

Luz, micro:bit, ação!

Illumine seus projetos com MicroPython

Juliana Karoline de Sousa | [@julianaklulo](https://twitter.com/julianaklulo)



Juliana Karoline de Sousa



- ★ **Bacharel em Ciência da Computação | UFSCar**
- ★ **PyLadies São Carlos | co-fundadora e organizadora**
- ★ **Grupy-Sanca | co-fundadora e organizadora**
- ★ **Omnivector | software engineer**
- ★ **IoT, robótica, impressão 3D, MicroPython :3**





Agenda

- 01 MicroPython**
- 02 BBC micro:bit**
- 03 Editor Python**
- 04 NeoPixels**
- 05 Exercícios**
- 06 Projeto final: Pacman**





Repositório do Tutorial



Exemplos, exercícios e slides disponíveis em:
<https://tinyurl.com/tutorial-neopixel>



01

MicroPython

Python para microcontroladores

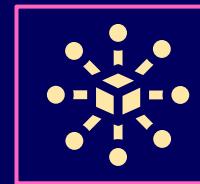
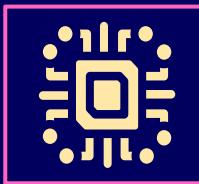
Microcontroladores?



O que é um microcontrolador?

Círcuito integrado

Processador, memória e periféricos em um único chip

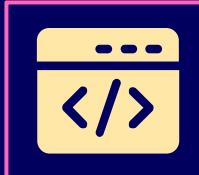


I/O

Portas integradas de entrada e saída para interagir com sensores e atuadores

Programável

Pode armazenar código customizado na memória



Baixo consumo de energia

Pode ser utilizado em projetos alimentados por baterias



O que é o MicroPython?



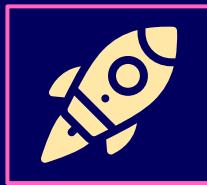
O **MicroPython** é uma implementação enxuta e eficiente do Python 3 que inclui um pequeno subconjunto da biblioteca padrão e é otimizada para execução em **microcontroladores** e em ambientes restritos.



Por que usar o MicroPython?

Produtividade

Sintaxe mais simples do que C/C++, poucas linhas já são o suficiente



Abstração

É possível abstrair a camada de hardware de acordo com a aplicação

Alto nível

Permite escrever código orientado a objetos



Python

Segue a filosofia da linguagem Python



Características do MicroPython

Features:

- ★ Prompt interativo (REPL)
- ★ List comprehensions e generators
- ★ Tratamento de exceções
- ★ Acesso aos protocolos GPIO, PWM, ADC, UART, SPI

Requisitos:

- ★ 256k de espaço de código
- ★ 16k de memória RAM

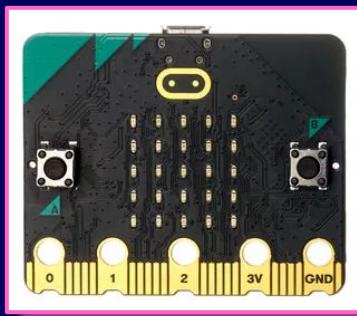
Microcontroladores que rodam MicroPython



PyBoard



ESP8266/ESP32



BBC micro:bit



Raspberry
Pi Pico



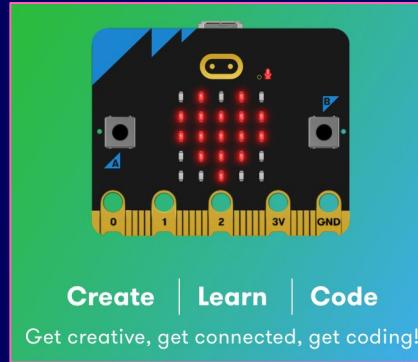
Arduino Nano
RP2040

02

BBC micro:bit

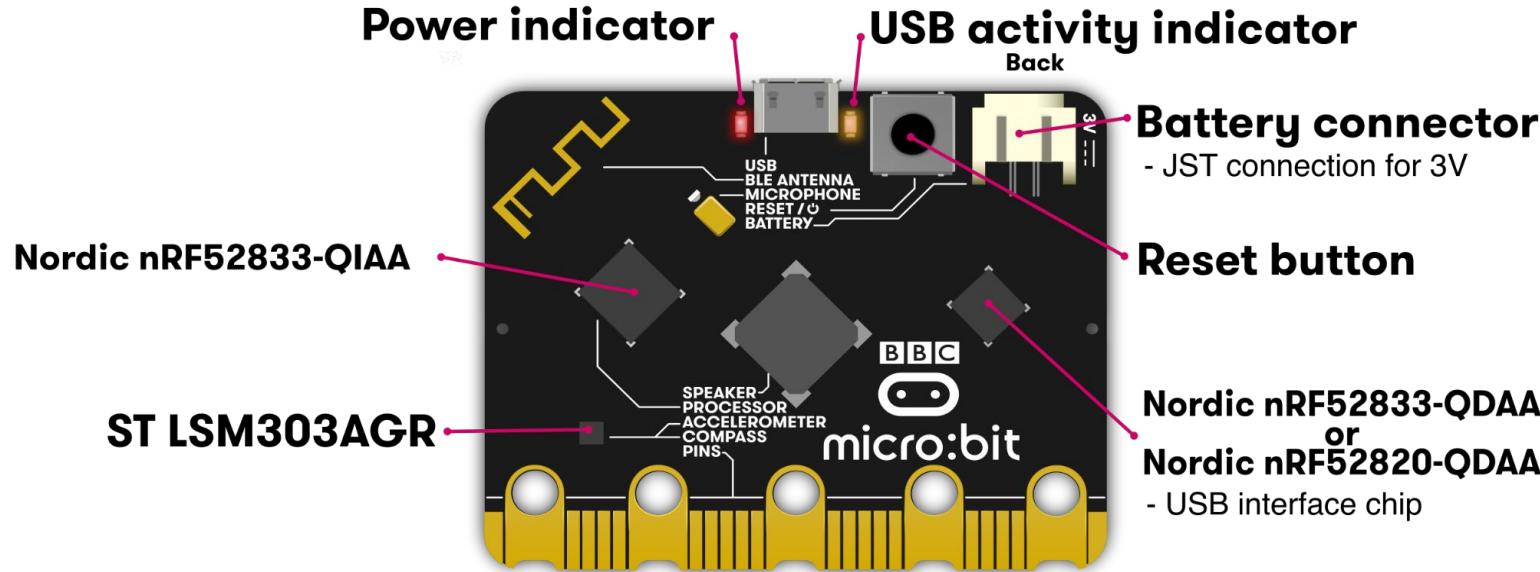
Computador de bolso para computação física

O que é a BBC micro:bit?

