LE/ESSE 2220 Algorithmic and Computational Methods

Lab 1: Telemetry Beacon (Blinking LED)

(Fall 2025-2026)

Z. ARJMANDI (ZARARJ@YORKU.CA)

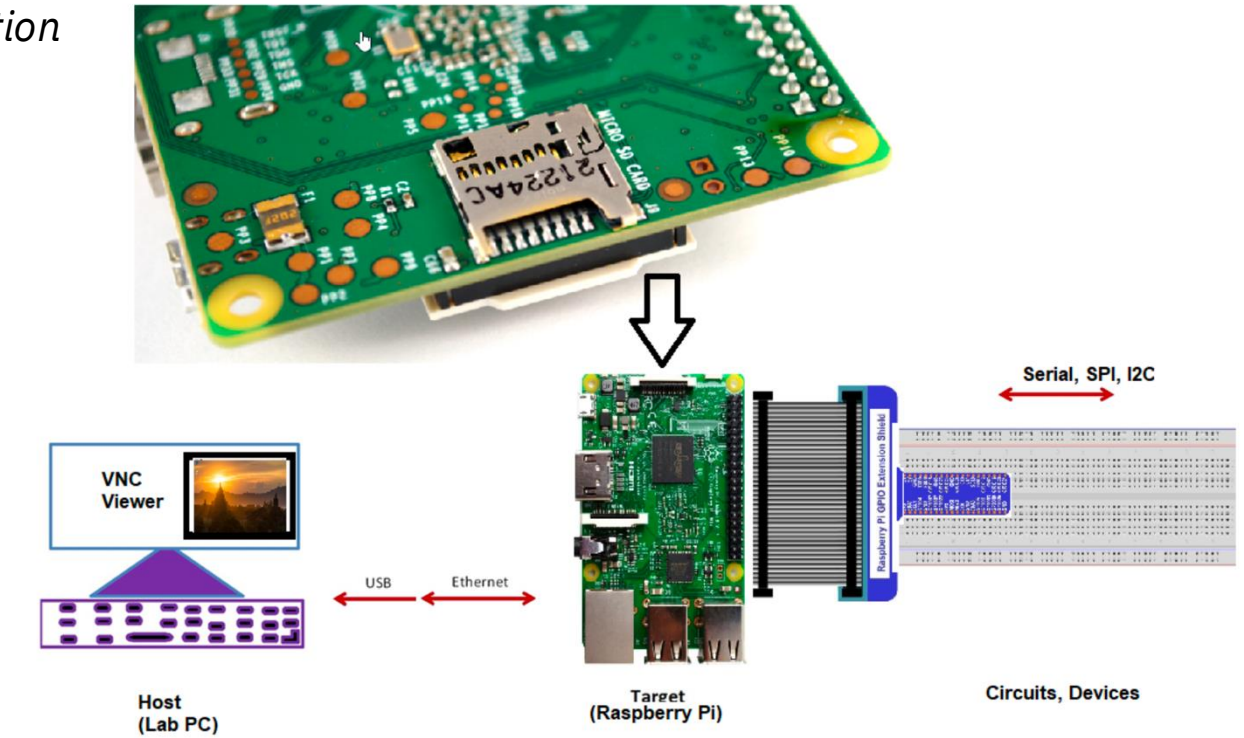
YORK U

Review Before Lab

Raspberry Pi

- **Required components to connect a Raspberry Pi:**
  - Power cable to supply electricity and turn it on
  - Ethernet cable to connect it to your PC (using VNC Viewer as the monitor*)
  - 5V power supply for the cooling fan

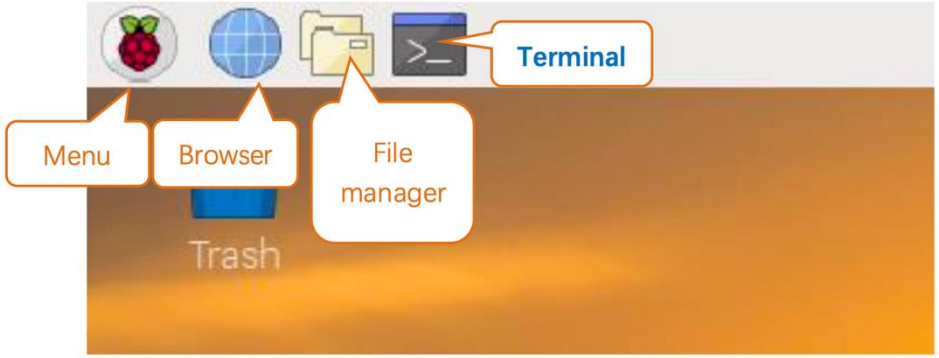
** Follow the tutorial to set up the software connection*



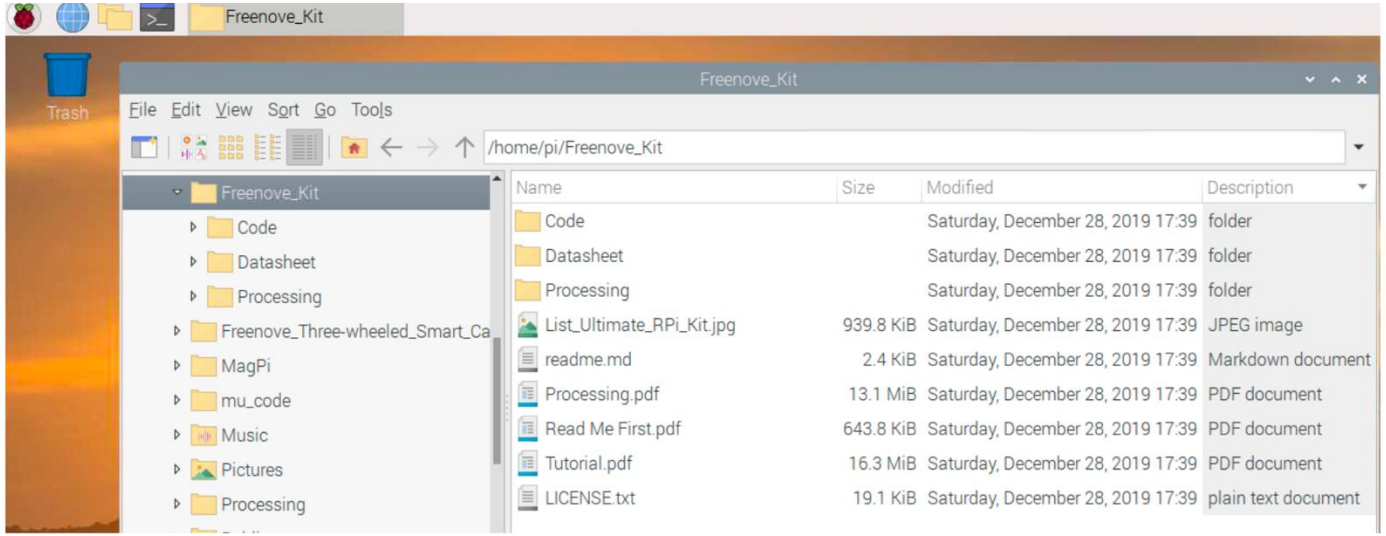
YORK U

Raspberry Pi OS

- Raspberry Pi OS is built on the Linux operating system, which means you can use standard Linux commands and follow its rules.



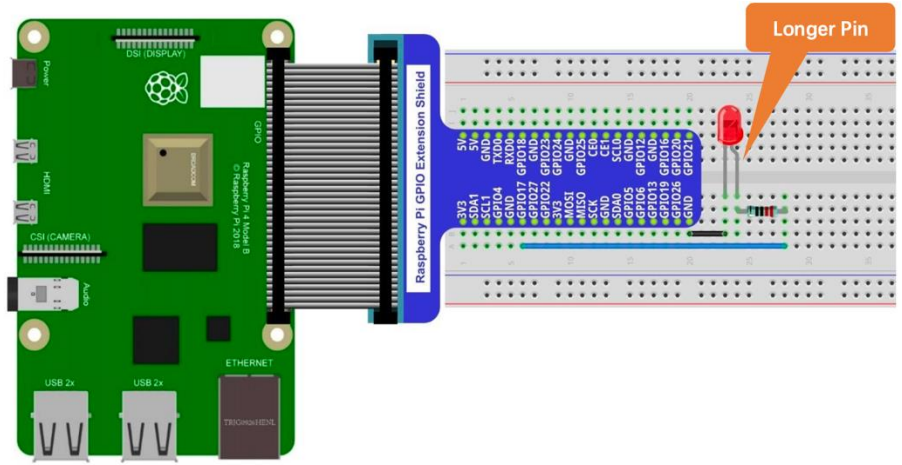
- Freenove prewritten codes

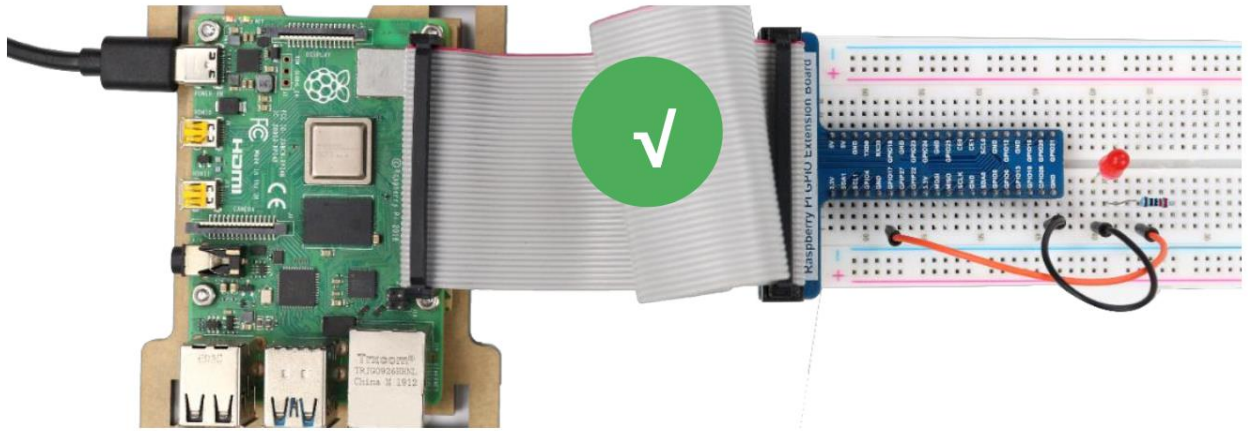


YORK U

Raspberry Pi 3 Model B: Connecting Hardware and Software

- Raspberry Pi
- GPIO Extension Board & Ribbon Cable



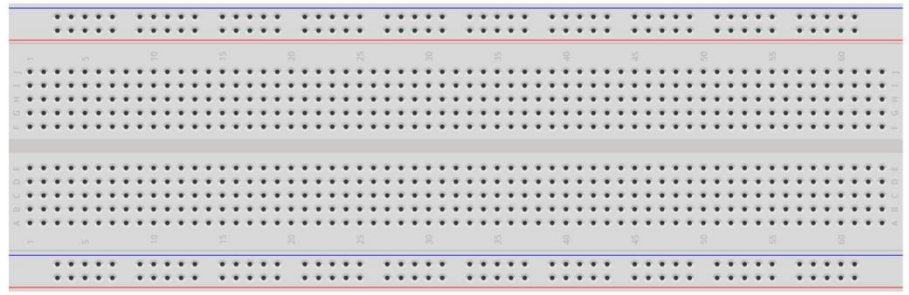


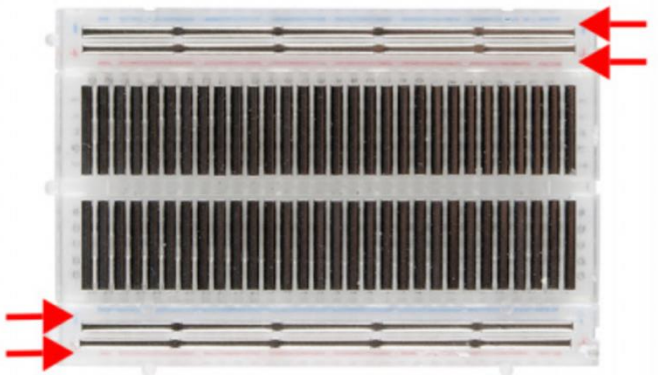
YORK U

Breadboard Connections

- A tool for building circuits **without soldering**.
- Reusable holes & strips connect electronic components with jumper wires.
- **Layout Basics**
  - **Horizontal Power Rows (sides):**
    - Long rows marked **+** and **–** .
    - Used for **power (3.3 V / 5 V)** and **ground**.
  - **Vertical (middle):**
    - Each column of **5 holes** is connected internally.
    - Components (resistors, LEDs, sensors) plug here.
  - **Center Divider (gap):**
    - Splits the board in half → designed for ICs (chips) to straddle the gap.

Breadboard x1

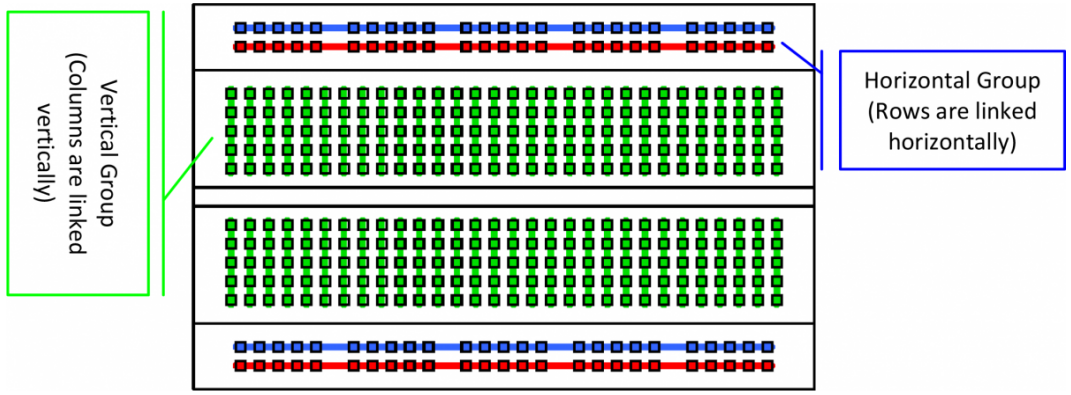


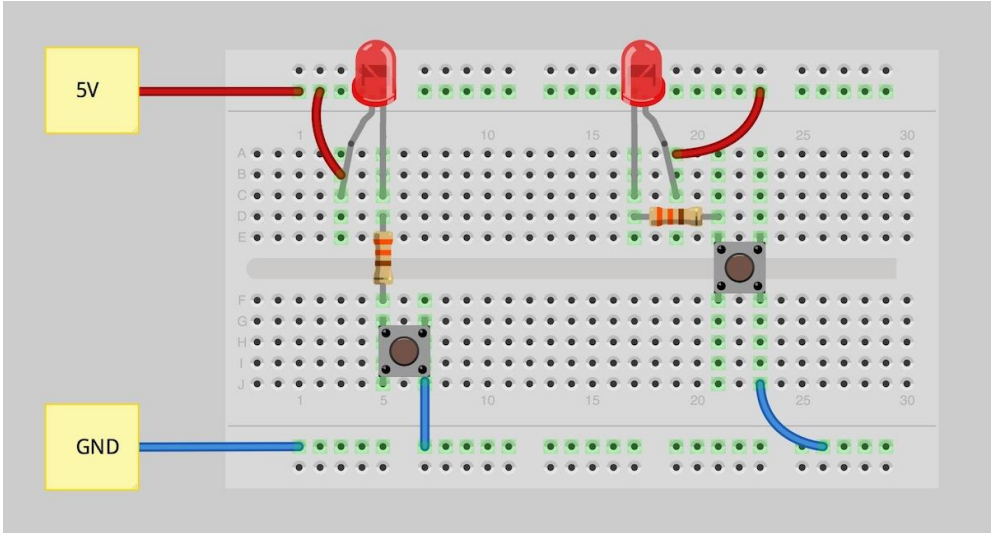


YORK U

Breadboard Connections

- Example:

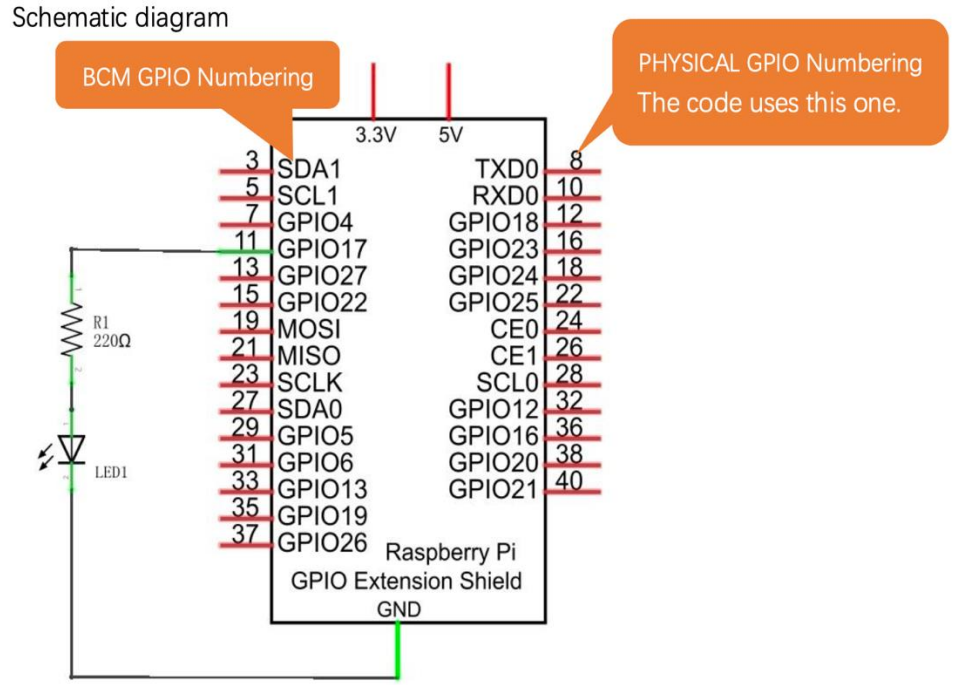




Raspberry Pi GPIO: BCM vs Physical

- **GPIO (General Purpose Input/Output):**
  - Pins on the Raspberry Pi used to control devices (LEDs, motors) or read data from sensors.
  - Can be configured as **input** or **output** in Python.

- Numbering Systems
  - **BCM GPIO (Broadcom SOC Channel):**
    - Refers to the *chip's internal numbering scheme*.
    - Example: **GPIO17**.
  - **Physical GPIO (Board Numbering):**
    - Refers to the *pin's actual position* on the 40-pin header.
    - Example: **Pin 11**.



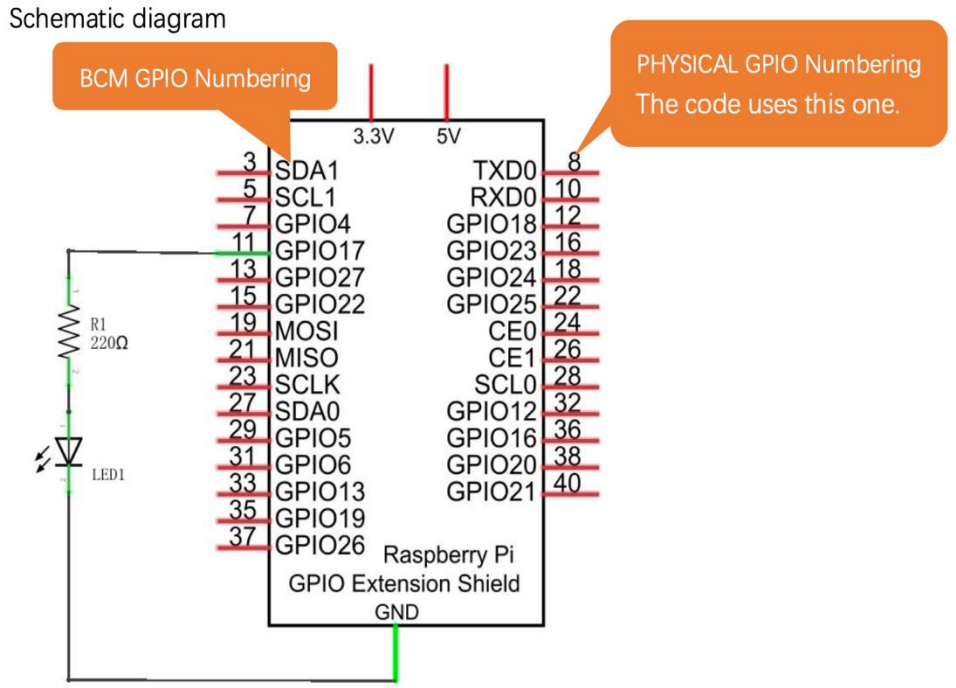
YORK U

Raspberry Pi GPIO: BCM vs Physical

- Here's the equivalent code side-by-side for **BCM** vs **BOARD** numbering. Both turn an LED on and off on the same physical pin (pin 11 on the header, which is GPIO 17 in BCM mode):

```python
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)       # Use BCM numbering
GPIO.setup(17, GPIO.OUT)     # GPIO 17 = physical pin 11

GPIO.output(17, GPIO.HIGH)   # LED ON
GPIO.output(17, GPIO.LOW)    # LED OFF

GPIO.cleanup()               # Reset pins
```

```python
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BOARD)     # Use physical pin numbers
GPIO.setup(11, GPIO.OUT)     # Pin 11 = GPIO 17 (same pin as above)

GPIO.output(11, GPIO.HIGH)   # LED ON
GPIO.output(11, GPIO.LOW)    # LED OFF

GPIO.cleanup()               # Reset pins
```



YORK U

Using GPIO on Raspberry Pi

- **GPIO.setmode()** → choose numbering system (**BCM** or **BOARD**)
- **GPIO.setup(pin, GPIO.OUT)** → set a pin as **output** (to send signals, e.g. turn on an LED)
- **GPIO.setup(pin, GPIO.IN)** → set a pin as **input** (to read signals, e.g. detect a button)

Example: Output (LED)

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)          # Use BCM numbers
GPIO.setup(17, GPIO.OUT)        # Pin 17 = Output

GPIO.output(17, GPIO.HIGH)      # LED ON
GPIO.output(17, GPIO.LOW)       # LED OFF
GPIO.cleanup()                  # Reset pins
```

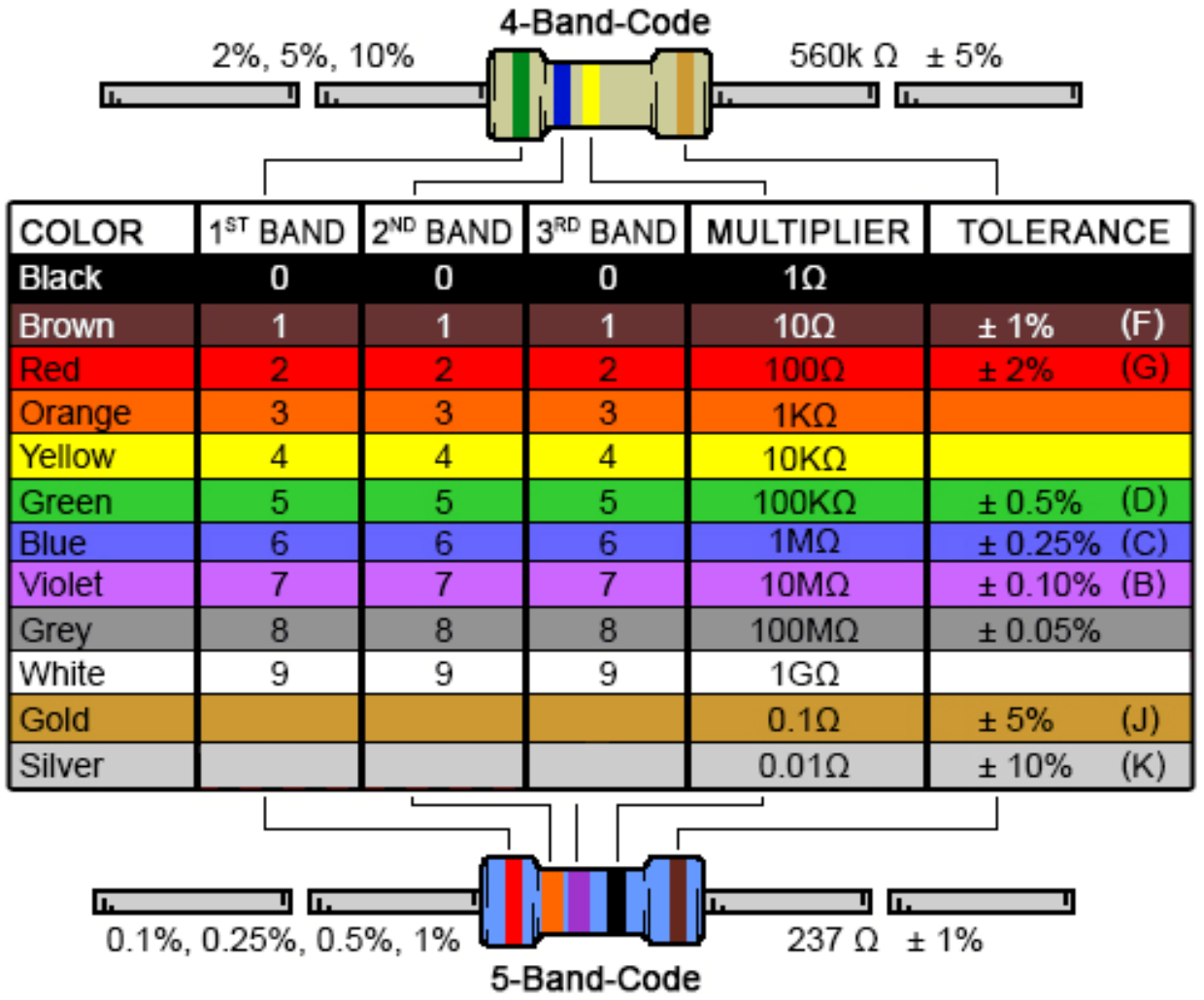
Example: Input (Button)

```
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)          # Use BCM numbers
GPIO.setup(18, GPIO.IN)         # Pin 18 = Input

if GPIO.input(18) == GPIO.HIGH:
    print("Button pressed")
else:
    print("Button not pressed")
GPIO.cleanup()
```

YORK U

Resistor Color Code Chart

- 4-band:
  - Band 1 = First digit
  - Band 2 = Second digit
  - Band 3 = **Multiplier**
  - Band 4 = Tolerance
- 5-band:
  - Band 1 = First digit
  - Band 2 = Second digit
  - Band 3 = Third digit
  - Band 4 = **Multiplier**
  - Band 5 = Tolerance



COLOR	1ST BAND	2ND BAND	3RD BAND	MULTIPLIER	TOLERANCE
Black	0	0	0	1Ω	
Brown	1	1	1	10Ω	± 1% (F)
Red	2	2	2	100Ω	± 2% (G)
Orange	3	3	3	1KΩ	
Yellow	4	4	4	10KΩ	
Green	5	5	5	100KΩ	± 0.5% (D)
Blue	6	6	6	1MΩ	± 0.25% (C)
Violet	7	7	7	10MΩ	± 0.10% (B)
Grey	8	8	8	100MΩ	± 0.05%
White	9	9	9	1GΩ	
Gold				0.1Ω	± 5% (J)
Silver				0.01Ω	± 10% (K)

Lab 1

ASKAP J1832–0911 — A Real Cosmic Beacon

What is it?

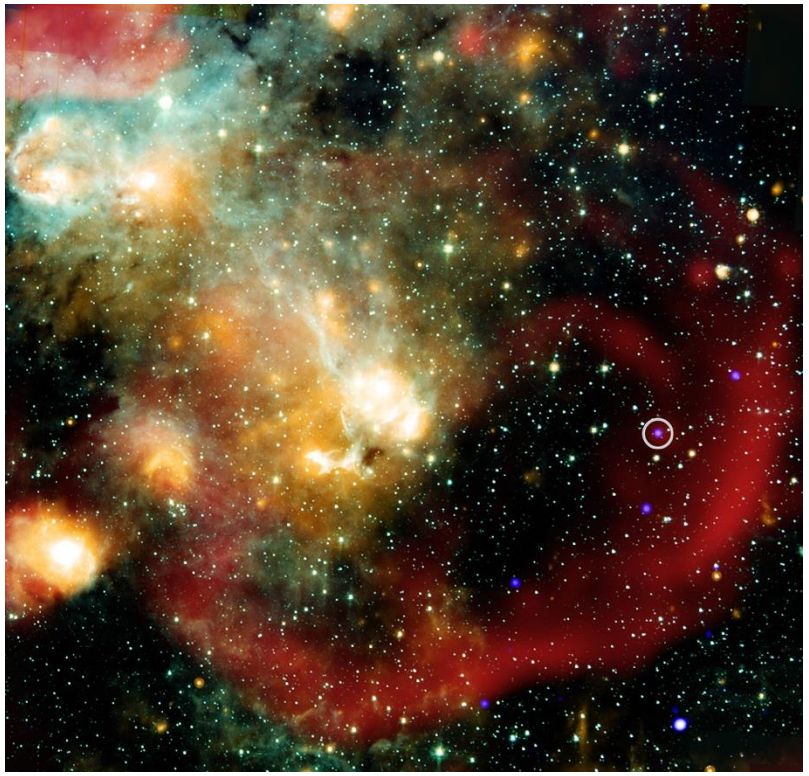
- A mysterious radio signal discovered in 2022 by the **ASKAP radio telescope** in Australia.
- Flashes on and off **every ~20 minutes**, but unlike a normal pulsar or star.
- Still unexplained, could be a new type of cosmic object.
- **Far too slow for a pulsar**, and too "on/off" to be a normal star.

Why is it interesting?

- Behaves like a "**cosmic lighthouse**," turning on and off with a repeating pattern.
- Astronomers use its **signal timing** to study extreme physics.

Your Project

- Just like ASKAP J1832–0911, your Raspberry Pi LED beacon sends **repeated signals**.
- Instead of random flashes, you control the pattern → your **student ID in Morse code**.
- You are building a **mini version of a space beacon** here on Earth!

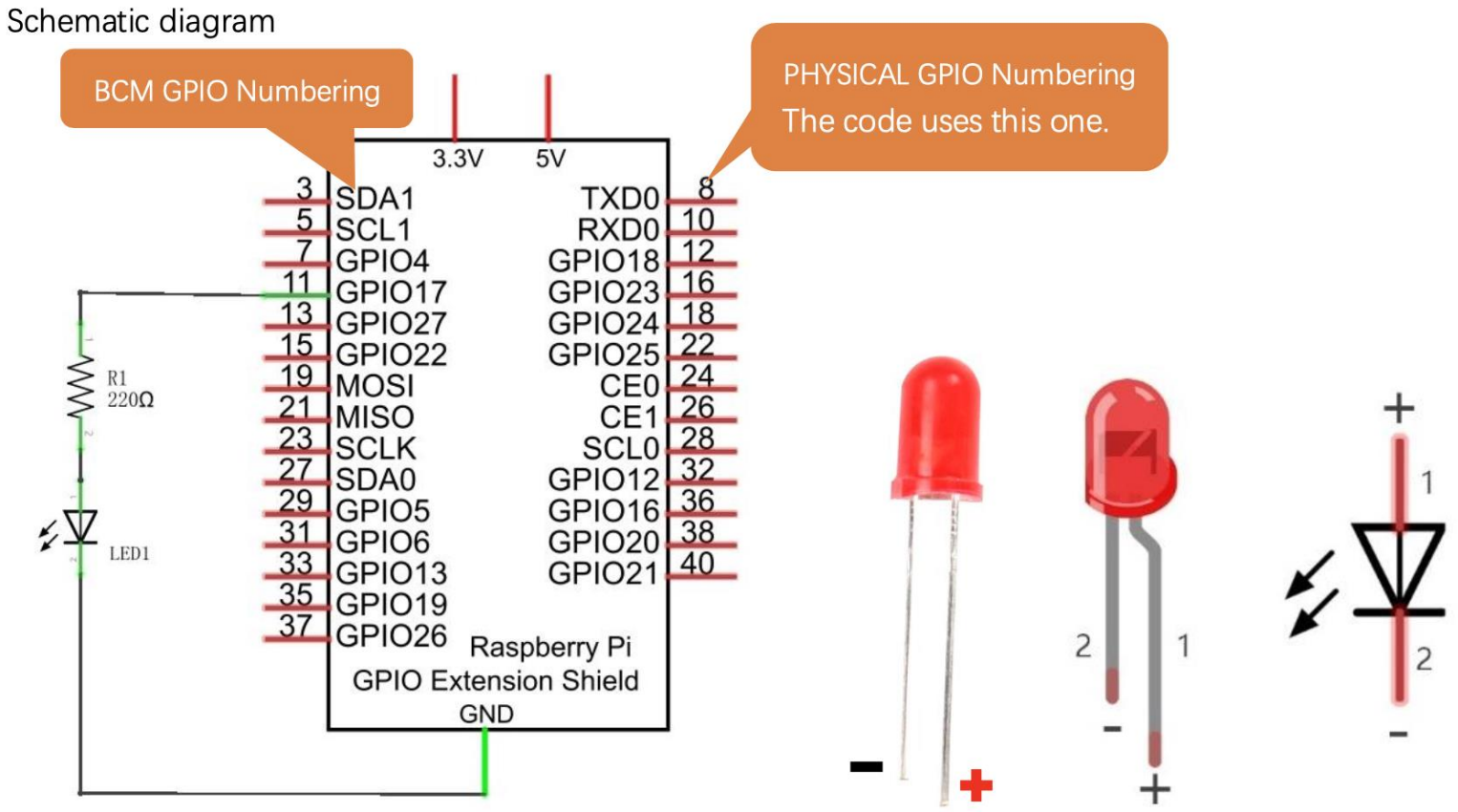


https://en.wikipedia.org/wiki/ASKAP_J1832%E2%88%920911

YORK U

Lab 1 Assignment: Spacecraft Status Beacon

- For more details on this circuit, please refer to Tutorial Chapter 1: LED (available on E-Class).



Lab 1 Assignment: Spacecraft Status Beacon

Step 1 – Blink.py

- Connect the LED (with a resistor) to the Raspberry Pi **according to the provided circuit diagram**.
- Test your setup by running the sample program Blink.py to make sure the LED works.

Step 2 – Customize

- Replace the simple on/off blinking with Morse code for your **student ID**.
- The beacon must:
  - Send each digit in your number using Morse.
  - Repeat continuously, with pauses between digits and between full transmissions.

Example Flow

- Program starts and prints your student ID.
- LED blinks the first digit’s Morse pattern.
- Pauses, then blinks the next digit.
- After all digits, waits a few seconds.
- Repeats forever until stopped.

Hints to Get Started (Optional)

- **Imports / Libraries**
  - import RPi.GPIO as GPIO → controls Raspberry Pi pins.
  - import time → allows delays using time.sleep(seconds: float).
- **Types you'll use**
  - int → pin number (11).
  - float → timing values (e.g., 0.2 seconds).
  - str → your student number ("12345678").
  - list/sequence → iterating through each digit in the string.
- **Functions**
  - dot() → turn LED **on**, wait **0.2s**, then off.
  - dash() → turn LED **on**, wait **0.6s**, then off.
  - send_digit(digit: str) → use if/elif to decide which Morse pattern to blink.
- **Loop**
  - Use for ch in STUDENT_ID: to go through digits.
  - After each digit, add a short time.sleep(0.6) pause.
  - After the whole ID, add a longer pause before repeating.
- **Stop safely**
  - Catch KeyboardInterrupt (Ctrl + C) to exit cleanly and call GPIO.cleanup().

Morse Code for Digits

- 0 = — — — — —
- 1 = · — — — —
- 2 = · · — — —
- 3 = · · · — —
- 4 = · · · · —
- 5 = · · · · ·
- 6 = — · · · ·
- 7 = — — · · ·
- 8 = — — — · ·
- 9 = — — — — ·

- **Dot (.)** = LED on for **1 unit (0.2s)**
- **Dash (-)** = LED on for **3 units (0.6s)**
- **Between symbols** = LED off for **1 unit**
- **Between digits** = pause for **3 units**
- **Between repeats** = pause for **7 units**

YORK U

Structure of a Python Program

- A Python program is made of **blocks of code** that work together:
  - **Imports (libraries)** extra tools you can use
  - **Functions** reusable instructions
    - **functions must be defined before they're called** in the main code.
  - **Main Code** where the program starts running

```python
# 1. Import libraries
import random

# 2. Define functions
def greet(planet):
    print("Hello,", planet, "!")

def countdown(n):
    for i in range(n, 0, -1):
        print(i)
    print("Liftoff!")

def choose_planet():
    planets = ["Mars", "Venus", "Jupiter"]
    return random.choice(planets)

# 3. Main code
countdown(5)
planet = choose_planet()
greet(planet)
```

YORK U

Program Entrance

```
if __name__ == '__main__':    # Program entrance
    print ('Program is starting ... \n')
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

- **__name__** → a special built-in variable in Python.
  - When you run a file directly (e.g., python Blink.py):
  - __name__ is set to "__main__".
  - The code under this block will run.

- **Why it's important**
  - Makes the file usable in **two ways**:
  - **As a script** → run it directly to blink the LED.
  - **As a module** → import its functions into another program

Handling Program Exit (try, except)

```
if __name__ == '__main__':    # Program entrance
    print ('Program is starting ... \n')
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

- **try:**
  - Run the main program (loop()) normally.
- **except KeyboardInterrupt:**
  - If the user presses **Ctrl + C** in the terminal, Python raises a special error called KeyboardInterrupt.
  - Instead of crashing, the program “catches” it.
- **destroy()**
  - Cleans up the GPIO pins safely (turns them off and releases them).
  - Prevents errors if you run another program later.

- **Why it’s important**
  - Keeps your Raspberry Pi stable.
  - Lets you stop the program safely.
  - Good coding practice for hardware control.

YORK U

Matching Types in Loops and Conditions

- When you loop through your student ID, the **type** matters:
  - **If the ID is stored as a string** ("1234"):
  - Each loop gives you a **string character** ("1", "2", ...).
  - Your if conditions must compare to **strings**:

```
if ch == "1":
    # Morse code for 1
```

- **If the ID is stored as a list of numbers** ([1, 2, 3, 4]):
  - Each loop gives you an **integer** (1, 2, ...).
  - Your if conditions must compare to **integers**:

```
if digit == 1:
    # Morse code for 1
```

- **If the ID is written as an integer** (1234) and converted with str():
  - The loop treats it as a **string of characters** again, so compare with strings ("1", "2").

Report Format (short, personal, verifiable)

- **1. Names & Group Info**
  - Group number, member names, date
- **2. Circuit Setup**
  - Write down which pin you used for the LED (e.g., pin 11, BOARD mode).
  - Add a short sentence in your own words:
    "We connected the LED to pin __ and ground, so that the Raspberry Pi could control it through Python."
- **3. Code (with comments)**
  - Paste your final Python code.
  - Add **comments** in your own words for each block (e.g., # dot = short blink, # this loop sends each digit of the ID).
- **4. Logic Explanation (in words)**
  - Write a short **step-by-step explanation** of what your program does.
- **4. Output (your run)**
  - Copy-paste the terminal printout (if any).
  - **Insert a photo of your physical circuit** (wires + LED connected to the Raspberry Pi).
- **5. Reflection** (Answer briefly in your own words)
  - What did you learn about Python (loops, functions, timing)?
  - What was easiest? What was hardest?
  - If you had more time, what is one improvement you would add (e.g., speed control, custom message, other components)?
- **7. In-Class Demonstration (**Each group will **show the instructor or the TA)**
  - The Python program running on the Raspberry Pi.
  - The LED blinking their student ID in Morse code.

Rubric

- **Rubric (10 points)**
  - Circuit setup (LED connected + pin explained with photo) – 1 pt
  - Code (runs correctly + comments show understanding) – 4 pts
  - Morse logic (dot/dash functions + digit patterns used correctly) – 2 pts
  - Loop & repetition (student ID transmitted continuously) – 2 pts
  - Reflection (original, not copy-paste) – 1 pt

- **Due Date:**
  - Monday, 11:59 PM

- **Quiz 2**
  - Release: Monday, 11:59 PM
  - Deadline: Friday, 12:00 PM (before labs)

Next Week

- Buttons & LEDs
- LED Bar Graph