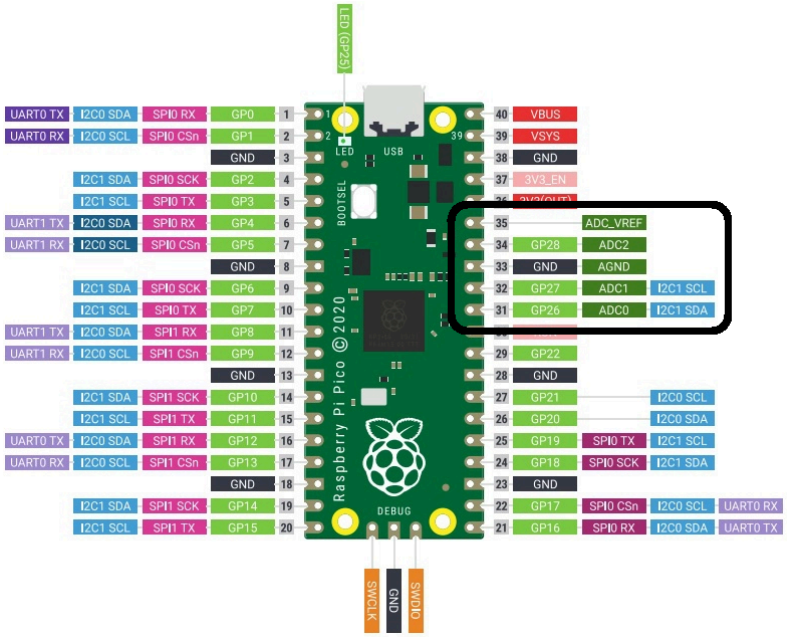
The ADC on RP2040 has a resolution of 12 bits, meaning that it can transform an analog signal into a number ranging from 0 to 4095 – though this is handled in MicroPython, transformed to a 16-bit number ranging from 0 to 65535. RP2040 has five ADC channels total, four of which are brought out to chip GPIOs: GP26, GP27, GP28, and GP29. On Raspberry Pi Pico, the first three of these are brought out to GPIO pins, and the fourth can be used to measure the VSYS voltage on the board. The ADC's fifth input channel is connected to a temperature sensor built into the RP2040.



It is possible to read any ADC channel either by using the PIN or by channel:

```
adc = machine.ADC(26)     # pin number
adc = machine.ADC(0)      # channel
```

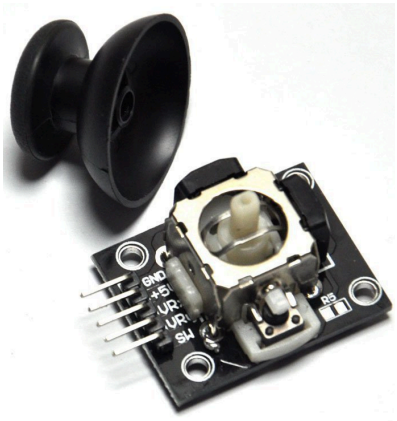
Task 1. Create a MicroPython script to read the internal temperature sensor and send this value to a computer.

```
from machine import ADC
from utime import sleep
```

```
#use channel instead of a number of a pin
temp_sensor = ADC(4)

while True:
    result = temp_sensor.read_u16()
     temperature = 27 - (result*3.3/65535 -
                         0.706)/0.001721
    print("Temperature: {:.2f}°C".format(temperature))
    sleep(2)
```

Typically, 0.706V means 27 degrees Celsius, with a slope of -1.721mV (0.001721) per degree.

Task 2. Connect a joystick to the Raspberry Pi Pico board.



