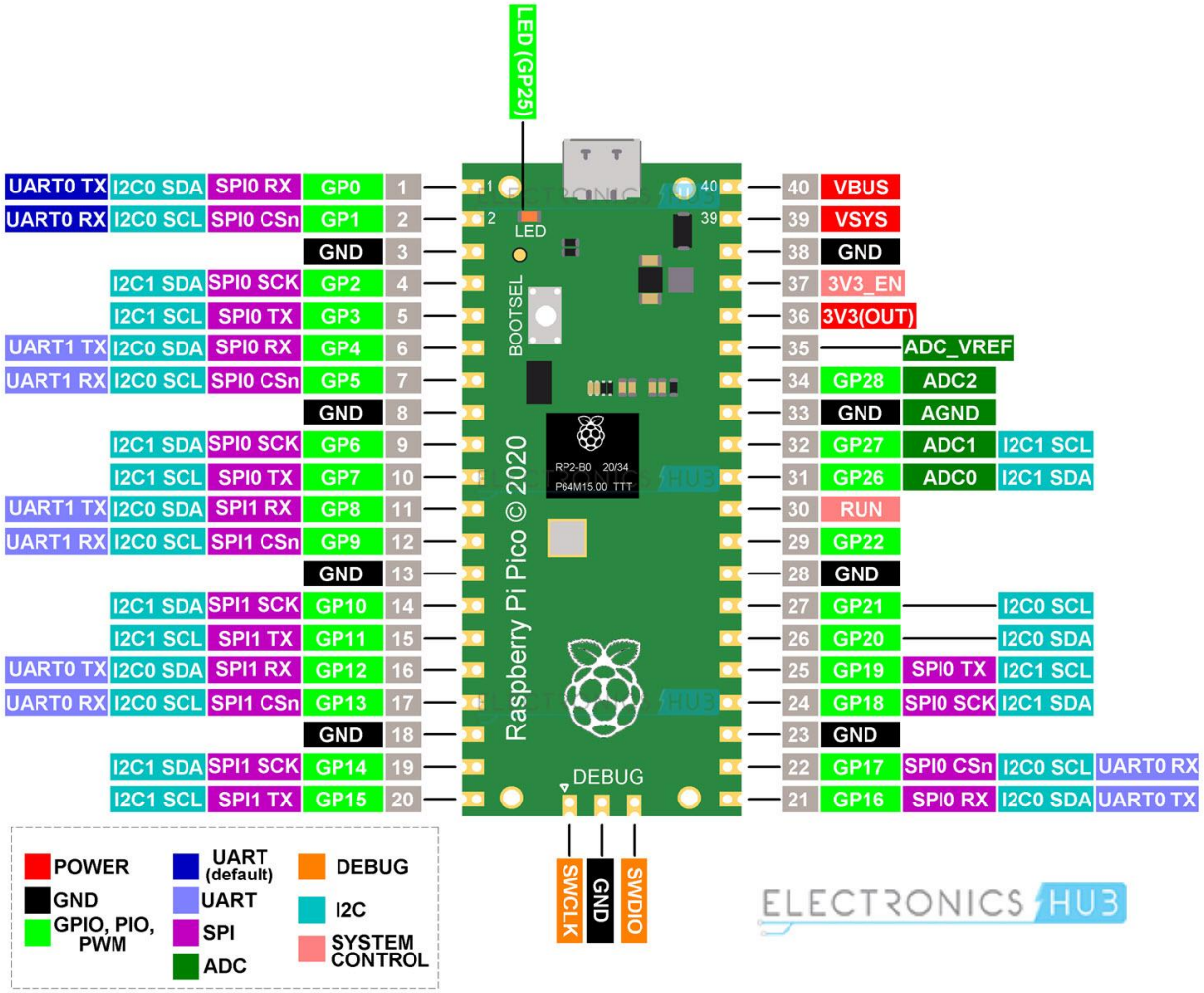
You should now see the page pictured below, to the left is the area where you will write your python code in the main.py file, the JSON file is actually what is represented graphically on the right hand side feel free to have a look but this is much easier to do with the pictures than type up the JSON data for, but we'll have to configure our components with the JSON data so we actually need both. On the right hand side the green button will run it, the purple button will bring in new components to connect to the device, and the ellipsis button is for layout options. Feel free to go ahead and hit run with the "hello Pi Pico" thing there to see what the serial output will look like.