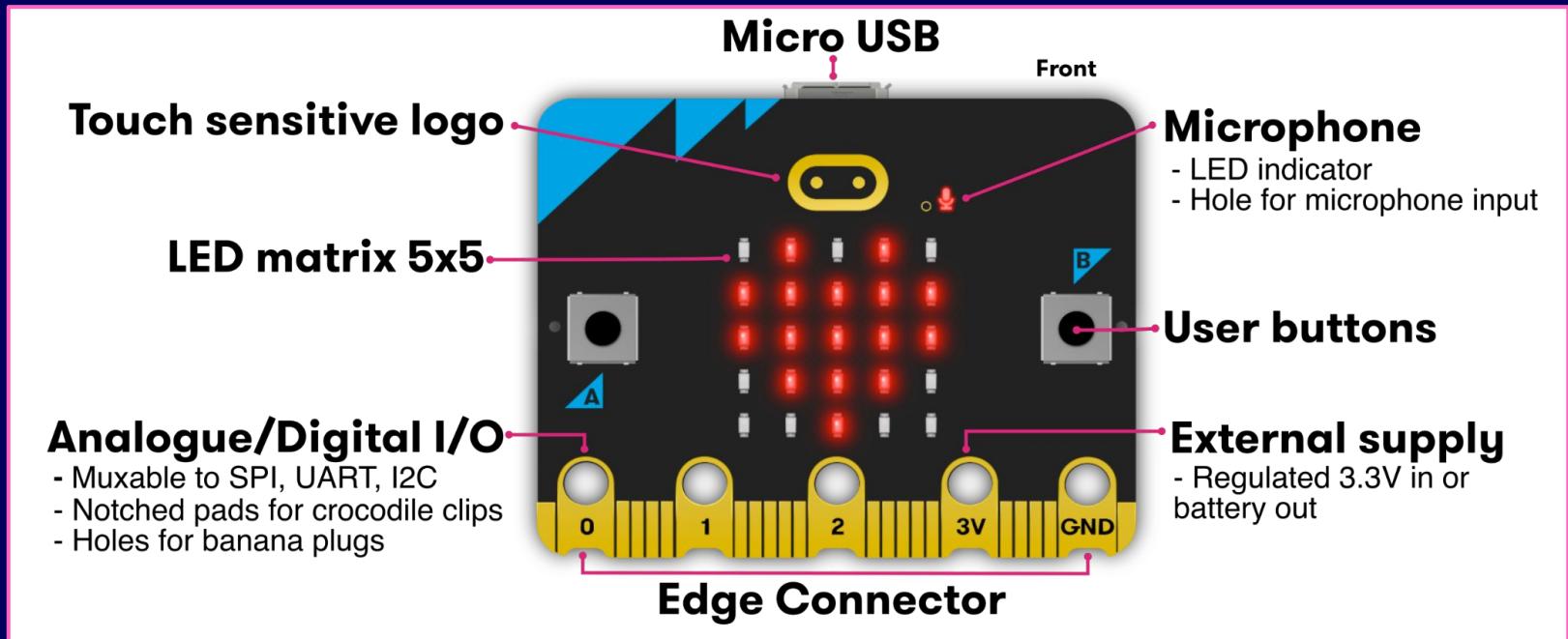


A **BBC micro:bit** é um computador de bolso que ensina como o software e o hardware funcionam juntos. Baseia-se no conceito de **computação física**, onde utiliza-se código para interagir com o mundo físico.

Componentes da BBC micro:bit



Componentes da BBC micro:bit



03

Editor Python

Programando a BBC micro:bit no navegador



<https://python.microbit.org>

The screenshot shows the Python code editor for the micro:bit. On the left, a sidebar lists various components: Variables, Display, Buttons, Loops, Logic, and Accelerometer. The 'Buttons' section is currently selected. The main area displays an 'Untitled project' with the following Python code:

```
1 # Imports go at the top
2 from microbit import *
3
4
5 # Code in a 'while True:' loop repeats forever
6 while True:
7     display.show(Image.HEART)
8     sleep(1000)
9     display.scroll('Hello')
10
```

To the right of the code is a preview of the micro:bit board showing a heart icon on the display and the text 'Hello' scrolling across it. Below the preview are several control sliders for shake, brightness, contrast, rotation, and volume. At the bottom are buttons for 'Send to micro:bit', 'Save', and 'Open...'. The overall interface has a purple and teal color scheme.

Como utilizar a BBC micro:bit?

Navegadores suportados (Google Chrome)

Coloque o cabo na placa e no computador

Escreva código .py no editor online

Clique em “**send to micro:bit**” e selecione o dispositivo

O arquivo .hex é salvo automaticamente na micro:bit

Outros navegadores (Firefox)

Coloque o cabo na placa e no computador

Escreva código .py no editor online

Clique em “**send to micro:bit**” e baixe o arquivo .hex no computador

Copie o arquivo .hex para o dispositivo da micro:bit





Hello,World!