Pico board	Joystick board
Gnd	Gnd
+3V3	+5V
GP26	VRx
GP27	VRy
GP15	SW

Create a MicroPython script to send values proportional to the stick position to the computer.

```
from machine import ADC
from utime import sleep
```

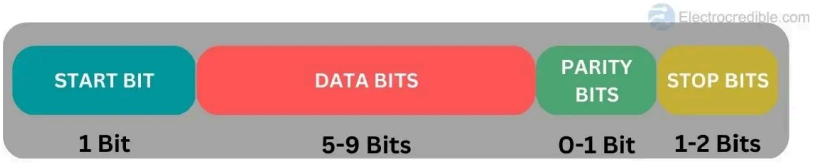
```
joy_X = ADC(26)
joy_Y = ADC(27)

while True:
    t_string = "X position = " + str(joy_X.read_u16())
    print(t_string)
    t_string = "Y position = " + str(joy_Y.read_u16())
    print(t_string)
    sleep(0.5)
```

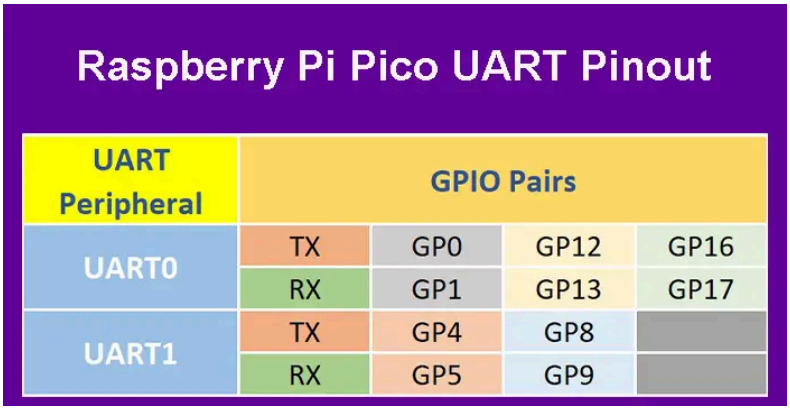
Task 3. Create a MicroPython script to send to the computer, values from 0 to 10, proportional to the current stick position.

```
from machine import ADC
from utime import sleep

def analog_map(x, in_min, in_max, out_min, out_max):
     return int((x-in_min)*(out_max-out_min)/
               (in_max-in_min)+ out_min)

joy_X = ADC(26)

while True:
     result = analog_map(joy_X.read_u16(),0,65535,0,10)
     t_string = "X position = " + str(result)
     print(t_string)
     sleep(1)
```

Task 4. UART communication. Connect the FTDI232 Serial-to-UART converter to a Raspberry Pi Pico board:

- RX should be connected to GPIO4
- TX should be connected to GPIO5






```
from machine import Pin,UART
from utime import sleep

uart1 = UART(1,baudrate=9600, tx=Pin(4), rx=Pin(5))
uart1.init(bits=8, parity=None, stop=1)

led = Pin('LED', Pin.OUT)

while True:
    uart1.write("a"+ "\n")
    if uart1.any():
        char = uart1.read()
        if char == b'x':
            led.toggle()
    sleep(0.5)
```

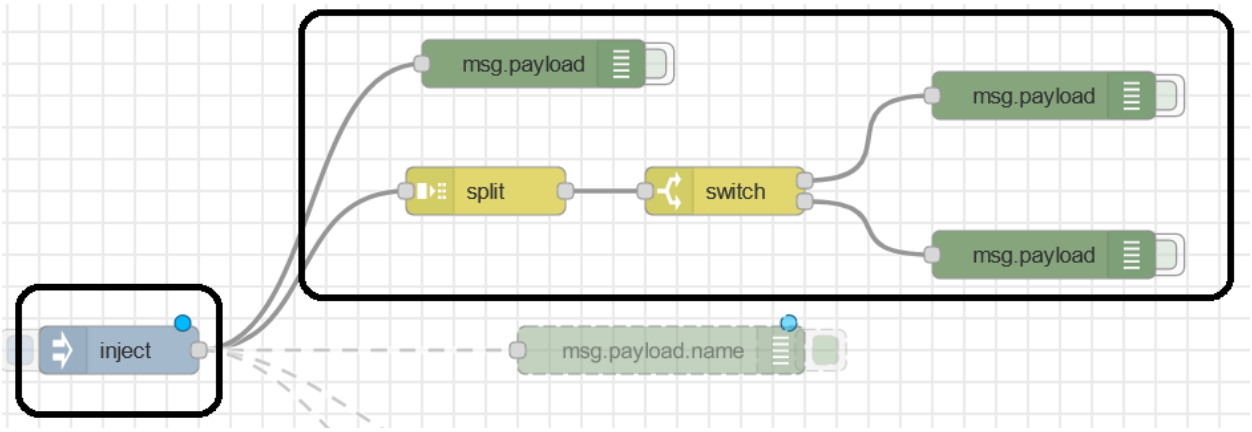
Task 5. JSON example.

```
from machine import Pin,UART
from utime import sleep
import ujson

uart1 = UART(1,baudrate=9600, tx=Pin(4), rx=Pin(5))
uart1.init(bits=8, parity=None, stop=1)

led = Pin('LED', Pin.OUT)

payload = {
    "command": "start",
    "value": 10
}

while True:
    uart1.write(ujson.dumps(payload) + "\n")
    if uart1.any():
        char = uart1.read()
        if char == b'x':
            led.toggle()
    sleep(0.5)
```

