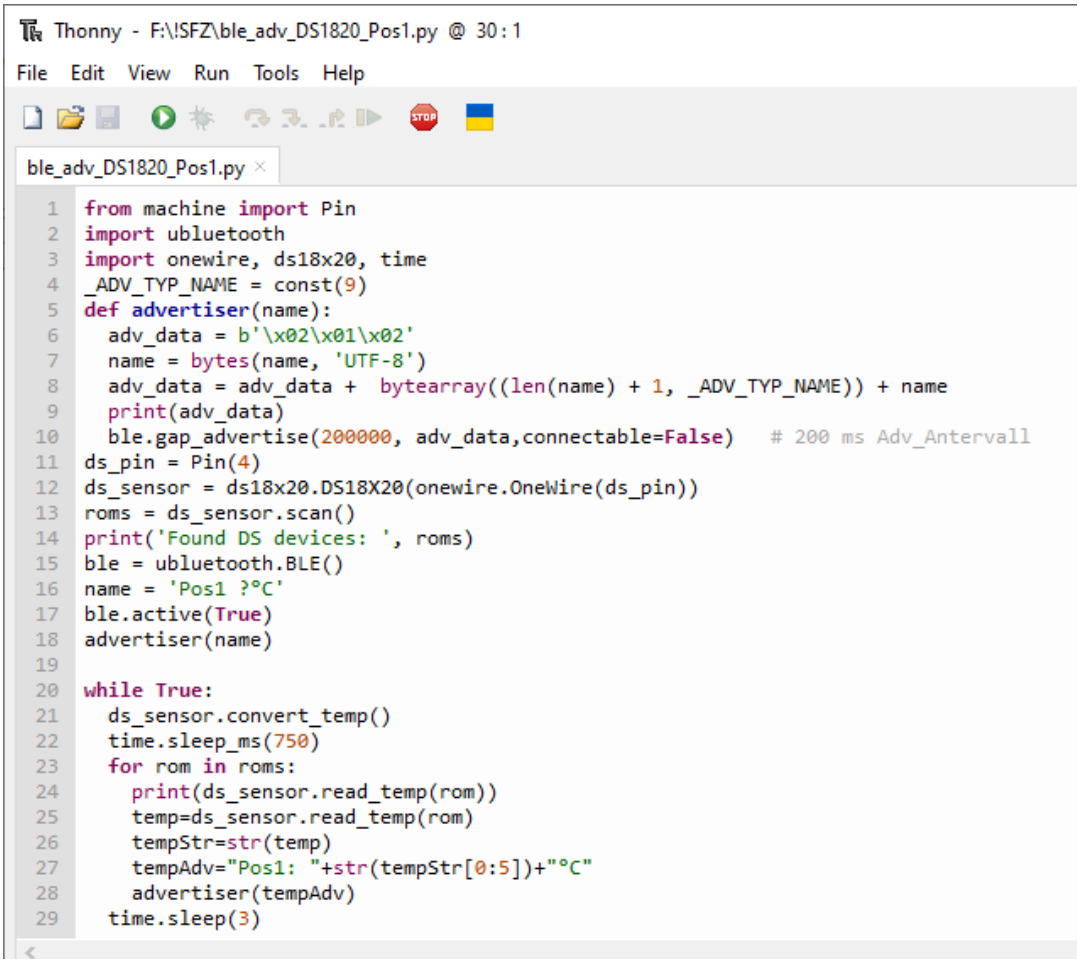
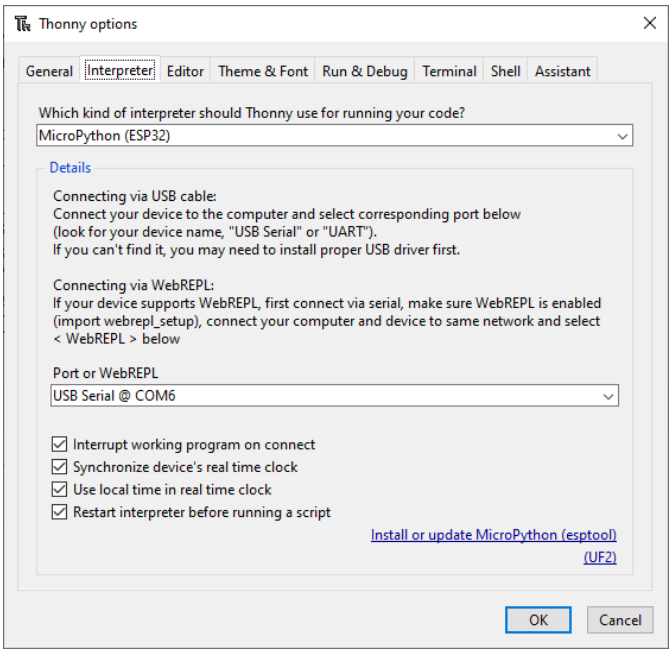
ble_adv_DS1820_Pos1.py:

