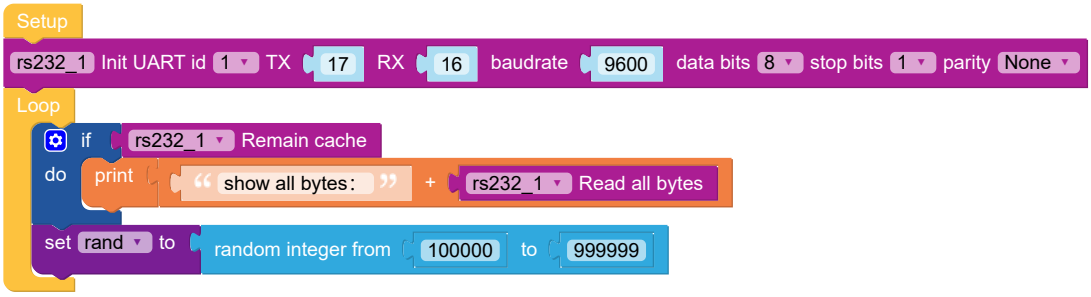
Module13.2 RS232F / Module13.2 RS232M

Example

Initialize the RS232 communication module (UART1), and in the loop, check if there is remaining buffered data. If there is, read and display all byte data, while also generating a random integer.

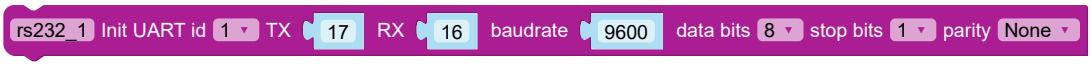



```
from m5stack import *
from m5ui import *
from uiflow import *
import module

setScreenColor(0x00006d)

rand = None

rs232 = module.get(module.RS232)

import random

rs232_1 = rs232.init(1, tx=17, rx=16, baudrate=9600, data_bits=8, stop_bits=1, parity=None)
while True:
  if rs232_1.any():
    print((str('show all bytes: ') + str((rs232_1.read()))))
  rand = random.randint(100000, 999999)
  wait_ms(2)
```

API


```
any()
```

- Check and retain the buffered data in the RS232 module. Used to read unprocessed input data.




```
rs232.init(1, tx=17, rx=16, baudrate=9600, data_bits=8, stop_bits=1, parity=None)
```

- Initialize the RS232 communication module, setting the UART interface ID, transmit (TX) pin, receive (RX) pin, baud rate, data bits, stop bits, and parity settings. This initialization must be done before using other RS232 functions.
  - UART ID: The number of the UART interface being used.
  - TX Pin: The pin number connected to the transmitter.
  - RX Pin: The pin number connected to the receiver.
  - Baud Rate: Communication speed (e.g., 9600).
  - Data Bits: The number of bits in each data frame (e.g., 8 bits).
  - Stop Bits: The number of stop bits used after each frame (e.g., 1 bit).
  - Parity: The error detection mechanism used for data transmission (e.g., no parity).


```
read()
```

- Read all available byte data from the RS232 module. This operation will read all the data in the buffer until no more bytes are available.


```
readline()
```

- Read a line of data from the RS232 module. Typically, this will read until a newline character or a specified end character is encountered.


```
read(10)
```

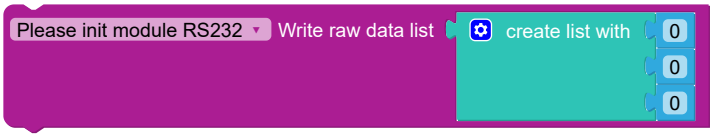
- Read a specified number of characters from the RS232 module (10 characters in the example). It will read the specified number of characters and then stop.


```
write(''+"\r\n")
```

- Write (send) the specified text through the RS232 module to the connected device or system. This operation sends the complete string.


```
write(''+"\r\n")
```

- Write (send) the specified text as a line through the RS232 module, automatically adding a newline character (e.g., \n) at the end of the text. This is commonly used for sending data line by line.



```
write(bytes([0, 0, 0]))
```

- Write (send) a list containing raw data (e.g., byte array) through the RS232 module. This method allows sending binary data, not just text strings.