Lecture 2

Single-board Microcontroller (I)

Artificial Intelligence of Things (SWS3025)
NUS SoC Summer Workshop 2024

Lecturer: A/P TAN Wee Kek

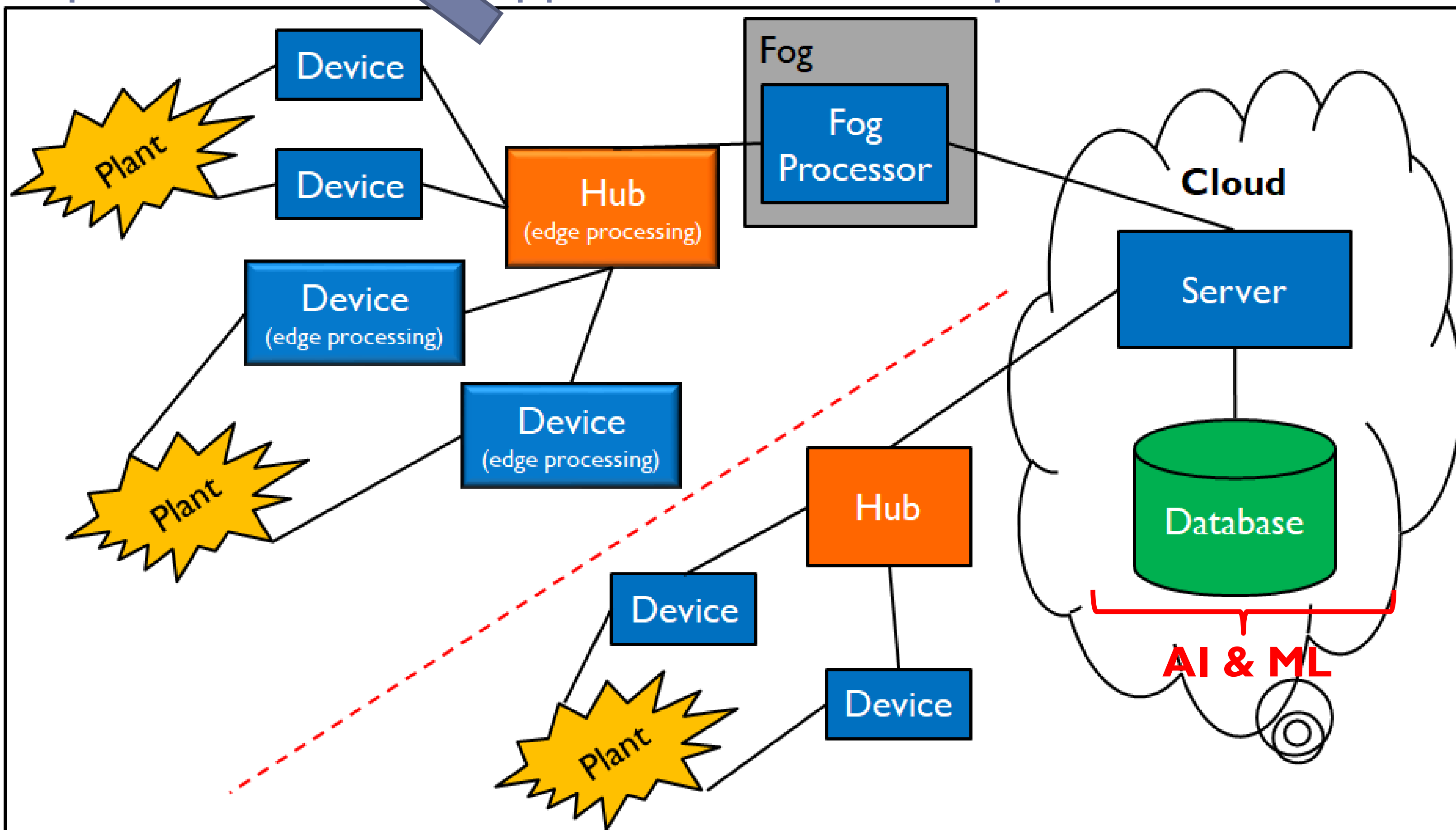
Email: tanwk@comp.nus.edu.sg :: **WeChat** :: tanweekek

Learning Objectives

- At the end of this lecture, you should understand:
  - Overview of single-board microcontrollers.
  - Overview of the micro:bit.
  - Technical characteristics and features of micro:bit.
  - Programming micro:bit with blocks and JavaScript.
  - Working with micro:bit onboard sensors.
  - Working with micro:bit computational and communication capabilities.

Technical Roadmap

Single-board Microcontroller | Single-board Computer | Server-side Backend Integration
Android Wear | Android App



Overview of Single-board Microcontrollers

- **A single-board microcontroller:**
  - Is built on a single printed circuit board.
  - Contains all the circuitry necessary to perform a useful control task:
    - A microprocessor.
    - I/O circuits.
    - A clock generator.
    - RAM.
    - Stored program memory.
    - Any other necessary support ICs (integrated circuits).
  - Can be used immediately by an application developer without the need to develop own controller hardware:
    - Save time and effort.

Overview of Single-board Microcontrollers (cont.)

- Microcontrollers can be used to automate the controlling of various products and devices:
  - E.g., automobile engine control systems, remote controls, office machines, appliances, power tools, toys and other embedded system.
- By using microcontrollers, we can reduce the size and cost of devices compared to alternative designs that are based on a separate microprocessor.
- Microcontrollers make it economical to digitally control many devices and processes:
  - Including IoT sensors and actuators.

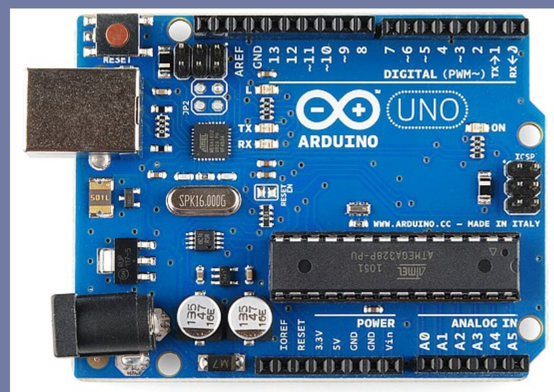
Overview of Single-board Microcontrollers (cont.)

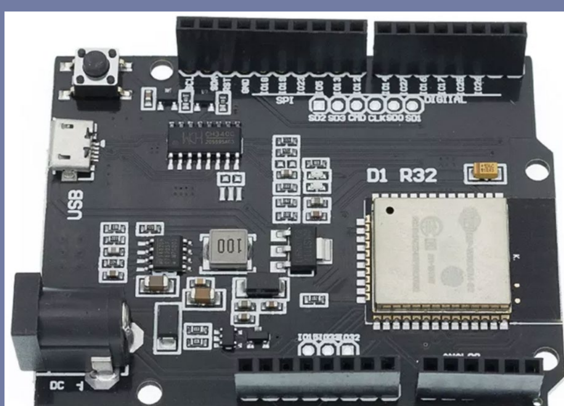
- Microcontroller is different from a single-board <u>computer</u>:

Property	Single-board Microcontroller	Single-board Computer
Interface	• No	• Yes • It has an interface that you can access by plugging it into a monitor of some kind.
Computational Capability	• You write a program on a computer and upload just the code to the board. • It has the capacity to store and run only one program at a time. • But can be reprogrammed as many times as you like.	• It has a full-fledged operating system, e.g., Raspberry Pi OS and Windows for IoT. • Capable of multiprogramming, multitasking and multithreading.

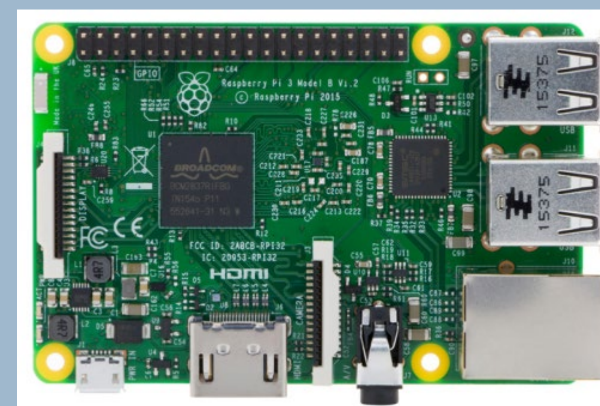
Overview of Single-board Microcontrollers (cont.)

- Single-board microcontrollers (left) versus single-board computer (right).





micro:bit (V1), Arduino UNO and D1 R32 (clockwise)