Ein BLE-Thermometer, das seinen aktuellen Messwert so sendet, dass ein Smartphone mit BLE-Scanner ihn ohne weitere Software lesen kann

Der ESP32 wird so konfiguriert, dass er die vom angeschlossenen DS1820 gemessenen Temperaturwerte als BLE-Beacon-Namen sendet (advertised, "bewirbt"). Das kann bei schlecht zugänglichen Messpunkten hilfreich sein.

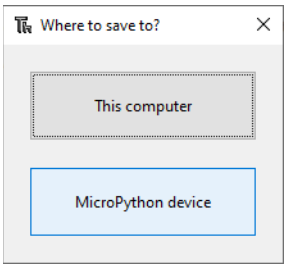


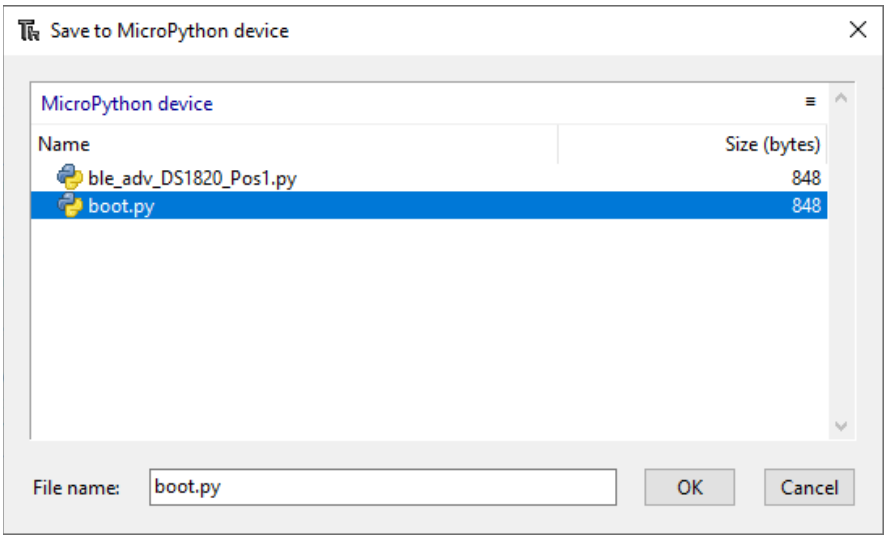

```
from machine import Pin
import ubluetooth
import onewire, ds18x20, time
_ADV_TYP_NAME = const(9)
def advertiser(name):
  adv_data = b'\x02\x01\x02'
  name = bytes(name, 'UTF-8')
  adv_data = adv_data +  bytearray((len(name) + 1, _ADV_TYP_NAME)) + name
  print(adv_data)
  ble.gap_advertise(200000, adv_data,connectable=False)   # 200 ms Adv_Antervall
ds_pin = Pin(4)
ds_sensor = ds18x20.DS18X20(onewire.OneWire(ds_pin))
roms = ds_sensor.scan()
print('Found DS devices: ', roms)
ble = ubluetooth.BLE()
name = 'Pos1 ?°C'
ble.active(True)
advertiser(name)

while True:
  ds_sensor.convert_temp()
  time.sleep_ms(750)
  for rom in roms:
    print(ds_sensor.read_temp(rom))
    temp=ds_sensor.read_temp(rom)
    tempStr=str(temp)
    tempAdv="Pos1: "+str(tempStr[0:5])+"°C"
    advertiser(tempAdv)
  time.sleep(3)
```



Die ESP-Programmierungen mittels Thonny und MicroPython ermöglichen es, den hochgeladenen Code auf dem Board zu lesen, zu ändern, zurückzuschreiben auf den ESP oder auf den Computer zu speichern:
File => Save as =>
Fenster "Where to save to?"
Auswahl:
This Computer
MicroPython device

Den Code für das autonome Laufen ohne Thonny-IDE auf dem ESP mit dem Namen "boot.py" speichern:





Den gleichen Code mit spezifischem Namen kopierbar und lesbar auf dem gleichen ESP zwecks Übersichtlichkeit speichern mit: