PRINCÍPIOS DE PROGRAMAÇÃO

Nigel Ward

Esse material se destina a auxiliar os estudantes a programarem o LEGO® MINDSTORMS EV3 utilizando a linguagem baseada em texto, **python 3**, através do **ev3dev**, um sistema operacional baseado em Debian-Linux que roda em várias plataformas compatíveis com LEGO® MINDSTORMS, incluindo o LEGO® MINDSTORMS EV3 e o BrickPi com processador Raspberry Pi. Esse material foi elaborado por **Nigel Ward** e publicado no site https://sites.google.com/site/ev3python/. Foi editado e traduzido por Luiz Gustavo N. Bonfim.

Nigel Ward

Edição: Luiz Gustavo Neri Bonfim

Tradução: Luiz Gustavo Neri Bonfim

Sumário

INTRODUÇÃO 3
POR QUE O EV3 PYTHON? 5
EV3 Basic 6
Python 7
LeJos 7
ROS 7
Outras línguas 7
INSTALAÇÃO DO EV3DEV 8
EDITE E EXECUTE PROGRAMAS 10
Execute seu script Python na linha de comando 11
Execute seu script Python de dentro do Brickman 11
Resetar o bloco 11
USE UMA IDE PYTHON 11
LEDs 13
Definir a cor de um LED 14
Faça os LEDs piscarem 15
Desligue os LEDs 16
DISPLAY LCD 16
Smiley / Grumpy 17
Exibir texto com a função print () 18
Exibir texto com a função draw.text () 19
Centralizar texto horizontalmente 19
Embrulhe o texto 20
Use fontes maiores 21
Instale fontes para usar texto ainda maior 21
Exibir um arquivo de imagem 22
Como capturar uma captura de tela da tela do EV3 23
Como executar a partir do terminal um script que grava no visor LCD do EV3 25
Tela de lcd (avançado) 26
BOTÕES 29
Pressione qualquer botão para sair 29
Reagir ao botão pressionar e liberar 30
Obter uma lista de estados de botão alterados 31
Verifica os estados dos botões 32
Outras funções de botão 32
USANDO MOTORES 33
Gire o motor através de um determinado ângulo 34
Ligue o motor por um tempo determinado 35

Acione o motor 'para sempre' ... 36
Aguarde a conclusão ... 37
Parando os motores ... 38
Regulação de velocidade ... 39
Funções motoras semelhantes a EV3G ... 39
Observações ... 43
Outras notas ... 45
CONTROLE REMOTO ... 45
USANDO SENSORES ... 47
Exemplo 1 (sensor de toque e sensor IR) ... 48
Exemplo 2 (sensor de toque e sensor ultrassônico EV3) ... 49
Exemplo 3 (dois sensores de toque) ... 51
Exemplo 4 (sensor giroscópio e sensor táctil) ... 51
Exemplo 5: Sensor de cor no modo COL-REFLECT ... 53
Exemplo 6: Sensor de cor EV3 no modo COL-AMBIENT ... 53
Exemplo 7: Sensor de cor EV3 no modo COL-COLOR e sensor de toque ... 54
Exemplo 8: Sensor de cor EV3 no modo RGB-RAW e sensor de toque ... 55
Modos de Sensor ... 56
Sensores EV3 ... 57
Sensores NXT ... 58
Modo RGB do Sensor de Cor EV3 ... 58
SOM ... 59
Reproduzir um sinal sonoro padrão ... 59
Reproduzir um som único ... 59
Toque uma SEQUÊNCIA de tons ... 59
Reproduzir um arquivo de som WAV ... 60
Texto para fala ... 61
MATEMÁTICA E TEMPO ... 62
Abs() and Round() ... 62
Divisão e Módulo ... 63
Funções trigonométricas ... 63
Números aleatórios ... 63
Tempo ... 64
MULTITHREADING ... 65
CONTROLE DO TECLADO ... 73
'Get character' Exemplo de interação de teclado ... 74
Controle de teclado de um robô ... 75

INTRODUÇÃO

Desde maio de 2018, uma ótima nova maneira de escrever e executar programas em Python EV3 está disponível! O novo fluxo de trabalho tem muitas vantagens sobre todos os fluxos de trabalho anteriores. Esse novo fluxo de trabalho é baseado no editor de código livre Microsoft Visual Studio Code. O VS Code é compatível com Windows, Mac OS e Linux.

Este conteúdo pode ser acessado em **www.ev3python.com** e destina-se a ajudar os usuários do robô Lego EV3 que já estão familiarizados com os conceitos básicos de programação em Python para começar a usar a linguagem de programação EV3 Python para controlar seu robô.