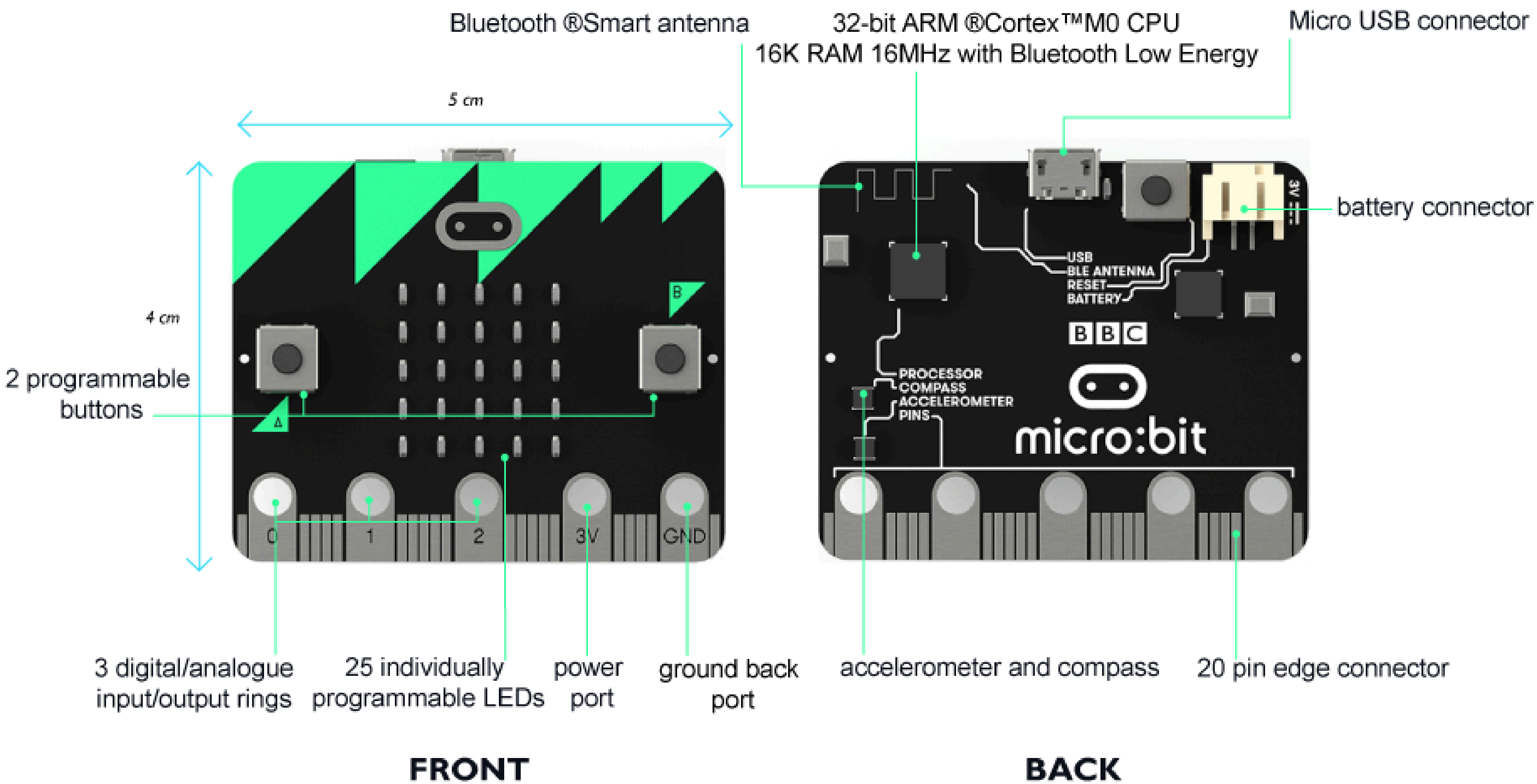


Raspberry Pi

Overview of micro:bit

- **micro:bit** is an ARM-based microcontroller designed by the BBC for use in computer education in the UK:
  - A.k.a. BBC Micro Bit, stylised as **micro:bit**.
  - ARM stands for Advanced RISC Machine.
  - RISC stands for reduced instruction set computing.
  - Processors with RISC architecture typically require fewer transistors than those with a complex instruction set computing (CISC) architecture.
  - RISC improves cost, power consumption and heat dissipation.
  - Good for light, portable and battery-powered devices:
    - E.g., smartphones, laptops and tablet computers, and other embedded systems.

Overview of micro:bit (cont.)



Overview of micro:bit (cont.)

- The board of micro:bit is 4 cm x 5 cm, described as "half the size of a credit card":
  - Contains an ARM Cortex-M0 processor.
  - On board motion (accelerometer, compass, magnetometer), temperature and light sensors.
  - Wired connectivity via micro-USB.
  - Wireless connectivity via Bluetooth and Radio.
  - On board output display consisting of 25 LEDs.
  - On board input via two programmable buttons (A and B).
  - Powered by either USB or an external battery pack (2 x AAA).
  - Device inputs and outputs are done through five ring connectors that are part of the 25-pin edge connector.

Features of micro:bit

- 25 individually-programmable LEDs:
  - Can display text, numbers and images.
  - Uses a scrolling interface.
- 2 programmable buttons:
  - Buttons are labelled as A and B.
  - Can detect when these buttons are pressed.
  - These buttons can be used to trigger code on the device.
- Physical connection pins:
  - There are 25 external connections or "pins" on the edge connector (5 large pins and 20 small pins).
  - Can program LEDs, motors or other electrical components with the pins, or connect extra sensors.

Features of micro:bit (cont.)

- Light sensor:
  - Uses some of the LEDs on the LED screen to measures the ambient light (how bright or dark it is).
  - Light level 0 means darkness and 255 means bright light.
- Temperature sensor:
  - Measures the current ambient temperature, in degrees Celsius.
- Accelerometer:
  - Measures the acceleration of the micro:bit.
  - Senses when the micro:bit is moved.
  - It can also detect other actions, e.g., shake, tilt and free-fall.

Features of micro:bit (cont.)

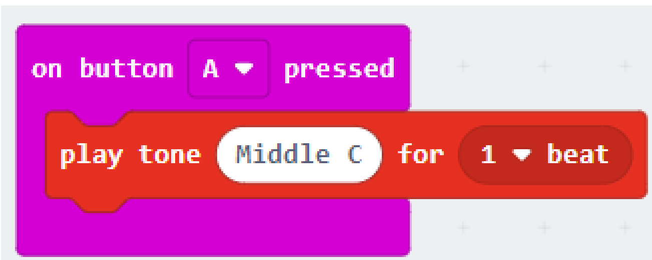
- Compass:
  - Detects the earth's magnetic field, and thus which direction the micro:bit is facing.
  - The compass has to be calibrated before it can be used to ensures the compass results are accurate.
- Radio:
  - Allows micro:bits to communicate wirelessly among themselves.
  - Use the radio to send messages to other micro:bits, build multiplayer games and implement other use cases.

Features of micro:bit (cont.)

- BLE (Bluetooth Low Energy) antenna:
  - Allows the micro:bit to send and receive Bluetooth signals.
  - Performs wireless communication with single-board computers, personal computers, phones and tablets:
    - E.g., can control micro:bit from a phone and send code wirelessly to the device from the phone.
  - Before using the Bluetooth antenna, need to pair micro:bit with the other device.
- USB interface:
  - Connect the micro:bit to a computer via a micro-USB cable.
  - Power the device and download program onto the device.

Programming the micro:bit

- **Microsoft MakeCode Editor:**
  - JavaScript Blocks Editor that supports visual programming (blocks to JavaScript) – https://makecode.microbit.org




```
input.onButtonPressed(Button.A, function () {
    music.playTone(262, music.beat(BeatFraction.Whole))
})
```

src00.js

  - MakeCode (based on PXT) compiles the JavaScript to ARM Thumb assembly, then links it against a pre-compiled .hex file of mbed + micro:bit runtime.
  - The .hex file is then downloaded to your micro:bit for flashing.

Programming the micro:bit (cont.)

- **Python Editor**:
  - Python is a very popular high-level programming language.
  - MicroPython is a lean version of Python specifically designed to run on microcontrollers (like the ARM Cortex-M0 on the micro:bit).
  - There are several ways to code in Python on the micro:bit:
    - Latest version of Microsoft MakeCode Editor provides integrated language support for blocks to Python – https://makecode.microbit.org
    - Standalone Python Editor on the official micro:bit website – https://python.microbit.org/v/beta
    - MakeCode Editor uses procedural style whereas standalone Python editor uses an infinite main control loop.

micro:bit as a Reactive System

- micro:bit is a **<u>reactive system</u>**:
  - It reacts continuously to external events, such as a person pressing the A button or shaking the device.
  - The reaction to an event may be to perform a computation, update variables and change the display.
  - After the device reacts to an event, it is ready to react to the next one.
  - E.g., most computer games are reactive systems.
- Reactive systems need to be <u>responsive</u>, i.e., react in a timely manner to events.
- To be responsive, a reactive system needs to be able to do several things at the same time, i.e., <u>concurrently</u>.

micro:bit as a Reactive System (cont.)

- But the micro:bit only has <u>one CPU</u> for executing program:
  - It can only execute one program instruction at a time.
  - However, it can execute millions of instructions in a single second.
- The micro:bit's <u>scheduler</u> provides the capability to concurrently execute different code sequences:
  - The <u>first job</u> of the scheduler is to allow multiple subprograms to be queued up for later execution.
  - The <u>second job</u> of the scheduler is to periodically interrupt execution to <u>read (poll) the various inputs</u> to the micro:bit (the buttons, pins, etc.) and <u>fire off events</u> (such as "button A pressed").

micro:bit as a Reactive System (cont.)

- The firing of an event causes the event handler subprogram associated with that event to be queued for later execution.
- The scheduler uses a timer built into the hardware to interrupt execution every 6 milliseconds and poll the inputs.

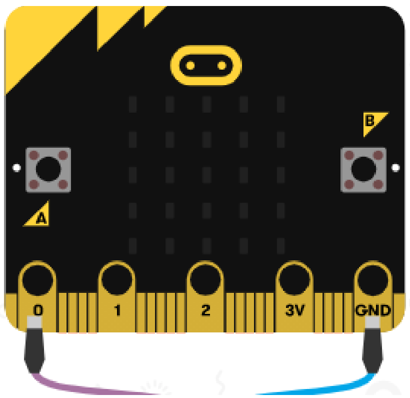
- The micro:bit's scheduler is non-preemptive:
  - This means that the runtime will never take control away from a subprogram.
  - The scheduler will wait for a subprogram to either:
    - finish execution; or
    - make a call to a runtime function that is blocking.

Event Driven Programming with micro:bit

- micro:bit uses an **event-driven programming model**.
- An empty default micro:bit project consists of:
  - `on start` event handler:
    - Runs code when the program starts.
    - This is essentially the `main()` equivalent that contains code executed sequentially.
  - `basic.forever loop`:
    - Code in this loop runs forever in the background.
    - It is an infinite loop handling the basic logic of the micro:bit.
    - But it allows other code to run on each iteration.
- All other code are written within **event handlers** that gets executed when the associated **event** is triggered.

Event Driven Programming with micro:bit (cont.)

- micro:bit version of "Hello World!"




```
input.onButtonPressed(Button.A, function () {
    basic.showString("Hello World!")
})
music.playTone(262, music.beat(BeatFraction.Whole))
basic.forever(function () {

})
```

src01.js

Cooperative Passing of Control

- How does the <u>forever loop</u> get to start execution?
- Furthermore, once the forever loop is running, how do any other subprograms (like event handlers) ever get a chance to execute?
- The forever loop periodically and voluntarily <u>pause its execution</u> so that other subprograms can execute:
  - But the forever loop is not aware of other subprograms.
  - Thus, the forever loop (and other subprograms) passes control of execution back to the scheduler.
  - The scheduler then determines the <u>next subprogram to pass control to</u>.

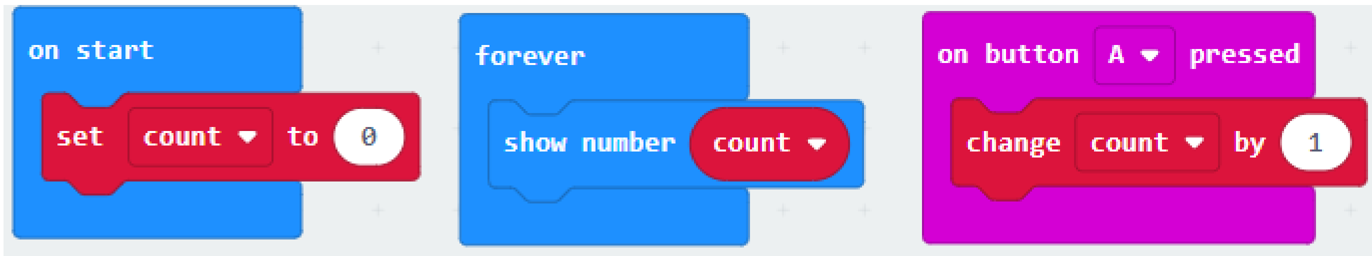
Round-robin Scheduling

- The third and final job of the scheduler is to determine which is the next subprogram to pass control to.
- The scheduler uses two queues to perform this task:
  - The run queue contains all non-sleeping subprograms, such as the event handlers queued by the firing of events.
  - The sleep queue contains previously running subprograms that have called the pause function and still have time left to sleep.
  - The scheduler moves the subprogram that has just paused into the sleep queue.
  - Then removes the subprogram at the head of the run queue and resumes its execution.
  - Once a subprogram's sleep period is over, the scheduler moves it from the sleep queue to the back of the run queue.