The hello world of microcontrollers is to make an LED flash, so let's start there. Pictured below is the pinout diagram for a pico, you'll want to refer back to this while writing your code, but there's a lot of info here beyond the scope of an intro lab: UART is the standard Serial communication (USB and Bluetooth for example), SPI and I2C are the standards for parallel processing devices, ADC is analog to digital conversion so we can get measurements of real world phenomena into a form we can use when programming machines, but what we need for

this lab is the innermost set of labels: lime green GPIO pin numbers which will be used in our script, chip connection 36 -the red 3.3 volts out to power our peripherals, and the many available ground connections to complete the circuit back to.

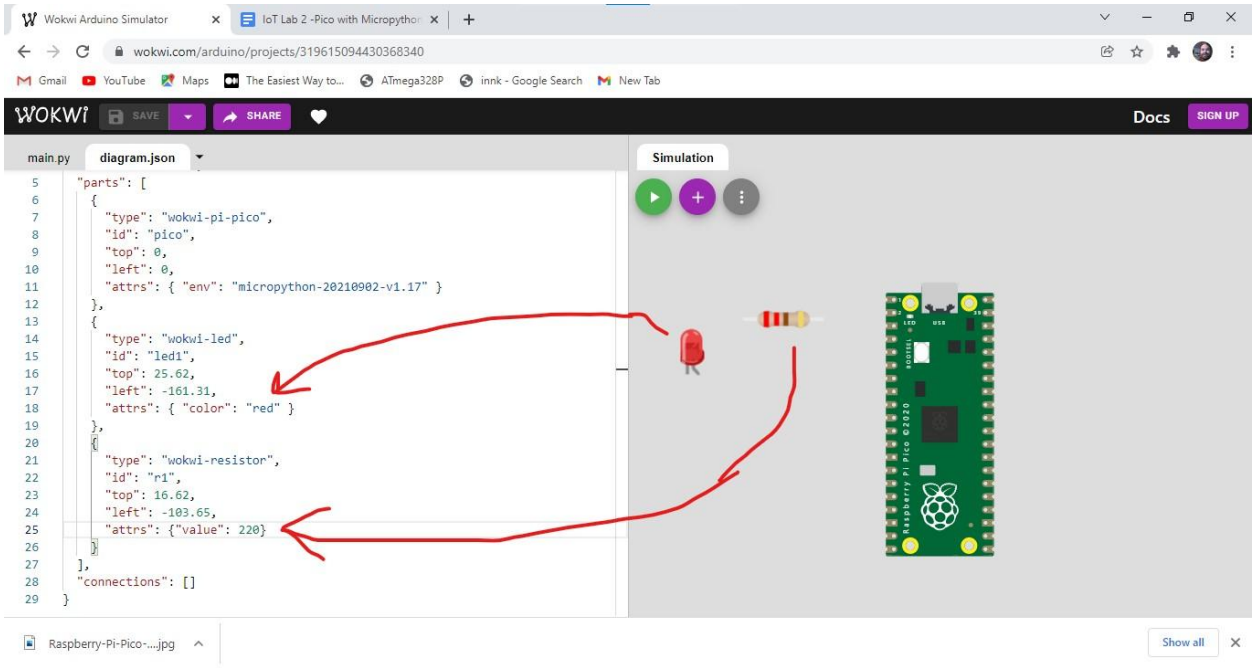


So let's hook up an LED and get it flashing. Click the new component button and search for an LED. You should see a red one appear, take a quick peek at the JSON file and you'll notice a data entry for an led with the color attribute set to "red" what do you think happens if you change what the color is set to there… give it a try with a few colors you expect LED's are available in. Now hooking up an LED directly would run far too much current through it and burn it out quicker than you can say "ahh… shucks…" so we need to try and keep that under 20 milliamps.

Add in a resistor component, and on the JSON file, find the resistor's data (when I was taking pictures this one came in as one long line, you can put line breaks after the commas in a JSON file to make it look a little nicer). Locate the spot in the code that says "Attrs: {} " and add the key

value pair “value”: 220. We use Ohm’s law to calculate current $I = \frac{V}{R}$ where I is the current, V is the voltage, and R is the resistance. We know we have 3.3 volts, and $I = \frac{V}{R} = \frac{3.3}{220} = 0.015$, 15 milliamps is a safe current to run an LED at.

