



# **RISPARMIO ENERGETICO - PROGETTO PON 2021-2022**

AMBROSINO LUCA  
CHIAPALE ASIA  
CUCCHIETTI GIOVANNI  
GUGNINO ALBERTO  
MUSSO LEONARDO

# MATERIALE

per la parte di misurazione e ricezione dei dati abbiamo utilizzato:

- 2 microbit alimentati a batteria (situati all'interno di un ambiente di misurazione)
- due microbit collegati al computer (per la ricezione dei dati)



- muEditor per la creazione del codice per la lettura dei dati
- python per la creazione del codice per il passaggio dei dati tra microbit



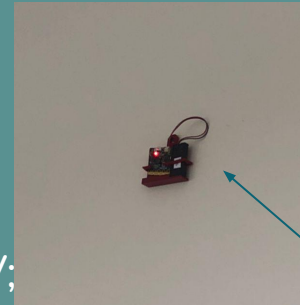
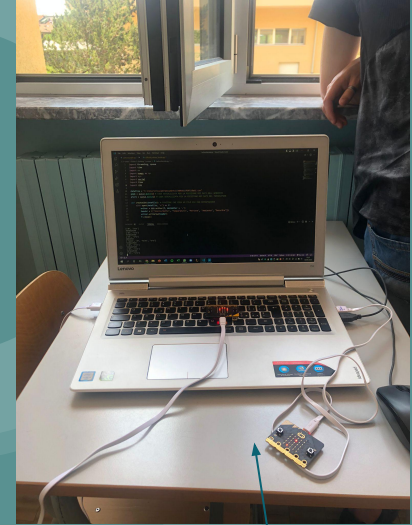
per la parte fisica del progetto abbiamo utilizzato:

- sketchup per la creazione dei supporti dei microbit
- la stampante 3d per la creazione fisica dei pezzi



# MANUALE DELL'UTENTE

- procurarsi quattro microbit: due alimentati a batteria e due che dovranno essere collegati a un pc tramite cavo;
- caricare i programmi “ambiente.py” e “termosifone.py” ai due microbit a batteria e “collegamentoAmbiente.py” e “collegamentoTermosifone.py” ai due microbit collegati al computer;
- posizionare i due microbit a batteria negli appositi supporti;
- appoggiare il microbit ambiente a un muro e quello termosifone accanto al termosifone;
- collegare gli altri due microbit al pc;
- far partire il programma “letturaSeriale.py” sul pc;
- attendere i risultati che verranno stampati su un file CSV;



microbit  
collegati al pc  
durante l'analisi

microbit  
ambiente  
appeso al muro



# **PARTE SOFTWARE - MICROBIT CHE PRENDE I DATI DALL'ESTERNO**

Musso Leonardo

# IL CODICE

Abbiamo creato due codici:

- Ambiente
- Temperatura

Il primo ha il compito di raccogliere i dati dell'ambiente circostante (Temperatura dell'ambiente, presenza di persone e luminosità).

Il secondo misura la temperatura dei termosifoni

Infine tutti e due invieranno i dati ad altri microbit via Radio



# AMBIENTE

```
# funzione per verificare se è giorno
def verifyDay():
    # try
    if display.read_light_level() < 100:
        d = False
    else:
        d = True
    return d
```

Funzione luminosità

```
# funzione che indica la presenza di persone
def findPeople():
    # se il microfono sente dei rumori
    if microphone.was_event(SoundEvent.LOUD):
        p = True
    else:
        p = False
    return p
```

Funzione presenza persone

```
# funzione per misurare la temperatura
def measureTemperature():
    t = temperature()
    return t
```

Funzione temperatura

```
radio.config(group = 1)
radio.on()
stringa = str(day)+';'+str(temp)+';'+str(people)+';'
radio.send(stringa)
sleep(1000)
```

Configurazione radio()

# Termosifone

```
# funzione per misurare la temperatura
def measureTemperature():
    t = temperature()
    return t
```

Funzione temperatura

```
radio.config(group = 2)
radio.on()
stringa =str(temp)+';'
radio.send(stringa)
sleep(1000)
```

Configurazione radio()

# INVIO E RICEZIONE DEI DATI

Per inviare e ricevere i dati abbiamo usato la libreria Radio() di microbit.