Create a Node-Red dashboard to test the communication channel. The procedure for starting the Node-RED environment is as follows:

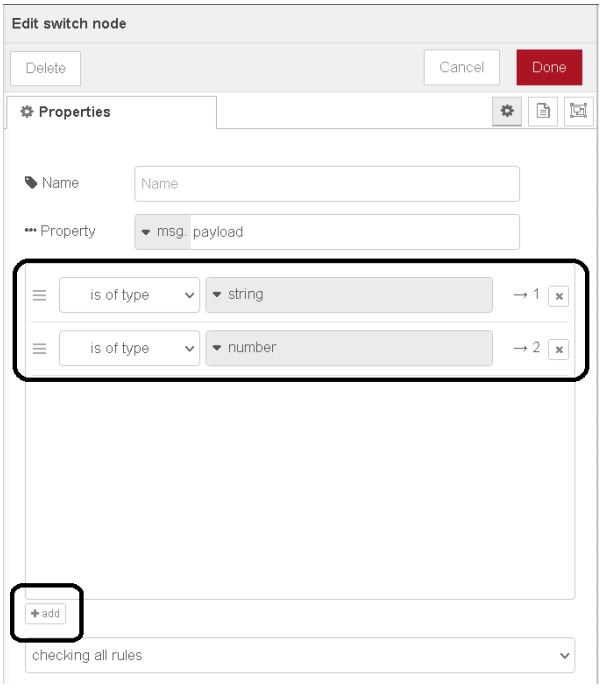
- launch the command line/Terminal;
- execute the `node-red` command;
- launch a web browser of Your choice;
- type `127.0.0.1:1880` in the address bar. Alternatively, you can enter `localhost:1880`.

Implement the indicated parts of the flow diagram.

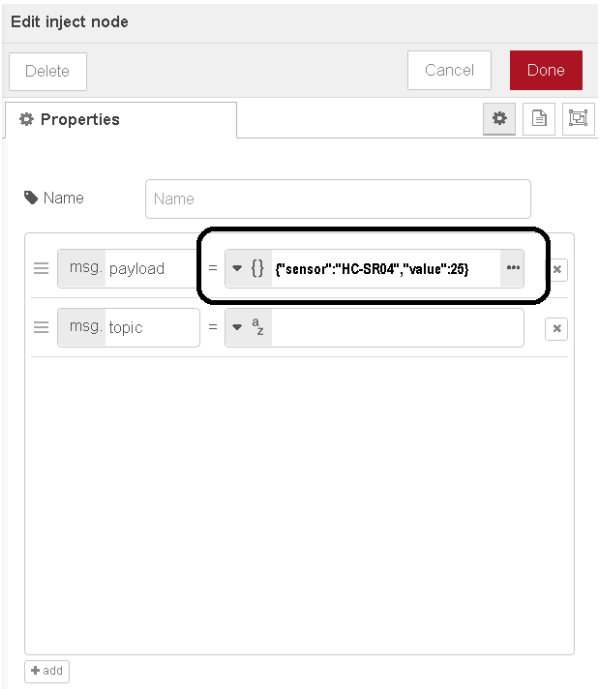


The *switch* node and the *inject* node configurations are presented below.