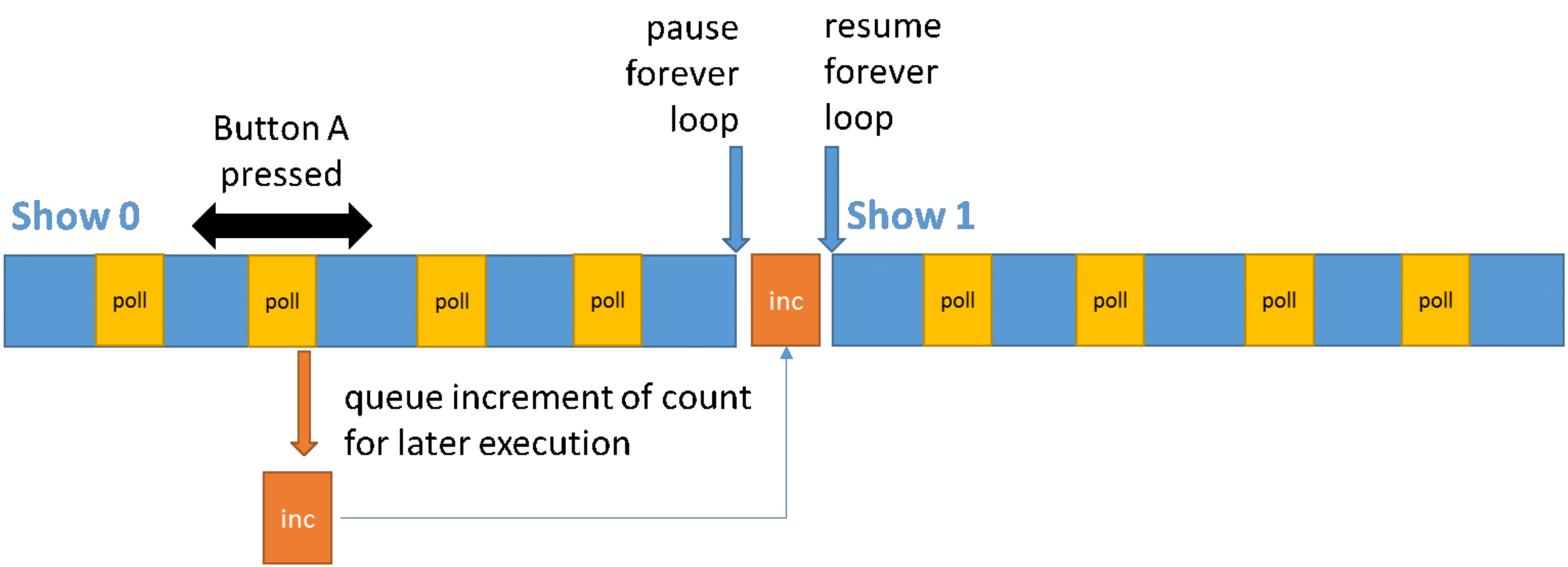
Round-robin Scheduling (cont.)

- Round-robin scheduling assumes that every subprogram:
  - Eventually runs to completion; or
  - Periodically enters the sleep queue.
- In this way, every subprogram will periodically get a chance to execute.
- Essentially, the micro:bit scheduler enables us to create a program that is composed of concurrent subprograms easily:
  - We do not need to worry about the low-level programming.

Round-robin Scheduling (cont.)



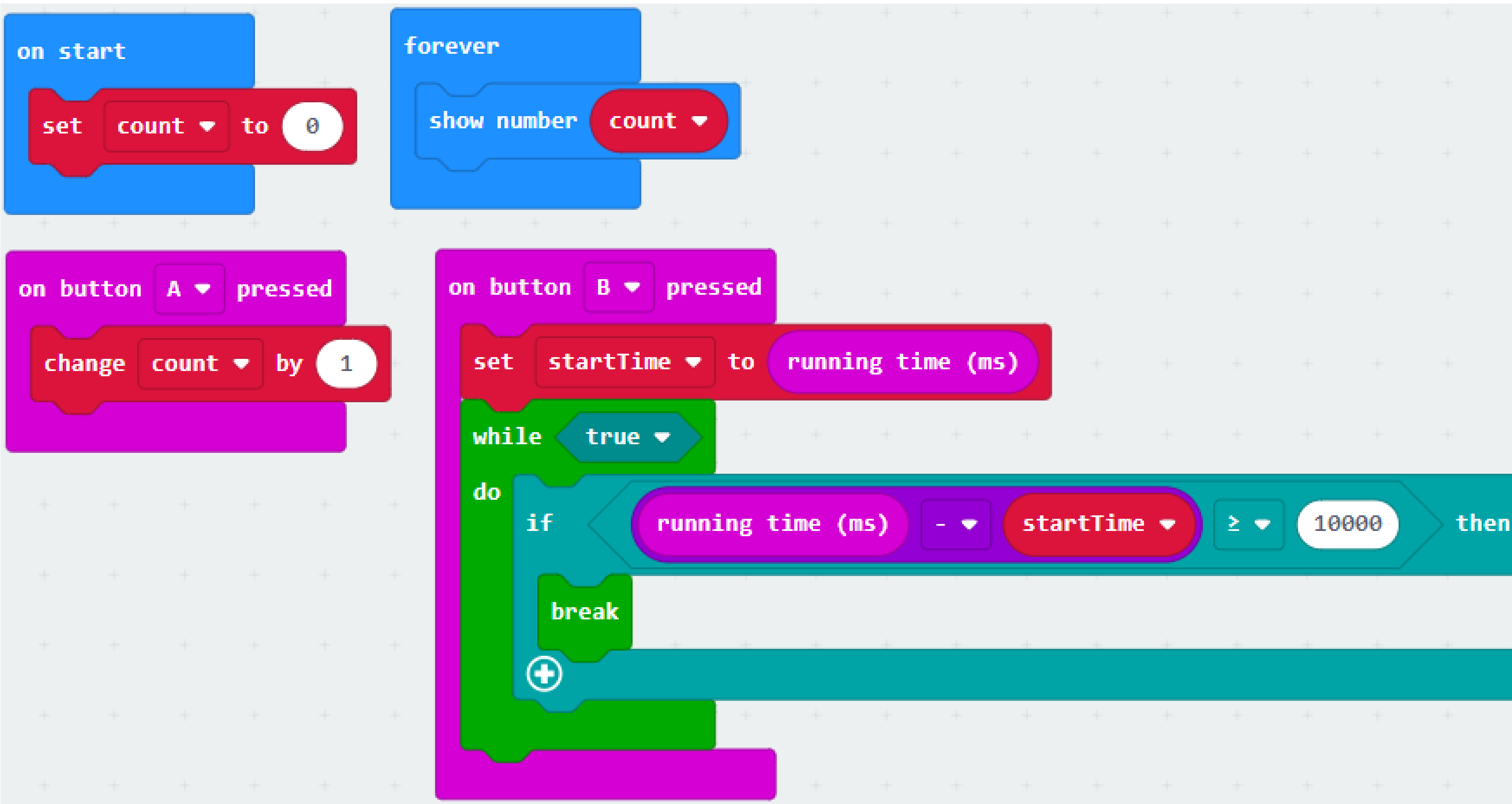
rrschedule.js



Round-robin Scheduling (cont.)

- Recall that micro:bit's scheduler is <u>non-preemptive</u> and assumes that every subprogram periodically pauses:
  - What happens if a subprogram misbehaves and refuses to pause?
  - In such a scenario, the forever loop and other subprograms would not be able to run.
  - But the scheduler's interrupt mechanism would still be able to respond and queue event handlers for future execution.
  - An event would not be lost unless it triggers faster than 6 milliseconds.

Round-robin Scheduling (cont.)



rrscheduleblocking.js

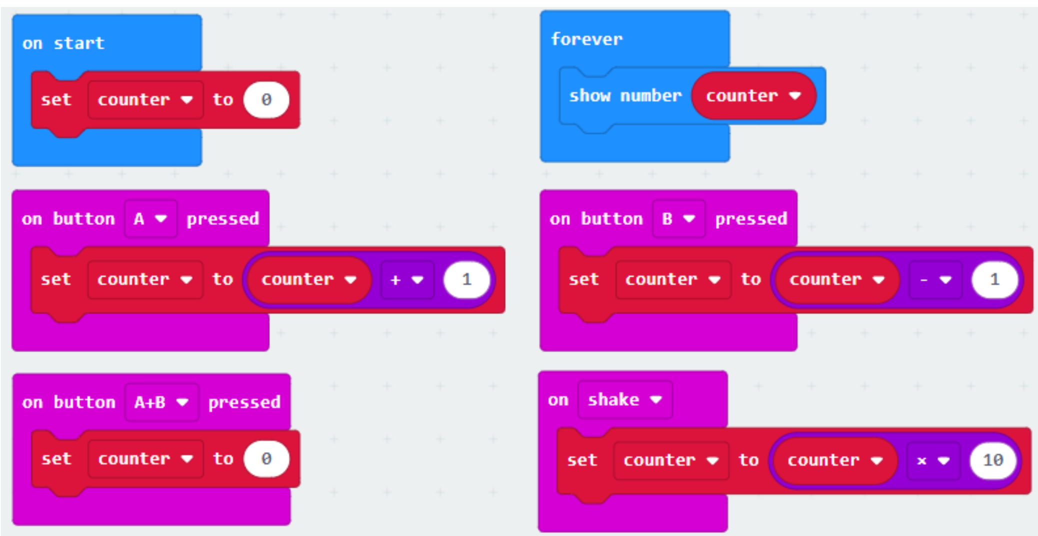
Variables and Operators

- You can define variables for holding micro:bit's state.
- The `let` statement is used to declare a block scope local variable and optionally initialising it to a value.
- In JavaScript, variables are dynamically typed and in general you can manipulate the following primitive data types:

Data Type	Description
String	Represents sequence of characters, e.g., "hello"
Number	Represents numeric values, e.g. ,100
Boolean	Represents boolean value either `false` or `true`
Undefined	Represents undefined value
Null	Represents `null`, i.e., no value at all

Variables and Operators (cont.)

- Simple example of visitor counter:
  - Use a single number counting variable initialised to 0.
  - Perform some simple arithmetic operations on the variable.

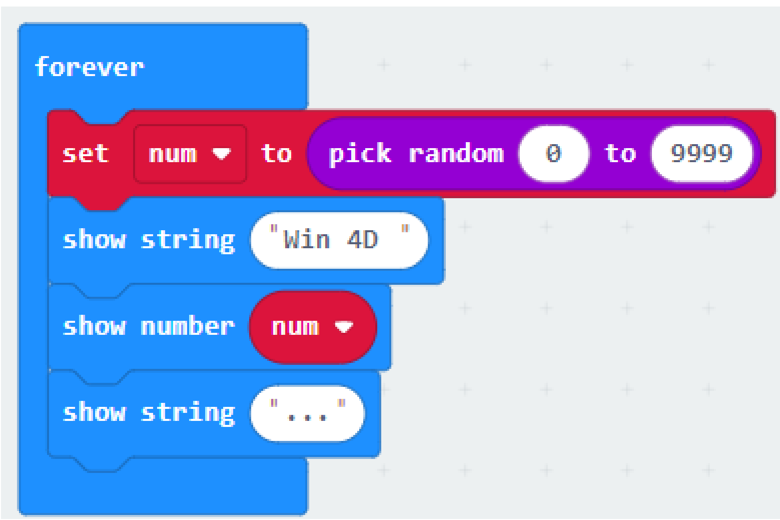



```
input.onButtonPressed(Button.A, function () {
    counter = counter + 1
})
input.onButtonPressed(Button.AB, function () {
    counter = 0
})
input.onButtonPressed(Button.B, function () {
    counter = counter - 1
})
input.onGesture(Gesture.Shake, function () {
    counter = counter * 10
})
let counter = 0
counter = 0
basic.forever(function () {
    basic.showNumber(counter)
})
```

src02.js

Variables and Operators (cont.)

- You can also perform advanced mathematical operations with your micro:bit:
  - `Math.randomRange()` can be used to generate a random number between 0 (inclusive) and limit (inclusive).



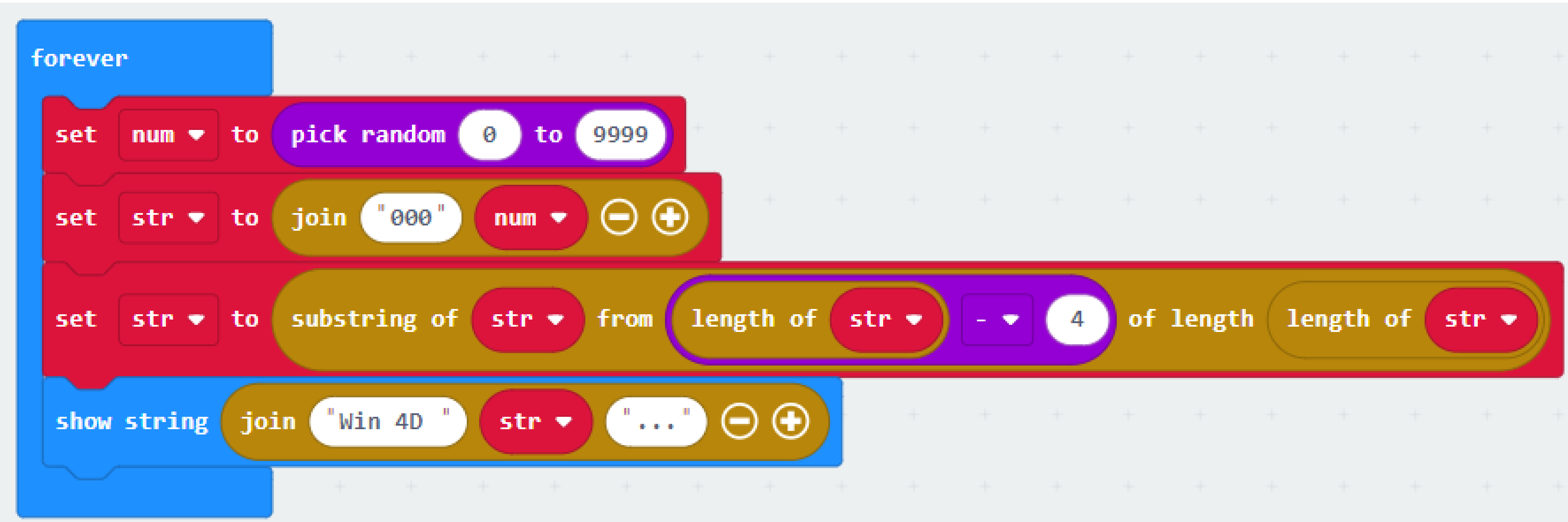

```
let num = 0
basic.forever(function () {
    num = Math.randomRange(0, 9999)
    basic.showString("Win 4D ")
    basic.showNumber(num)
    basic.showString("...")
})
```

src03.js

Variables and Operators (cont.)

- Is there any potential problem with sample code `src03.js`?
  - What is the problem?
  - How can you solve the problem?
  - The JavaScript runtime in micro:bit does not support string padding.
  - Thus, we need to manually pad the 4D number with 0s.

Variables and Operators (cont.)




```
let num = 0
let str = ""
basic.forever(function () {
    num = Math.randomRange(0, 9999)
    str = "000" + num
    str = str.substr(str.length - 4, str.length)
    basic.showString("Win 4D " + str + "...")
})
```

src04.js

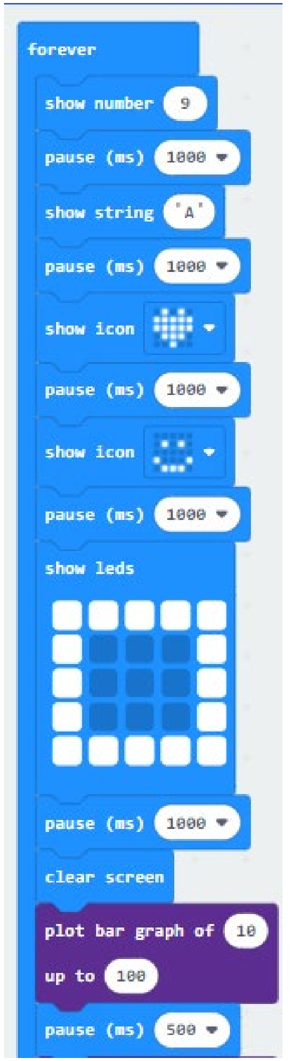
Basic Input/Output

- Input can be obtained from user via:
  - Buttons – A, B or A+B
  - Gesture:
    - Movement – shake, logo up, logo down, screen up, screen down, tilt left, tilt right, free fall
    - Amount of g force being applied – 3g, 6g, 8g
  - Pin Pressed – Pin 0, Pin 1 or Pin 2 (GPIO ADC)
- Sensors are classified as input too and we will come to that later.

src05.js

Basic Input/Output (cont.)

- Onboard output is achieved using the 25 individually-programmable LEDs (5 by 5):
  - Show numbers.
  - Show string.
  - Show predefined icons.
  - Show LEDs by controlling which of the individual LED gets turned on/off.
- Advanced output can be achieved by:
  - (Un)Plotting a x,y coordinate
  - Toggling a x,y coordinate
  - Plot a graph of n to max.

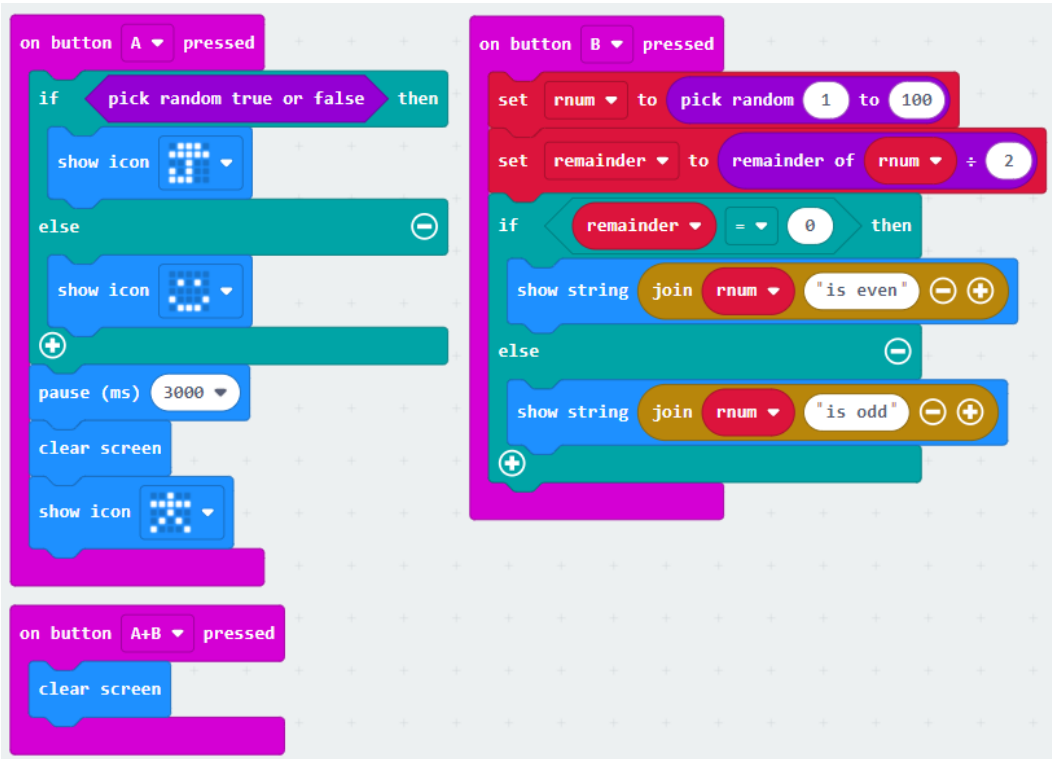


src06.js

Conditional Control Flow

- Standard JavaScript conditional control flow with `if` statement can be used in micro:bit:
  - `if` statement for optional action.
  - `if … else` for alternative actions.
  - `if … else if … else` for multiple alternative actions.
- Relational operators can be used for comparison – `=`, `!=`, `<`, `<=`, `>` and `>=`.
- micro:bit also supports the `and` as well as `or` boolean operators to form complex expressions.
- Easier to edit the JavaScript directly.

Conditional Control Flow (cont.)



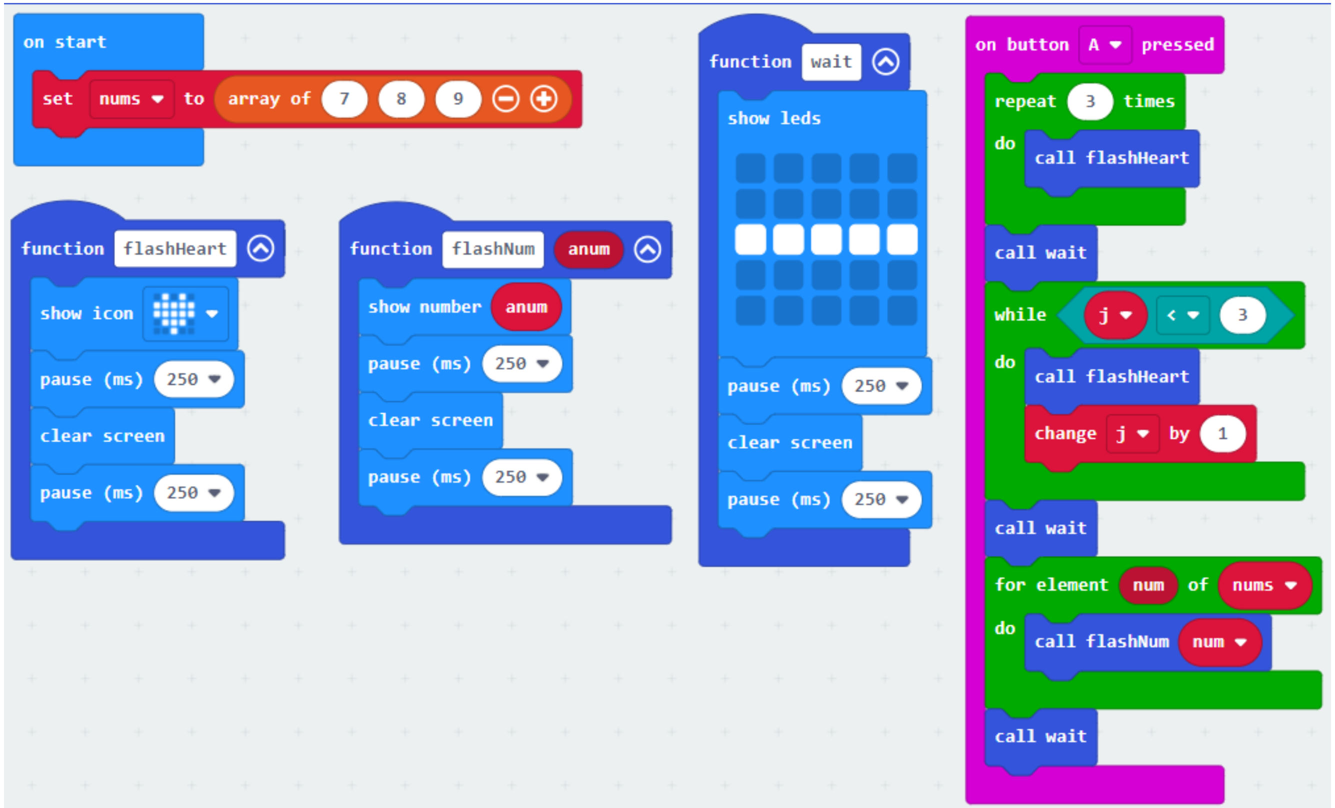

```
let rnum = 0
let remainder = 0
input.onButtonPressed(Button.A, function () {
    if (Math.randomBoolean()) {
        basic.showIcon(IconNames.Umbrella)
    } else {
        basic.showIcon(IconNames.Happy)
    }
    basic.pause(3000)
    basic.clearScreen()
    basic.showIcon(IconNames.StickFigure)
})
input.onButtonPressed(Button.B, function () {
    rnum = Math.randomRange(1, 100)
    remainder = rnum % 2
    if (remainder == 0) {
        basic.showString("" + rnum + "is even")
    } else {
        basic.showString("" + rnum + "is odd")
    }
})
input.onButtonPressed(Button.AB, function () {
    basic.clearScreen()
})
```

src07.js

Iterative Control Flow

- Standard JavaScript iterative control flow can be used in micro:bit:
  - `while` statement for general looping base on a boolean condition.
  - `for` statement for looping with a counting variable.
  - `for...of` statement iterating through a list.
- It is also possible to define functions, including parameterised functions, to reuse JavaScript code in micro:bit.
- Procedural programming is useful once the micro:bit program becomes big.

Iterative Control Flow (cont.)




```
let j = 0
let nums: number[] = []
nums = [7, 8, 9]
input.onButtonPressed(Button.A, function () {
    for (let index = 0; index < 3; index++) {
        flashHeart()
    }
    wait()
    while (j < 3) {
        flashHeart()
        j += 1
    }
    wait()
    for (let num of nums) {
        flashNum(num)
    }
    wait()
})
function flashNum (anum: number) {
    basic.showNumber(anum)
    basic.pause(250)
    basic.clearScreen()
    basic.pause(250)
}
function flashHeart () {
    basic.showIcon(IconNames.Heart)
    basic.pause(250)
    basic.clearScreen()
    basic.pause(250)
}
function wait () {
    basic.showLeds(`
        . . . . .
        . . . . .
        # # # # #
        . . . . .
        . . . . .
        `)
    basic.pause(250)
    basic.clearScreen()
    basic.pause(250)
}
```

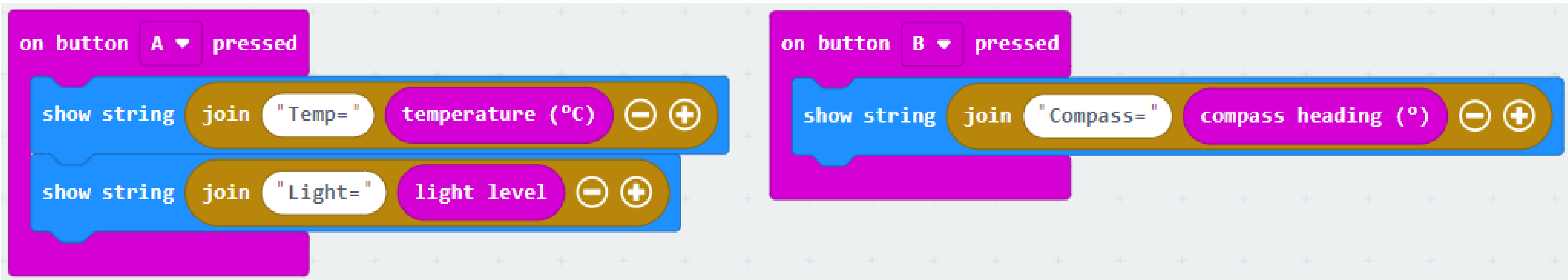
src08.js

Sensors Input

- micro:bit onboard sensors can be read easily:
  - Data value can be displayed on the LEDs; or
  - Perform further processing.
- The following sensor data can be obtained:

Sensor Data	Description
compassHeading	Get the current compass heading in degrees
temperature	Get the temperature in Celsius degrees
acceleration	Get the acceleration value in milli-gravitys (when the board is laying flat with the screen up, x=0, y=0 and z=-1024).
lightLevel	Reads the light level applied to the LED screen in a range from "0" (dark) to "255" (bright).
rotation	The pitch or roll of the device, rotation along the "x-axis" or "y-axis", in degrees.
magneticForce	Get the magnetic force value in "micro-Teslas" ("µT")

Sensors Input (cont.)

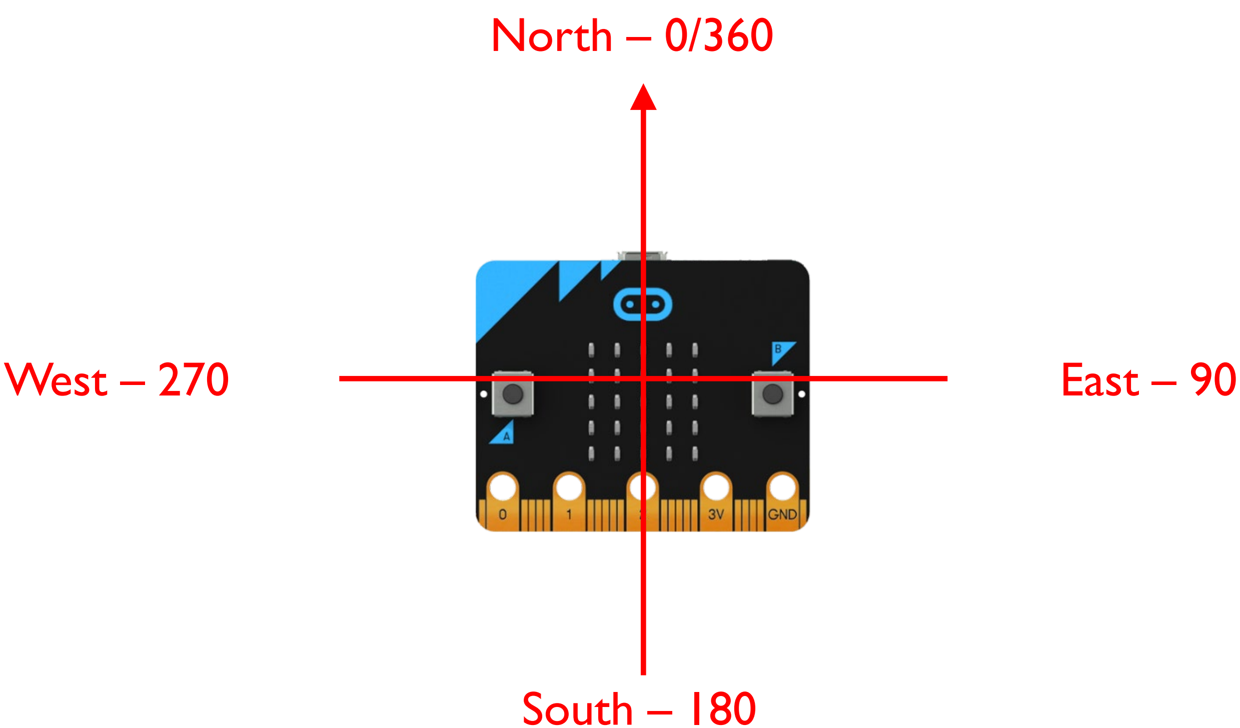



```
input.onButtonPressed(Button.A, function () {
    basic.showString("Temp=" + input.temperature())
    basic.showString("Light=" + input.lightLevel())
})
input.onButtonPressed(Button.B, function () {
    basic.showString("Compass=" + input.compassHeading())
})
```