## INVIO DI DATI

Tramite la funzione `radio.config()` abbiamo configurato il canale di comunicazione. Abbiamo utilizzato la funzione `radio.on()` per attivare il canale e la funzione `radio.send()` per mandare la stringa di dati concatenata da `;`.

```
radio.config(group = 2)
radio.on()
stringa = str(temp) + ';'
radio.send(stringa)
sleep(1000)
```

## RICEZIONE DEI DATI

Tramite la funzione `radio.config()` abbiamo configurato il canale di comunicazione. Abbiamo utilizzato la funzione `radio.on()` per attivare il canale e la funzione `radio.receive()` per ricevere la stringa di dati concatenata da `;`.

```
message = radio.receive()
if message:
    print(message)
    display.show(message)
    sleep(100)
```





# **PARTE SOFTWARE - MICROBIT CHE PRENDE I DATI CON I THREAD**

Ambrosino Luca

# IL CODICE

Per permettere al computer di ricevere i dati dai microbit, abbiamo utilizzato la libreria serial di Python che permette di inviare i dati tramite cavo USB.

```
#serial config  
port = "COM8"  
s = serial.Serial(port)  
s.baudrate = 115200
```

Abbiamo creato due classi Thread, una per microbit, per gestire in parallelo i due flussi di dati.

```
class Read_Microbit_Termosifone(threading.Thread):
```

```
class Read_Microbit_Ambiente(threading.Thread):
```

Per gestire al meglio gli scambi di dati tra i microbit e il computer ed evitare conflitti tra i Thread abbiamo creato due code sincronizzate

```
qAmb = queue.Queue()
qTerm = queue.Queue()
```

Thread microbit ambiente:

```
def run(self):
    #serial config
    port = "COM8"
    s = serial.Serial(port)
    s.baudrate = 115200
    self._running = True
    while self._running:
        data = s.readline().decode()
        print("Termosifone")
        messaggio = [space for space in data.split(';')]
        qTerm.put(messaggio)
        print(messaggio)
        time.sleep(0.01)
```

Thread microbit termosifone:


```
def run(self):
    #serial config
    port = "COM5"
    s = serial.Serial(port)
    s.baudrate = 115200
    self._running = True
    while self._running:
        data = s.readline().decode()
        messaggio = [space for space in data.split(';')]
        print("Ambiente")
        qAmb.put(messaggio)
        print(messaggio)
        time.sleep(0.01)
```

## Metodi gestione File CSV:

```
def createCSV(dataFile):  
    with open(dataFile, 'w') as f:  
        writer = csv.writer(f, delimiter = ',')  
        header = (["Giorno/Notte", "Temperatura", "Persone", "Ambiente", "Data/Ora"])  
        writer.writerow(header)  
        f.close()  
  
def addData(dataFile, data):  
    with open(dataFile, 'a') as f:  
        writer = csv.writer(f, delimiter=';')  
        writer.writerow(data)  
        f.close()
```

Main del programma:

```
running = True
rmt = Read_Microbit_Termosifone()
rma = Read_Microbit_Ambiente()
rma.start()
rmt.start()
while running:
    messaggioAmb = qAmb.get()
    print(messaggioAmb)
    print("#####")
    messaggioTerm = qTerm.get()
    print(messaggioTerm)
    dataOra = str(time.strftime('%Y-%m-%d %H:%M:%S', time.localtime()))
    data = (messaggioTerm[0], messaggioTerm[1], messaggioTerm[2], messaggioAmb[0], dataOra)
    addData(dataFile, data)
rma.terminate()
rmt.terminate()
rmt.join()
rma.join()
```



# **PARTE HARDWARE - DESIGN E STAMPA DELLA STRUTTURA**

Cucchietti Giovanni  
Gugnino Alberto

# DESIGN

## Supporto scheda termosifone

**Questo supporto è stato progettato per posizionarsi su un qualsiasi termosifone tramite un doppio gancio che si inserisce all'interno dei fori superiori del termosifone.**

**Sul supporto, la scheda è collocata frontalmente verso il termosifone, in modo che riesca a percepire la corretta temperatura rilasciata da quest'ultimo.**

**Invece la batteria che alimenta il Micro Bit è posta sul retro del supporto per evitare rischi di surriscaldamento date le elevate temperature nei pressi del termosifone.**

# DESIGN

## Supporto rilevatore di presenza

Questo supporto è stato ideato per rilevare la temperatura ambientale, la presenza di persone e il livello di luminosità. E' stato realizzato per essere facilmente appeso ad un chiodo, in questo modo può essere collocato dove si preferisce.