Before forming the connections make sure yours looks kind of like this image



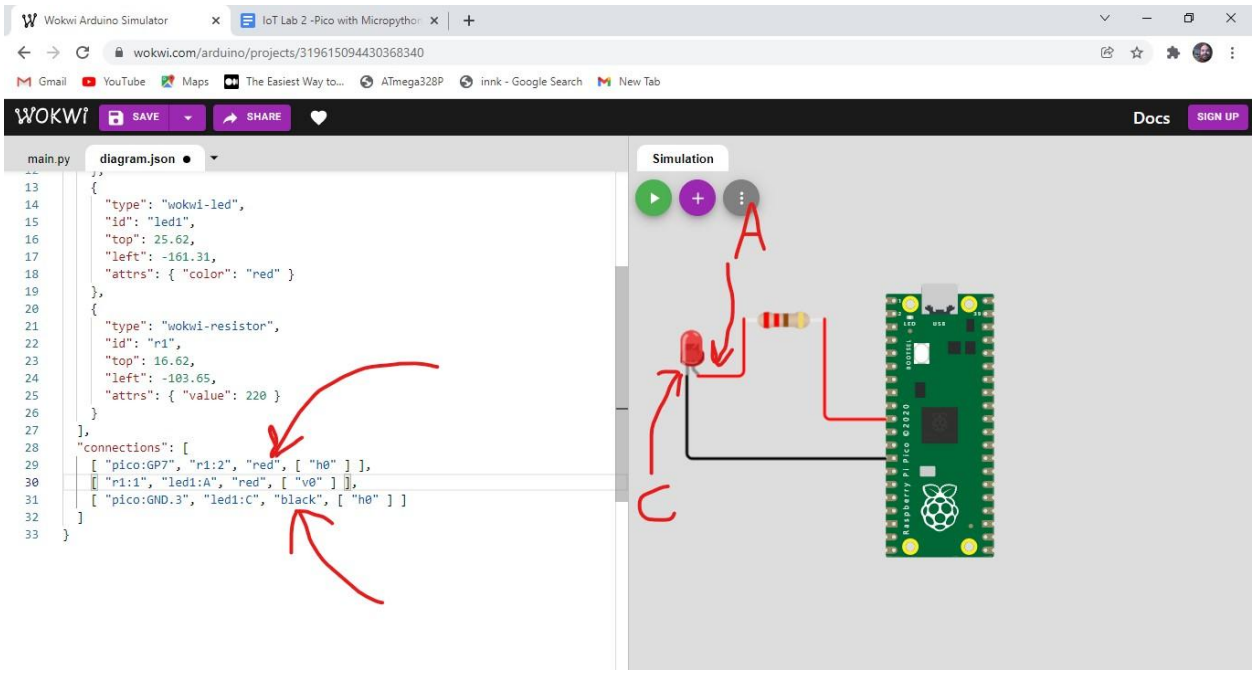
Notice how the Pinout Diagram above shows the top left two pins with a different color for the UART label, this is because these are the default UART, in other words these are the same connection as that USB port we can see on there, so you probably don’t want to use GPIO 0 or GPIO 2 since those can’t be used simultaneously with the USB port. Pick any other GPIO pin and hover your mouse over it, you’ll see the name of the connection pop up. To simulate a wire click on that connection and then click on the spot you want the wire to go to.

At this point take notice how the JSON file shows the connections as lists/arrays of strings, which in JSON is pretty much identical to a list of strings in python if you would prefer to type some of these things up. In fact you’ll probably notice that JSON objects are very similar to dictionaries in Python, and the main noticeable difference is that we set functions/methods or other objects as the value differently in JSON and Python. Those won’t show up here though so we might as well act like we’re talking about a Python Dictionary for the rest of this lab.

The voltage will be coming from the GPIO pin you chose, go through the resistor and LED, and return to one of the ground pins. Note that LED’s are polarized so the longer side (bent lead in

these pictures, known as the “Anode”) is the positive side and the short (straight lead in these graphics, known as the “Cathode”) connects to the ground side.

After making the connections it should appear somewhat similar to the image below except for the choice in GPIO and Ground pin. I chose GPIO 7 and GND 3, I also changed the default color of the wire to keep “hot” and “ground” straight with just a glance.



Now we’re ready to code up the control code. Switch the editor back to the python file and let’s start by importing a library to keep track of timing: Raspberry Pi foundation says to use the “utime” module instead of the regular time module but they have the same methods so the names should seem familiar at this point in your experience with Python. We will also need to import the “machine” library which has methods for the pins on the Pico. For the lab here I am just going to do the import all notation and hopefully it’s clear what comes from utime and what comes from to the machine via context.

```
from machine import *
from utime import *
```

Now to set up our connections we’ll use a pin object, and pass in arguments for the GPIO pin number we used and whether it is to be used as an input or an output.

```
myLED = Pin(7, Pin.OUT) ## you may have chosen a different pin than I did
```

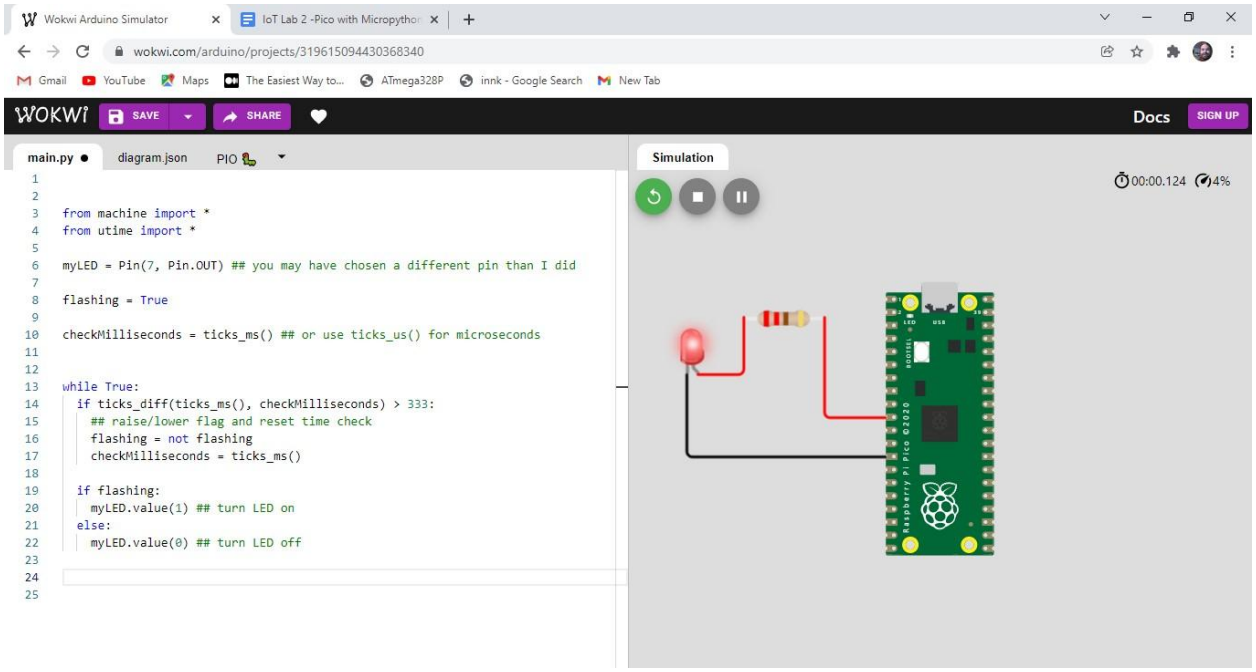
The utime module has a sleep method, but in larger situations we probably don't want to just shut down the device temporarily preventing other pins from doing stuff as well, so it's better form to take a measurement of time, and then just check how much time has passed since that measurement. I'm going to set a boolean flag to check later to keep the code organized, but you could very easily just nest an if-else structure where this flag is changed instead.