O robô Lego Mindstorms EV3 é um robô educacional / entretenimento muito popular que é normalmente programado usando um sistema de programação baseado em ícones que se assemelha a Scratch. Leia mais sobre programação Lego EV3 baseada em ícones no meu site mind-storms.com. Mas o EV3 tem um computador Linux em seu coração e pode, portanto, ser programado usando vários outros sistemas de programação, como Python, C ++, Java ou EV3 Basic (veja meu site ev3basic.com). A execução dessas linguagens de programação mais avançadas permite que o robô EV3 faça coisas que não seriam possíveis com o sistema de programação EV3 padrão.

Este texto não pretende mostrar como tornar os programas muito complexos, apenas para ajudá-lo a começar a programar EV3 trabalhando com programas muito curtos, simples e fáceis de entender. Mas talvez um dia você possa fazer seus próprios programas tão impressionantes quanto de um programador profissional.

Além de capacitar você a escrever programas que podem fazer coisas que não podem ser alcançadas com o sistema padrão de programação EV3, outro grande motivo para aprender EV3 Python é que o EV3 Python é uma linguagem de programação textual, o tipo usado pelos programadores profissionais. Aprender uma linguagem de programação textual (especialmente Python) pode impulsionar enormemente suas perspectivas de carreira, enquanto aprender a programar o EV3 com o sistema baseado em ícones padrão terá muito menos valor profissionalmente.

Você provavelmente está curioso sobre como é um programa do EV3 em Python. Aqui está um exemplo que usa um sensor de cor para ajudar um veículo do tipo rover a seguir uma linha preta.

Isso é apenas para dar uma ideia de um script do EV3 Python - a explicação completa está nesta seção. Este programa inclui, no entanto, comentários (precedidos de cada vez por um sinal '#') que existem apenas para o ajudar a compreender o código.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *

btn = Button() # usará qualquer botão para parar o script

# Conecta o sensor de cor EV3.
cl = ColorSensor()

# põe o sensor de cor em modo COL-REFLECT
# para medir a intensidade da luz refletida.
cl.mode='COL-REFLECT'

# Anexa os motores grandes às portas B e C.
mB = LargeMotor('outB')
mC = LargeMotor('outC')

while not btn.any():    # sai do loop quando algum botão for
pressionado
    if cl.value()<30:   # reflexo fraco sobre a linha preta
        # curva media à direita
        mB.run_forever(speed_sp=450)
        mC.stop(stop_action='brake')
    else:   # forte reflexo (>=30) então sobre a superfície branca
        # curva media à esquerda
        mB.stop(stop_action='brake')
        mC.run_forever(speed_sp=450)

mB.stop(stop_action='brake')
mC.stop(stop_action='brake')
```

O EV3 Python precisa ser executado sobre o sistema operacional EV3dev. O site oficial do EV3dev e do EV3 Python é www.ev3dev.org e vou encaminhá-lo para esse site para obter detalhes sobre a instalação e a documentação das várias funções do EV3 em Python. Os desenvolvedores de EV3dev e EV3dev Python fizeram um trabalho fantástico, disponibilizando um sistema operacional estável e comandos Python relacionados ao robô com a mais simples sintaxe possível.

Ao contrário de alguns dos desenvolvedores do EV3dev, eu não sou um programador profissional, mas tenho décadas de experiência no ensino de informática para adolescentes, incluindo Python e EV3. Por isso, me sinto bem qualificado para ajudar pessoas que já conhecem o básico do Python e que desejam começar a usar o EV3 Python para controlar seu robô Lego EV3. Este site não é endossado pela Lego Corporation.

Eu sou um homem do Windows, então este site será orientado para os usuários do Windows, mas os usuários de outras plataformas também devem achar o site útil. Este site foi projetado para ser compatível com os dois tipos de kit de robô EV3: a edição doméstica e a edição educacional. Este site discutirá apenas o que pode ser obtido com o hardware padrão da Lego - não haverá discussão sobre hardware de terceiros. Dito isso, é bom saber que deve ser possível usar EV3dev e EV3 Python com sensores e motores de terceiros e até mesmo executar o EV3 Python em outras plataformas, além do EV3, como o Raspberry Pi com o escudo BrickPi ou o escudo PiStorms. Esses 'escudos' se conectam ao Raspberry Pi e atuam como uma interface entre os motores e sensores Raspberry Pi e Lego EV3. Isso significa que você pode usar o poder extra do Pi comparado ao EV3 e, no caso do PiStorms, você também terá uma bela tela colorida.

POR QUE O EV3 PYTHON?

O robô Lego EV3 geralmente roda um sistema operacional conhecido como EV3-G, no qual os programas são criados usando um método baseado em ícones, como o Scratch. Este modo de programação tem vantagens e desvantagens:

Vantagens:

- Conectar blocos juntos significa menos digitação, portanto, menor risco de erros ortográficos e de sintaxe (gramática)
- A natureza visual dos ícones torna a programação mais clara para falantes não ingleses do que uma linguagem de programação textual baseada no inglês.
- Algumas pessoas podem achar o layout visual dos loops etc mais claro do que um programa textual

Desvantagens:

- Programadores profissionais usam programação textual, então aprender EV3-G não é a melhor maneira de se preparar para uma carreira em programação profissional (também chamada de codificação)

- Programas feitos com o sistema de programação EV3-G baseado em ícones podem ser executados mais lentamente do que programas feitos com linguagens de programação textuais
- O sistema EV3-G disponibiliza um conjunto limitado de funções - funções adicionais estarão geralmente disponíveis em uma linguagem de programação textual

Em qualquer caso, o método padrão de programação do EV3 é usar o sistema de programação embutido, EV3-G, então certamente faz sentido dominar isso antes de pensar em passar para a programação textual. Uma vez que um bom nível de domínio do EV3-G tenha sido alcançado, faz sentido que os usuários motivados queiram usar a programação textual para programar seu robô, e há várias opções disponíveis:

EV3 Basic

Estou convencido de que a linguagem de programação textual EV3, que é a mais fácil de instalar e aprender, é o EV3 Basic. Passei centenas de horas trabalhando com o EV3 Basic e gostei tanto que criei um site para promovê-lo: ev3basic.com. Basic foi originalmente projetado para iniciantes, mas evoluiu ao longo dos anos em uma forma sofisticada chamada Microsoft Visual Basic, que é menos fácil para iniciantes aprenderem. Felizmente a Microsoft reconheceu esse problema e lançou em 2008 uma versão do Basic que é mais uma vez simples e fácil de aprender: Microsoft Small Basic. O Small Basic foi projetado para atrair os jovens, com ênfase em jogos e gráficos. Adicionando a 'extensão EV3' converte Small Basic em EV3 Basic e permite que ele trabalhe com o robô EV3. O Basic foi uma das primeiras linguagens de programação a serem criadas e algumas pessoas acham que se tornou desatualizado e que os iniciantes devem aprender uma linguagem mais moderna. Mas o Basic sempre foi muito popular e até hoje (outubro de 2016) Visual Basic.Net e sua variante mais antiga Visual Basic estão entre as linguagens de programação mais populares do mundo - veja por si mesmo na Tiobe.com que acompanha a popularidade de mais de 100 línguas. De fato, se você juntar os dois tipos principais de Basic (Visual Basic e Visual Basic.net), então podemos dizer que Basic é tão popular quanto a linguagem Python (outubro de 2016)! Outro motivo para considerar o uso do EV3 Basic é que é mais fácil de instalar do que outras linguagens de programação textual do EV3, pois ele trabalha diretamente com o bloco EV3 não modificado, enquanto outras linguagens textuais exigem que você modifique o firmware do brick ou prepare um cartão microSD que contenha sistema operacional alternativo que será usado no lugar do firmware do tijolo. Observe que o EV3 Basic é executado apenas em computadores com Windows.

Python

O Python é uma linguagem de programação moderna e muito poderosa que é concisa, legível e, portanto, mais fácil de aprender do que a maioria das outras, como C ++ ou Java, sendo, portanto, muito popular em escolas e universidades. Uma coisa que ajuda a tornar mais fácil o aprendizado é que um recurso avançado e bastante difícil chamado 'programação orientada a objetos' (OOP) está disponível, mas não é obrigatório, então você pode evitá-lo inicialmente e começar a usá-lo quando estiver pronto.

LeJos

O LeJos permite que você use a linguagem de programação Java para programar o EV3. Atualmente, o Java é a linguagem de programação mais popular do mundo, mas é muito mais difícil de aprender do que o Basic ou o Python, portanto, é provavelmente uma boa escolha apenas para os alunos mais motivados. Alunos menos motivados podem ser deixados de lado a programação por Java! Dito isto, algumas pessoas sentem que, embora seja mais fácil escrever programas simples em Python, é mais fácil escrever programas complexos em Java.

ROS

ROS (Robot Operating System) é projetado especificamente para robôs. No entanto, é significativamente mais difícil de aprender do que o EV3 Python ou o EV3 Basic.

Outras línguas

Outras linguagens de programação textuais podem ser usadas para programar o EV3, como o C ++ (rodando no sistema operacional EV3dev), mas o C ++ também é muito mais difícil de aprender do que o Python.

Conclusão: se você está procurando a maneira mais fácil possível de tentar programar o EV3 textualmente e não pretende seguir carreira em codificação, então eu o encorajo a visitar o EV3Basic.com e considerar a possibilidade de tentar o EV3 Basic. Se você quiser usar uma linguagem de programação mais moderna, talvez com a intenção de se tornar um programador profissional um dia, então eu recomendo que você use o EV3 Python. Se esta é realmente a sua escolha, então o próximo passo é instalar o EV3Dev e o EV3 Python.

Existem centenas de linguagens de programação por aí e o vídeo a seguir coloca o Python em quarto lugar "por empregos, por dinheiro, por construir coisas novas". O autor, um desenvolvedor de software, coloca Java, Java Script e C # (C Sharp) à frente do Python. Mas eu li que Java é tão

detalhado que um programa Java pode ser 3 ou 4 vezes mais longo que um programa em Python que faz a mesma coisa!

INSTALAÇÃO DO EV3DEV

O Bloco EV3 permite que um sistema operacional alternativo seja usado, ignorando o sistema interno, colocando o sistema operacional alternativo em um cartão micro SD e inserindo-o no slot ao lado do bloco EV3 (de agora em diante eu vou me referir ao cartão micro SD como um 'cartão SD' por brevidade). O EV3 Python precisa ser executado no sistema operacional EV3dev (uma versão modificada do sistema operacional Linux Debian Jessie), portanto, precisamos preparar um cartão SD que contenha EV3dev e EV3 Python (a instalação do EV3dev inclui automaticamente o EV3 Python). O EV3 não é compatível com cartões SD com capacidade superior a 32 GB, portanto, verifique se você possui um cartão SD com capacidade entre 2 e 32 GB. Esteja ciente de que quaisquer arquivos existentes no cartão serão excluídos durante o procedimento de instalação. As instruções oficiais para instalar o EV3dev estão na página de 'introdução' do site oficial EV3dev.org e você precisará seguir as instruções nessa página com muito cuidado. Aqui está uma versão esquelética das instruções, não suficiente para você poder seguir os passos (veja a página de introdução para isso), mas suficiente para dar a essência do processo antes de trabalhar com as instruções oficiais. Recordo-lhe mais uma vez que este site destina-se principalmente a utilizadores do Windows e que outros utilizadores irão considerá-lo menos relevante.

1. Baixe um arquivo de imagem (no formato .zip) que será copiado para o cartão SD
2. Faça o download de um programa especial chamado Etcher para copiar a imagem para o cartão SD (você não pode simplesmente copiar o arquivo de imagem para o cartão da maneira usual).
3. Copie a imagem para o cartão SD usando Etcher. Para mim, Etcher às vezes relatou erros quando o processo é concluído, mas na verdade o processo sempre foi satisfatoriamente concluído e produziu uma carteira de trabalho. Portanto, no caso improvável de você receber uma mensagem de erro, sugiro que você experimente o cartão de qualquer maneira - provavelmente funcionará bem.
4. Conecte o cartão no bloco EV3 desligado, ligue o bloco e espere-o terminar a inicialização
5. Conecte o EV3 ao seu computador com um cabo USB. É melhor usar essa conexão simples. Posteriormente, você poderá usar um dongle Ethernet ou Bluetooth ou WiFi para se conectar ao bloco. O computador fará uma conexão ao bloco.
6. Agora você precisa mudar para a página 'Conectando à Internet via USB'.
7. Nomeie a conexão do computador para o bloco 'EV3'

8. Possibilite que o brick compartilhe a conexão do computador com a internet (o EV3 precisa de acesso à internet para que certos arquivos possam ser baixados da internet para o brick).
9. No EV3, configure um endereço IP estático para tornar a conexão à Internet compartilhada um pouco mais confiável.
10. Conecte o EV3 à internet através do computador
11. Agora precisamos editar nossos programas em Python no computador, baixá-los no bloco e executá-los no bloco. Também gostaríamos de poder carregar programas em Python do bloco para o computador. Para usuários do Windows, o site ev3dev.org recomenda o uso de um programa chamado PuTTY para lidar com a cópia de arquivos entre o computador e o bloco e o lançamento dos programas no bloco. Ele também nos permitirá fornecer outros comandos ao bloco, como baixar e instalar atualizações no sistema operacional ev3dev. Eu prefiro usar o programa gratuito MobaXTerm Home Edition em vez de PuTTY, pois ele pode fazer muito mais. Por exemplo, o MobaXTerm tem um editor de texto embutido que você pode usar para editar seus programas em Python, embora deva ser dito que é melhor usar um 'Ambiente de Desenvolvimento Integrado' (IDE) para isso já que um IDE pode executar o autocompletar e dar ajuda na depuração.
12. Baixe PuTTY ou MobaXTerm para o seu computador e inicie-o.
13. em PuTTY ou MobaXTerm, crie uma conexão 'secure shell' (SSH) entre o computador e o bloco. Uma 'janela de terminal' será aberta. Esta é uma interface de linha de comando onde você digita comandos um por um e o bloco (não o computador) responde aos comandos. Por exemplo, você pode digitar 'ls' para obter uma listagem na janela do terminal do conteúdo da pasta atual no bloco. Observe que os comandos fazem distinção entre maiúsculas e minúsculas, por exemplo, você não pode obter a listagem digitando 'Ls' em vez de 'ls'.
14. Faça o login com o nome de usuário 'robot' e a senha 'maker'.
15. Existem duas versões do Python, o Python v2, que agora é obsoleto, e o Python v3, a versão que você usará. Você vai querer usar o Python v3, que está pronto para ser executado no bloco. Pode ser iniciado com o comando Python3.
16. Com o Python3 em execução, você pode inserir comandos Python na linha de comando do PuTTY ou do MobaXTerm, como print ('Hello, world'), o que fará com que 'Hello, World' seja gerado na janela do terminal.

EDITE E EXECUTE PROGRAMAS

Seguindo as instruções dos vídeos na página de instalação, você estabeleceu uma conexão de rede, mas ainda não começou a usar o SSH (Secure SHell) para gerenciar a comunicação entre o EV3 e o computador. O texto abaixo mostra como usar o programa de SSH, MobaXTerm.

O fluxo de trabalho que eu recomendo é primeiro usar seu programa SSH (PuTTY ou MobaXTerm, por exemplo) para criar um arquivo de programa Python vazio no bloco, abrir e editar isso no computador e salvar o arquivo modificado de volta no bloco. Esta é, na minha opinião, a melhor maneira de trabalhar porque evita a confusão de ter duas cópias do mesmo arquivo, uma no bloco e outra no computador, e talvez não ter certeza de qual cópia você está trabalhando ou se as duas cópias estão em sincronia.

Você precisa criar um novo arquivo Python no bloco usando o MobaXTerm: basta clicar no botão 'Novo arquivo', nomear o novo arquivo, certificando-se de finalizar o nome do arquivo com .py e clicar em OK. Também é possível usar a linha de comando do seu programa SSH para criar um novo arquivo Python no bloco usando o caractere '>'. Por exemplo, para criar um arquivo Python vazio com o nome myscript.py, digite> myscript.py.

```
robot@ev3dev:~$ > myscript.py
robot@ev3dev:~$
```

Observe que você retorna ao prompt sem indicação de que um arquivo foi criado. Você pode verificar se o arquivo existe usando o comando "ls", se desejar. Alternativamente, se você estiver usando o MobaXTerm, clique no botão 'Atualizar pasta' para ver seu novo arquivo no painel que mostra a lista de arquivos na pasta atual. Aviso: este método irá sobrescrever um arquivo existente com esse nome sem avisar você!

Outro comando que funciona de maneira semelhante ao uso da seta para a direita (>) para criar um arquivo de texto vazio é o comando “touch”. Digite “toque” seguido por um nome de arquivo. Por exemplo, digite toque em myscript.py para criar um arquivo Python vazio com o nome myscript.py. Isso NÃO irá sobrescrever um arquivo existente com esse nome e é, portanto, o método preferido.

Você gostaria que fosse possível executar seu programa a partir da linha de comando do programa SSH ou da interface Brickman no bloco. Para tornar possível executar o script Python dentro do Brickman, você precisa fazer duas coisas:

- Você precisa tornar o script executável, que por padrão não é. Para fazer isso, execute o comando `chmod +x my-file.py`, usando o nome real do seu arquivo.
- Adicione a linha `#!/Usr/bin/env python3` ao topo do seu programa. Certifique-se que é a primeira linha! Você notará que na maioria dos scripts de amostra no meu site esta linha está incluída, pois sem ela o script não pode ser executado a partir do Brickman. By the way, uma linha começando #! não é apenas um comentário que o computador irá ignorar, é chamado de 'shebang' e realmente faz alguma coisa! Neste caso, ele diz ao interpretador para abrir o script usando o Python3.

Depois de ter feito o script do Python executável, um asterisco (*) será exibido após o nome do arquivo no navegador de arquivos do Brickman, para que você possa ver quais arquivos .py são executáveis e quais não são.

Execute seu script Python na linha de comando

`python3 <filename>` é usado para iniciar um arquivo Python na pasta atual dentro do Python3. Por exemplo, para iniciar um arquivo em Python chamado `drive.py`, use `python3 drive.py`. A maioria dos programas terminará sozinhos, mas alguns continuarão a ser executados até que você os force a parar, digitando Ctrl + C na linha de comando.

Execute seu script Python de dentro do Brickman

Para executar um programa a partir do Brickman, basta abrir o gerenciador de arquivos no menu principal, selecionar seu `arquivo.py` e pressionar o botão enter (meio). Isso iniciará seu arquivo Python, desde que ele inclua o 'shebang' correto e que o arquivo tenha se tornado executável, conforme explicado mais acima nessa página, caso contrário, você receberá uma mensagem de erro. A maioria dos programas terminará sozinhos, mas alguns continuarão a ser executados até que você os force a parar. Se o programa foi iniciado dentro do Brickman, você pode pressionar o botão 'voltar' para encerrar qualquer programa.

Resetar o bloco

Se você precisar resetar o bloco, use o atalho 'back' + 'enter'. Ou digite `sudo reboot` na linha de comando.

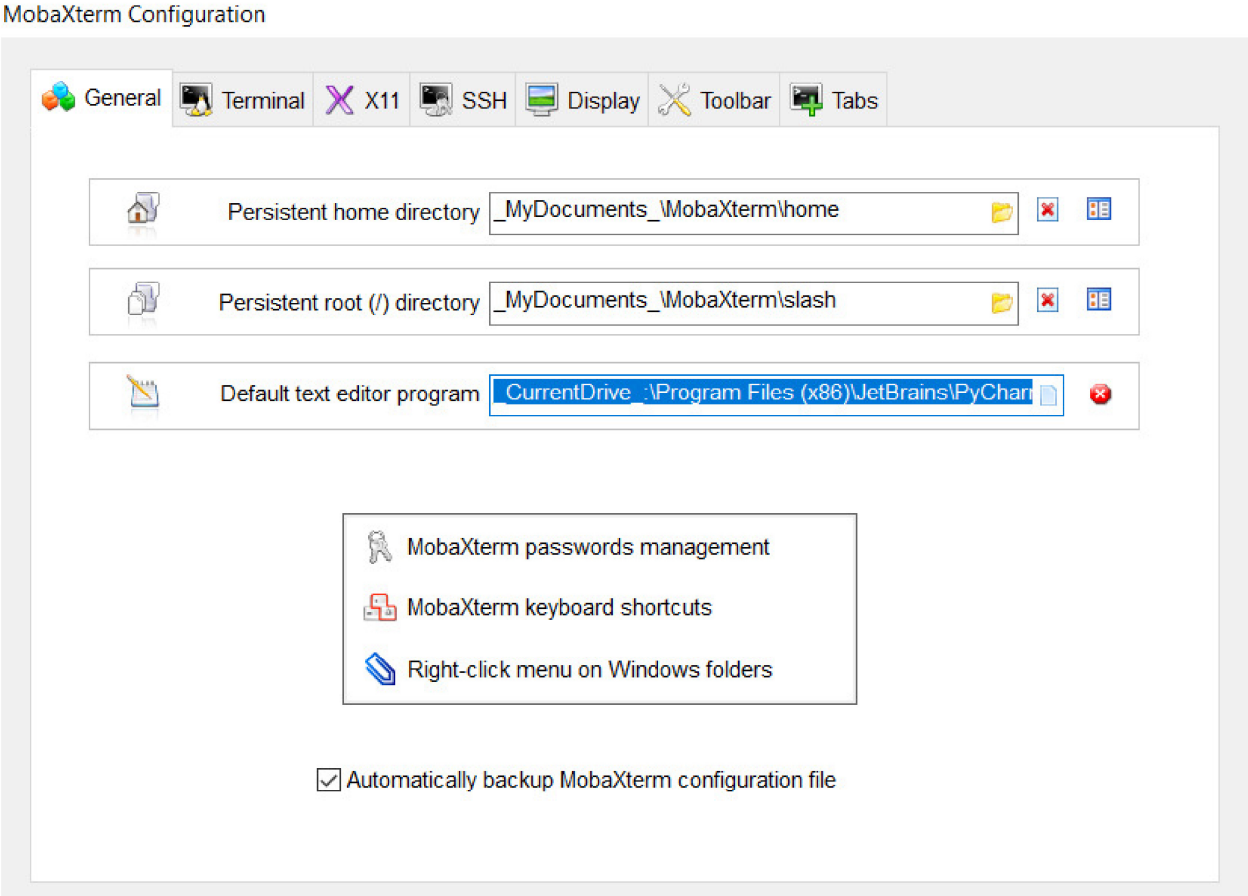
