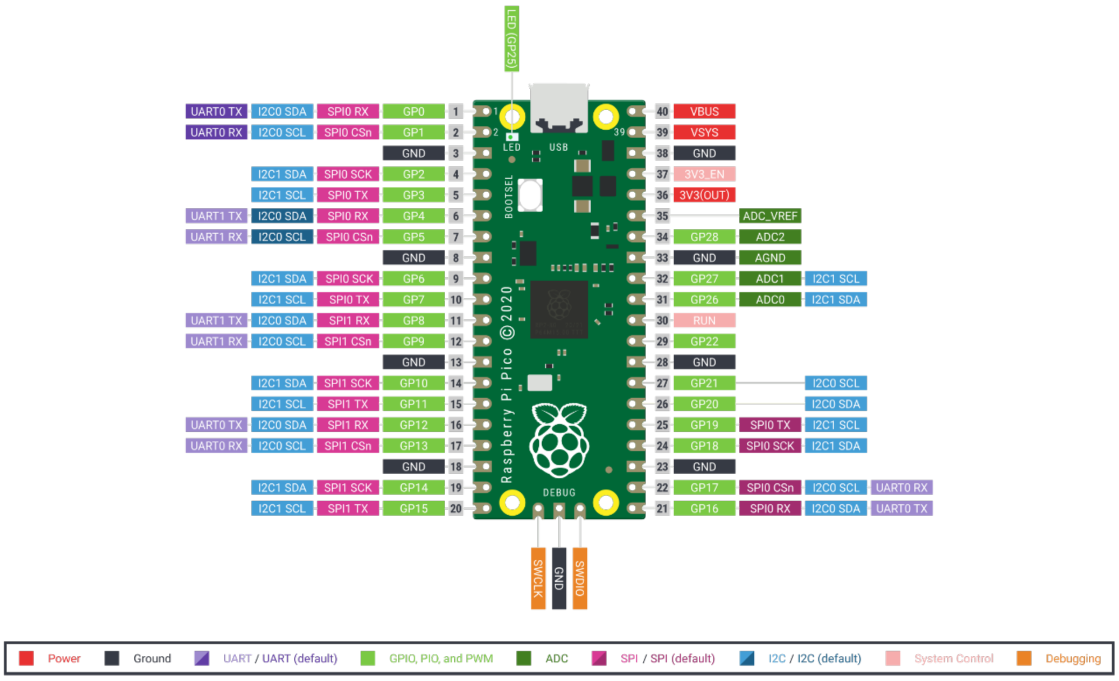
1- Introducing Raspberry Pi Pico

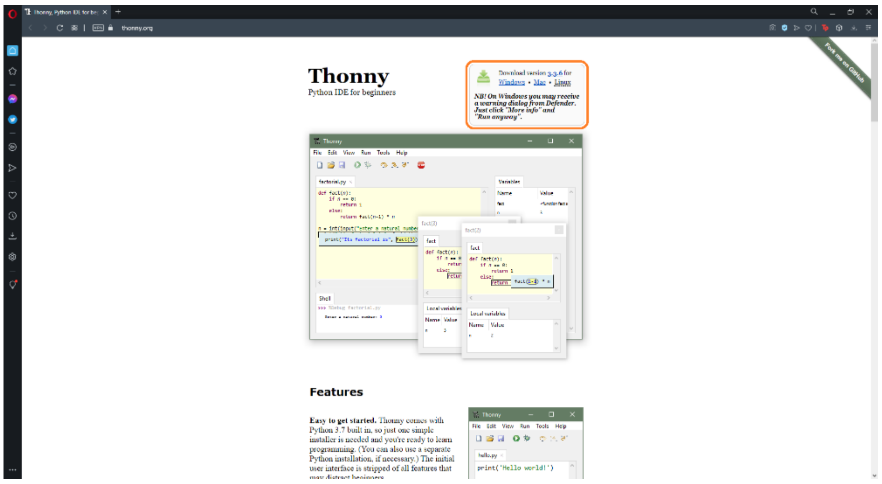
Raspberry Pi Pico that was released by Raspberry Pi Foundation is a microcontroller that might use in our Embedded systems projects, prototyping or to the Micropython presented to us for both elect-ronic and in learning coding. Raspberry Pi Pico, which is quite powerful compared to Arduino Nano, stands out with its 32-bit Arm Cortex M0 + 133 MHz processor. Thanks to the temperature sensor on it, it will be very easy for us to make projects such as a weather monitor.



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

2- Installing Thonny IDE

Firstly, You should download Thonny Ide which right version for your operating system on link https://thonny.org. In this blog, you will see the version for Windows 10.



Thonny
Python IDE for beginners

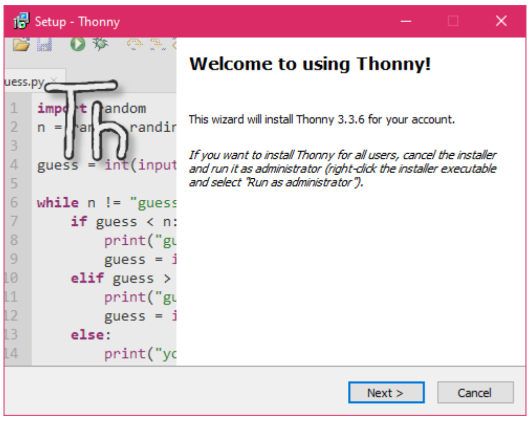
Download version 3.3.6 for Windows • Mac • Linux

NB! On Windows you may receive a warning dialog from Defender. Just click "More info" and "Run anyway".

Features

Easy to get started. Thonny comes with Python 3.7 built in, so just one simple installer is needed and you're ready to learn programming. (You can also use a separate Python installation, if necessary.) The initial user interface is stripped of all features that may distract beginners.

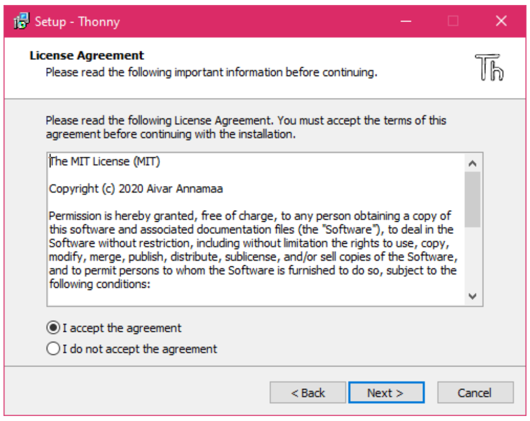
You run the installation file you downloaded. Follow the these screenshots to complete the installation process. You just need to click "Next".



Welcome to using Thonny!

This wizard will install Thonny 3.3.6 for your account.

If you want to install Thonny for all users, cancel the installer and run it as administrator (right-click the installer executable and select 'Run as administrator').



License Agreement
Please read the following important information before continuing.

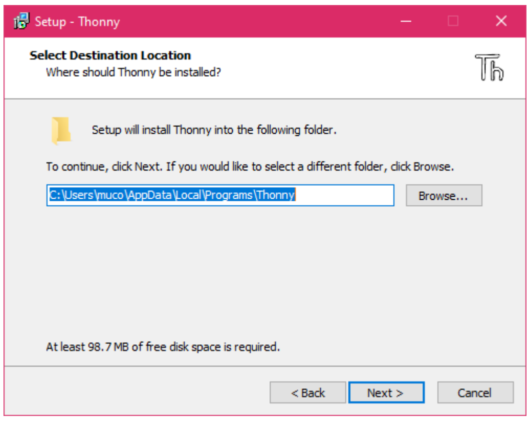
Please read the following License Agreement. You must accept the terms of this agreement before continuing with the installation.

The MIT License (MIT)

Copyright (c) 2020 Aivar Annamaa

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

[✓ I accept the agreement]
I do not accept the agreement



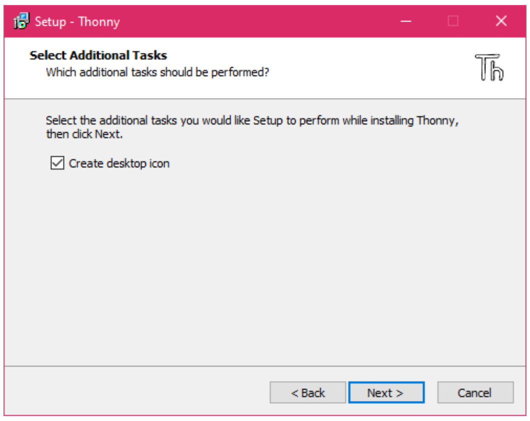
Select Destination Location
Where should Thonny be installed?

Setup will install Thonny into the following folder.

To continue, click Next. If you would like to select a different folder, click Browse.

C:\Users\muco\AppData\Local\Programs\Thonny

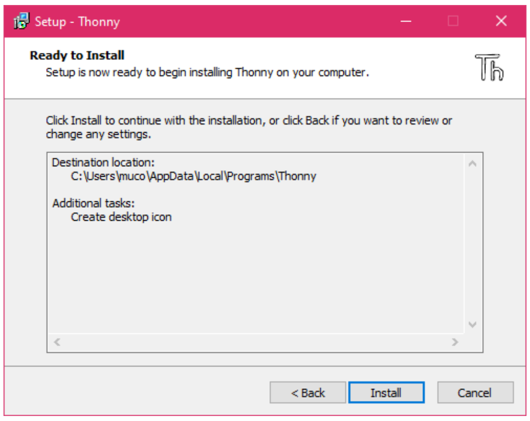
At least 98.7 MB of free disk space is required.



Select Additional Tasks
Which additional tasks should be performed?

Select the additional tasks you would like Setup to perform while installing Thonny, then click Next.

[✓] Create desktop icon

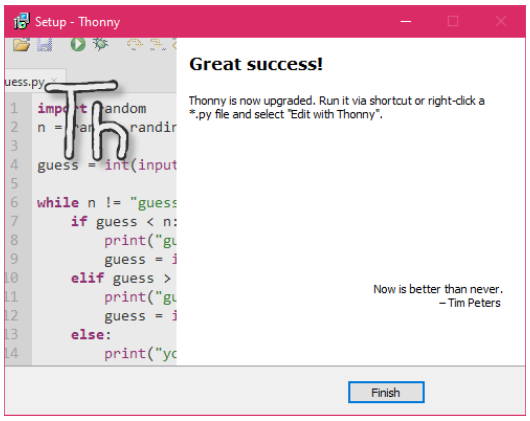


Ready to Install
Setup is now ready to begin installing Thonny on your computer.

Click Install to continue with the installation, or click Back if you want to review or change any settings.

Destination location:
C:\Users\muco\AppData\Local\Programs\Thonny

Additional tasks:
Create desktop icon



Great success!

Thonny is now upgraded. Run it via shortcut or right-click a *.py file and select "Edit with Thonny".

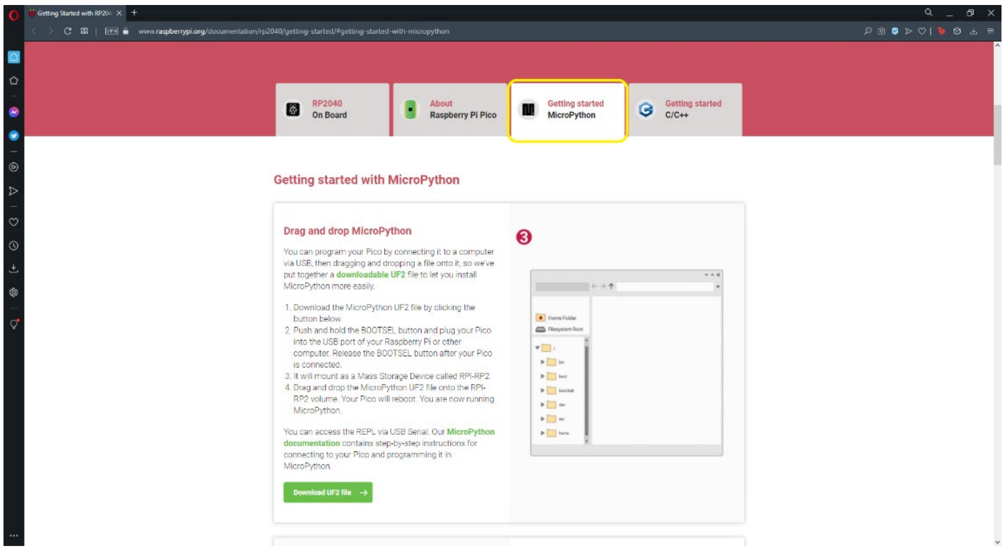
Now is better than never.
– Tim Peters

Lastly, You click "Finish"

3- Installing Micropython Firmware on Raspberry Pi Pico

Raspberry Pi Pico is programmable in two programming language that MicroPython and “C/C++”. You need to download the suitable "firmware" file for programming your Pico.