Execute o código de exemplo na placa



04

NeoPixels

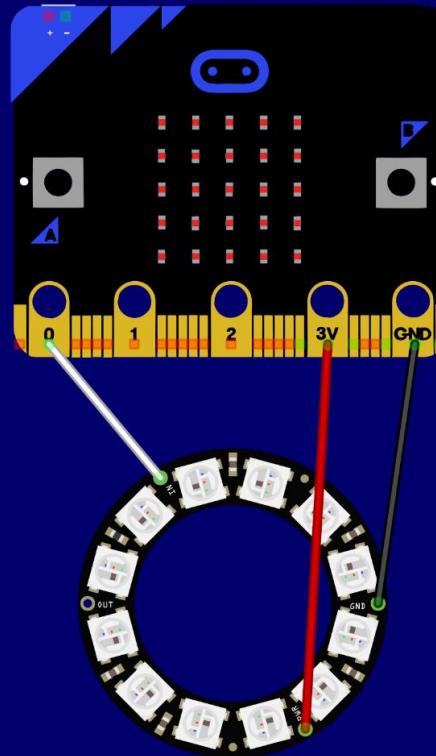
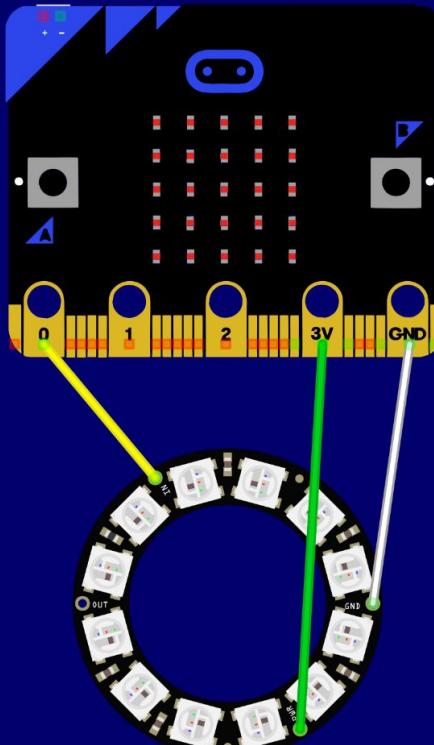
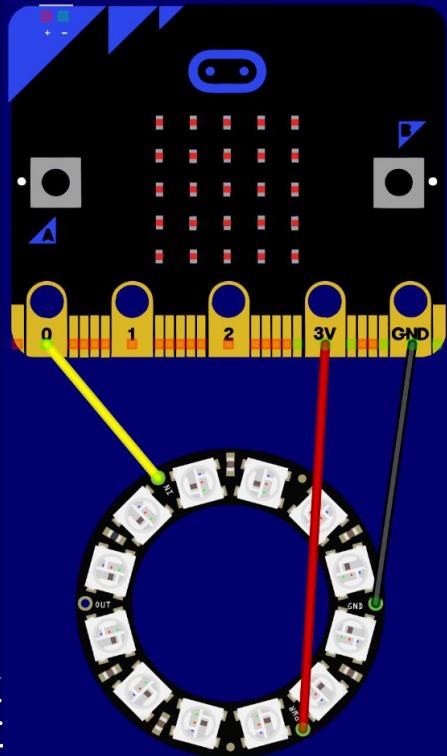
Controlando LEDs endereçáveis

O que são NeoPixels?



NeoPixels são sequências de LEDs endereçáveis que podem ser controlados individualmente utilizando somente um fio para comunicação.

Conectando o anel de LED na micro:bit



05

Exercícios

Controlando os LEDs com os sensores da micro:bit

Exercício #0

Usando os exemplos da documentação

Exemplo #1 - Acender um LED

```
from microbit import *
import neopixel

np = neopixel.Neopixel(pin0, 12)
np[0] = (63, 63, 0)
np.show()
```

Exemplo #2 - Acender todos os LEDs

```
from microbit import *
import neopixel

np = neopixel.NeoPixel(pin0, 12)
np.fill((0,0,63))
np.show()
```

Exemplo #3 - Acender vários LEDs

```
from microbit import *
import neopixel

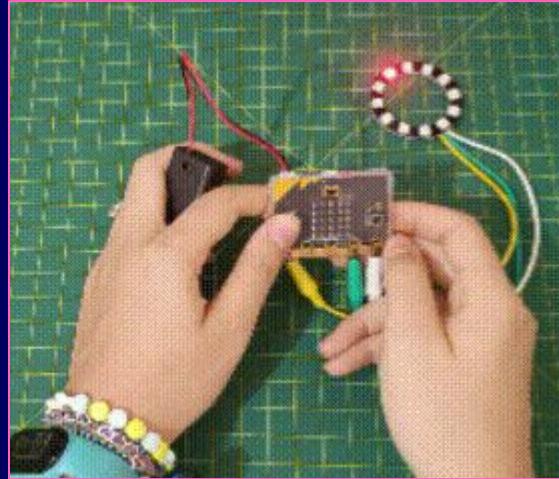
np = neopixel.NeoPixel(pin0, 12)
for pixel_id in range(len(np)):
    np[pixel_id] = (127, 0, 127)
    np.show()
    sleep(1000)
```

Exemplo #4 - Apagar LEDs

```
from microbit import *
import neopixel

np = neopixel.NeoPixel(pin0, 12)
for pixel_id in range(5):
    np[pixel_id] = (0, 63, 0)
    np.show()
    sleep(1000)

np.clear()
```

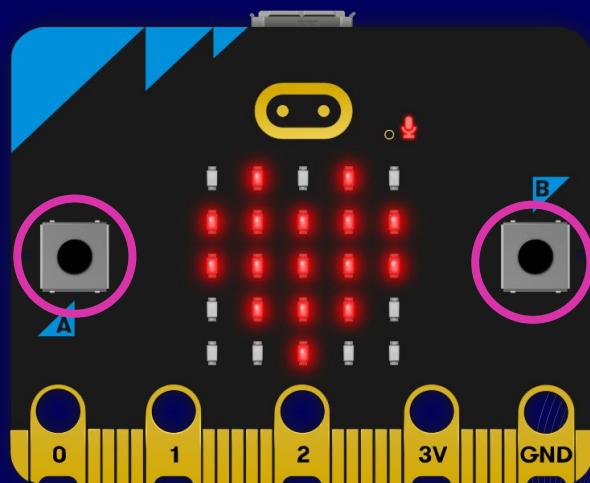


Exercício #1

Mover o LED aceso com os botões A e B

Como usar os botões da BBC micro:bit?

```
from microbit import *\n\n# Botão A foi pressionado?\nif button_a.was_pressed():\n    pass\n\n# Botão A está sendo pressionado?\nif button_a.is_pressed():\n    pass\n\n# Quantas vezes o botão A foi pressionado?\nquantity = button_a.get_presses()
```