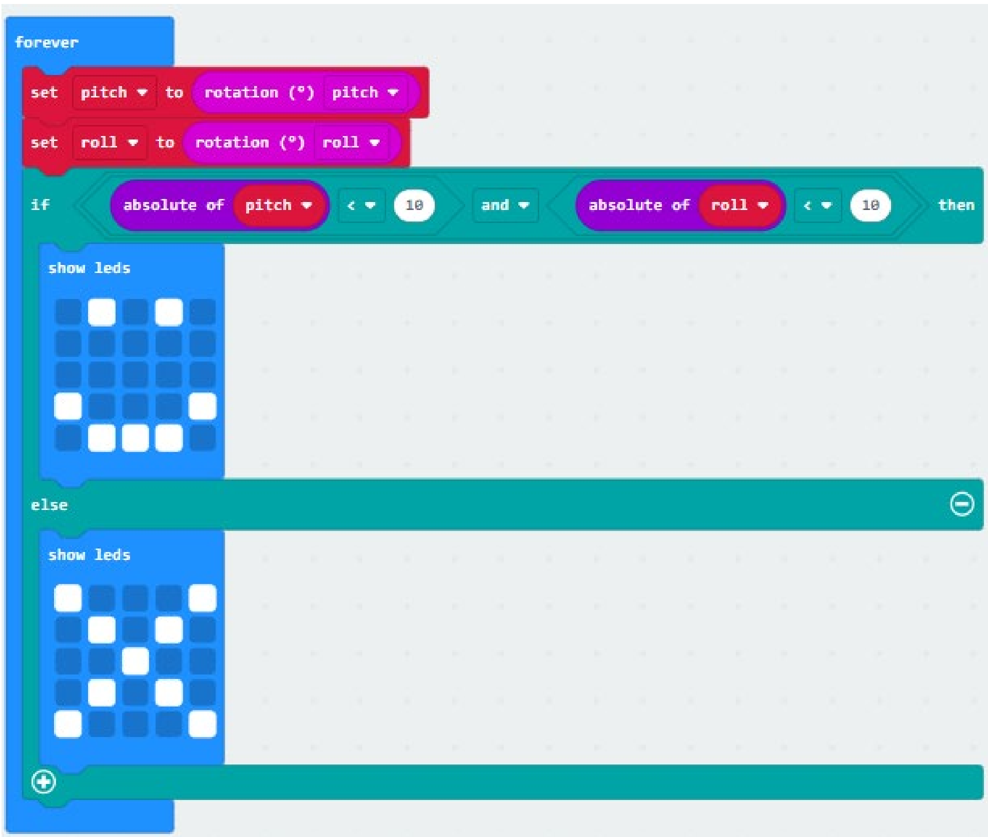
src09.js

Sensors Input (cont.)

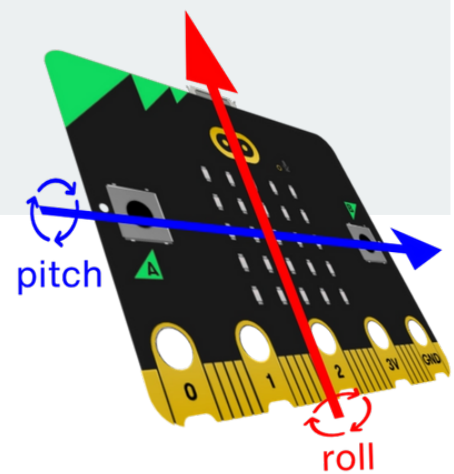
- Compass heading is read with the:
  - micro:bit's logo facing up and the microcontroller flat.
  - The front of the micro:bit pointing at the required direction.



Sensors Input (cont.)




```
let pitch = 0
let roll = 0
basic.forever(function () {
    pitch = input.rotation(Rotation.Pitch)
    roll = input.rotation(Rotation.Roll)
    if (Math.abs(pitch) < 10 && Math.abs(roll) < 10) {
        basic.showLeds(`
            . # . # .
            . . . . .
            . . . . .
            # . . . #
            . # # # .
            `)
    } else {
        basic.showLeds(`
            # . . . #
            . # . # .
            . . # . .
            . # . # .
            # . . . #
            `)
    }
})
```



src10.js

Wireless Communication

- micro:bit supports two forms of wireless communication:
  - **Radio** (2.4 GHz) communication between two or more micro:bit devices.
  - **Bluetooth Low Energy (BLE)** communication between other non-micro:bit devices.
- The central processing unit (CPU) of micro:bit is the Nordic Semiconductor nRF51822:
  - The nRF51 series combines Nordic Semiconductor's 2.4GHz transceiver technology with the powerful but low power ARM Cortex-M0 core.
  - The built-in 2.4GHz radio module can be configured in a number of ways and is primarily designed to run the BLE protocol.

Wireless Communication (cont.)

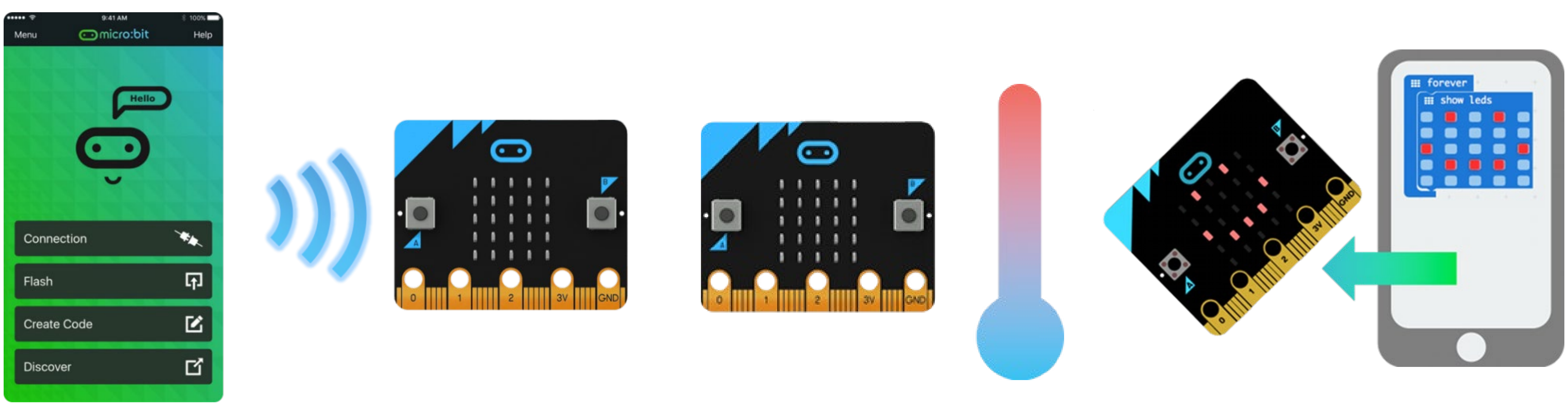
- It can also be placed into a much simpler mode of operation that allows simple, direct micro:bit to micro:bit communication.
- However, it is not currently possible to run the Radio component and BLE at the same time:
  - If you want to use the Radio functionality, you need to disable the BLE stack on your micro:bit and vice-versa.

Radio Wireless Communication

- A micro:bit device specifies a <u>radio group ID</u> from 0 to 255:
  - micro:bit can only send or receive in one group at a time.
  - If we load the very same program onto two different micro:bits, they will be able to talk to each other because they will have the same radio group ID.
- Transmission power is set from 0 (-30 dBm) to a strength of 7 (+4 dBm):
  - When operating in an open area with the highest transmission power of 7, a micro:bit signal can reach as far as 70 m.

BLE Wireless Communication

- A device such as a smartphone can use any of the Bluetooth "services" provided by a micro:bit.
  - However, it must first be paired with the micro:bit.
  - Once paired, the other device may connect to the micro:bit and exchange data relating to many of the micro:bit's features.
- Data communication can take place both ways, but each "service" is unidirectional.



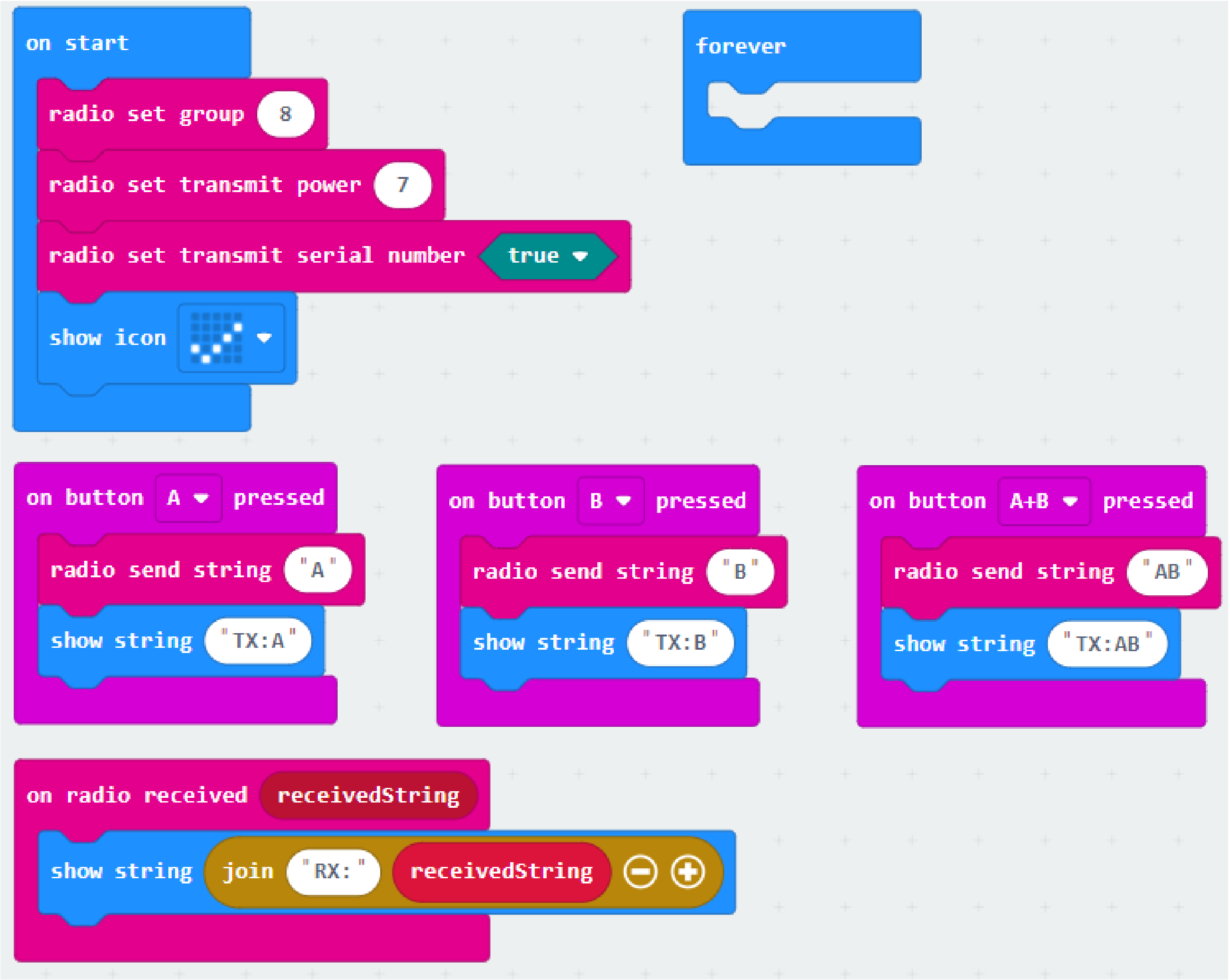
BLE Wireless Communication (cont.)