1- Just follow the instructions to download. https://www.raspberrypi.org/documentation/rp2040/getting-started/


www.raspberrypi.org/documentation/rp2040/getting-started/#getting-started-with-micropython
RP2040 On Board
About Raspberry Pi Pico
Getting started MicroPython
Getting started C/C++
3
Home Folder
Filesystem Root
Download UF2 file

Getting started with MicroPython

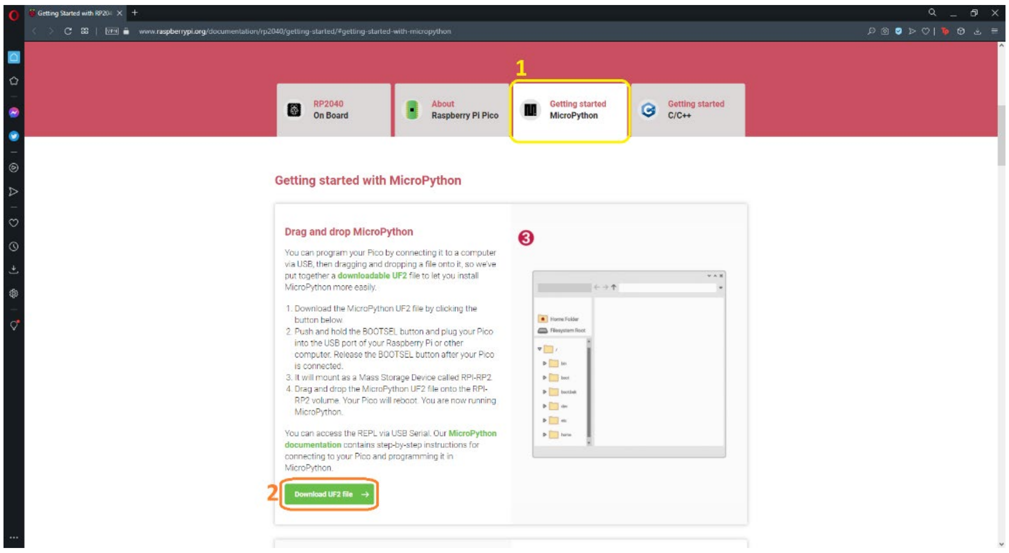
Drag and drop MicroPython

You can program your Pico by connecting it to a computer via USB, then dragging and dropping a file onto it, so we've put together a downloadable UF2 file to let you install MicroPython more easily.

1. Download the MicroPython UF2 file by clicking the button below.
2. Push and hold the BOOTSEL button and plug your Pico into the USB port of your Raspberry Pi or other computer. Release the BOOTSEL button after your Pico is connected.
3. It will mount as a Mass Storage Device called RPI-RP2.
4. Drag and drop the MicroPython UF2 file onto the RPI-RP2 volume. Your Pico will reboot. You are now running MicroPython.

You can access the REPL via USB Serial. Our MicroPython documentation contains step-by-step instructions for connecting to your Pico and programming it in MicroPython.

2- You click “Download UF2 file” and download the "firmware"



Getting started with MicroPython

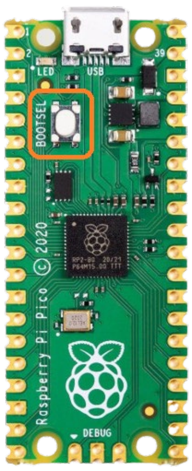
Drag and drop MicroPython

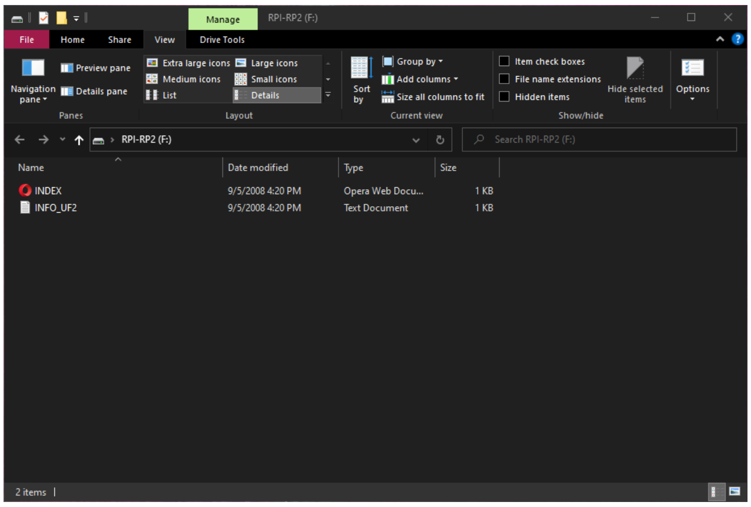
You can program your Pico by connecting it to a computer via USB, then dragging and dropping a file onto it, so we've put together a downloadable UF2 file to let you install MicroPython more easily.

1. Download the MicroPython UF2 file by clicking the button below.
2. Push and hold the BOOTSEL button and plug your Pico into the USB port of your Raspberry Pi or other computer. Release the BOOTSEL button after your Pico is connected.
3. It will mount as a Mass Storage Device called RPI-RP2.
4. Drag and drop the MicroPython UF2 file onto the RPI-RP2 volume. Your Pico will reboot. You are now running MicroPython.

You can access the REPL via USB Serial. Our MicroPython documentation contains step-by-step instructions for connecting to your Pico and programming it in MicroPython.

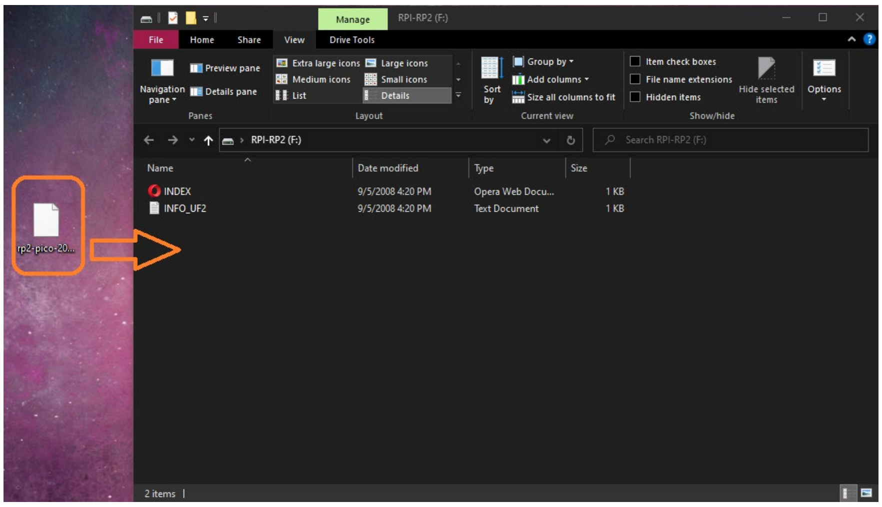
3- Connect to our computer by holding down the "BOOTSEL" button on your Pico. You will see the folder when you connected.





Name	Date modified	Type	Size
INDEX	9/5/2008 4:20 PM	Opera Web Docu...	1 KB
INFO_UF2	9/5/2008 4:20 PM	Text Document	1 KB

4- Copy the “*.UF2” file that downloaded on that folder.



Name	Date modified	Type	Size
INDEX	9/5/2008 4:20 PM	Opera Web Docu...	1 KB
INFO_UF2	9/5/2008 4:20 PM	Text Document	1 KB

When the copying process is finished, the folder will close automatically.

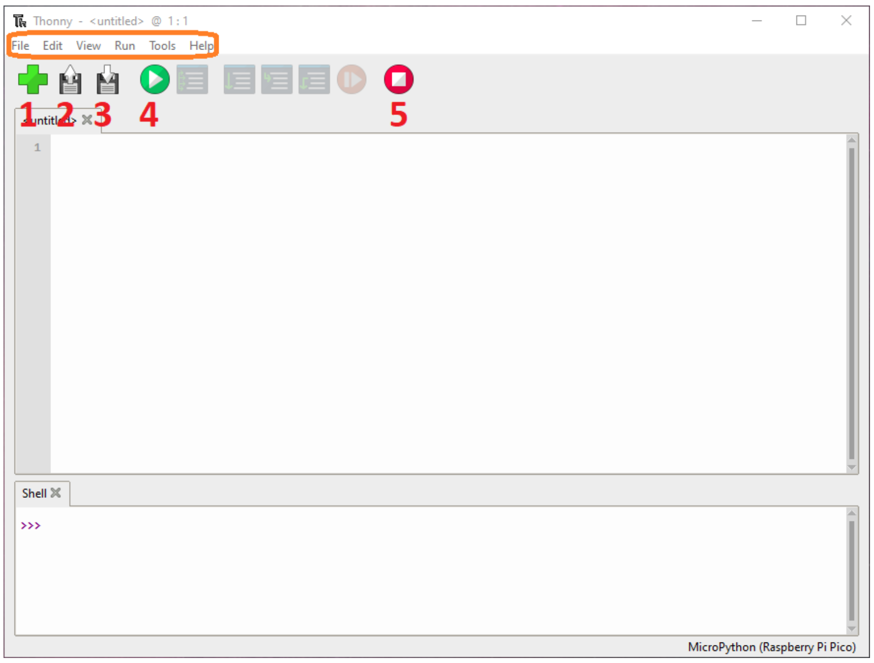
4- Introducing Thonny IDE Interface

Unlike Arduino, Raspberry Pi Pico has its own memory ,and library or scripts what you want can be uploaded into the this memory with Thonny IDE. There are some useful things in Thonny IDE

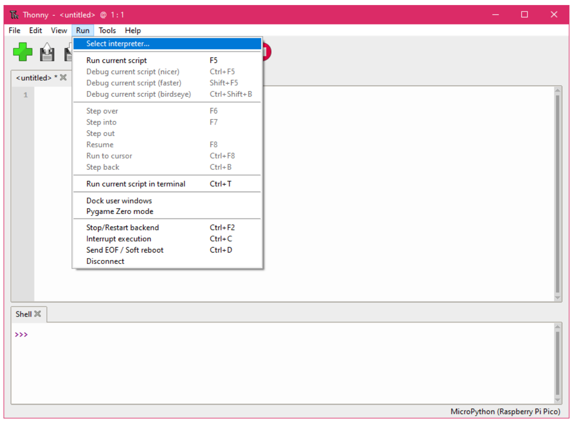
When you open the Thonny IDE, an interface like in the photo will welcome you. The functions of the tabs at the top, the section I have included in the orange box, respectively;

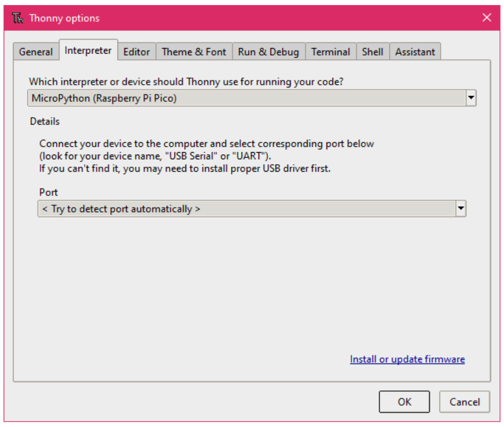
- **Files:** On this tab, you can save scripts what you wrote, open the scripts you have written or new Project file
- **Edit:** on this tab you can undo, forward, copy or select all in scripts what you have written
- **View:** On this tab ,you can customize the interface
- **Run:** on this tab, you can run code you have written, choose Interpreter for your Raspberry Pi Pico or operate like Debug on code
- **Tools:** On this tab, you can manage packets or edit Thonny settings
- **Help:** On this tab, you can get information about Thonny version or view help content regarding your issue.

1. You can open a new file with this icon
2. You can open scripts you have saved with this icon
3. *3.* You can save the script with this icon
4. You can run the script with this icon abi bu ne salak mı adamlar?
5. *5.* You can stop the script with this icon



You adjust the Interpreter Settings on Thonny IDE for using your Raspberry Pi Pico. Click the the run tab at the top, then click Select Interpreter. Select "MicroPython (Raspberry Pi Pico)" as the "Interpreter" and "Try to detect port automatically" as the port from the window that opens.


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
Select interpreter...
Run current script F5
Debug current script (nicer) Ctrl+F5
Debug current script (faster) Shift+F5
Debug current script (birdseye) Ctrl+Shift+B
Step over F6
Step into F7
Step out
Resume F8
Run to cursor Ctrl+F8
Step back Ctrl+B
Run current script in terminal Ctrl+T
Dock user windows
Pygame Zero mode
Stop/Restart backend Ctrl+F2
Interrupt execution Ctrl+C
Send EOF / Soft reboot Ctrl+D
Disconnect
Shell
MicroPython (Raspberry Pi Pico)


Thonny options

General | Interpreter | Editor | Theme & Font | Run & Debug | Terminal | Shell | Assistant

Which interpreter or device should Thonny use for running your code?

MicroPython (Raspberry Pi Pico)

Details

Connect your device to the computer and select corresponding port below
(look for your device name, "USB Serial" or "UART").
If you can't find it, you may need to install proper USB driver first.