USE UMA IDE PYTHON

Na página "Editar e executar programas", mostrei como criar um arquivo Python e editá-lo no editor de texto interno do MobaXTerm personal Edition. O MobaXTerm é apenas para Windows, mas acredito que existam programas equivalentes para Mac e Linux. Um editor de texto PODE

ser usado para criar programas em Python, mas é melhor usar um ambiente de desenvolvimento integrado (IDE) porque uma IDE oferece recursos extras, como depuração e autocomplete.

De acordo com minha pesquisa, o melhor IDE para iniciantes do Python é provavelmente o PyCharm Edu. Este programa está disponível para macOS, Windows e Linux. Como o nome sugere, esta IDE é projetada para alunos - inclui a possibilidade de trabalhar com lições de Python com exercícios do tipo preencher o espaço em branco e, de fato, um bom curso introdutório está incluído no pacote. Além disso, o PyCharm Edu oferece um caminho de atualização fácil para a versão paga do PyCharm, que é muito popular entre os programadores profissionais de Python.

É muito fácil configurar a edição pessoal do MobaXTerm para que, ao clicar duas vezes em um arquivo Python, ele seja aberto automaticamente no PyCharm Edu. Salvando o arquivo irá salvar automaticamente o arquivo de volta para o bloco. Basta escolher Configurações> Configuração> Geral para obter o seguinte diálogo:



Em seguida, clique no pequeno quadrado azul à direita da caixa 'programa de edição de texto padrão' e navegue até o arquivo pycharm.exe em seu computador. É simples assim! Agora, toda vez que você clicar duas vezes em um arquivo na lista de diretórios do MobaXTerm, ele será aberto no PyCharm Edu e quando você salvar o arquivo, ele será salvo diretamente no bloco EV3.

É importante NÃO usar o MobaXTerm para criar novos arquivos Python para o bloco, pois seria difícil transformá-los diretamente no bloco ou movê-los para o bloco posteriormente. Se você quiser criar um novo arquivo Python no EV3, simplesmente clique no ícone 'Novo arquivo' no topo do painel de listagem de diretórios do MobaXTerm e dê ao novo arquivo um nome que termine com .py, como `myfile.py`. Então você pode abrir o novo arquivo clicando duas vezes e começar a editá-lo no PyCharm Edu!

LEDs

ATUALIZAÇÃO EM SETEMBRO DE 2017: Embora o controle dos LEDs (no plural, já que o EV3 Python controla os LEDs esquerdo e direito separadamente) seja simples e direto, na verdade experimentei uma frustração significativa com isso. Minhas experiências me levaram a pensar que erros podem ocorrer se uma operação de LEDs não tiver tempo para ser concluída antes que outra seja tentada, e também tenho a impressão de que tentar fazer com que os LEDs pisquem frequentemente resultados inesperados, especialmente para as cores laranja âmbar e amarelo. Estas são cores 'compostas' feitas misturando as cores vermelha e verde que são as cores dos LEDs reais. Se você tentar piscar em vermelho e verde, provavelmente terá mais sucesso. Se você tiver problemas, então meu conselho é incluir sleep (0.01) após cada comando de LED e evitar tentar fazer com que os LEDs pisquem. Aqui está um exemplo de código que segue meu conselho - ele define os dois LEDs para laranja:

```
from ev3dev.ev3 import*
from time import sleep

Leds.set_color(Leds.LEFT, Leds.ORANGE)
sleep(0,01)
Leds.set_color(Leds.RIGHT, Leds.ORANGE)
sleep(0,01)
```

No entanto, como não tenho certeza de que outras pessoas estejam passando por esses problemas, assumirei que esse não é o caso do restante desta página.
Controlar os dois LEDs localizados sob os botões do bloco EV3 deve ser uma das coisas mais simples que podemos fazer no EV3 Python, então vamos começar com isso.

Eu acho que tudo que você realmente precisa saber como iniciante é como definir a cor de cada LED e como fazer os LEDs piscarem. Para outras funções, como definir o brilho de cada LED, consulte a documentação oficial.

Definir a cor de um LED

Use `Leds.set_color (grupo, cor)`
Grupo pode ser `Leds.LEFT` ou `Leds.RIGHT`.
A cor pode ser Leds.RED, Leds.GREEN, Leds.YELLOW, Leds.ORANGE ou Leds.AMBER.

Exemplo

Este exemplo transforma o led vermelho esquerdo quando o botão do sensor de toque é pressionado. Quando você executar este ou qualquer outro programa, esteja ciente de que o programa levará alguns segundos para iniciar. Pressione Ctrl-C para interromper o loop infinito.

```
#!/usr/bin/env python3
# Linha acima é necessária para que o programa possa ser executado a
partir do Brickman

# Conecte um sensor de toque em qualquer porta de sensor
from ev3dev.ev3 import*
ts = TouchSensor ()

while True:
    if ts.value () == 1: sensor #touch pressionado
        Leds.set_color (Leds.LEFT, Leds.RED)
    else:
        Leds.set_color (Leds.LEFT, Leds.GREEN)

# Pressione Ctrl-C para sair
```

Aqui está o mesmo exemplo de uma forma mais clara, porém mais obscura. Destacada é uma tupla com dois elementos. Se ts.value () for zero, o elemento zero da tupla (verde) será usado, etc.

```
#!/usr/bin/env python3
# Então o programa pode ser executado a partir do Brickman

# Conecte um sensor de toque em qualquer porta de sensor
```

```
from ev3dev.ev3 import*
ts = TouchSensor ()

while True:
    Leds.set_color (Leds.LEFT, (Leds.GREEN, Leds.RED) [ts.value ()])

# Pressione Ctrl-C para sair
```

Aqui está um exemplo semelhante que irá alternar os dois LEDs ao mesmo tempo:
Chegando...

Faça os LEDs piscarem

Aqui está uma demonstração de LED piscando. Aviso: atualmente, isso não funciona corretamente para mim se eu escolher cores diferentes de RED ou GREEN, que são as cores verdadeiras dos LEDs. As outras cores, LARANJA, AMARELA e AMARELA, são feitas usando os LEDs VERMELHO e VERDE juntos, e para mim isso às vezes faz com que a cor errada apareça ou que mensagens de erro sejam acionadas.