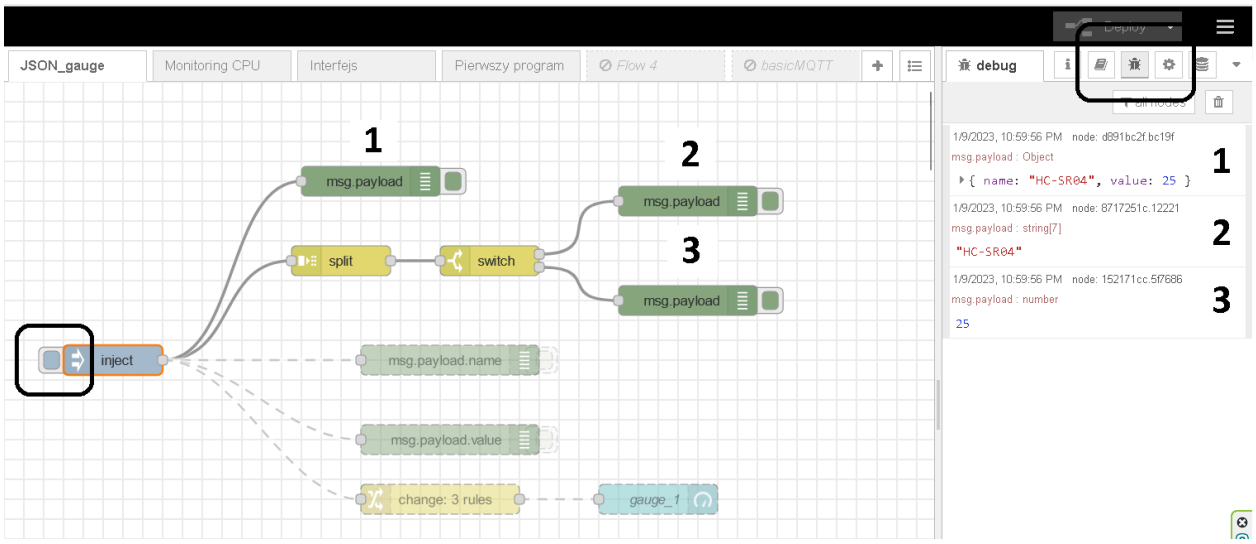
switch node configuration



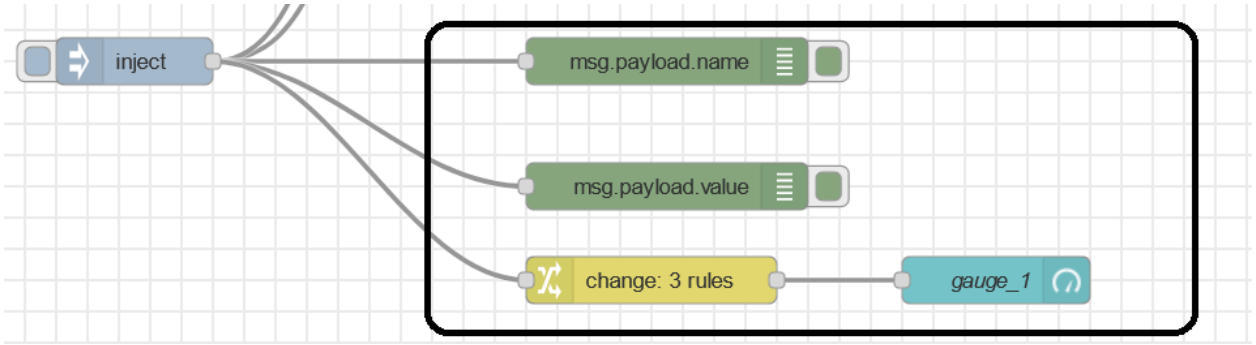
inject node configuration



Press the ***Deploy*** button. Trigger the *inject* node with the button on the left side of the block. Observe results in the *Debug* window.