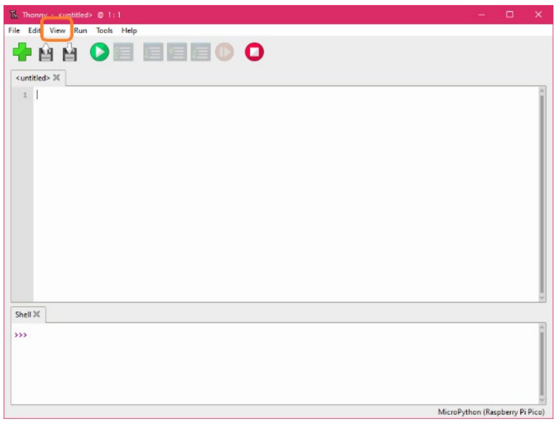
Port

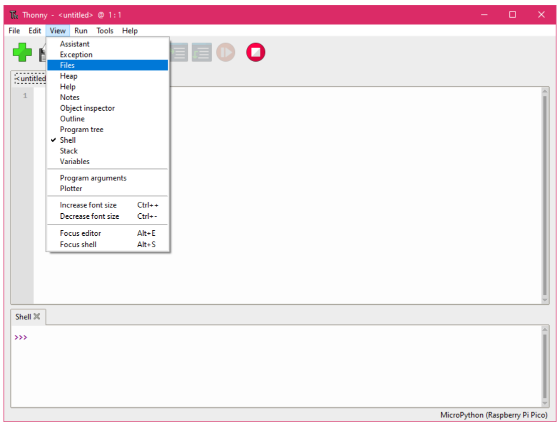
< Try to detect port automatically >

Install or update firmware

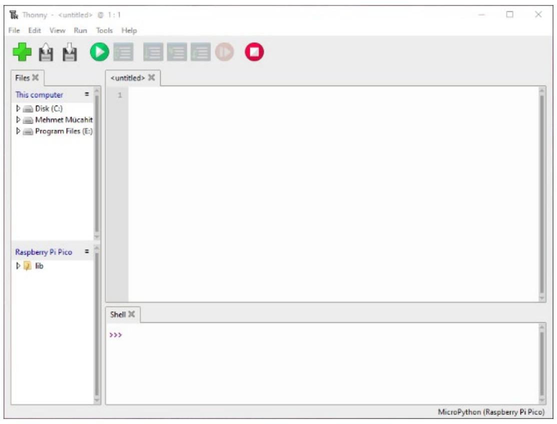
OK Cancel

Add “Files” tab that will contribute on the interface.Click the the“View” tab at the top and select “Files”.


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
<untitled>
Shell
MicroPython (Raspberry Pi Pico)



After that,you will see a tab like these screenshots..

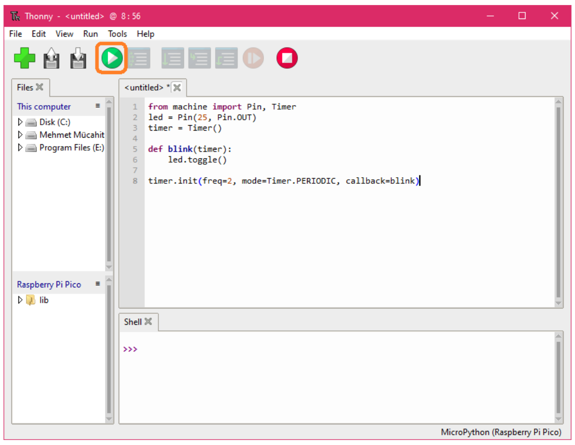


5- Blink Internal LED

Type the following code into Thonny IDE this code provides the Internal LED to blink every half second When you examine this code, you wrote the code "from machine import Pin, Timer" so that be able to access the hardware on the Pico. In line 2, you stated that the internal LED on the Pico is connected to pin 25. In line 3, you started timer .In line 5,you defined a blink function. This is a simple function that makes the LED light. In line 8, you ran the timer. We specified the frequency of the process with "freq" in parentheses, the mode of the timer with the "mode" and the function we would call with the "callback".

```
1 from machine import Pin, Timer
2 led = Pin(25, Pin.OUT)
3 timer = Timer()
4
5 def blink(timer):
6     led.toggle()
7
8 timer.init(freq=2, mode=Timer.PERIODIC, callback=blink)
```

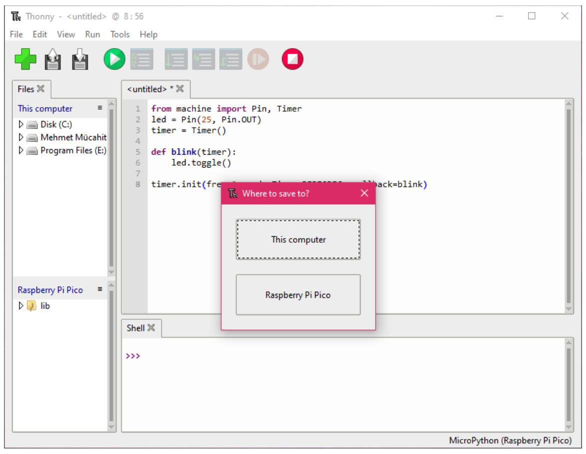
Upload this code to your Pico. Let's click on the green "Run" icon at the top.



```
1 from machine import Pin, Timer
2 led = Pin(25, Pin.OUT)
3 timer = Timer()
4
5 def blink(timer):
6     led.toggle()
7
8 timer.init(freq=2, mode=Timer.PERIODIC, callback=blink)
```

```
>>>
```

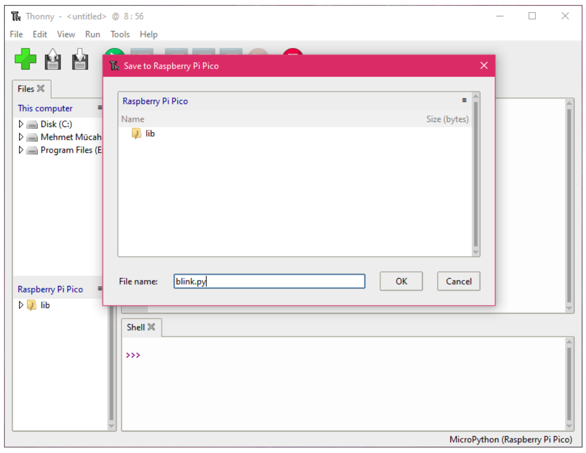
It asks where you want to save the code. If save the code to the computer and run it, the code will only run when the Pico is plugged into the computer. For the code to always run. You need to click on the "Raspberry Pi Pico" option and save it to your Pico.



```
1 from machine import Pin, Timer
2 led = Pin(25, Pin.OUT)
3 timer = Timer()
4
5 def blink(timer):
6     led.toggle()
7
8 timer.init(fre                          ack=blink)
```

```
>>>
```

After clicking on the "Raspberry Pi Pico" text, a window like in the photo will open. In this window, it asks us to which location you want to save the code you wrote in Pico. You can save it directly in Pico. For this, you need to write a name and file extension to save the code youhave written. Since you are working with MicroPython, your file extensions will always be "* .py". I typed "blink.py" as the filename. You can also type any file name you want. After typing the file name, you can save and run the code by clicking the "OK" button.



When the code runs, the internal LED on your Pico will blink and flash at half-second intervals.

6- LED Control with Button

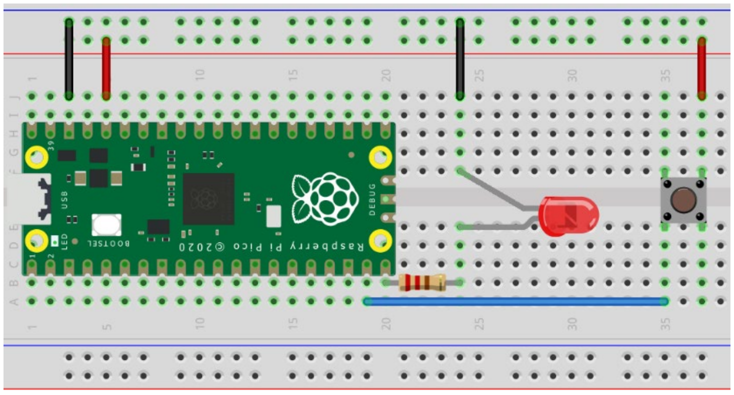
Once you lit LEDs, but this time, you need to realize this project in order to warm up our Pico's hardware and breadboard usage.

Necessary Materials:

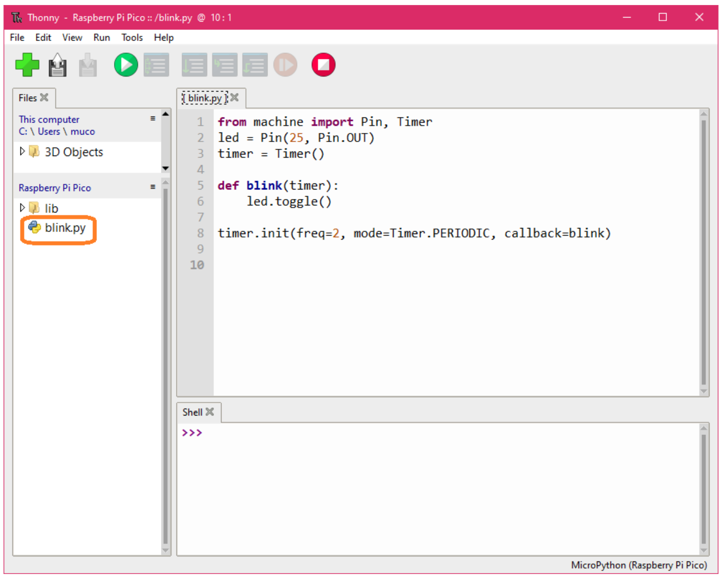
- Raspberry Pi Pico
- Breadboard
- Push Button
- LED
- Male – Male Jumper
- 220 Ω or 330 Ω Resistor

When you look at the pin diagram of your Pico, the 38th pin is "GND", ie the ground pin, the 36th pin is the "3V3 (OUT)" pin. You will use these two pins frequently in your projects.

First, build the circuit on the breadboard according to the diagram below.



Then open your Thonny IDE and edit the "blink" code you have written in the previous project. Open it by double clicking on the "blink.py" file on the left side of Thonny IDE. The codes you have written in the previous project will be opened. Now you will delete these codes completely and write new code.


Thonny - Raspberry Pi Pico :: /blink.py @ 10 : 1
File Edit View Run Tools Help
Files
This computer
C: \ Users \ muco
3D Objects
Raspberry Pi Pico
lib
blink.py
Shell
MicroPython (Raspberry Pi Pico)

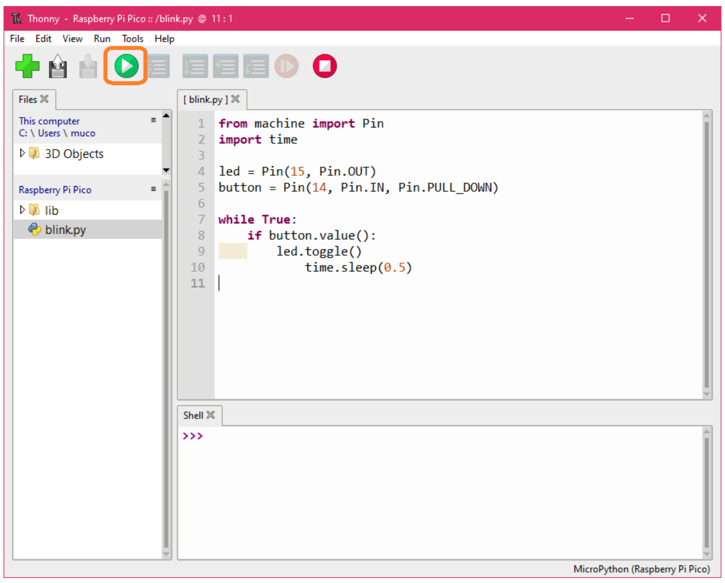
```
from machine import Pin, Timer
led = Pin(25, Pin.OUT)
timer = Timer()

def blink(timer):
    led.toggle()

timer.init(freq=2, mode=Timer.PERIODIC, callback=blink)
```

```
>>>
```

When you examine the codes below, you defined the hardware of Pi-co with the code "from mac-hine import Pin" as in the previous project and added the "time" library. With the code "led = Pin (15, Pin.OUT)", you assigned the 15 nu-mara pin as the pin you connected to the LED. With the code "but-ton = Pin (14, Pin.IN, Pin.PULL_DOWN)", you defined the pin number 14 to which you connect the button as the input pin. Then you created a "while loop". In this cycle, it will change the status of the led according to the value read from the button. For example, if press the button once while the LED is off, it will turn on, if the LED is on, it will turn off when you press the button. When press and hold the button with the code "time.sleep (0.5)", the LED will flash and flash at half-second intervals.

```python
from machine import Pin
import time

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

while True:
    if button.value():
        led.toggle()
            time.sleep(0.5)
```

After writing the codes, press the "Run" icon at the top and save your codes to your Pico and run them.


Thonny - Raspberry Pi Pico :: /blink.py @ 11 : 1
File Edit View Run Tools Help
Files
This computer
C: \ Users \ muco
3D Objects
Raspberry Pi Pico
lib
blink.py
[blink.py]
Shell
MicroPython (Raspberry Pi Pico)

```python
from machine import Pin
import time

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

while True:
    if button.value():
        led.toggle()
            time.sleep(0.5)
```

```
>>>
```

Now you will be able to turn the LED on and off by pressing the button. Move on to the next project.

