Four simple experiments to practice computational thinking

Hans-Martin Hilbig, July 2023

T³ DEUTSCHLAND

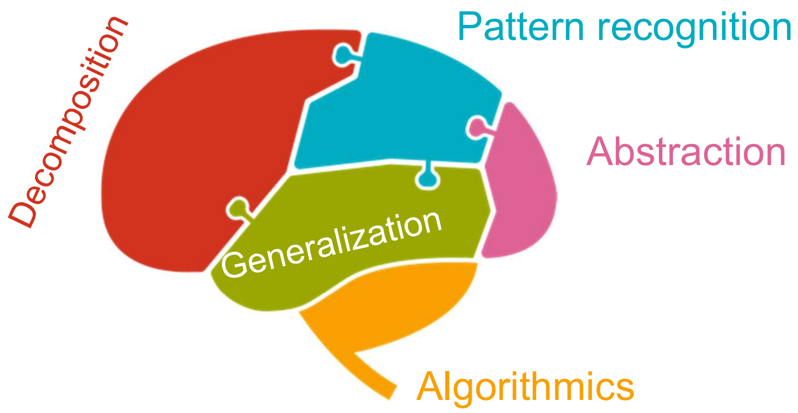
Teachers Teaching with Technology^{TM}

Overview

Many STEM projects are centered around coding. They require the knowledge of at least one coding language. They often involve many lines of code to achieve the solution of a problem. This all takes time, often exceeding the 45 minutes of a normal classroom lesson. In contrast, computational thinking does not require a lot of knowledge up front, as it is rather skill based than based on knowledge. It does require practice, however. This is what the four simple experiments of this project are aiming at.

What is computational thinking?

Computational thinking is anchored around five different skills [1]:



Obviously, computational thinking is important for creating efficient and effective code a computer, i.e. a machine, understands. But computational thinking also is important in the modern world of Artificial Intelligence (AI) where a large language model (LLM) eliminates the knowledge of a specific coding language, but still requires the user's ability to describe a problem in a structured and precise way the machine understands. Maybe computational thinking will become a more important skill than coding when it comes to human-machine interactions in the future.

Four simple experiments

All four experiments included in this project require only basic experience with using the TI-Nspire CXII Handheld. Coding experience in Python is helpful for those who like to understand what's happening in the background. All Python code provided with the experiments is based on TI-Nspire CXII and BBC micro:bit technology, just the micro:bit transmitter part of the Find & Rescue! experiment (exercise #4) requires the use of an open source Python editor (Thonny [3]) to achieve the required performance.
The four experiments to some extent build upon each other, but can be used as standalone classroom exercises as well.

Experiment #1: Computer Art

For students who have not worked with Python and the TI-Nspire CXII a lot, this is a good exercise to start with.

Objective:
- Create geometric objects on the screen of a Nspire CXII Handheld using simple functions of the turtle.tns module.

Hardware needed:
- TI-Nspire CXII Handheld

Software needed:
- turtle.tns module installed in the pylib library of the Nspire CXII Handheld
- exp_1_turtle.tns provided with this project [2]

Learning objectives:
- Get hands on experience with the Nspire CXII and the most basic functions of the turtle library
- Train the five computational thinking skills: Decomposition, Pattern Recognition, Algorithmics, Abstraction, Generalization
- Train geometric imagination
- Work with an object-oriented coding style, using the available methods of the turtle class.

Tasks:
1. my_project.py of exp_1_turtle.tns: Find the <move> functions sub-menu contained in the Turtle Graphics menu of the Nspire CXII Handheld, try them out and see what happens
2. square.py of exp_1_turtle.tns: Think about a square. Decompose the square into four separate lines. How would turtle have to move across the screen to draw a square?
3. polygon.py of exp_1_turtle.tns: Apply your learnings from 2 and do some Abstraction, Algorithmics and Generalization thinking.
   a. What would turtle need to know to draw a polygone?
   b. How would the number of edges of a polygon translate into the angular movements of turtle?
   c. Translate your conclusions into Python code and put it into a function structure (def polygon(a,b):).
   d. Call the function with different numbers of edges and side length of your polygon. What happens to the Nspire CXII screen if the side length of your polygon is exceeding the screen size of the CXII?
   e. Can you find out how many pixels (horizontally and vertically) a Nspire CXII screen is composed of?
4. rosette.py of exp_1_turtle.tns: Look at the code example provided here. It uses the polygon function in another function called rosette(), plus some color change and color fill commands of turtle. Play around with the parameters of the function and the code to find out what the individual functions do.
5. t_init.py of exp_1_turtle.tns: with all the visualization features switched on, it can become pretty time consuming to draw more complex graphics. The t_init.py module is switching off all the visualization features, to speed up the drawing process. You may use it in all your code by adding <from t_init import *>, followed by <turtleinit(t,s)>, where t is the name of your turtle object and s is the desired drawing speed (10=fastest, no argument defaults to 5=medium)

Experiment #2: Rover Slalom

With using the TI-Rover, the same basic skills from exercise #1, turtle graphics, are reused and applied to a real object, with all the pro's and con's involved: You can touch it, you can hear it. Its movement is subject to what has been coded, similar like turtle, but external, environmental factors come into play, like: it can fall off the table, its movements can be hampered by uneven or dirty surfaces, the Rover is not always placed at the exact same position, thus subsequent execution of the same code can produce different results.
In this experiment, no coding skills will be needed. You will input your coordinates in a spreadsheet-like window on the Nspire CXII mounted to a TI-Rover.

Objective:

- Create a list of x,y-coordinate pairs that will maneuver the TI-Rover through a slalom parcour, without falling off the table or touching any of the traffic cones positioned in equal distances on a straight line.

Hardware needed:

- TI-Nspire CXII Handheld, TI-Innovator Rover, three model traffic cones (approx 4 inches high) per student (team)

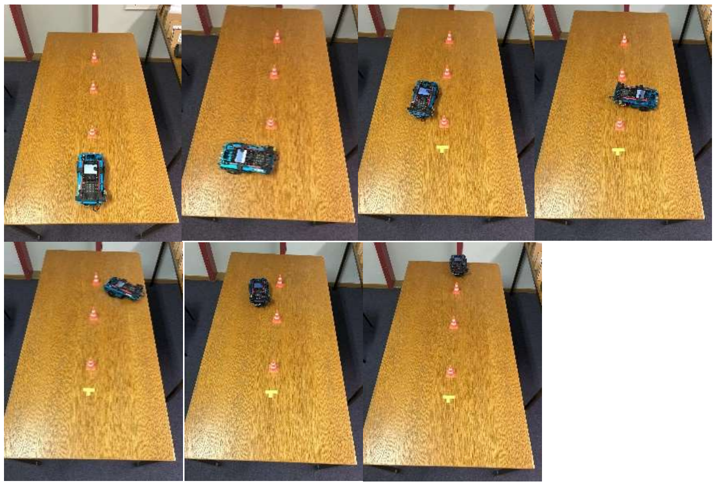
Software needed:

- exp_2_roverslalom.tns provided with this project

Learning objectives:

- Train the five computational thinking skills: Decomposition, Pattern Recognition, Algorithmics, Abstraction, Generalization
- Train geometric imagination with a real physical object (TI-Rover)
- Practice a trial & error method to come to a robust and repeatable result
- Work with an object-oriented coding style, using the available methods of the rv class.

Tasks:



1. Set up a slalom parcour on a table big enough to fit the entire track. See pictures above. Mark a starting point you will align your Rover to whenever you restart your code on the next trial & error round. Use the Rover as a ruler to position your three traffic cones. A spacing of 15 inches between the cones has worked well. Think about possible strategies to maneuver the TI-Rover thru the parcour. Is zick-zack a good strategy? Are orthogonal movements (i.e. 90 Deg turns) better? What are the pro's and con's of either approach?
2. Make sure your coordinate list in the lists & spreadsheet window is empty (i.e. blank cells, no numbers at all).
   As a starting point, experimentally find out how the coordinate system of the TI-Rover

works. Start with just one coordinate tuple (e.g. 1,0 or 0,1) to see where and how far the Rover moves.

3. Based on your findings from task #2 above, determine your first set of coordinate points to make the Rover move around the first traffic cone. Apply trial & error to refine your coordinate values. Make sure you always position the Rover precisely at its starting point markers, to make sure you get repeatable results.
4. Repeat task #3 above and work your way around all traffic cones, all the way to the finish line.
5. Compare your result with the approach your peer students took. What are the pro's and con's of the different solutions? Which one is most reliable and repeatable?

Experiment #3: Drone mission

For those who do experiment #3 right after experiment #2, Drone mission is just adding a third dimension to the 'control a real object' challenge. The Tello library is object oriented coding style like the Rover library, expanded by methods only a drone can perform in 3D space.

Objective:

- Get familiar with the basic motion control methods of the tello library to maneuver a Tello drone in 3D space.

Hardware needed:

- TI-Nspire CXII Handheld
- Ryze Robotics Tello Edu drone
- BBC micro:bit V2 or better
- micro:bit Bitmaker expansion board or similar
- Grove WiFi module
- External battery

Software needed:

- exp_3_drone_mission.tns provided with this project
- Getting started with Tello drone documentation [4]
- tello.tns library installed in your pylib folder

Learning objectives:

- Train the five computational thinking skills: Decomposition, Pattern Recognition, Algorithmics, Abstraction, Generalization
- Train geometric imagination in 3D with a real physical object (Tello drone)
- Get familiar with the basic operating principles using Nspire CXII with BBC micro:bit and Tello hardware
- learn about head-lock-style and FPV-style maneuvers with a drone
- Work with an object-oriented coding style, using the available methods of the tello class.

Tasks:

1. Position your drone at a safe distance in front of you, with its head facing away from you.
   Execute the takeoff_land.py example provided in exp_3_drone_mission.tns. This is a very basic code to ensure:
   a. All hardware has been assembled and cabled correctly
   b. A Tello drone has previously been paired with the BBC micro:bit module you are using
   c. Communication between all hardware parts – Nspire CXII, micro:bit, expansion board, WiFi module and Tello drone is working properly
   d. The Tello drone is operational and performs a simple take-off and landing method.
2. The tello_square_headlock.py code demonstrates how a drone can fly a square with a 20cm-side length. Feel free to increase side length as classroom space allows. Position your drone at a safe distance in front of you, with its head facing away from you. Make sure you have ample space in front and to the left, as the drone will fly 20cm forward and 20cm to the left to complete its square mission. Execute the code. Observe the drone heading will always remain in the same direction, facing away from you. This is called a headlock flying style.
3. The tello_square_fpv.py code performs basically the same flight pattern like (2) above, but the Tello drone will change its heading around its vertical axis at each corner of the square. Feel free to increase side length as classroom space allows. Position your drone at a safe distance in front of you, with its head facing away from you. Make sure you have ample space in front and to the left, as the drone will fly 20cm forward and 20cm to the left to complete its square mission. Execute the code.

Observe the drone heading will change at the corners of the square, facing towards its flight path. This is called a FPV (first person view) flying style.

4. Discuss the headlock flying against the FPV flying style. What does the drone front camera ‘see’ in each flying style? What are the pro’s and con’s? When would you prefer headlock flying over FPV flying and vice versa?

Experiment #4: Find & Rescue!

Experiment #4 simulates a scenario where somebody is buried under an avalanche or under debris after an earthquake. Fortunately, the buried person has a BBC micro:bit Bluetooth transmitter at hand. This transmitter periodically (about every 1.5s) sends the peak noise level received by its microphone.
Each student has a BBC micro:bit Bluetooth receiver which can receive the data sent by the buried person's transmitter, as long as the receiver is within reach of the transmitter's beacon. When there is no beacon received, a 'no signal!' message is displayed on the students Nspire CXII screen. Otherwise, the signal strength received from the transmitter as well as the detected peak noise level during the recent 1.5s interval is displayed on each of the Nspire CXII screens.

Objective:

- Act as a team, decide on a collaborative strategy to find the buried micro:bit transmitter as quickly as possible!

Hardware needed:

- One TI-Nspire CXII Handheld per student
- One BBC micro:bit V2 per student (receiver) with a micro USB cable to connect to a Nspire CXII
- One BBC micro:bit V2 with a microbit.hex file onboard and visible in the Thonny [3] MicroPython IDE (integrated development environment) application. (transmitter)
- One external battery (2 AAA cells connected to the external power connector of the micro:bit transmitter or a USB battery connected to the USB port of the micro:bit transmitter)

Software needed:

- exp_4_find_rescue.tns provided with this project
- microbit.tns module version 3.4.0 or better installed in pylib library
- ti_runtime_xxx.hex firmware version 3.2.0 or better installed on each micro:bit receiver board
- Thonny application [3]
- A non-TI microbit.hex file installed on the micro:bit transmitter board

Learning objectives:

- Solve a problem as a team, not as competitors
- Understand how signal strength of a radio transmitter is influenced by proximity, antenna orientation and obstacles of different materials
- Understand how discrete/momentary noise can be captured by fast sampling the micro:bit's microphone

Tasks:

1. Without the students in the room, the teacher hides (buries) a micro:bit connected to a battery, executing the transmitter Python main.py code flashed on board by the Thonny application before. See details about how to use Thonny below.
2. The teacher explains the technical background:
    - **a.** Received Signal Strength (RSS) is a measure of the proximity of a given transmitter from a given receiver. This measure can be used to obtain information about how close each student is to the buried transmitter.
    - **b.** Obstacles (doors, walls, sheet metal, wood, debris, snow) are attenuating radio signals, resulting in lower signal strength compared to an unobstructed (line of sight) environment between transmitter and receiver. A weak RSS value might be caused by an obstacle attenuating the radio signal of the transmitter. So there is no absolute measure of RSS values vs. proximity, but rather a relative measure of one student standing closer to the transmitter than others.

   c. Antenna orientation of the receiver relative to the transmitter influences RSS as well. RSS value will be lower (i.e. more negative) if the receiver antenna orientation is orthogonal to the transmitter antenna orientation.

   d. The microphone of the transmitter is acting as a sensor for noise. Ambient noise, such as students talking, music, machines, etc. will be measured as a continuous noise floor, which will not provide a useful peak level to locate the buried transmitter. A quiet environment with a minimal noise floor will be a good base for capturing discrete noise as a peak level standing out.

   e. Repeatable sources of discrete noise are best suited for a meaningful result of the peak loudness level measured by the transmitter's microphone. A knock or a hand clap creates a sharp discrete noise. If students are spread all over the classroom, with one student knocking at a time, the resulting peak loudness level sent from the buried transmitter is a good indicator of the proximity of the student having knocked to the buried transmitter.

   f. To be able to catch such short, sharp discrete loudness signals, the Python code of the transmitter micro:bit must be fast enough to sample the microphone sensor. For that reason, the code needs to run on the micro:bit itself and not on the Nspire CXII handheld. The microbit.hex firmware used along with the Thonny software will provide a high-speed sampling of up to 20,000 samples per seconds (20kS/s) and the peak noise provided by a knock will be caught by the Python code.

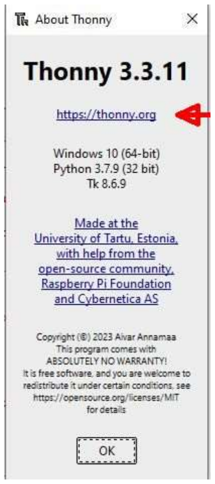
3. After having explained the technical background to the students, let the students solve the search & rescue challenge on their own. Measure the time till the transmitter was found.

4. Discuss with the students about what went well, what went wrong, what could have been done better to find the buried transmitter quickly.

A quick guide to use Thonny and microbit.hex with the BBC micro:bit module

Thonny is an integrated development environment to use MicroPython code with standalone microcontroller boards. Similar to the TI-Nspire desktop, Thonny provides the coding, editing and debugging infrastructure to develop MicroPython code. Thonny has been developed by the University of Tartu, Estonia and is an open source platform. Here is a simple step-by-step instruction about how to store the transmitter Python code on a micro:bit board:

1. Label a micro:bit board with a 'Thonny' mark in order to distinguish the transmitter board from all the student receiver boards you have on hand.
2. Download the Thonny IDE from the Thonny website shown here:



About Thonny

Thonny 3.3.11

https://thonny.org

Windows 10 (64-bit)
Python 3.7.9 (32 bit)
Tk 8.6.9

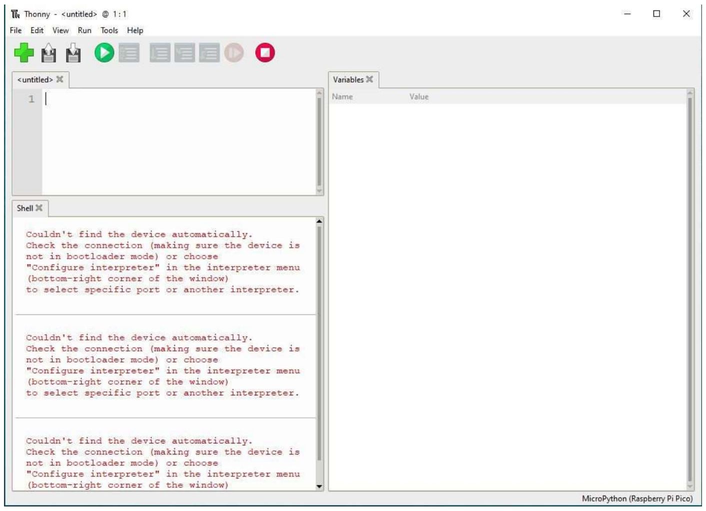
Made at the
University of Tartu, Estonia,
with help from the
open-source community,
Raspberry Pi Foundation
and Cybernetica AS

Copyright (©) 2023 Aivar Annamaa
This program comes with
ABSOLUTELY NO WARRANTY!
It is free software, and you are welcome to
redistribute it under certain conditions, see
https://opensource.org/licenses/MIT
for details

OK

3. Connect the micro:bit board to your PC. A Windows Explorer window should appear, showing the micro:bit board as a USB memory device.

4. Start Thonny. Initially, Thonny may show a screen like this:

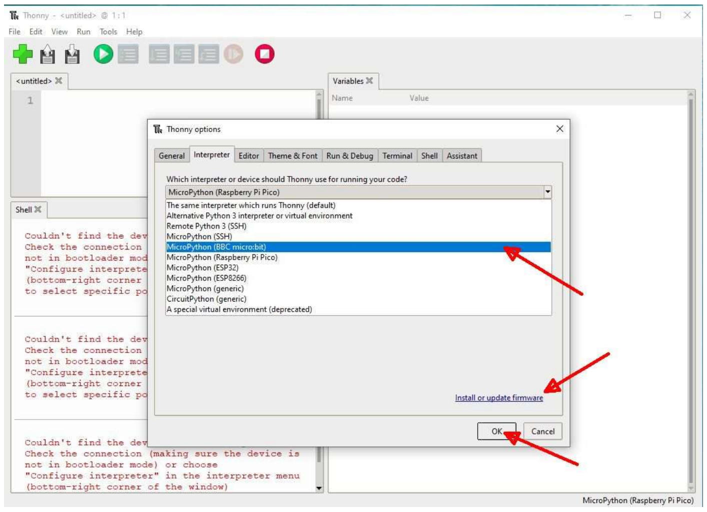


```
Couldn't find the device automatically.
Check the connection (making sure the device is
not in bootloader mode) or choose
"Configure interpreter" in the interpreter menu
(bottom-right corner of the window)
to select specific port or another interpreter.

Couldn't find the device automatically.
Check the connection (making sure the device is
not in bootloader mode) or choose
"Configure interpreter" in the interpreter menu
(bottom-right corner of the window)
to select specific port or another interpreter.

Couldn't find the device automatically.
Check the connection (making sure the device is
not in bootloader mode) or choose
"Configure interpreter" in the interpreter menu
(bottom-right corner of the window)
```

MicroPython (Raspberry Pi Pico)

Look at the lower right of the Thonny window. In the example above, Thonny was expecting a Raspberry Pi Pico board, but didn't find such.

5. Click onto that text message on the lower right of the Thonny window:


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
The same interpreter which runs Thonny (default)
Alternative Python 3 interpreter or virtual environment
Remote Python 3 (SSH)
MicroPython (SSH)
MicroPython (BBC micro:bit)
MicroPython (Raspberry Pi Pico)
MicroPython (ESP32)
MicroPython (ESP8266)
MicroPython (generic)
CircuitPython (generic)
A special virtual environment (deprecated)
Install or update firmware
OK
Cancel

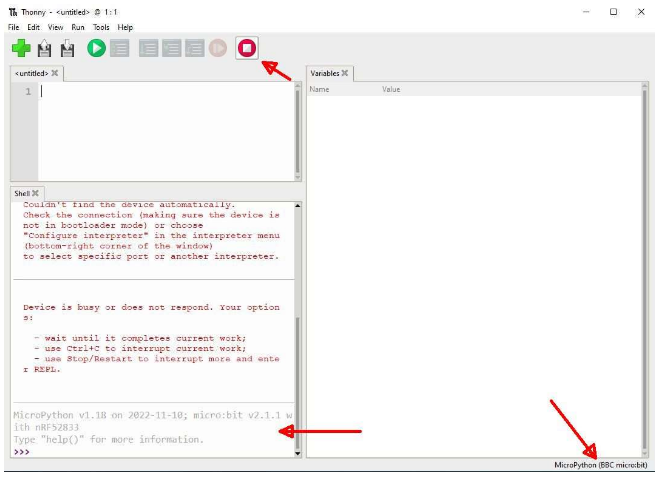
Which interpreter or device should Thonny use for running your code?

MicroPython (Raspberry Pi Pico)

MicroPython (Raspberry Pi Pico)

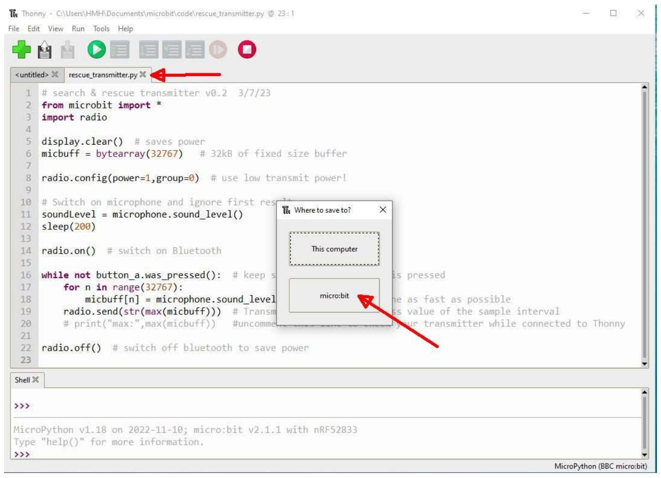
From the menu appearing, select the BBC micro:bit board. If there hasn't been a microbit.hex file installed on the board, select 'Install or update Firmware'. Click 'ok'.

6. Press the red ‘stop’ button in the menu bar of Thonny. Thonny now should find the micro:bit board and display the correct board in both, the Shell window and the lower right hand corner of Thonny.


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
<untitled>
Variables
Name
Value
Shell
MicroPython (BBC micro:bit)

```
Couldn't find the device automatically.
Check the connection (making sure the device is
not in bootloader mode) or choose
"Configure interpreter" in the interpreter menu
(bottom-right corner of the window)
to select specific port or another interpreter.


Device is busy or does not respond. Your option
s:

  - wait until it completes current work;
  - use Ctrl+C to interrupt current work;
  - use Stop/Restart to interrupt more and ente
r REPL.

MicroPython v1.18 on 2022-11-10; micro:bit v2.1.1 w
ith nRF52833
Type "help()" for more information.
>>>
```

7. Having both, Thonny and the TI-Nspire desktop application open on your PC, copy the transmitter MicroPython code from the radio_xmit_thonny.py tab of the exp4_find_rescue.tns file in the TI-Nspire desktop into the ‘untitled’ editor window of Thonny:

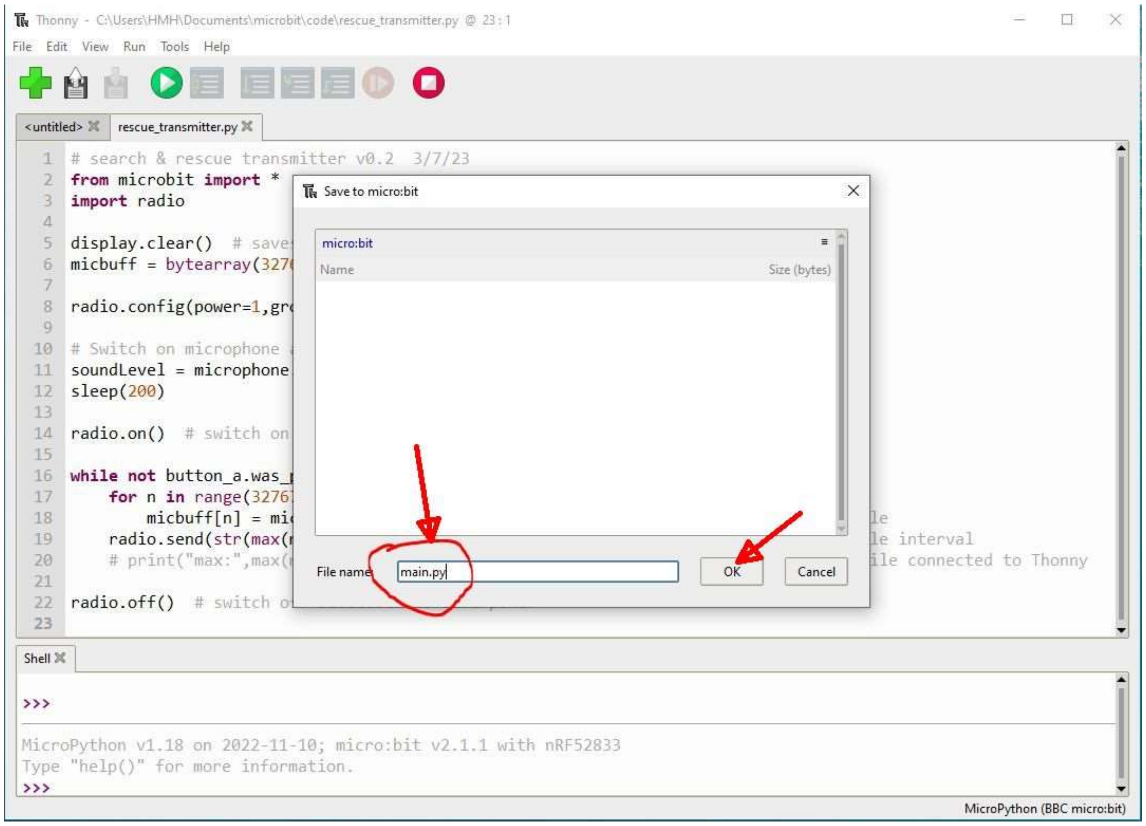

Thonny - C:\Users\HMH\Documents\microbit\code\rescue_transmitter.py @ 23 : 1
File Edit View Run Tools Help
<untitled>
rescue_transmitter.py
Where to save to?
This computer
micro:bit
Shell
MicroPython (BBC micro:bit)

```
# search & rescue transmitter v0.2  3/7/23
from microbit import *
import radio

display.clear()  # saves power
micbuff = bytearray(32767)   # 32kB of fixed size buffer

radio.config(power=1,group=0)  # use low transmit power!

# Switch on microphone and ignore first res
soundLevel = microphone.sound_level()
sleep(200)

radio.on()  # switch on Bluetooth

while not button_a.was_pressed():  # keep s                is pressed
    for n in range(32767):
        micbuff[n] = microphone.sound_level                 e as fast as possible
    radio.send(str(max(micbuff)))  # Transm                ss value of the sample interval
    # print("max:",max(micbuff))   #uncomme                ur transmitter while connected to Thonny

radio.off()  # switch off bluetooth to save power
```

```
>>>

MicroPython v1.18 on 2022-11-10; micro:bit v2.1.1 with nRF52833
Type "help()" for more information.
>>>
```

Select ‘save as’ from the Thonny file menu and press micro:bit to store the code onto the micro:bit board.

8. Make sure you type the entire filename, including .py extension and save the code as ‘main.py’ on the micro:bit. This will ensure that the transmitter MicroPython code will start as soon as the micro:bit is connected to a power source (USB port or battery pack):