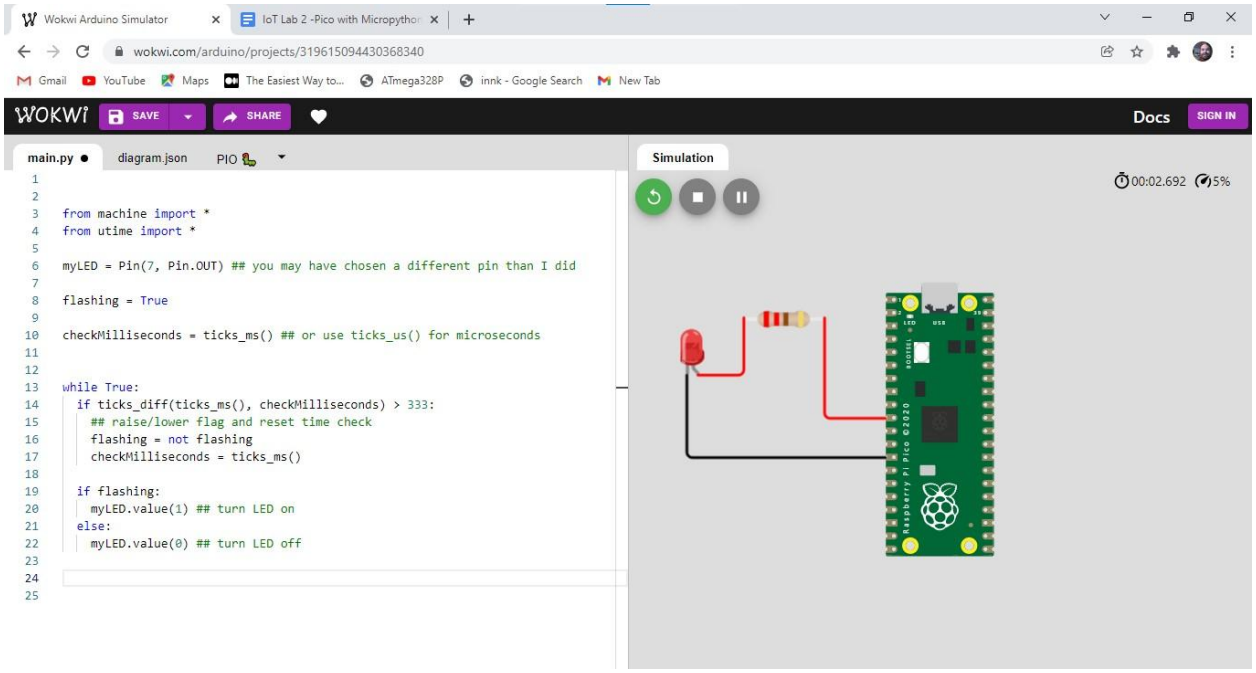
```
flashing = True

checkMilliseconds = ticks_ms() ## or use ticks_us() for microseconds
```

That's the setup we need, so now we're ready to go into an INFINITE loop and adjust whether the LED is on or off using a selection structure checking if enough milliseconds have passed, the ticks_diff() method from the utime module just subtracts them but with the method they don't have to be in the same units of time (i.e. one could be in days and the other in microseconds, that would take a bunch of conversions just to subtract the two values).

```
while True:
  if ticks_diff(ticks_ms(), checkMilliseconds) > 333:
    ## raise/lower flag and reset time check
    flashing = not flashing
    checkMilliseconds = ticks_ms()

  if flashing:
    myLED.value(1) ## turn LED on
  else:
    myLED.value(0) ## turn LED off
```

You should see the LED flash on and off when you hit the run button.





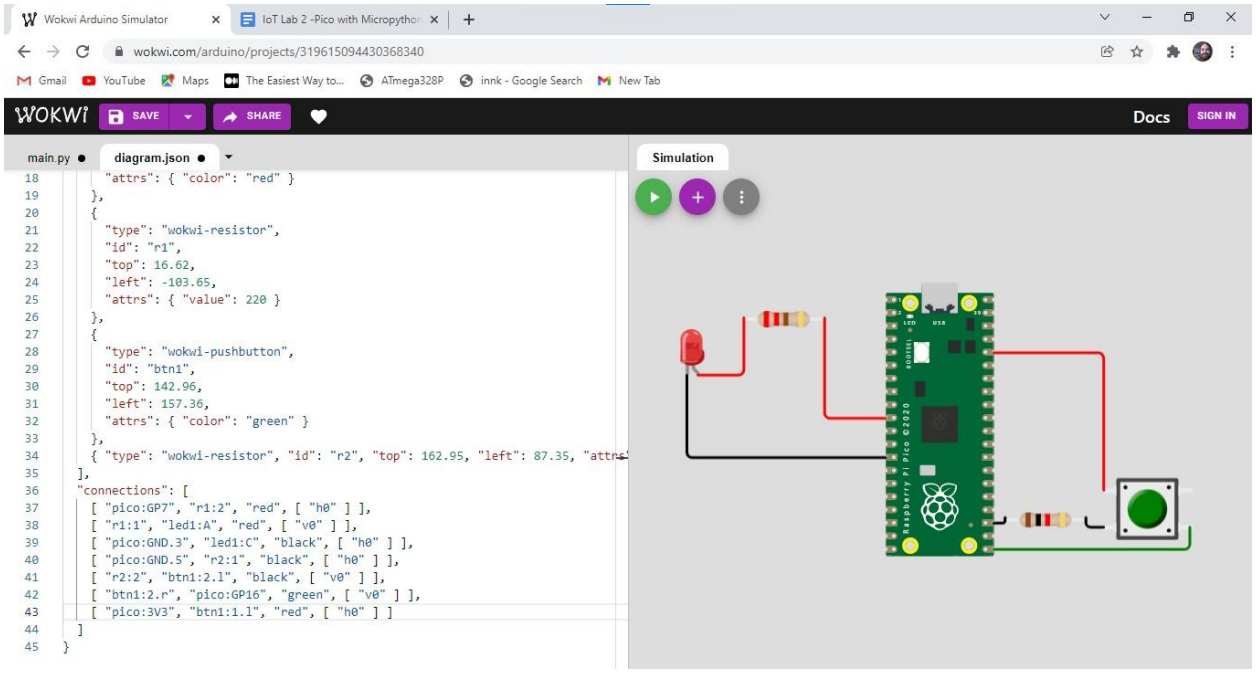
So that's the basic idea of output with GPIO pins, and if we can write code that can turn on and off an LED we can write code that can turn on or off any electrical device.