Exercício #1: Mover o LED aceso com os botões

01 Defina um index

Será utilizado para indicar o LED aceso

02 While True

Execute o código dentro de um loop, com um sleep no final

03 Apague o index

Coloque (0, 0, 0) na cor do LED para apagá-lo

04 Verifique os botões

Aumente ou diminua o index de acordo com o botão apertado

05 Acenda o index

Escolha uma cor e use `np.show()` para acender o LED

Dica: use `% 12` ao ajustar o index para dar a volta no anel

Exercício #1: Mover o LED aceso com os botões

```
from microbit import *
import neopixel

pixels = 12
np = neopixel.NeoPixel(pin0, pixels)

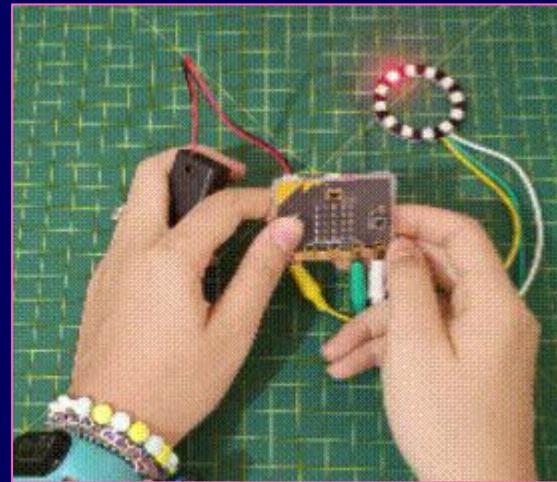
index = 0

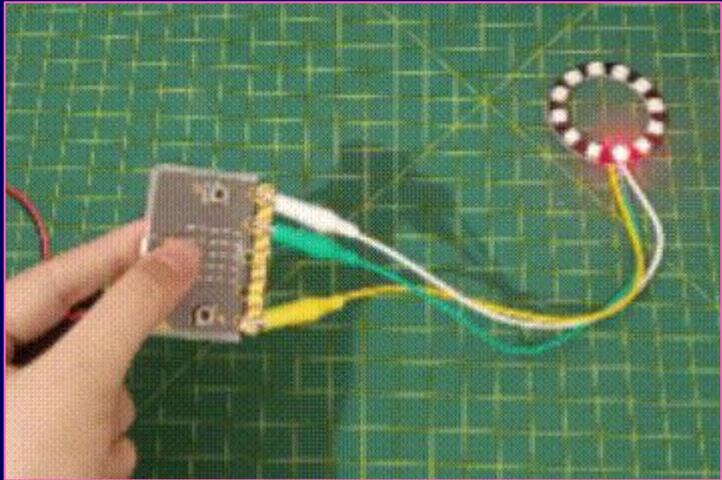
while True:
    np[index] = (0, 0, 0)

    if button_a.was_pressed():
        index = (index - 1) % pixels
    elif button_b.was_pressed():
        index = (index + 1) % pixels

    np[index] = (255, 0, 0)
    np.show()

    sleep(100)
```



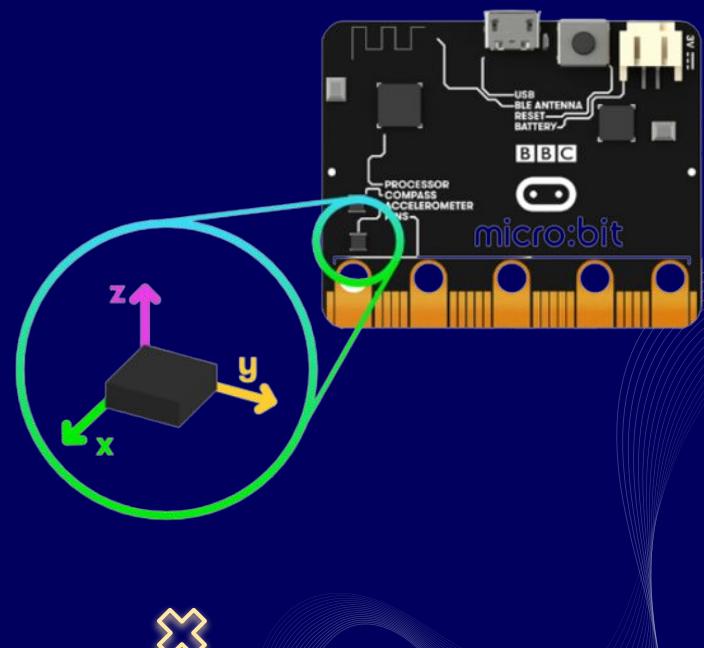


Exercício #2

Acender um LED aleatório quando balança a placa

Como usar acelerômetro da BBC micro:bit?

```
from microbit import *\n\n# Ler aceleração em cada eixo\nx = accelerometer.get_x()\ny = accelerometer.get_y()\nz = accelerometer.get_z()\n\n# Gesto está acontecendo?\nif accelerometer.is_gesture("face up"):\n    pass\n\n# Gesto aconteceu desde a última leitura?\nif accelerometer.was_gesture("shake"):\n    pass
```