Thonny - C:\Users\HMH\Documents\microbit\code\rescue_transmitter.py @ 23 : 1
File Edit View Run Tools Help
<untitled>
rescue_transmitter.py
Save to micro:bit
micro:bit
Name
Size (bytes)
OK
Cancel
Shell
MicroPython (BBC micro:bit)

```
# search & rescue transmitter v0.2  3/7/23
from microbit import *
import radio

display.clear()  # save
micbuff = bytearray(327

radio.config(power=1,gr

# Switch on microphone
soundLevel = microphone
sleep(200)

radio.on()  # switch on

while not button_a.was_
    for n in range(3276
        micbuff[n] = mi                                  le
    radio.send(str(max(                                  le interval
    # print("max:",max(                                  ile connected to Thonny

radio.off()  # switch o

```

File name: main.py

```
>>>
MicroPython v1.18 on 2022-11-10; micro:bit v2.1.1 with nRF52833
Type "help()" for more information.
>>>
```

9. Disconnect the micro:bit board from your PC. Connect a power source, e.g. a powerbank to the USB port of the micro:bit or use a AAA battery pack connected to the battery port of the micro:bit. The red LED next to the USB port should turn on. The transmitter now periodically sends the peak noise level sampled from the microphone every 1.5 seconds.

10. Start the radio_rcv.py code on a Nspire CXII connected to a micro:bit board containing the ti_runtime_xxx.hex firmware. Every 1.5 seconds, you should receive the RSS level and a peak loudness level displayed on the Nspire CXII screen.

11. Check out the entire setup by moving the receiver micro:bit away from the transmitter and making some noise. The RSS and noise level values should change accordingly.

Summary:

These four simple experiments should demonstrate that there is no much coding needed to practice computational thinking in the classroom. At the same time, these experiments demonstrate the depth and the breadth of experiments coded in MicroPython using the TI-Nspire and BBC micro:bit platforms.
Please provide feedback, suggestions and ideas to hm-hilbig@web.de.

Happy coding!

Sources:

[1] Credits go to Bert Wikkerink, Cathy Baars & colleagues from T3 Netherlands

[2] Credits of some of the code used in Turtle goes to Veit Berger from T3 Germany

[3] https://thonny.org

[4] Make coding take flight: https://education.ti.com/en/product-resources/tello-drone

T³ DEUTSCHLAND

Teachers Teaching with Technology™