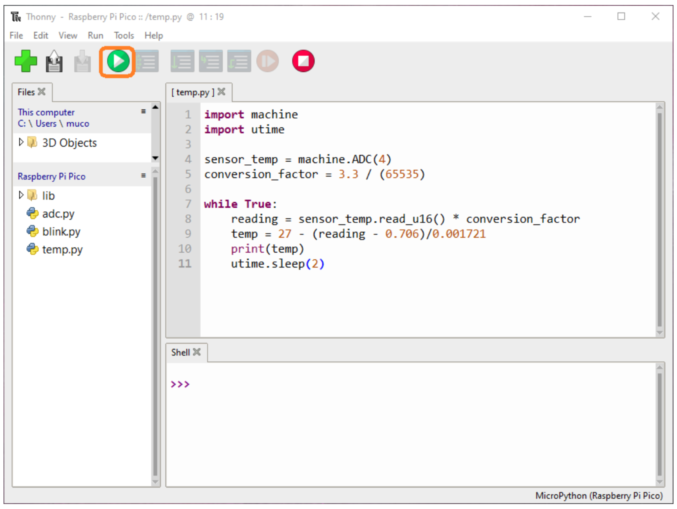
7- Temperature Measurement with Pico

As I mentioned at the beginning, Raspberry Pi Pico has an internal temperature sensor. So why not take a temperature measurement?

First, open the Thonny IDE and open a new file and write the following codes. Now, when you examine the codes, you have added the necessary "machine" and "utime" libraries as usual. Since made analog readings with the temperature sensor, you defined that it was connected to pin number 4 with the code "sensor_temp = machine. ADC (4)", then you converted the 16-bit data from the sensor into voltage data that could be meaningful for you with the code "conversion_factor". Since the Pico pin gives 3.3 volts, you divided 3.3 volts by 216 - 1 = 65535. Now, when you examine the While loop, read the data from the sensor. In the code on line 8, multiply the data from the temperature sensor with the "conversion_factor" you just wrote, and write the temperature data in terms of voltage. You convert the data you have obtained to "Celsius" in the 9th line. Finally, print the temperature data to Shell with the code "print (temp)".

```
 1 import machine
 2 import utime
 3
 4 sensor_temp = machine.ADC(4)
 5 conversion_factor = 3.3 / (65535)
 6
 7 while True:
 8     reading = sensor_temp.read_u16() * conversion_factor
 9     temp = 27 - (reading - 0.706)/0.001721
10     print(temp)
11     utime.sleep(2)
```

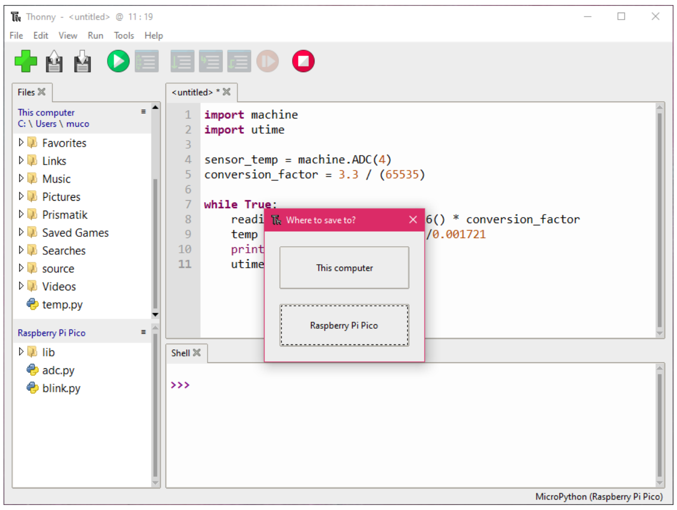
Now save these codes to your Pico and run them, then press the "Run" icon at the top.


Thonny - Raspberry Pi Pico :: /temp.py @ 11 : 19
File Edit View Run Tools Help
Files
This computer
C: \ Users \ muco
3D Objects
Raspberry Pi Pico
lib
adc.py
blink.py
temp.py
[temp.py]
Shell
MicroPython (Raspberry Pi Pico)

```
 1 import machine
 2 import utime
 3
 4 sensor_temp = machine.ADC(4)
 5 conversion_factor = 3.3 / (65535)
 6
 7 while True:
 8     reading = sensor_temp.read_u16() * conversion_factor
 9     temp = 27 - (reading - 0.706)/0.001721
10     print(temp)
11     utime.sleep(2)
```

```
>>>
```

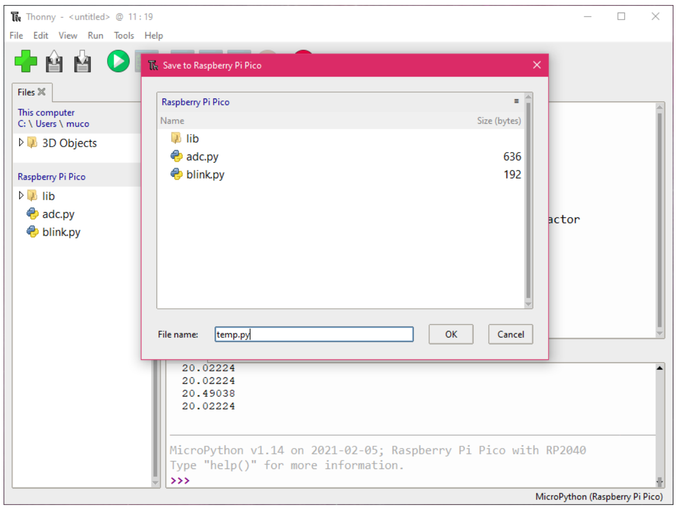
It asks where you want to save the code. For the code to always run, you need to click on the "Raspberry Pi Pico" option and save it to our Pico.



```
import machine
import utime

sensor_temp = machine.ADC(4)
conversion_factor = 3.3 / (65535)

while True:
    readi                6() * conversion_factor
    temp                 /0.001721
    print
    utime
```

```
>>>
```

Write "temp.py" as the file name, click the "OK" button and now you can read the temperature data from Shell.

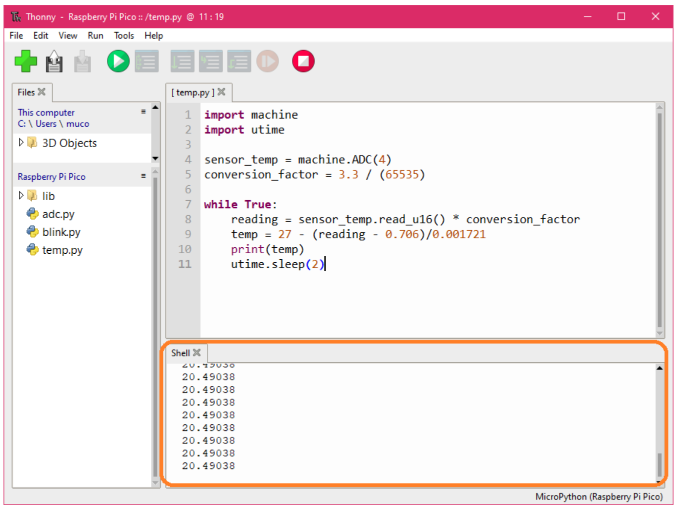


Name	Size (bytes)
lib	
adc.py	636
blink.py	192

File name: temp.py

```
  20.02224
  20.02224
  20.49038
  20.02224

MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>>
```

In the Shell window at the bottom, you can see the temperature data from your Pico every 2 seconds.


Thonny - Raspberry Pi Pico :: /temp.py @ 11 : 19
File Edit View Run Tools Help
Files
This computer
C: \ Users \ muco
3D Objects
Raspberry Pi Pico
lib
adc.py
blink.py
temp.py
[temp.py]
Shell
MicroPython (Raspberry Pi Pico)

```
import machine
import utime

sensor_temp = machine.ADC(4)
conversion_factor = 3.3 / (65535)

while True:
    reading = sensor_temp.read_u16() * conversion_factor
    temp = 27 - (reading - 0.706)/0.001721
    print(temp)
    utime.sleep(2)
```

```
20.49038
20.49038
20.49038
20.49038
20.49038
20.49038
20.49038
20.49038
20.49038
```

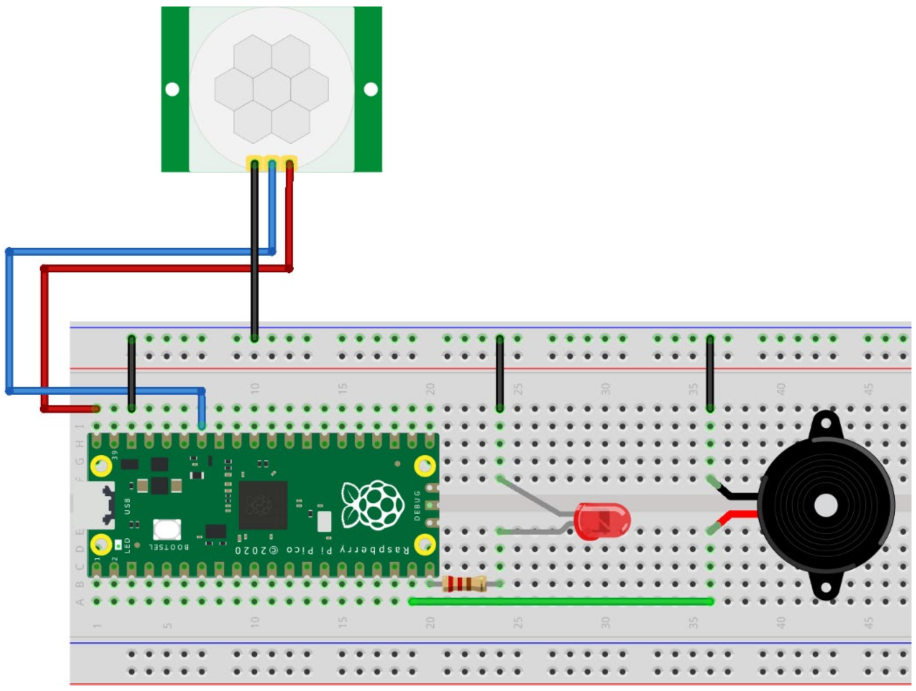
8- Hırsız Alarmı

In this project, you will make a burglar alarm using PIR sensor, LED and buzzer.

Necessary materials:

- Raspberry Pi Pico
- Breadboard
- PIR Sensor
- Buzzer
- LED
- Male – Male Jumper
- Male – Female Jumper
- 220 Ω or 330 Ω Resistor

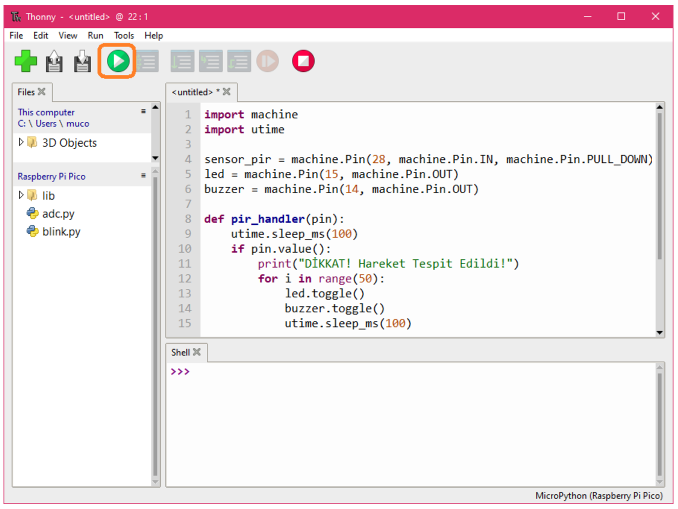
First, build your circuit according to the diagram below.