Let's round out this lab by adding a push button for input so that the light comes on when we push the button and goes off when we release it.

Click the purple add new part button and find a push button component. You'll notice that the push button has two pins sticking out each side. When the button is pressed a connection is formed between the sets of pins, otherwise the pins on the same side are connected to each other.

Connect the Pico's 3.3 volt output pin to one of the pins on the push button. Then add a resistor (set the value to 10000, ten thousand) on the same side as the connection to the hot line. This is known as a pull down resistor, when the button is not being pressed it will allow the electricity to just continue on its path back to ground, but when the button does get pressed the resistance on the other side will be less than the 10,000 Ohms going straight back to ground, so the electricity will flow that way instead ("electricity takes the path of least resistance"). On the pin straight across from the pull down resistor add a wire heading back to one of the GPIO pins on the Pico.

If necessary adjust the colors of the wires, and that "h0" and "v0" element of the connection array defines horizontal or vertical as it's drawn coming out of the place it connects to the component. Compare your connections and JSON to the image below.



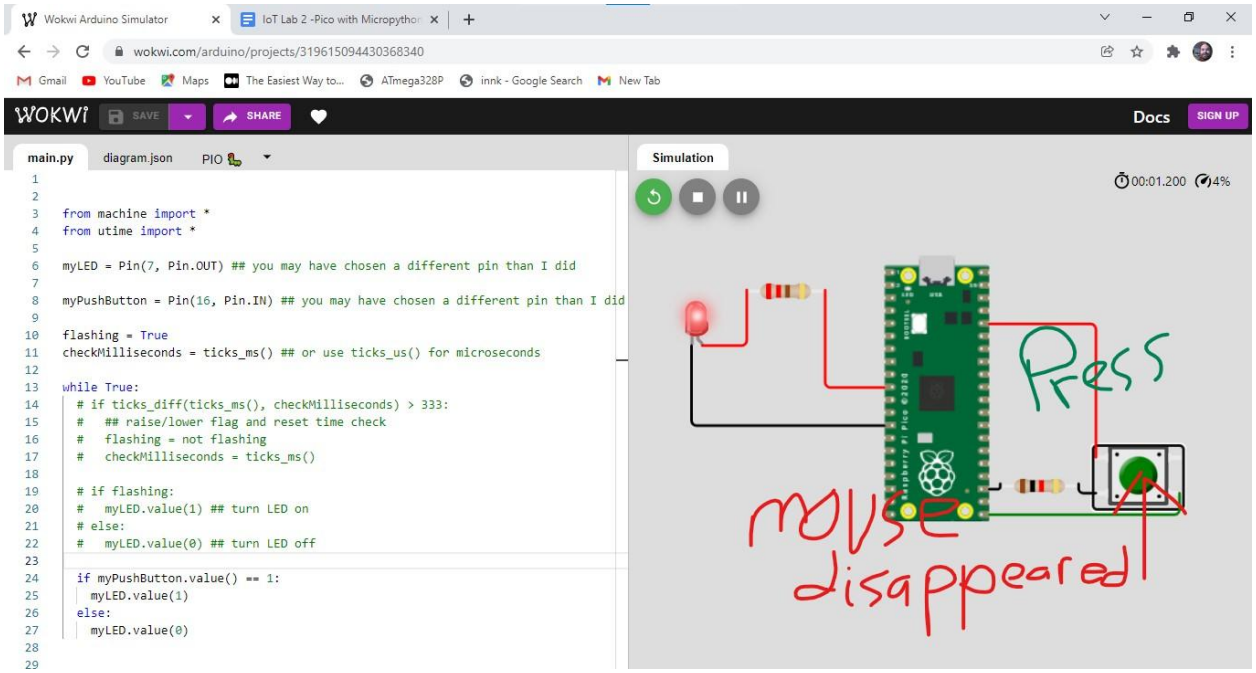
To code this up we'll initialize the pin in a similar fashion, except we want an input to read whether there is a signal on that line or not.