Exercício #2: Acender LED aleatório ao balançar

01 Defina um index

Será utilizado para indicar o LED aceso

02 While True

Execute o código dentro de um loop, com um sleep no final

03 Apague o index

Coloque (0, 0, 0) na cor do LED para apagá-lo

04 Verifique o gesto

Verifique se a placa está sendo balançada e sorteie um index

05 Acenda o index

Escolha uma cor e use `np.show()` para acender o LED

Dica: use `pixels - 1` como intervalo do random

Exercício #2: Acender LED aleatório ao balançar

```
from microbit import *
import neopixel
import random

pixels = 12
np = neopixel.NeoPixel(pin0, pixels)

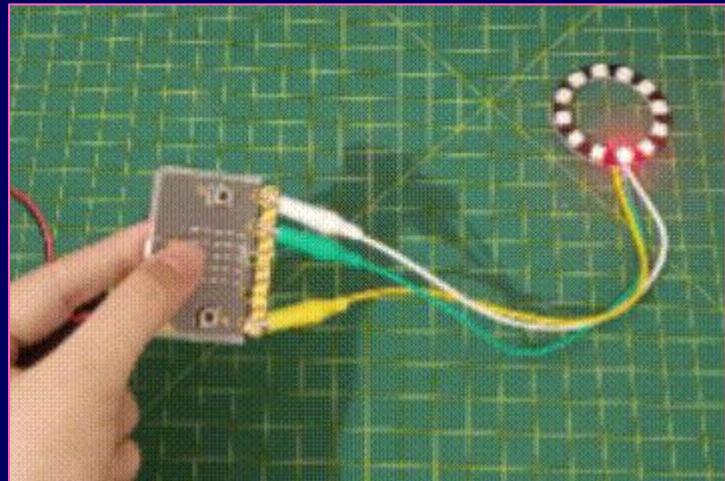
index = 0

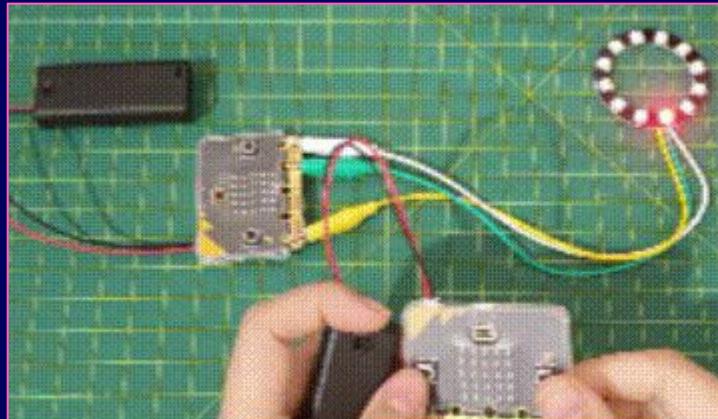
while True:
    np[index] = (0, 0, 0)

    if accelerometer.is_gesture("shake"):
        index = random.randint(0, pixels - 1)

    np[index] = (255, 0, 0)
    np.show()

    sleep(100)
```





Exercício #3

Mover LED através de mensagens de rádio de outra placa

Como usar rádio da BBC micro:bit?

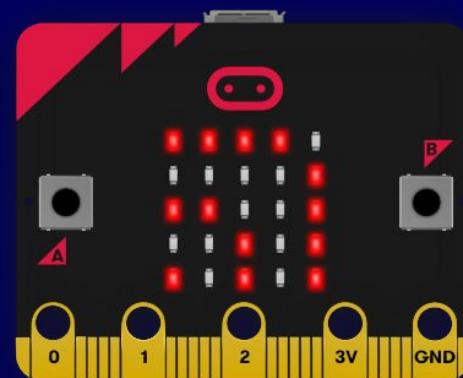
```
import radio

# Ligar o rádio
radio.on()

# Configurar o grupo e a força do sinal
# Group: 0-255, Power: 0-7
radio.config(group=23, power=7)

# Receber uma mensagem do grupo
message = radio.receive()

# Enviar uma mensagem para o grupo
radio.send("Python Brasil 2024")
```



Exercício #3: Mover LED através do rádio



Sender

02 Calcule o index

Leia o x do acelerômetro e calcule o index de acordo

03 Envie a mensagem

Use `radio.send()` para transmitir o index para a outra placa

01 Defina um grupo

Ambas as placas precisam estar no mesmo grupo para funcionar

Receiver

02 Receba a

Verifique se uma mensagem foi recebida com `radio.receive()`

03 Acenda o LED indicado

Apague o index antigo e acenda o novo index recebido pelo rádio



Exercício #3: Mover LED através do rádio

```
from microbit import *
import radio

channel = 0
radio.config(channel=channel)
radio.on()

pixels = 12
limit = 250
index = 0

while True:
    if accelerometer.get_x() > limit:
        index = (index - 1) % pixels
    elif accelerometer.get_x() < -limit:
        index = (index + 1) % pixels

    radio.send(str(index))
    sleep(100)
```

Sender

```
from microbit import *
import neopixel
import radio

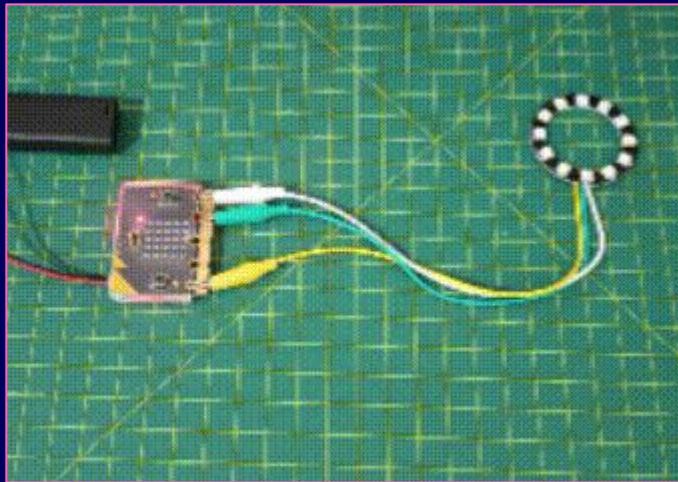
channel = 0
radio.config(channel=channel)
radio.on()

pixels = 12
np = neopixel.NeoPixel(pin0, pixels)

index = 0

while True:
    message = radio.receive()
    if message:
        np[index] = (0, 0, 0)
        index = int(message)
        np[index] = (255, 0, 0)
        np.show()
        sleep(100)
```