Now open a new file in Thonny IDE and write the following codes. . When you examine the codes, you first added the libraries and then defined the pins to which you connected the PIR sensor, Buzzer and LED. You created a function with "def pir_handler (pin)". When the motion is detected with the if block in the function, “ATTENTION! Motion Detected! ” printing, buzzer and LED are working.

```python
import machine
import utime

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
buzzer = machine.Pin(14, machine.Pin.OUT)

def pir_handler(pin):
    utime.sleep_ms(100)
    if pin.value():
        print("ATTENTION! Motion Detected!")
        for i in range(50):
            led.toggle()
            buzzer.toggle()
            utime.sleep_ms(100)
sensor_pir.irq(trigger=machine.Pin.IRQ_RISING, handler=pir_handler)

while True:
    led.toggle()
    utime.sleep(5)
```

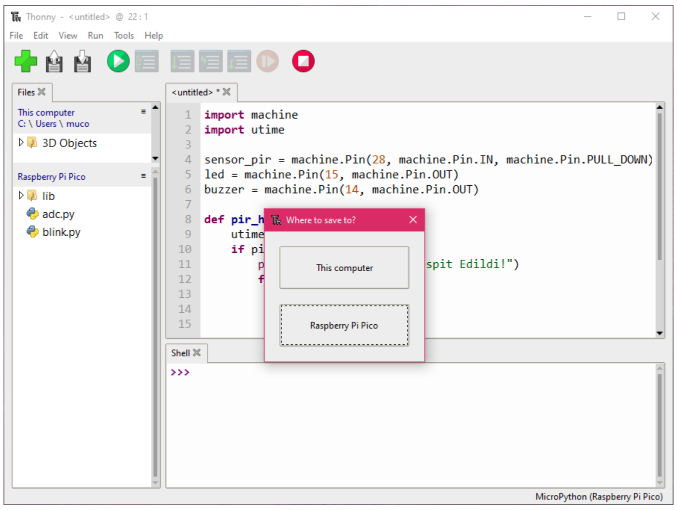
Now save these codes to your Pico and run them, then press the "Run" icon at the top.



```python
import machine
import utime

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
buzzer = machine.Pin(14, machine.Pin.OUT)

def pir_handler(pin):
    utime.sleep_ms(100)
    if pin.value():
        print("DİKKAT! Hareket Tespit Edildi!")
        for i in range(50):
            led.toggle()
            buzzer.toggle()
            utime.sleep_ms(100)
```

```
>>>
```

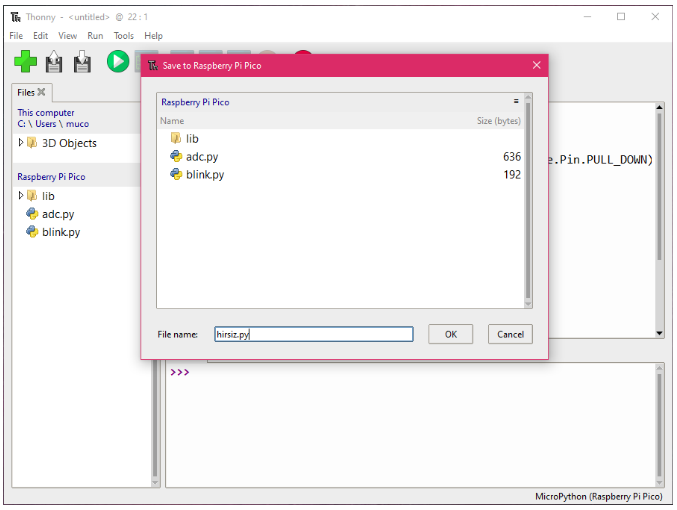
It asks where you want to save the code. For the code to always run, you need to click on the "Raspberry Pi Pico" option and save it to our Pico.



```
import machine
import utime

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
buzzer = machine.Pin(14, machine.Pin.OUT)

def pir_h
    utime
    if pi
        p                                        spit Edildi!")
        f
```

```
>>>
```

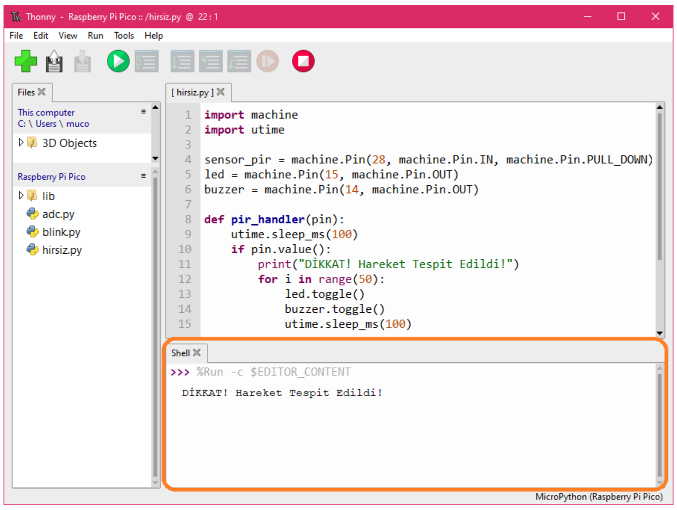
Write "hirsiz.py" as the file name and click on the "OK" button and now, in case of any situation that triggers our motion sensor, the buzzer will sound and our LED will light.


Thonny - <untitled> @ 22 : 1
File Edit View Run Tools Help
Files
This computer
C: \ Users \ muco
3D Objects
Raspberry Pi Pico
lib
adc.py
blink.py
Save to Raspberry Pi Pico
Raspberry Pi Pico
OK
Cancel
MicroPython (Raspberry Pi Pico)

Name	Size (bytes)
lib	
adc.py	636
blink.py	192

File name: hirsiz.py

In the Shell window at the bottom, when motion is detected, “ATTENTION! Motion Detected! ” text will appear.


Thonny - Raspberry Pi Pico :: /hirsiz.py @ 22 : 1
File Edit View Run Tools Help
Files
This computer
C: \ Users \ muco
3D Objects
Raspberry Pi Pico
lib
adc.py
blink.py
hirsiz.py
[hirsiz.py]
Shell
MicroPython (Raspberry Pi Pico)

```
import machine
import utime

sensor_pir = machine.Pin(28, machine.Pin.IN, machine.Pin.PULL_DOWN)
led = machine.Pin(15, machine.Pin.OUT)
buzzer = machine.Pin(14, machine.Pin.OUT)

def pir_handler(pin):
    utime.sleep_ms(100)
    if pin.value():
        print("DİKKAT! Hareket Tespit Edildi!")
        for i in range(50):
            led.toggle()
            buzzer.toggle()
            utime.sleep_ms(100)
```

```
>>> %Run -c $EDITOR_CONTENT
 DİKKAT! Hareket Tespit Edildi!
```

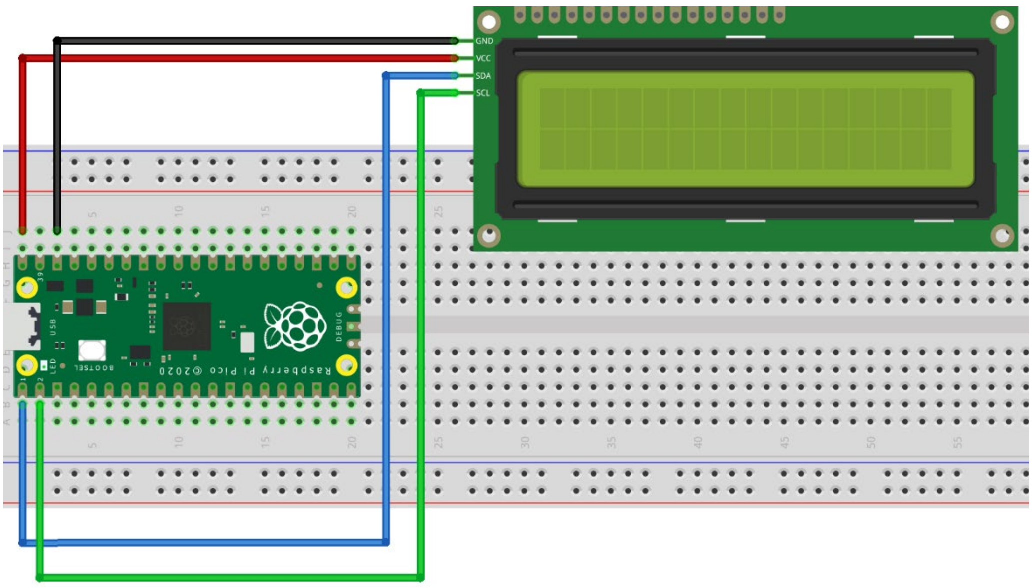
9- Weather Monitor

In this project, you will make a weather monitor using a 2X16 LCD screen. In this project, you will learn how to add a library and automatically save the temperature data you measure in a text file.

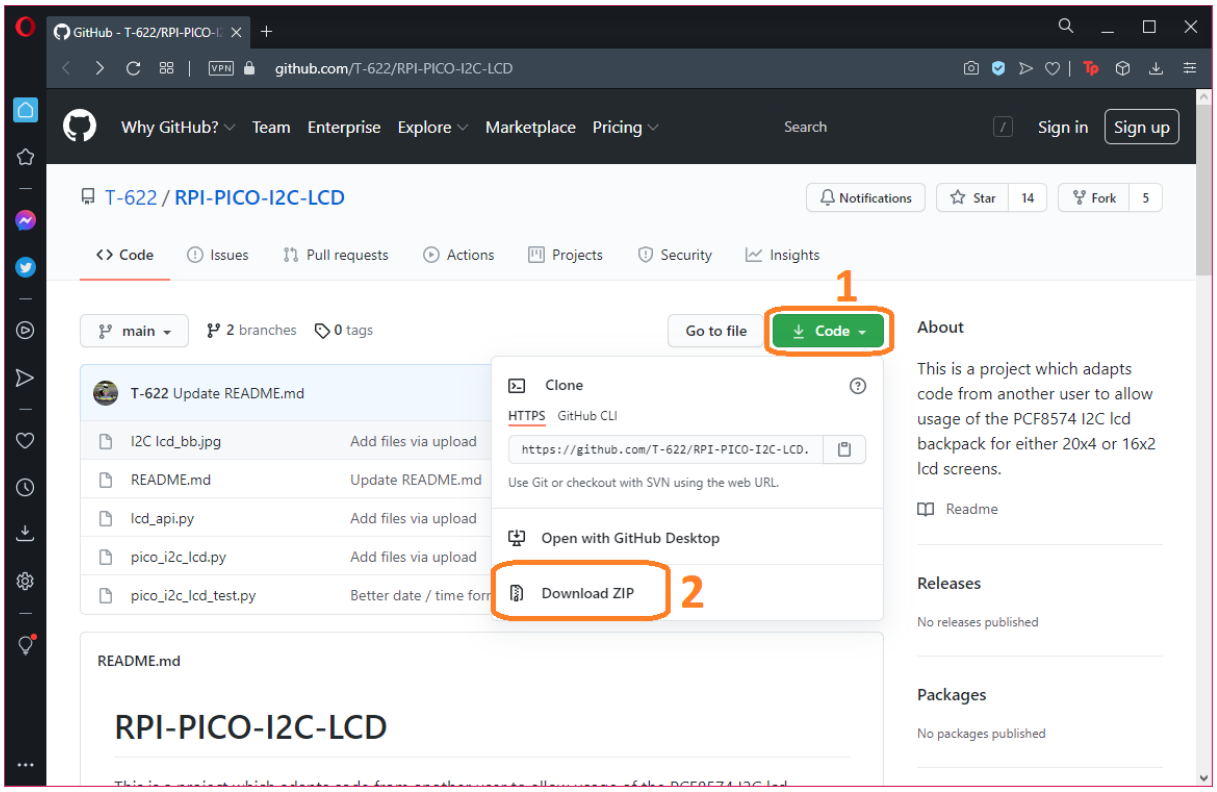
Necessary materials:

- Raspberry Pi Pico
- Breadboard
- 2X16 LCD Screen
- Male – Female Jumper

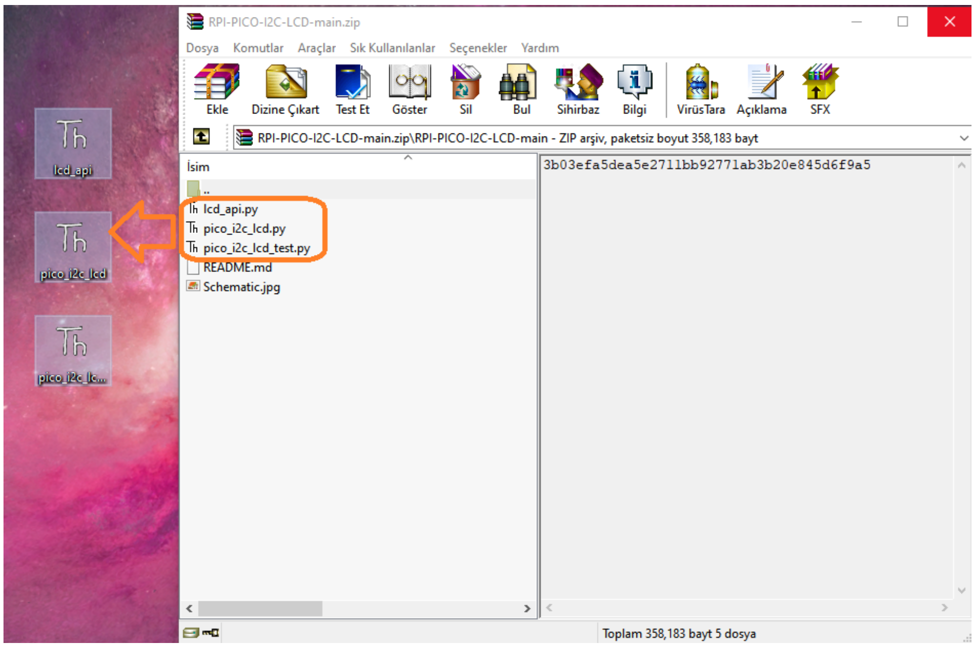
First, build your own circuit on the breadboard according to the circuit below.