- Examples:
  - The <u>Bluetooth button service</u> allows the micro:bit to notify the other device about the three possible states (not pressed, pressed and long press) of each of the two buttons.
  - On the contrary, the <u>Bluetooth LED service</u> allow the other device to control the 25 LEDs on the micro:bit by writing string data or toggling individual LED.
  - The former is micro:bit-to-device while the latter is device-to-micro:bit.
  - On an Android phone, you can use the Bitty Blue app for demonstration.

Programming Radio Wireless Communication

- There are two basic use cases for peer-to-peer radio communication between two or more micro:bit devices.
- **Exchanging data**:
  - micro:bit can send a number, string or a name-value pair.
  - Number – micro:bit supports signed 32-bit integer.
  - String – Maximum string length is 19 characters.
  - Name-value pair – Maximum string length of the key is 8 characters.

Programming Radio Wireless Communication (cont.)



src11.js

Programming Radio Wireless Communication (cont.)

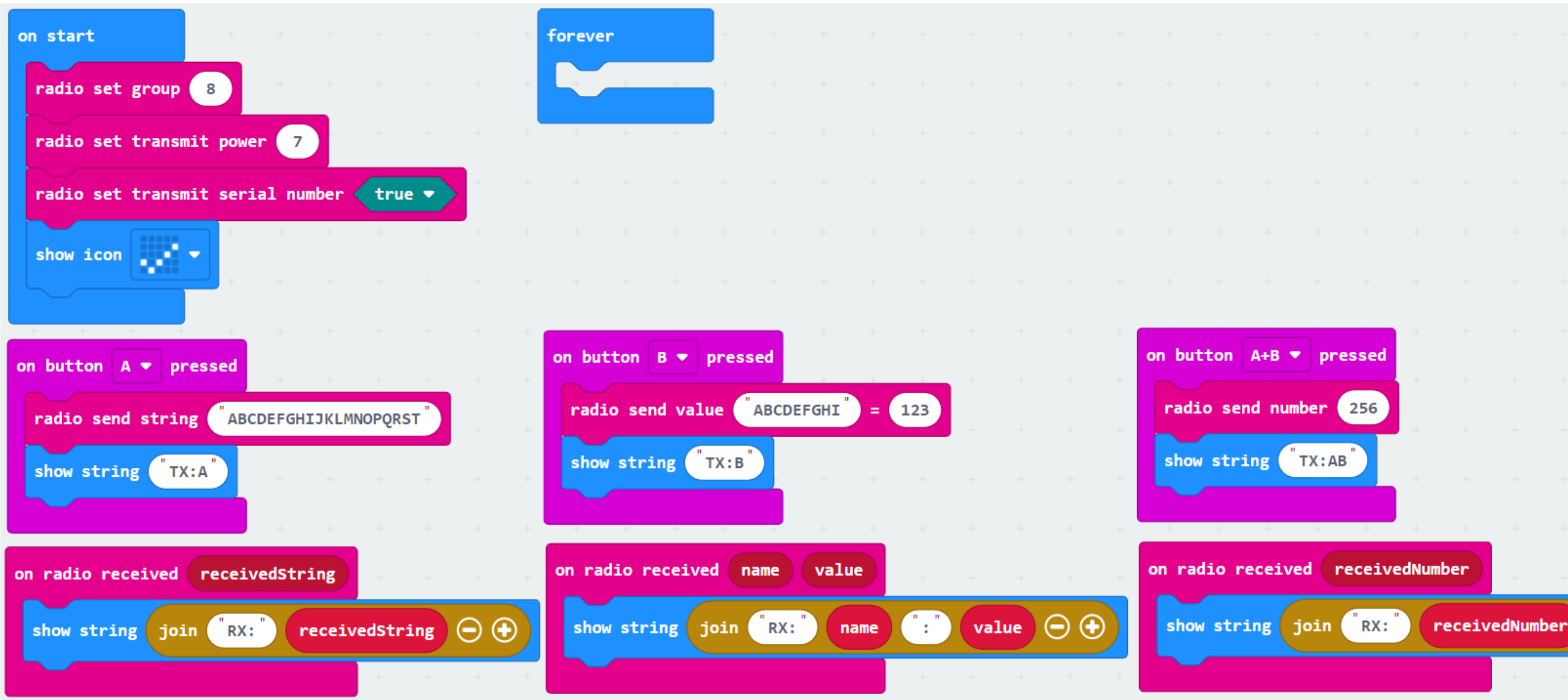
```javascript
input.onButtonPressed(Button.A, function () {
    radio.sendString("A")
    basic.showString("TX:A")
})
input.onButtonPressed(Button.AB, function () {
    radio.sendString("AB")
    basic.showString("TX:AB")
})
radio.onReceivedString(function (receivedString) {
    basic.showString("RX:" + receivedString)
})
input.onButtonPressed(Button.B, function () {
    radio.sendString("B")
    basic.showString("TX:B")
})
radio.setGroup(8)
radio.setTransmitPower(7)
radio.setTransmitSerialNumber(true)
basic.showIcon(IconNames.Yes)
basic.forever(function () {

})
```

src11.js

Programming Radio Wireless Communication (cont.)

- There are three radio sending blocks and three matching radio receiving blocks:
  - One pair for each of number, string and name-value pair.
  - Do take note of the difference in the maximum string length between string (19 characters) and the key of the name-value pair (8 characters).

Programming Radio Wireless Communication (cont.)



src12.js

Programming Radio Wireless Communication (cont.)

```
input.onButtonPressed(Button.A, function () {
    radio.sendString("ABCDEFGHIJKLMNOPQRST")
    basic.showString("TX:A")
})
radio.onReceivedString(function (receivedString) {
    basic.showString("RX:" + receivedString)
})
input.onButtonPressed(Button.B, function () {
    radio.sendValue("ABCDEFGHI", 123)
    basic.showString("TX:B")
})
radio.onReceivedValue(function (name, value) {
    basic.showString("RX:" + name + ":" + value)
})
input.onButtonPressed(Button.AB, function () {
    radio.sendNumber(256)
    basic.showString("TX:AB")
})
radio.onReceivedNumber(function (receivedNumber) {
    basic.showString("RX:" + receivedNumber)
})
radio.setGroup(8)
radio.setTransmitPower(7)
radio.setTransmitSerialNumber(true)
basic.showIcon(IconNames.Yes)
basic.forever(function () {

})
```

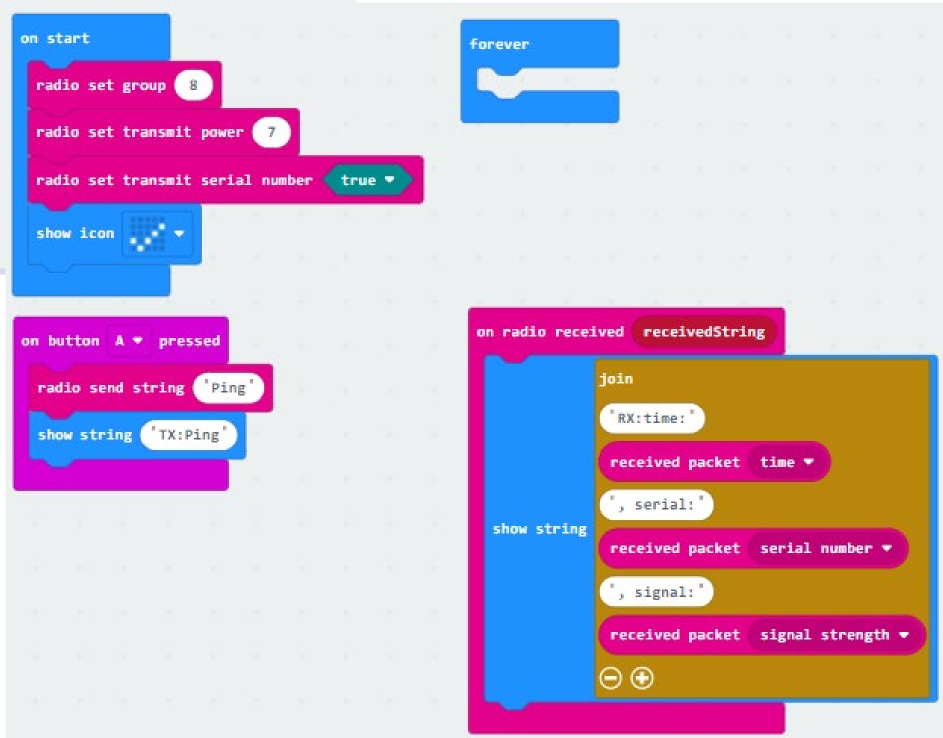
src12.js

Programming Radio Wireless Communication (cont.)

- The `radio.receivedPacket` block allows us to retrieve the following properties of the last received radio packet:
  - `time` – The system time of the micro:bit (elapsed time since the start of the program in ms) that sent this packet at the time the packet was sent.
  - `serial` – The serial number of the micro:bit that sent this packet, or 0 if the micro:bit did not include its serial number.
  - `signal` – How strong the radio signal is from -128 (weak) to -42 (strong).

Programming Radio Wireless Communication (cont.)

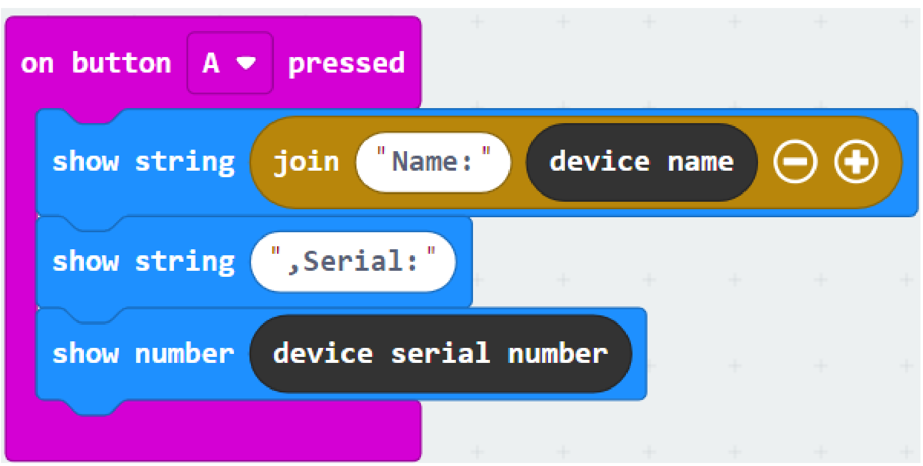
```
input.onButtonPressed(Button.A, function () {
    radio.sendString("Ping")
    basic.showString("TX:Ping")
})
radio.onReceivedString(function (receivedString) {
    basic.showString("RX:time:" + radio.receivedPacket(RadioPacketProperty.Time)
    + ", serial:" + radio.receivedPacket(RadioPacketProperty.SerialNumber)
    + ", signal:" + radio.receivedPacket(RadioPacketProperty.SignalStrength))
})
radio.setGroup(8)
radio.setTransmitPower(7)
radio.setTransmitSerialNumber(true)
basic.showIcon(IconNames.Yes)
basic.forever(function () {

})
```



src13.js

Programming Radio Wireless Communication (cont.)