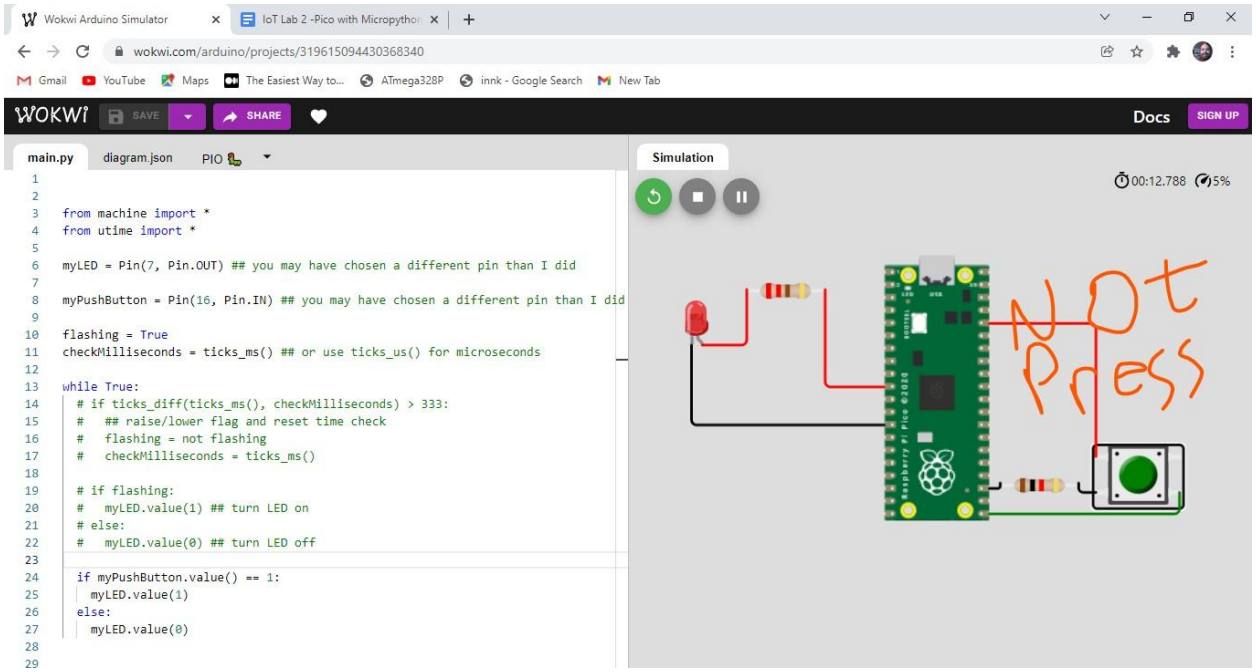
```
myPushButton = Pin(16, Pin.IN) ## you may have chosen a different pin than I did
```

We can go ahead and comment out anything that has to do with timing, and replace it with a selection structure that checks whether the signal on that input pin is a 1 or a 0. Notice that the

value method does not take in any arguments when reading but takes an argument when writing… Do you see anything that would make this code a single line long?

```
if myPushButton.value() == 1:
  myLed.value(1)
else:
  myLed.value(0)
```





So that's it for these instructions, play around with this a little, check out the other parts that are available for connection and see if you can get something else working from these basic building blocks, or more lights on different timers or something. Or maybe see if you can make that python script more efficient, there are several ways it could be improved upon. When you're ready take a couple screenshots showing your circuit in action and submit that to receive the points for this lab.