Receiver

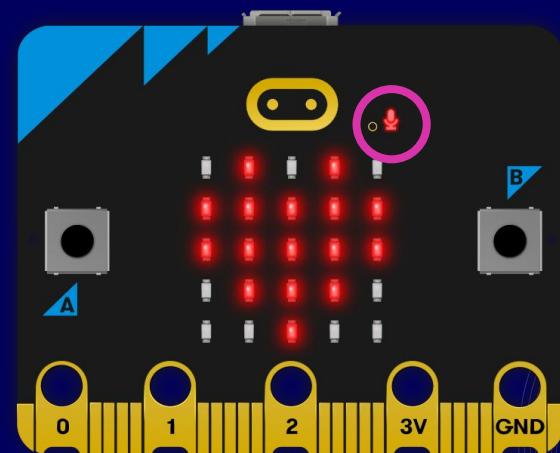


Exercício #4

Trocar a cor do anel de acordo com o som do microfone

Como usar microfone da BBC micro:bit?

```
from microbit import *\n\n# Ler a intensidade do som (0-255)\nlevel = microphone.sound_level()\n\n# Ler evento de som atual (LOUD ou QUIET)\nevent = microphone.current_event()\n\n# Ler os eventos que já ocorreram\ncurrent_events = microphone.get_events()
```



Exercício #4: Trocando a cor de acordo com o som

01 Defina um limite

Será utilizado para saber se o som é alto o suficiente

02 While True

Execute o código dentro de um loop, com um sleep no final

03 Verifique o som

Leia o `sound_level()` e verifique se é maior que o limite definido

04 Sorteie uma cor

Use o `random` para sortear um valor para cada canal RGB

05 Acenda o anel todo

Use `np.fill()` e `np.show()` para acender o anel de LED

Dica: use **255** como intervalo do random

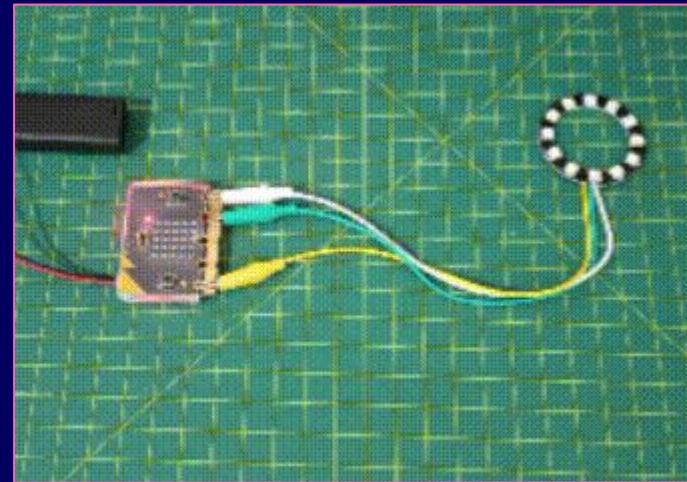
Exercício #4: Trocando a cor de acordo com o som

```
from microbit import *
import neopixel
import random

pixels = 12
np = neopixel.NeoPixel(pin0, pixels)

sound_limit = 50

while True:
    if microphone.sound_level() > sound_limit:
        r = random.randint(0, 255)
        g = random.randint(0, 255)
        b = random.randint(0, 255)
        np.fill((r, g, b))
    np.show()
    sleep(100)
```