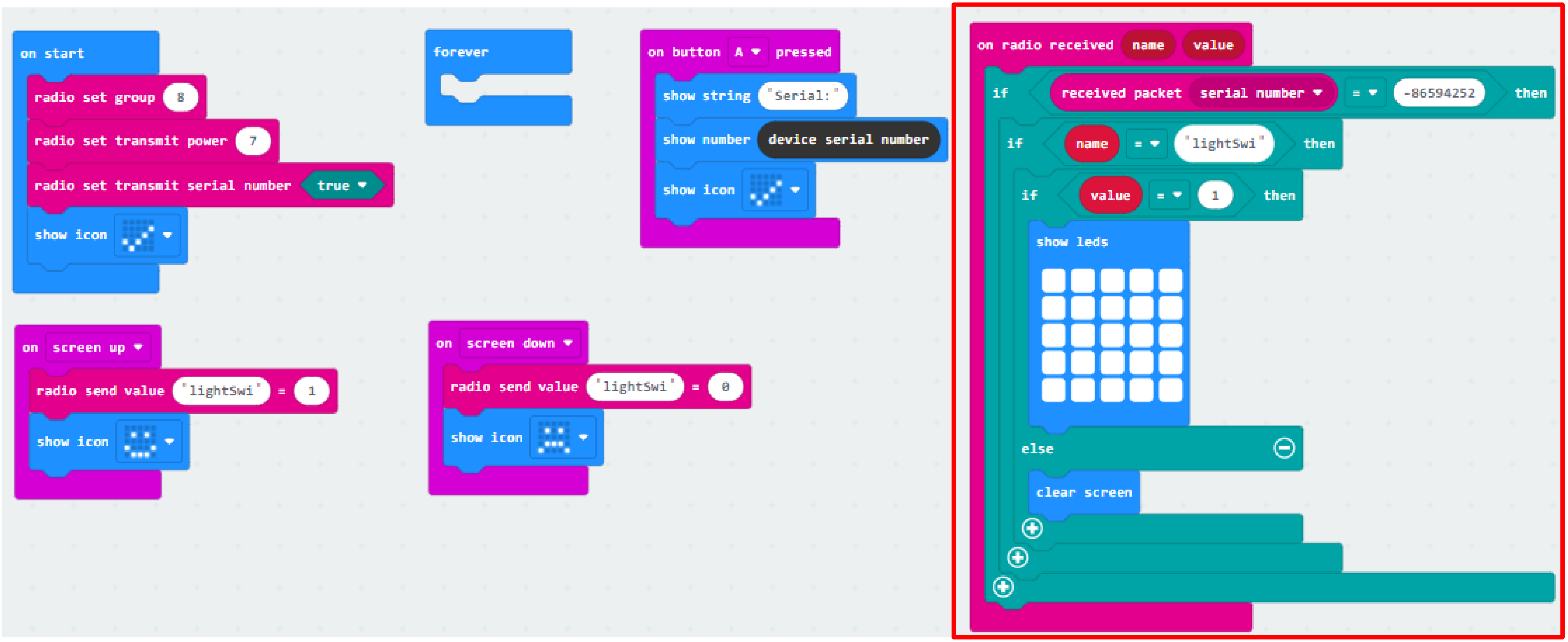
- **Controlling another micro:bit device:**
  - We can extend the base use case for exchanging data to send commands to other micro:bit devices.
  - The receiving micro:bit devices then take certain actions and may reply to the controlling device.
  - By using the device serial number, we can identify the sending device.




```
input.onButtonPressed(Button.A, function () {
    basic.showString("Name:" + control.deviceName())
    basic.showString(",Serial:")
    basic.showNumber(control.deviceSerialNumber())
})
basic.forever(function () {

})
```

src14.js

Programming Radio Wireless Communication (cont.)



src15.js

Programming Radio Wireless Communication (cont.)

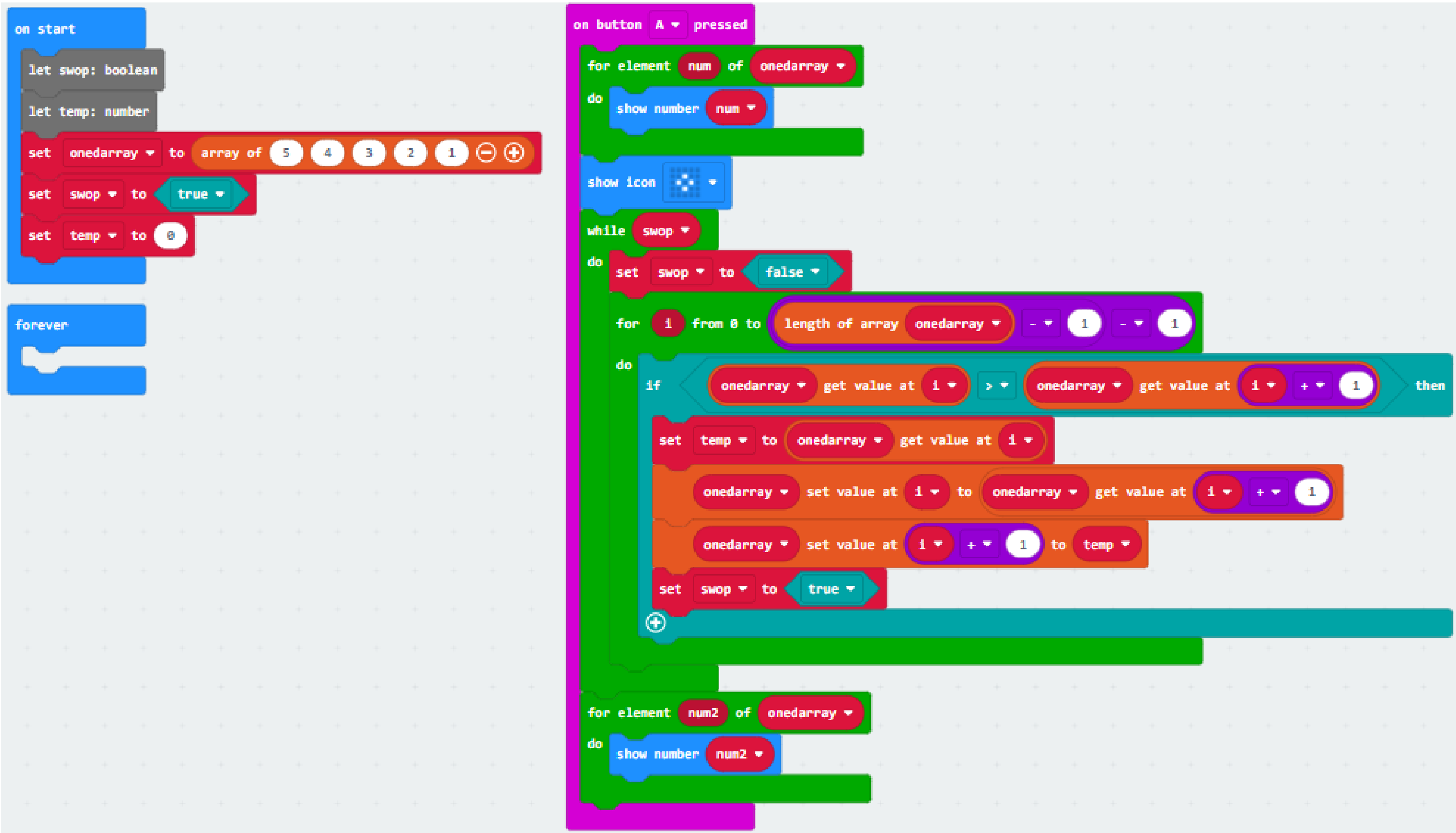
```
input.onButtonPressed(Button.A, function () {
    basic.showString("Serial:")
    basic.showNumber(control.deviceSerialNumber())
    basic.showIcon(IconNames.Yes)
})
input.onGesture(Gesture.ScreenUp, function () {
    radio.sendValue("lightSwi", 1)
    basic.showIcon(IconNames.Happy)
})
input.onGesture(Gesture.ScreenDown, function () {
    radio.sendValue("lightSwi", 0)
    basic.showIcon(IconNames.Sad)
})
radio.onReceivedValue(function (name, value) {
    if (radio.receivedPacket(RadioPacketProperty.SerialNumber) == -86594252) {
        if (name == "lightSwi") {
            if (value == 1) {
                basic.showLeds(`
                    # # # # #
                    # # # # #
                    # # # # #
                    # # # # #
                    # # # # #
                    `)
            } else {
                basic.clearScreen()
            }
        }
    }
})
radio.setGroup(8)
radio.setTransmitPower(7)
radio.setTransmitSerialNumber(true)
basic.showIcon(IconNames.Yes)
basic.forever(function () {

})
```

src15.js

Data Structure

- An **Array** is a list of items of a particular basic (primitive) type, e.g., numbers, strings or booleans.
- Arrays have a length indicating the number of items they contain.
- The values of items at different elements in an array can be accessed by a zero-based index number.
- Arrays are flexible and can grow and shrink in size by adding and removing items at any place in the array:
  - `push` and `pop` – Append or delete an item to/from the end of the array.
  - `insertAt()` and `removeAt()` – Add or delete an item at the specified index position.

Data Structure (cont.)



src16.js

Data Structure (cont.)

```
input.onButtonPressed(Button.A, function () {
    for (let num of onedarray) {
        basic.showNumber(num)
    }
    basic.showIcon(IconNames.SmallDiamond)
    while (swop) {
        swop = false
        for (let i = 0; i <= onedarray.length - 1 - 1; i++) {
            if (onedarray[i] > onedarray[i + 1]) {
                temp = onedarray[i]
                onedarray[i] = onedarray[i + 1]
                onedarray[i + 1] = temp
                swop = true
            }
        }
    }
    for (let num2 of onedarray) {
        basic.showNumber(num2)
    }
})
let onedarray: number[] = []
let swop: boolean
let temp: number
onedarray = [5, 4, 3, 2, 1]
swop = true
temp = 0
basic.forever(function () {

})
```

src16.js

Summary

- Micro:bit is a relatively powerful microcontroller with various onboard sensors.
- The hard buttons and sensors allows micro:bit to obtain input from user and its environment.
- The 25 LEDs provide some limited onboard output capability to micro:bit.
- Micro:bit supports both radio and BLE wireless communication.
- Micro:bit can be used to implement simple data structures and algorithms for supporting more complex computational use cases.