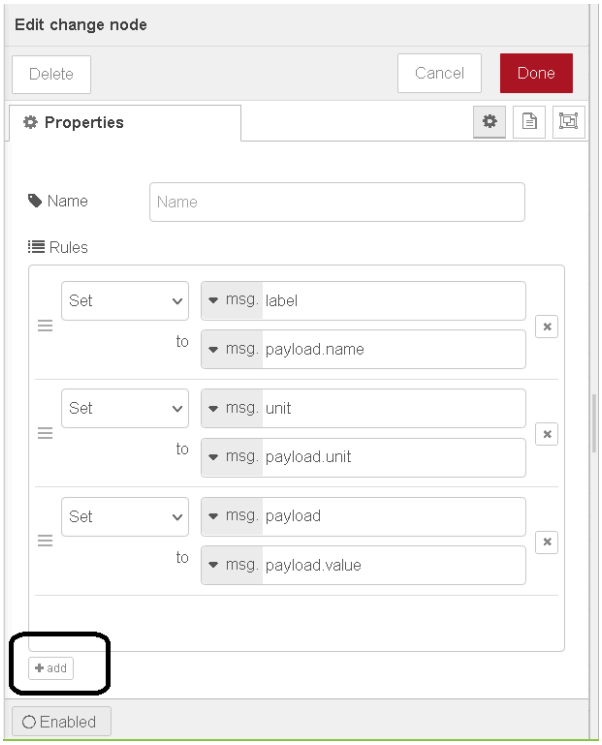


Implement the rest of the following flow diagram.

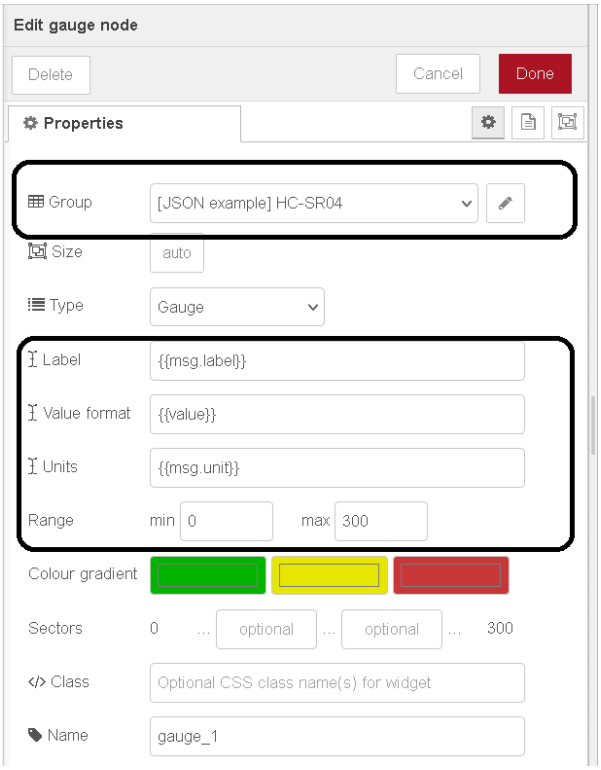


The *change* node and the *gauge* node configurations are presented below.

change node configuration



gauge node configuration



By default, only one rule is available inside a *change* node. To add more rules press the +*add* button.

A *gauge* node allows for building a custom user interface. In the Node-RED environment, the interface is called the dashboard. Flows that allow You to build a dashboard are not part of a typical *Node-RED* package installation. They must be added separately using the options *Manage palette* (*Menu → Manage palette*). After selecting the *Manage palette* option, the *User Settings* window opens. The first step in the installation process is to switch to the *Install* tab. Then, in the search box, type *dashboard.* From the list of available options, select *node-red-dasboard* and press the *Install* button. After completing the installation process, press the *Close* button. We can

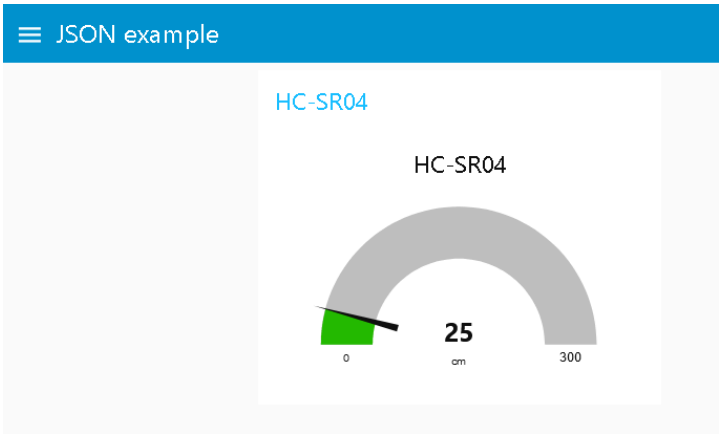
check the correctness of the installation by scrolling the window with available blocks to the *dashboard* tab.