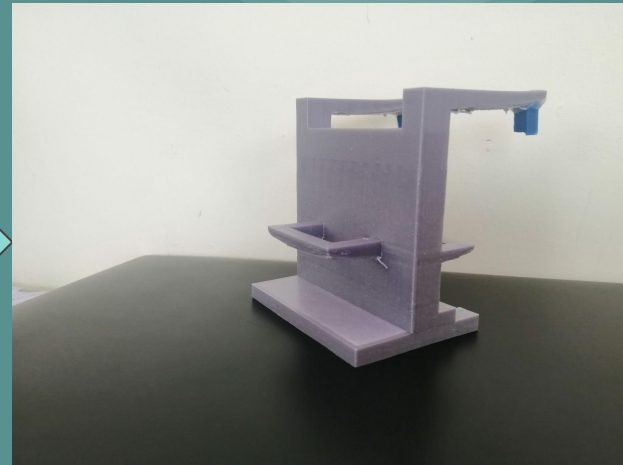
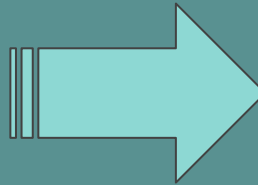
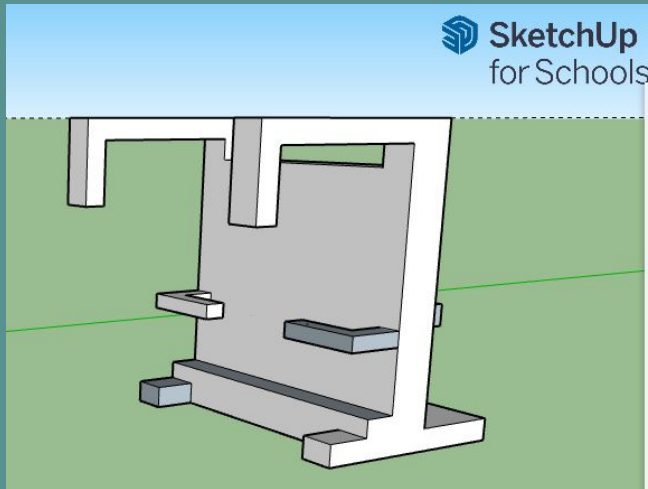
Il supporto è realizzato per stare vicino al muro, quindi sul retro è completamente piatto. Di fronte il pezzo si presenta con il microbit sulla facciata affiancata dalla batteria e un "asola" sopra di essi per essere appeso.



# STAMPA

## Supporto scheda termosifone

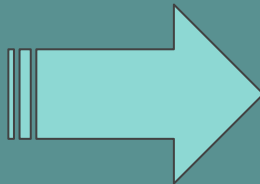
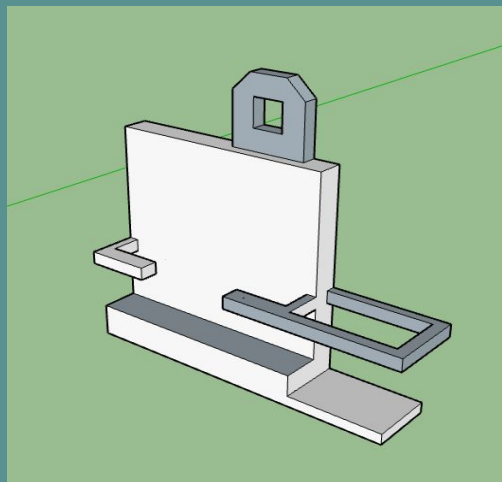
La stampa del supporto è stata realizzata per mezzo della stampante 3D MK3 PRUSA importando il progetto da CuraUltimaker ,software utile a convertire il progetto creato in formato “STL” su Sketch Up, in un file con estensione “gCode” ,leggibile dalla stampante. La stampa del pezzo è durata all'incirca 5 ore ma con un buon risultato finale.



# STAMPA

## Supporto rilevatore di presenza

La stampa del supporto è stata realizzata per mezzo della stampante 3D MK3 PRUSA importando il progetto da CuraUltimaker ,software utile a convertire il progetto creato in formato “STL” su Sketch Up, in un file con estensione “gCode” ,leggibile dalla stampante. La stampa del pezzo è durata 2 ore e 15 minuti ma con un buon risultato finale.



# **ANALISI CONCLUSIVA DEI DATI OTTENUTI**