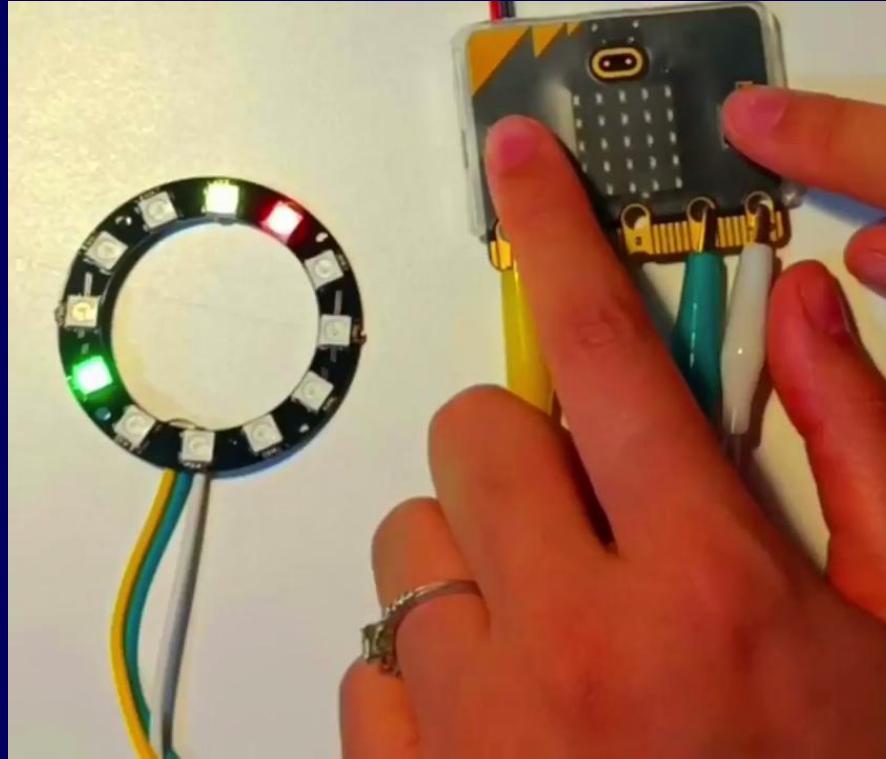


06

Projeto Final

Criando o jogo do Pacman

Projeto Final: Jogo do Pacman



Projeto Final: Jogo do Pacman



Pacman



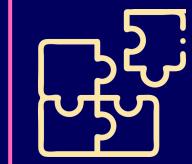
Fantasma

Direção do movimento é definida pelos botões

Muda de cor quando o Pacman fica forte



Fruta



Jogo

Dá força para o Pacman e permite captura

Roda até o Pacman ser capturado pelo fantasma

Recursos novos

Aprendendo a usar o display, o alto-falante e o contador de tempo

Como usar display da BBC micro:bit?

```
from microbit import *

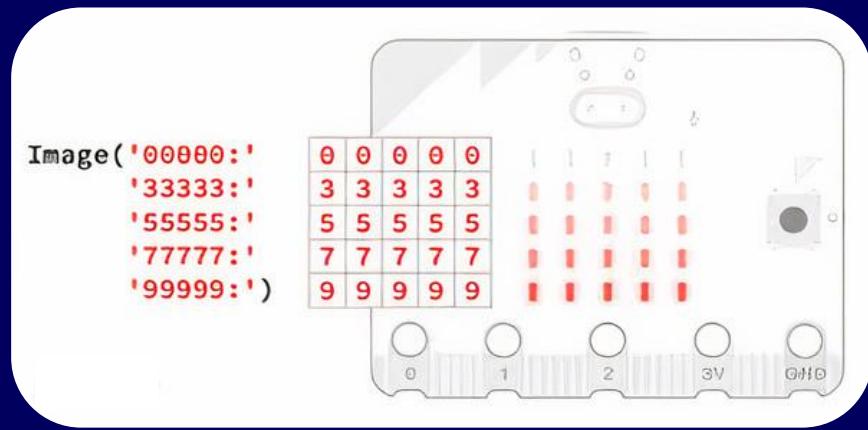
# Mostrar uma imagem predefinida
display.show(Image.HEART)

# Mostrar uma imagem personalizada
display.show(
    Image(
        "00000:"
        "33333:"
        "55555:"
        "77777:"
        "99999"
    )
)

# Mostrar um texto
display.scroll("Python Brasil 2024")

# Mostrar um pixel
display.set_pixel(0, 0, 9)

# Limpar o display
display.clear()
```



Como usar alto-falante da BBC micro:bit?

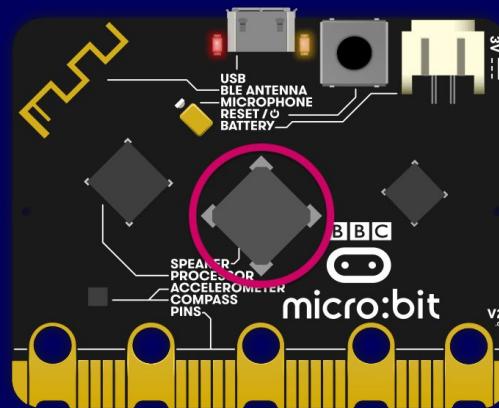
```
from microbit import *
import music, speech, audio

# Tocar música predefinida
music.play(music.POWER_UP)

# Criar sua própria música - formato: '{nota}{oitava}:{duração}'
music.play(
    [
        "C4:2", "D4:2", "E4:2", "F4:2",
        "G4:2", "A4:2", "B4:2",
    ]
)

# Texto para fala
speech.say("Hello, Python Brasil!")

# Tocar sons expressivos
audio.play(Sound.YAWN)
```



time.ticks_ms() e time.ticks_diff()

time.ticks_ms() é uma função que gera um contador de tempo, em milisegundos, que começa no momento em que foi instanciado.

time.ticks_diff() serve para calcular a diferença de tempo entre dois contadores criados com **time.ticks_ms()**.

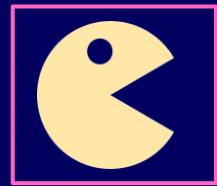
```
tempo_1 = time.ticks_ms() # começou às 10:00:00  
tempo_2 = time.ticks_ms() # começou às 10:00:05  
time.ticks_diff(tempo_2, tempo_1) # retorna 5000ms
```

Exemplo: time.ticks_ms() e time.ticks_diff()

```
from microbit import *
import time

start_time = time.ticks_ms()

while True:
    now = time.ticks_ms()
    if time.ticks_diff(now, start_time) > 3000:
        display.show(Image.HEART)
        break
```



Parte #1

Criando o Pacman

Projeto Final #1: Pacman

01 `_init_(self, pixels)`

Configura a posição, a cor, velocidade, direção e força

02 `next_position(self)`

Calcula a próxima posição baseado na direção

03 `move(self)`

Move a cada 1000ms, na direção definida pelos botões

04 `show()`

Atualiza o index com a nova posição calculada

Pacman.__init__()

```
class Pacman:  
    def __init__(self, pixels):  
        self.pixels = pixels  
        self.position = random.randint(0, self.pixels - 1)  
        self.color = (238, 173, 45)  
  
        self.speed = 1000  
        self.direction = 1  
        self.last_moved = time.ticks_ms()  
  
        self.strong = False
```

Pacman.next_position() and Pacman.move()

```
class Pacman:  
    ...  
    def next_position(self):  
        return (self.position + self.direction) % self.pixels  
  
    def move(self):  
        if button_a.was_pressed():  
            self.direction = -1  
        elif button_b.was_pressed():  
            self.direction = 1  
  
        if time.ticks_diff(time.ticks_ms(), self.last_moved) > self.speed:  
            self.position = self.next_position()  
            self.last_moved = time.ticks_ms()
```

Pacman.show()

```
class Pacman:  
    ...  
    def show(self, np):  
        np[self.position] = self.color
```



Parte #2

Criando o Fantasma

Projeto Final #2: Ghost

01 `_init_(self, pixels)`

Configura a posição, as cores, velocidade e direção

02 `next_position(self)`

Calcula a próxima posição baseado na direção

03 `move(self)`

Move a cada 1500ms, seguindo no sentido horário

04 `randomize_position()`

Calcula uma nova posição aleatória quando é capturado

05 `set_color()`

Decide se vai ficar azul ou verde de acordo com a força

06 `show()`

Atualiza o index com a nova posição calculada

Ghost.__init__()

```
class Ghost:  
    def __init__(self, pixels):  
        self.pixels = pixels  
        self.position = random.randint(0, self.pixels - 1)  
  
        self.regular_color = (0, 255, 0)  
        self.strong_color = (0, 0, 255)  
        self.current_color = self.regular_color  
  
        self.speed = 1500  
        self.direction = 1  
        self.last_moved = time.ticks_ms()
```

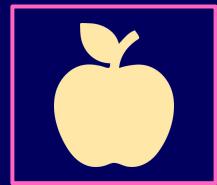
Ghost.next_position(), move() e randomize()

```
class Ghost:  
    ...  
    def next_position(self):  
        return (self.position + self.direction) % self.pixels  
  
    def move(self):  
        if time.ticks_diff(time.ticks_ms(), self.last_moved) > self.speed:  
            self.position = self.next_position()  
            self.last_moved = time.ticks_ms()  
  
    def randomize_position(self):  
        self.position = random.randint(0, self.pixels - 1)
```