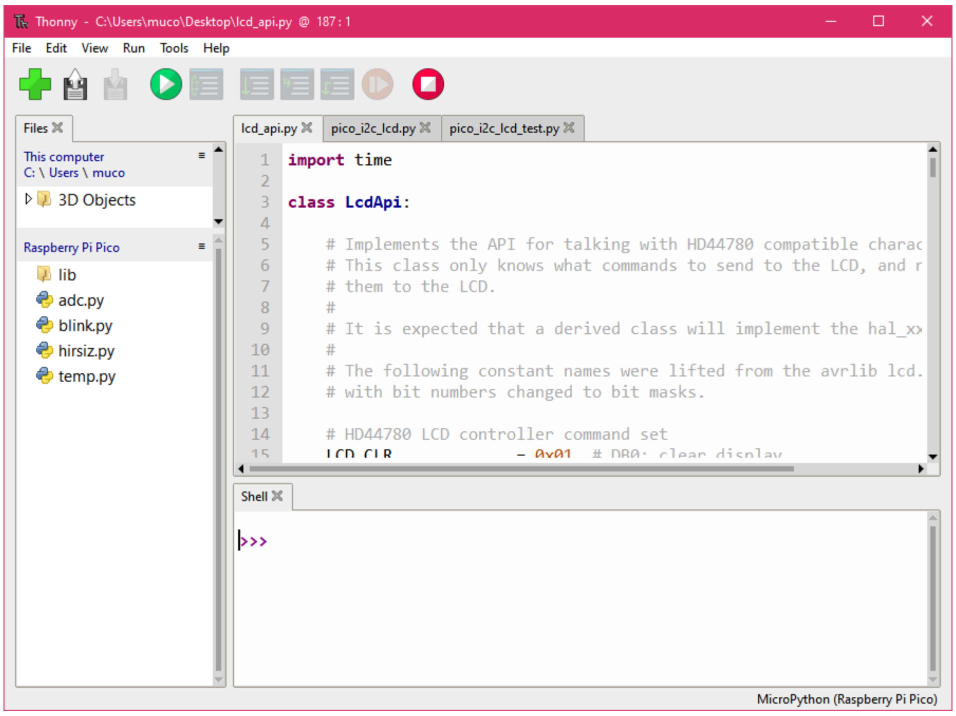
Now download the necessary library to use our LCD screen from https://github.com/T-622/RPI-PICO-I2C-LCD. When open the link, click the green button that says "Code" and then click the "Download ZIP" button to download the library.


GitHub - T-622/RPI-PICO-I2C-LCD
github.com/T-622/RPI-PICO-I2C-LCD
Why GitHub? Team Enterprise Explore Marketplace Pricing
Search
Sign in Sign up
T-622 / RPI-PICO-I2C-LCD
Notifications Star 14 Fork 5
Code Issues Pull requests Actions Projects Security Insights
main 2 branches 0 tags
Go to file
1
Code
About
This is a project which adapts code from another user to allow usage of the PCF8574 I2C lcd backpack for either 20x4 or 16x2 lcd screens.
Readme
Releases
No releases published
Packages
No packages published
T-622 Update README.md
I2C lcd_bb.jpg Add files via upload
README.md Update README.md
lcd_api.py Add files via upload
pico_i2c_lcd.py Add files via upload
pico_i2c_lcd_test.py Better date / time for
Clone
HTTPS GitHub CLI
https://github.com/T-622/RPI-PICO-I2C-LCD.
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP
2
README.md
RPI-PICO-I2C-LCD

Open the downloaded "* .zip" file and extract the "lcd_api.py", "pico_i2c_lcd.py", "pico_i2c_lcd_test.py" files to the desktop.


lcd_api
pico_i2c_lcd
pico_i2c_lc...
RPI-PICO-I2C-LCD-main.zip
Dosya Komutlar Araçlar Sık Kullanılanlar Seçenekler Yardım
Ekle Dizine Çıkart Test Et Göster Sil Bul Sihirbaz Bilgi VirüsTara Açıklama SFX
RPI-PICO-I2C-LCD-main.zip\RPI-PICO-I2C-LCD-main - ZIP arşiv, paketsiz boyut 358,183 bayt
İsim
..
lcd_api.py
pico_i2c_lcd.py
pico_i2c_lcd_test.py
README.md
Schematic.jpg
3b03efa5dea5e2711bb92771ab3b20e845d6f9a5
Toplam 358,183 bayt 5 dosya

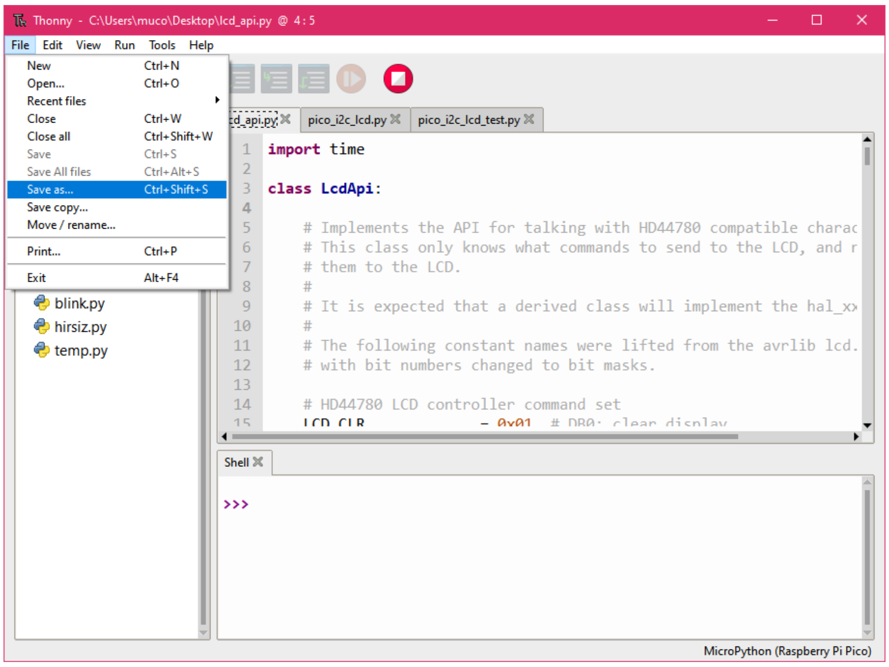
Open them in Thonny IDE by double clicking on these files that you extract to the desktop.



```
import time

class LcdApi:

    # Implements the API for talking with HD44780 compatible charac
    # This class only knows what commands to send to the LCD, and r
    # them to the LCD.
    #
    # It is expected that a derived class will implement the hal_x
    #
    # The following constant names were lifted from the avrlib lcd.
    # with bit numbers changed to bit masks.

    # HD44780 LCD controller command set
    LCD_CLR                 = 0x01  # DB0: clear display
```

```
>>>
```

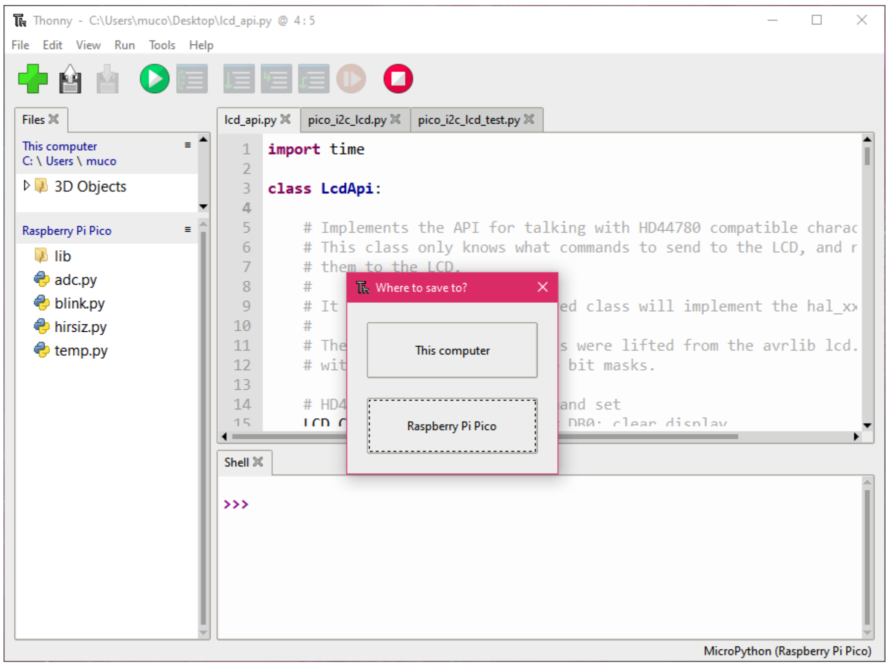
Now save these files to the "lib" folder in your Pico in order, click on the "Files" tab on the top and then click on "Save as ...".



```
import time

class LcdApi:

    # Implements the API for talking with HD44780 compatible charac
    # This class only knows what commands to send to the LCD, and r
    # them to the LCD.
    #
    # It is expected that a derived class will implement the hal_x
    #
    # The following constant names were lifted from the avrlib lcd.
    # with bit numbers changed to bit masks.

    # HD44780 LCD controller command set
    LCD_CLR                 = 0x01  # DB0: clear display
```

```
>>>
```

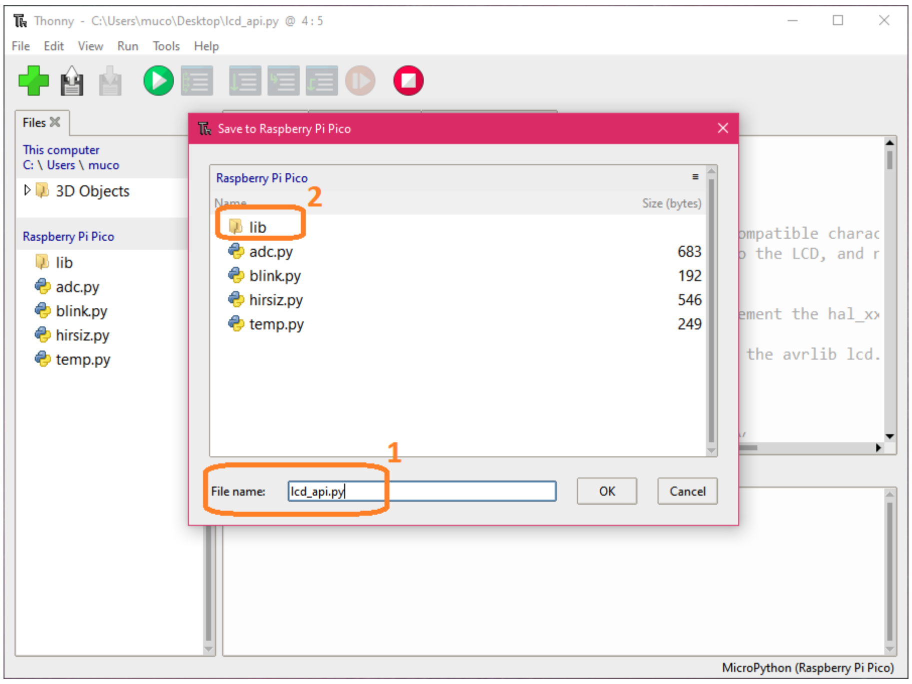
It asks where you want to save the library file. You will choose the Raspberry Pi Pico.



```
import time

class LcdApi:

    # Implements the API for talking with HD44780 compatible charac
    # This class only knows what commands to send to the LCD, and r
    # them to the LCD.
```

```
>>>
```

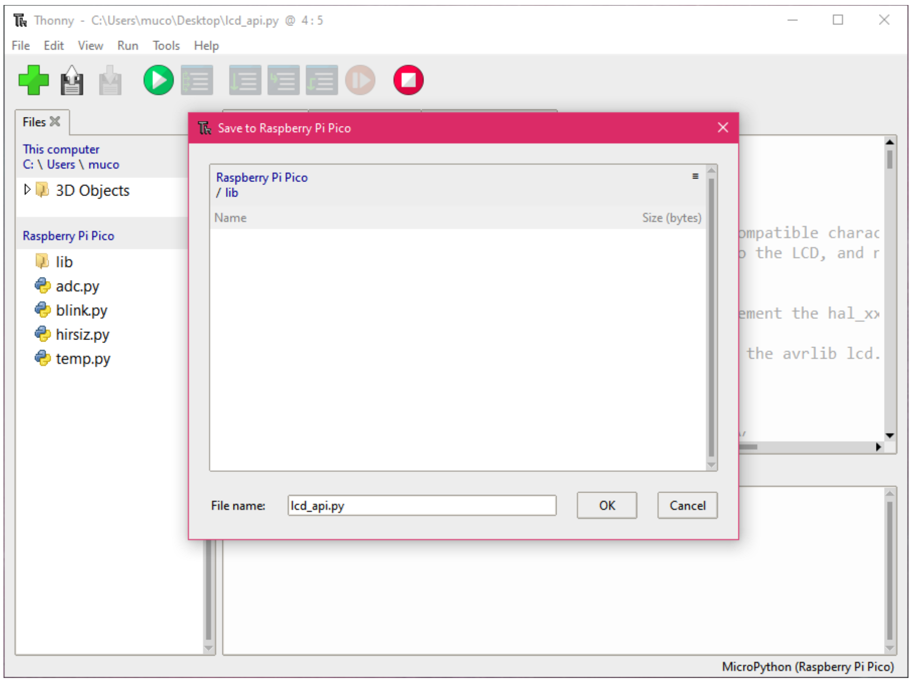
In this part, you have to choose the location where want to save the file, then type the file name first. After that double click on the "lib" folder.