Each object on the interface must be assigned to a tab (*Tab*) and a group (*Group*). If You do not have defined groups and tabs in the newly built diagram, so with the help of *Add new dashboard group...* enter the name of the group as *HC-SR04*. A tab is entered with *Add new dashboard tab...* Call it as You wish eg. *JSON example*. Confirm with the *Add* button - 2 times. When the *Done* button is pressed, the configuration of the *Button* control is complete. Change *Label* and *Unis* properties according to `{{msg.label}}` and `{{msg.unit}}` respectively.

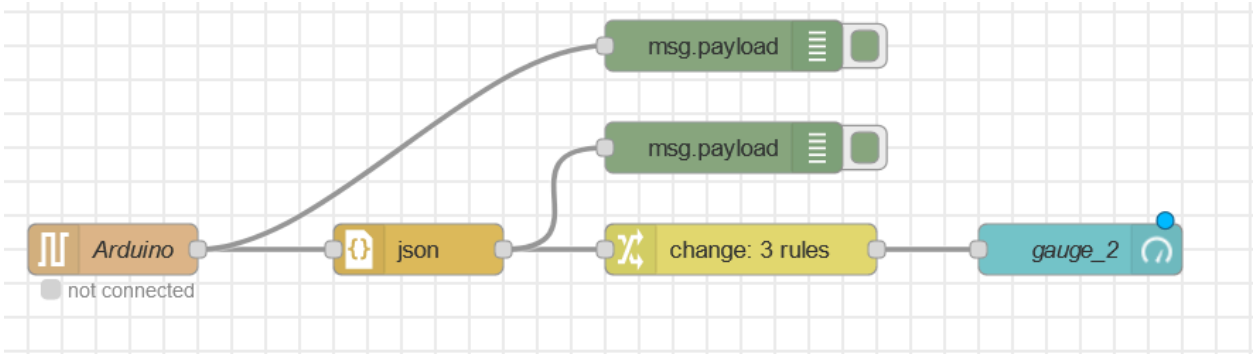
Modify the JSON object inside the *inject* node:

```
{ "name" : "HC-SR04", "value" : 0.0, "units" : "cm" }
```

Press the ***Deploy*** button. Open a new window in a web browser. Type `127.0.0.1:1880/ui` in the address bar. Alternatively, you can enter `localhost:1880/ui`.



Add the *serial in* node. Configure connection parameters as we did in exercise no 5. Change the name of this node to Arduino. The *json* node is available in the *parser* palette.



Task 6. (*own work*) Create a MicroPython script that displays ambient temperature in Celsius degrees, Kelvins, and Fahrenheit degrees values using a *Node-RED*-based interface.

Task 7. (*own work*) Display temperature vs time graph on the Node-RED-based interface.

Task 6 or Task 7. Presenting and sending the solution afterward is mandatory.

For those interested:

1. MicroPython web page:
   micropython.org/download/rp2-pico/
2. RandomNerd tutorial on analog inputs:
   randomnerdtutorials.com/raspberry-pi-pico-analog-inputs-micropython/
3. UART library:

docs.micropython.org/en/latest/library/machine.UART.html#machine.UART.irq

Project no 1.

The project must meet the following requirements:

1. User Interface:
   The user interface shall be developed using Node-RED.
2. Sensor Data Presentation:
   The interface shall display measurement results obtained from the built-in temperature sensor.
3. Control Elements:
   The interface shall include a minimum of three control buttons: one to start measurement, one to stop measurement, and one to display instant measurement.
4. Data Visualization:
   The interface shall contain at least one chart presenting the collected measurement data.
5. Indicators:
   The interface shall include at least one indicator showing the temperature value.
6. Data Exchange Format:
   Communication and data exchange shall be carried out using the JSON format.