Ghost.set_color() e Ghost.show()

```
class Ghost:  
    ...  
    def set_color(self, strong):  
        if strong:  
            self.current_color = self.strong_color  
        else:  
            self.current_color = self.regular_color  
  
    def show(self, np):  
        np[self.position] = self.current_color
```



Parte #3

Criando a Fruta

Projeto Final #3: Fruit

01 `_init_(self, pixels)`

Configura a posição, a cor, e o contador de tempo

02 `hide(self)`

Esconde a fruta após o Pacman ter capturado e inicia contador

03 `randomize_position()`

Calcula uma nova posição aleatória quando é capturado

04 `show()`

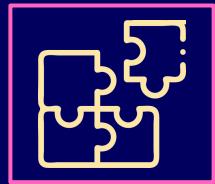
Atualiza o index com a nova posição calculada

Fruit_init_0

```
class Fruit:  
    def __init__(self, pixels):  
        self.pixels = pixels  
        self.position = random.randint(0, self.pixels - 1)  
        self.color = (255, 0, 0)  
  
    self.got_fruit = time.ticks_ms()
```

Fruit.hide(), randomize() e show()

```
class Fruit:  
    ...  
    def hide(self):  
        self.color = (0, 0, 0)  
        self.got_fruit = time.ticks_ms()  
  
    def randomize_position(self):  
        self.position = random.randint(0, self.pixels - 1)  
        self.color = (255, 0, 0)  
  
    def show(self, np):  
        np[self.position] = self.color
```



Parte #4

Criando a Jogo

Projeto Final #4: Game

01 `_init_(self, pixels)`

Configura os NeoPixels e instancia Pacman, Ghost e Fruit

02 `check_fruit_collision(self)`

Verifica se o Pacman capturou a fruta para ficar forte

03 `check_strong(self)`

Verifica se já esgotou o tempo que o Pacman fica forte

04 `check_ghost_collision(self)`

Verifica se o Pacman capturou um fantasma

05 `got_ghost()` e `end()`

Fica feliz quando o Pacman estava forte e triste quando fraco

06 `run()`

Implementa a lógica do jogo

Game.__init__()

```
class Game:  
    def __init__(self, pin, pixels):  
        self.pixels = pixels  
        self.np = neopixel.NeoPixel(pin, self.pixels)  
  
        self.pacman = Pacman(self.pixels)  
        self.ghost = Ghost(self.pixels)  
        self.fruit = Fruit(self.pixels)  
  
        self.strong_time = 5000
```

Game.check_fruit_collision()

```
class Game:  
    ...  
    def check_fruit_collision(self):  
        if self.pacman.position == self.fruit.position and not self.pacman.strong:  
            music.play(music.JUMP_UP, wait=False)  
            display.show(Image.HAPPY)  
            self.pacman.strong = True  
            self.fruit.hide()  
            self.ghost.set_color(self.pacman.strong)
```

Game.check_strong()

```
class Game:  
    ...  
    def check_strong(self):  
        if (  
            time.ticks_diff(time.ticks_ms(), self.fruit.got_fruit) > self.strong_time  
            and self.pacman.strong  
        ):  
            music.play(music.JUMP_DOWN, wait=False)  
            display.clear()  
            self.pacman.strong = False  
            self.fruit.randomize_position()  
            self.ghost.set_color(self.pacman.strong)
```

Game.check_ghost_collision()

```
class Game:  
    ...  
    def check_ghost_collision(self):  
        if self.pacman.position == self.ghost.position:  
            if self.pacman.strong:  
                self.got_ghost()  
                self.ghost.randomize_position()  
                return False  
            else:  
                self.end()  
                return True
```

Game.got_ghost() e Game.end()

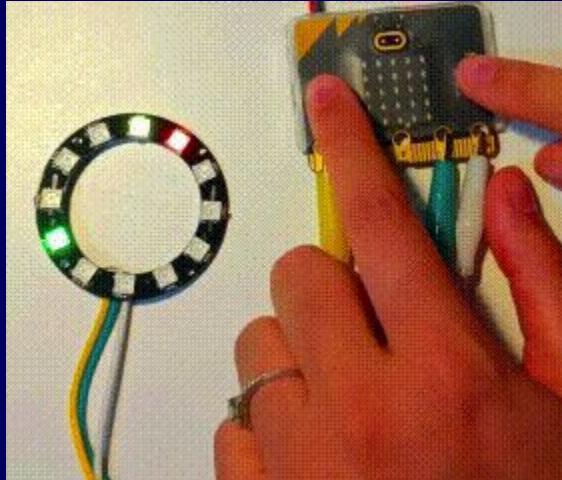
```
class Game:  
    ...  
    def got_ghost(self):  
        display.show(Image.YES)  
        music.play(music.BA_DING, wait=False)  
  
    def end(self):  
        display.show(Image.SAD)  
        music.play(music.WAWAWAWAA)
```

Game.run()

```
class Game:  
    ...  
    def run(self):  
        while True:  
            self.np.clear()  
            self.pacman.move()  
            self.ghost.move()  
  
            self.check_fruit_collision()  
            if self.check_ghost_collision():  
                break  
            self.check_strong()  
  
            self.fruit.show(self.np)  
            self.ghost.show(self.np)  
            self.pacman.show(self.np)  
            self.np.show()  
  
            sleep(100)
```

game = Game()

```
game = Game(pin=pin0, pixels=12)  
game.run()
```



Testando o jogo

Execute o código completo na placa e divirta-se!



Conclusão

Com a biblioteca **NeoPixel**, é possível usar a mesma abordagem para controlar sequências de LED independente do tamanho delas.

Ideias de projetos:

- ★ Acender luzes de escadas com sensor de presença
- ★ Acender luzes externas com sensor de luminosidade

O que você implementaria?

Obrigada!

Dúvidas?



@julianaklulo