Thonny - C:\Users\muco\Desktop\lcd_api.py @ 4 : 5
File Edit View Run Tools Help
Save to Raspberry Pi Pico
Raspberry Pi Pico
Name
Size (bytes)
2
1
File name:
OK
Cancel
MicroPython (Raspberry Pi Pico)

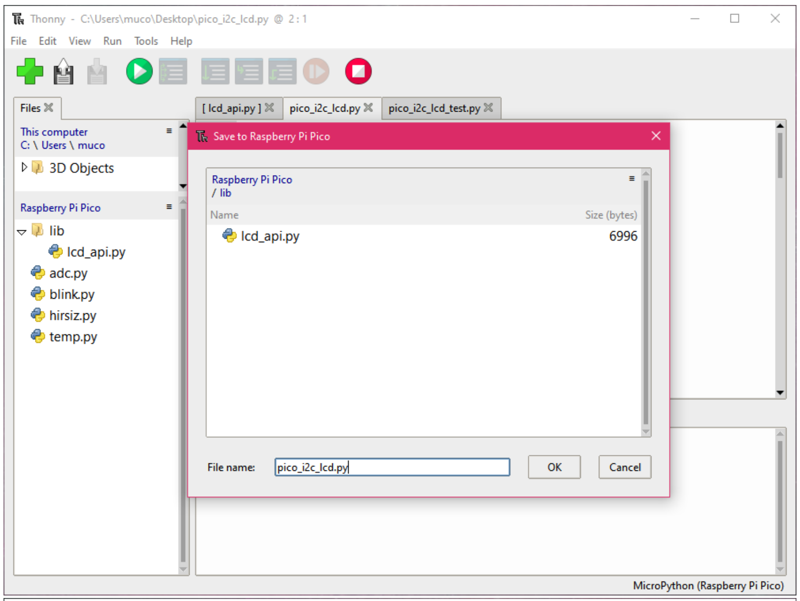
Name	Size (bytes)
lib	
adc.py	683
blink.py	192
hirsiz.py	546
temp.py	249

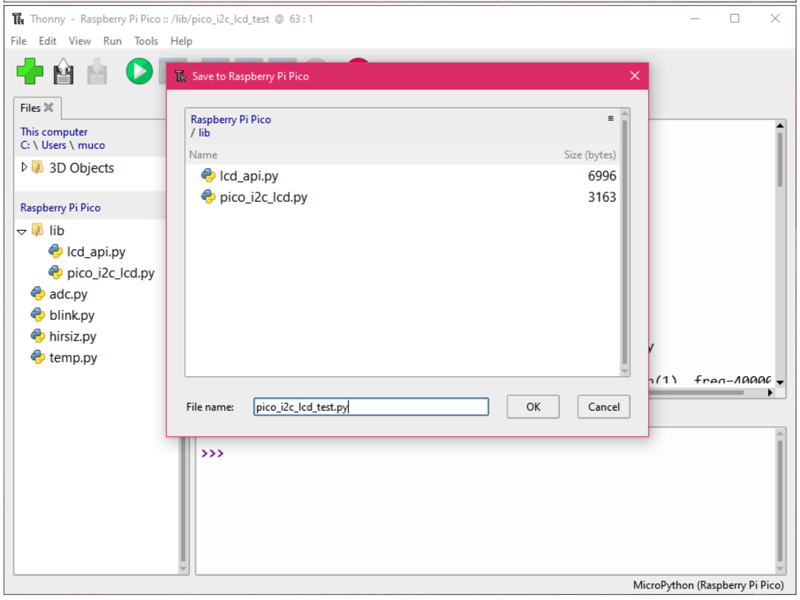
File name: lcd_api.py

Then click the "OK" button to save it to the folder



Do the same for the other two files.

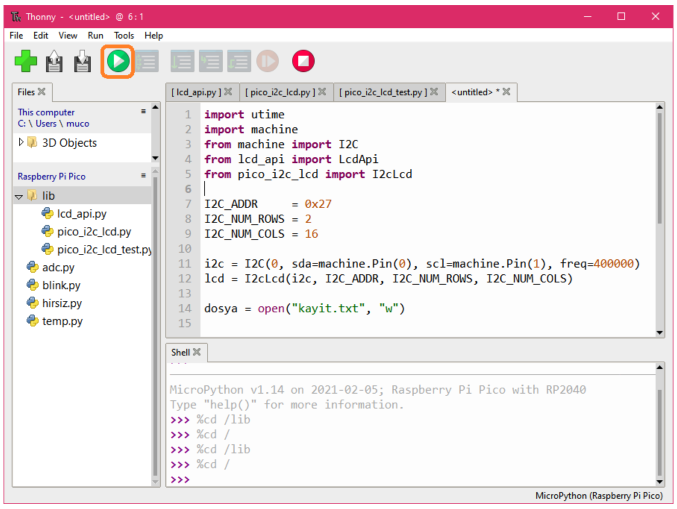




Now that you have added libraries, you can open a new file by clicking the "Plus" icon on the top of a new Thonny IDE and write the following codes. When you examine the codes, first added the libraries as in other projects, then made the settings of our 2X16 LCD screen and created a new text document named "save" with the code "file = open (" record.txt "," w ").

```python
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")

sensor_temp = machine.ADC(4)
conversion_factor = 3.3 / (65535)

while True:
    lcd.clear()
    reading = sensor_temp.read_u16() * conversion_factor
    temp = 27 - (reading - 0.706)/0.001721
    lcd.putstr("Temp: ")
    lcd.move_to(0,1)
    lcd.putstr(str(temp))
    dosya.write(str(temp) + "\n")
    dosya.flush()
    utime.sleep(2)
```

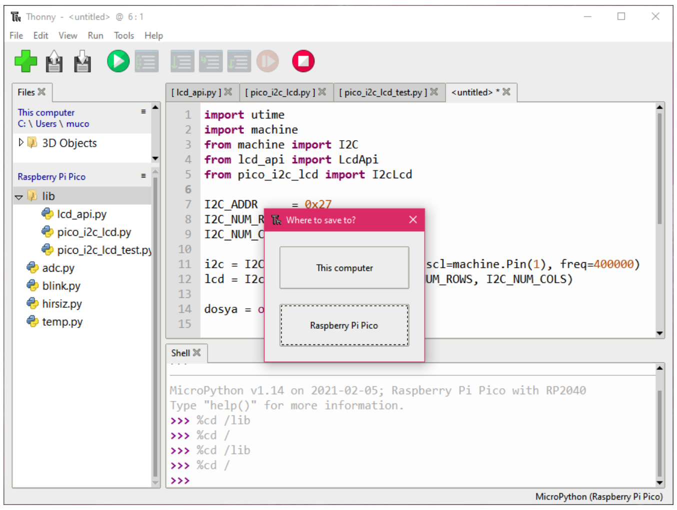
Now save these codes to your Pico and run them, then press the "Run" icon at the top.



```python
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")

```

```
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /lib
>>> %cd /
>>> %cd /lib
>>> %cd /
>>>
```

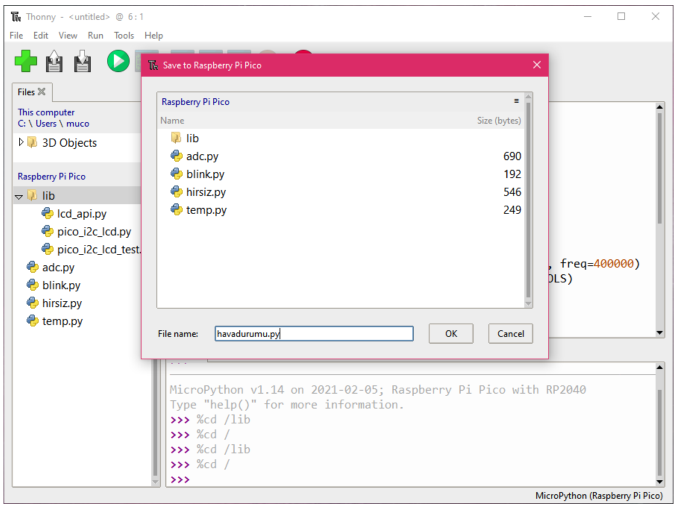
It asks where you want to save the code. For the code to always run, you need to click on the "Raspberry Pi Pico" option and save it to our Pico.



```
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_R
I2C_NUM_C

i2c = I2C                    scl=machine.Pin(1), freq=400000)
lcd = I2c                    UM_ROWS, I2C_NUM_COLS)

dosya = o
```

```
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /lib
>>> %cd /
>>> %cd /lib
>>> %cd /
>>>
```

Type "havadurumu.py" as the file name and click the "OK" button. Now, you will be able to see the temperature data on your LCD screen and at the same time this data will be recorded in a text document.

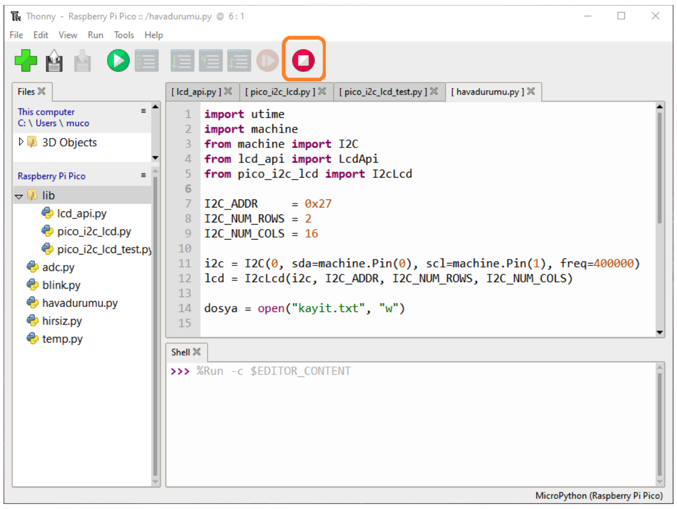


Name	Size (bytes)
lib	
adc.py	690
blink.py	192
hirsiz.py	546
temp.py	249

File name: havadurumu.py

```
MicroPython v1.14 on 2021-02-05; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %cd /lib
>>> %cd /
>>> %cd /lib
>>> %cd /
>>>
```

Now wait for some data to collect, then click the "Stop" icon at the top to stop the code from running.

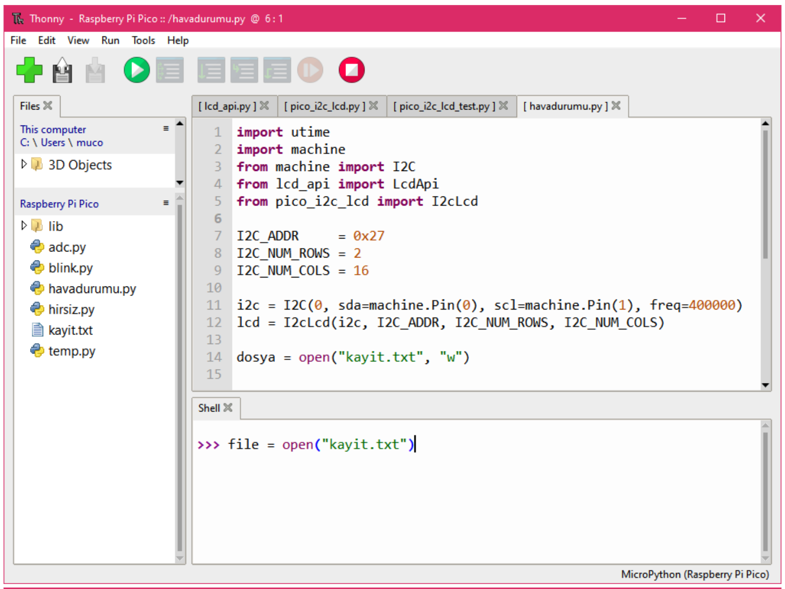


```
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")
```

```
>>> %Run -c $EDITOR_CONTENT
```

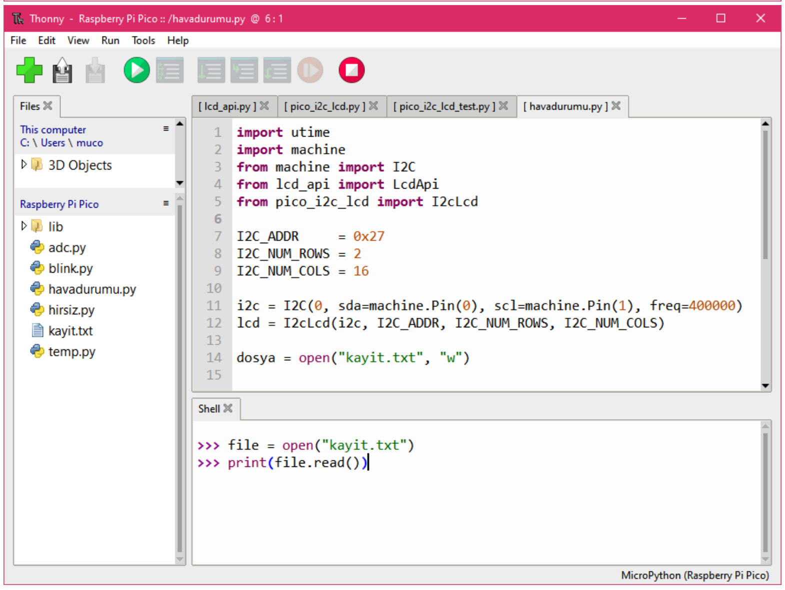
Write the following codes in the Shell section at the bottom so that you can see the data have saved.

```
1 file = open("kayit.txt")
2 print(file.read())
3 file.close()
```



```
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")
```

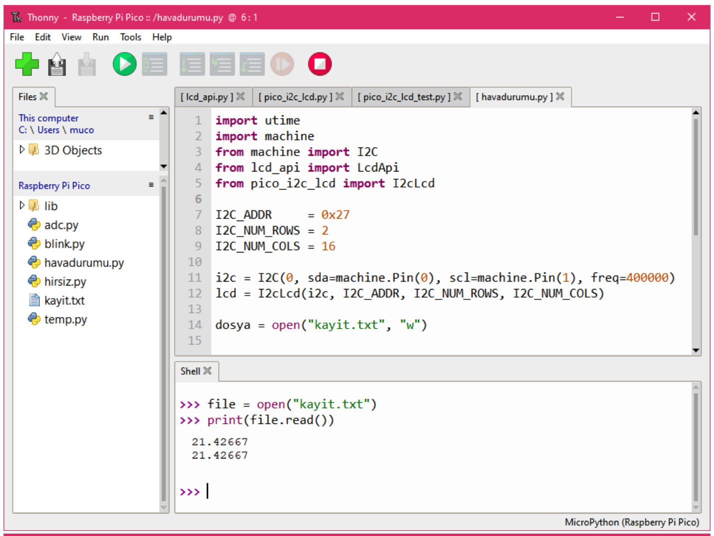
```
>>> file = open("kayit.txt")
```



```
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")
```

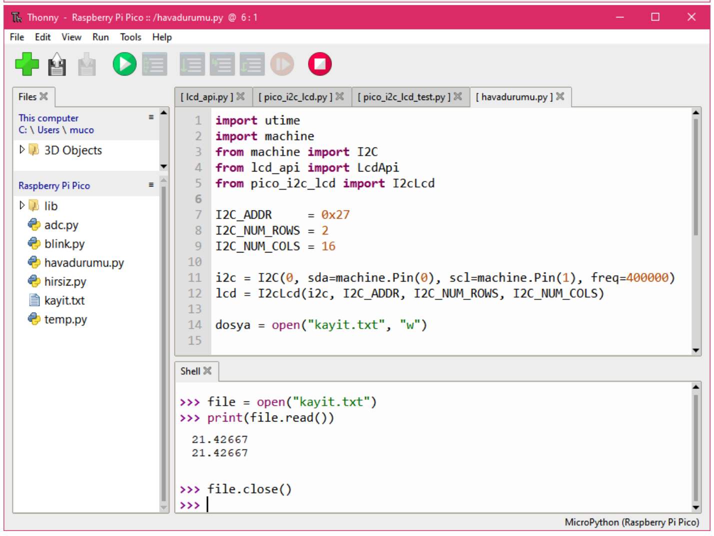
```
>>> file = open("kayit.txt")
>>> print(file.read())
```

After typing the "print (file.read ())" code, you can see the saved data as in the photo below.



```python
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")
```

```
>>> file = open("kayit.txt")
>>> print(file.read())
  21.42667
  21.42667

>>> 
```



```python
import utime
import machine
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd

I2C_ADDR     = 0x27
I2C_NUM_ROWS = 2
I2C_NUM_COLS = 16

i2c = I2C(0, sda=machine.Pin(0), scl=machine.Pin(1), freq=400000)
lcd = I2cLcd(i2c, I2C_ADDR, I2C_NUM_ROWS, I2C_NUM_COLS)

dosya = open("kayit.txt", "w")
```

```
>>> file = open("kayit.txt")
>>> print(file.read())
  21.42667
  21.42667

>>> file.close()
>>> 
```

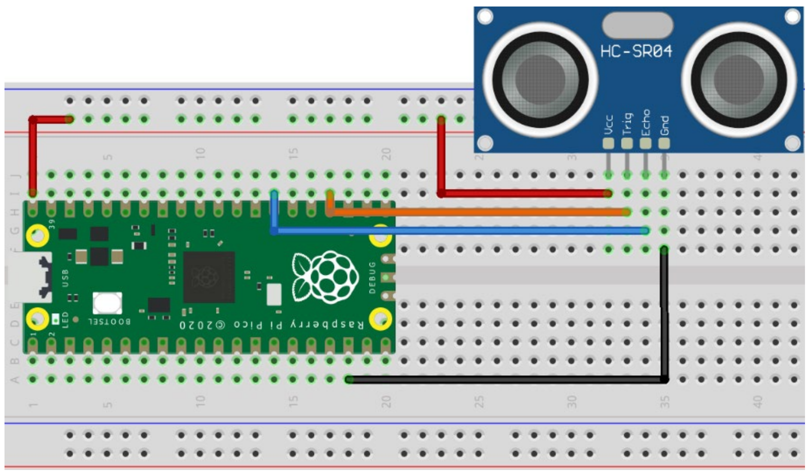
10- Using of Distance Sensor

In this project, you will measure distance with HC-SR04 distance sensor.

Necessary materials:

- Raspberry Pi Pico
- Breadboard
- HC-SR04 Distance Sensor
- Male – Male Jumper

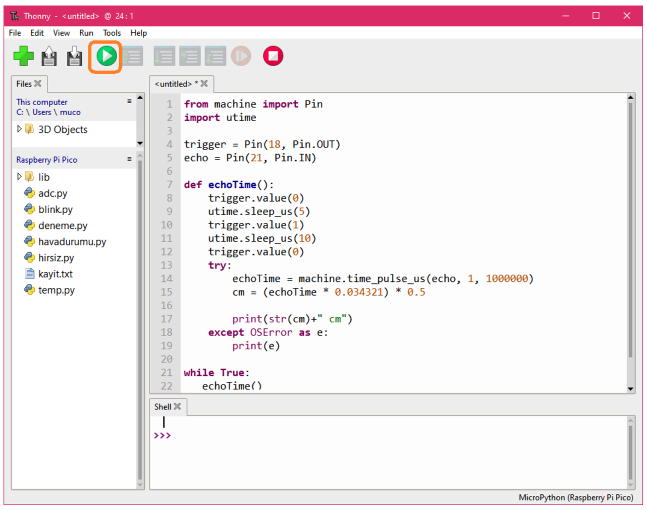
First, create your circuit on the breadboard according to the diagram below.



Now open a new file in Thonny IDE and write your code. When you examine the codes, you defined the libraries in the first two lines, and the pins to which the distance sensor is connected. Also wrote the distance measurement function with the function “def echoTime ():”. You have made the distance measurement function, which defined with the while loop, run every 5 seconds.

```python
from machine import Pin
import utime

trigger = Pin(18, Pin.OUT)
echo = Pin(21, Pin.IN)

def echoTime():
    trigger.value(0)
    utime.sleep_us(5)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)
    try:
        echoTime = machine.time_pulse_us(echo, 1, 1000000)
        cm = (echoTime * 0.034321) * 0.5

        print(str(cm)+" cm")
    except OSError as e:
        print(e)

while True:
    echoTime()
    utime.sleep(5)
```

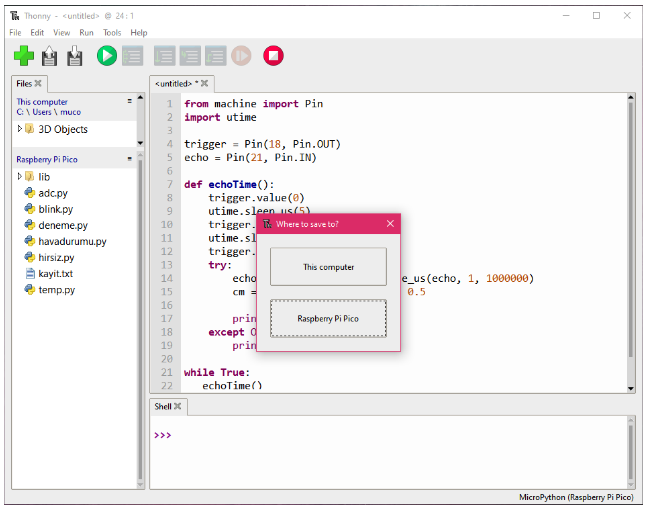
Now save these codes to your Pico and run them, then press the "Run" icon at the top.



```python
from machine import Pin
import utime

trigger = Pin(18, Pin.OUT)
echo = Pin(21, Pin.IN)

def echoTime():
    trigger.value(0)
    utime.sleep_us(5)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)
    try:
        echoTime = machine.time_pulse_us(echo, 1, 1000000)
        cm = (echoTime * 0.034321) * 0.5

        print(str(cm)+" cm")
    except OSError as e:
        print(e)

while True:
    echoTime()
```

```
>>>
```

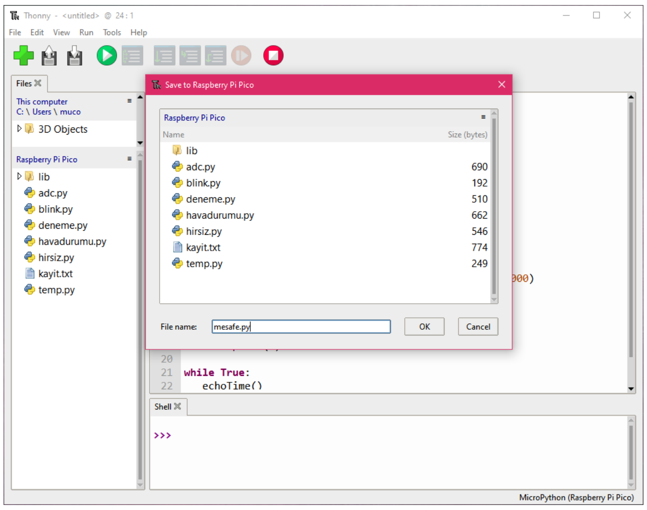
It asks where you want to save the code. For the code to always run, you need to click on the "Raspberry Pi Pico" option and save it to our Pico.



```
from machine import Pin
import utime

trigger = Pin(18, Pin.OUT)
echo = Pin(21, Pin.IN)

def echoTime():
    trigger.value(0)
    utime.sleep_us(5)
    trigger.
    utime.sl
    trigger.
    try:
        echo                                e_us(echo, 1, 1000000)
        cm =                                0.5

        prin
    except O
        prin

while True:
    echoTime()
```

```
>>>
```

Write "distance.py" as the file name and click the "OK" button.



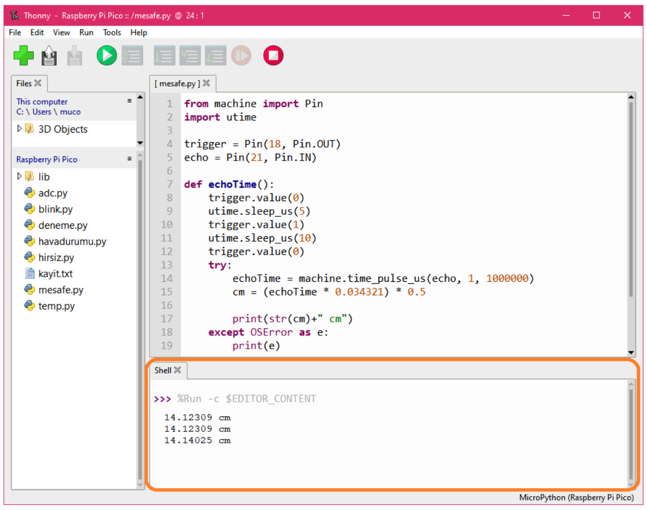
Name	Size (bytes)
lib	
adc.py	690
blink.py	192
deneme.py	510
havadurumu.py	662
hirsiz.py	546
kayit.txt	774
temp.py	249

File name: mesafe.py

```
while True:
    echoTime()
```

```
>>>
```

You can now see the distance in the Shell section of the Thonny IDE.



```
from machine import Pin
import utime

trigger = Pin(18, Pin.OUT)
echo = Pin(21, Pin.IN)

def echoTime():
    trigger.value(0)
    utime.sleep_us(5)
    trigger.value(1)
    utime.sleep_us(10)
    trigger.value(0)
    try:
        echoTime = machine.time_pulse_us(echo, 1, 1000000)
        cm = (echoTime * 0.034321) * 0.5

        print(str(cm)+" cm")
    except OSError as e:
        print(e)
```

```
>>> %Run -c $EDITOR_CONTENT
  14.12309 cm
  14.12309 cm
  14.14025 cm
```