```
#!/usr/bin/env python3
from ev3dev.ev3 import*
Leds.set (Leds.LEFT, brightness_pct = 0,5, trigger = 'temporizador')
Leds.set (Leds.LEFT, delay_on = 3000, delay_off = 500)
Leds.set_color (Leds.LEFT, Leds.GREEN)
```

Este exemplo usa o método “Leds.set” para definir algumas propriedades no grupo de LEDs à esquerda (“LEFT”). Ele define o brilho para 50% e, em seguida, define o acionador como "timer", o que significa que ele será "acionado" em um timer definido. “`Delay_off`” e “`delay_on`” ditam a quantidade de tempo (em milissegundos) em que o LED está desligado ou ligado, respectivamente. Com os números dados aqui, ele permanecerá desligado por 3 segundos e depois ligará por meio segundo, depois repetirá. O trigger deve ser ajustado para 'timer' antes de `Delay_off` e `delay_on` serem definidos, e na prática isso significa que eles devem ser definidos dentro de comandos separados. É possível que quando os LEDs esquerdo e direito pisquem, eles podem não estar sincronizados.

Observe que as configurações manterão seus valores depois que o script parar de ser executado, portanto, os LEDs continuarão piscando até que algum outro script os interrompa.

Desligue os LEDs
Desligue os dois LEDs com `Leds.all_off()`

DISPLAY LCD

Trabalhar com a tela LCD é bastante complicado e, em qualquer caso, a tela do EV3 é pequena e difícil de ver (sem luz de fundo), então eu sugiro que trabalhar com a tela deve ser uma prioridade baixa para você aprender o EV3 Python.

O EV3 tem um LCD monocromático (preto e branco) de 178 x 128 pixels. As coordenadas do pixel superior esquerdo são (0, 0) e as coordenadas do pixel inferior direito são (177, 127).

Para interagir com o LCD, você usará principalmente uma biblioteca gráfica poderosa chamada Pillow, que faz parte do Python padrão, em vez de algo exclusivo do EV3 Python. Observe que o Pillow evoluiu da Python Image Library (PIL). Há alguma documentação na página de ligações do EV3 Python, mas você encontrará documentação muito mais detalhada na documentação oficial do Pillow em https://pillow.readthedocs.io/en/3.3.x/, especialmente a documentação no módulo ImageDraw. O módulo ImageDraw fornece os seguintes métodos:

- `arc(xy, start, end, fill=None)`
- `bitmap(xy, bitmap, fill=None)`
- `chord(xy, start, end, fill=None, outline=None)`
- `ellipse(xy, fill=None, outline=None)`
- `line(xy, fill=None, width=0)`
- `pieslice(xy, start, end, fill=None, outline=None)`
- `point(xy, fill=None)`
- `polygon(xy, fill=None, outline=None)`
- `rectangle(xy, fill=None, outline=None)`
- `text(xy, text, fill=None, font=None, anchor=None, spacing=0, align="left")`
- ~~`multiline_text`~~`(xy, text, fill=None, font=None, anchor=None, spacing=0, align="left")`
- `textsize(text, font=None, spacing=0)`
- `multiline_textsize(text, font=None, spacing=0)`

Aviso: Os scripts do EV3 Python que usam a tela LCD podem ser executados com facilidade a partir da interface do Brickman ou usando o VS Code com a extensão EV3, mas se você quiser executar esses scripts de dentro de uma sessão SSH (PuTTY ou MobaXTerm, por exemplo), deve seguir as instruções na parte inferior desta página.

Smiley / Grumpy

Este script desenha dois círculos para os olhos, em seguida, um arco (neste caso, meia elipse), que alterna entre abrir para cima (um sorriso) e abrir para baixo (rabugento). Aviso: NÃO execute este script de dentro do Brickman porque ele é executado para sempre e, portanto, é necessário pará-lo com um toque longo no botão Voltar, mas isso NÃO interrompe o programa adequadamente, presumivelmente devido a algum tipo de erro no Brickman. Você pode executar este programa com êxito a partir do SSH, parando-o com Ctrl-C, mas não se esqueça de ler a nota no final desta página.

```
#!/usr/bin/env python3
from time import sleep
from ev3dev.ev3 import *
lcd = Screen()
smile = True
while True:
    lcd.clear()

    # lcd.draw retorna um identificador PIL.ImageDraw
    lcd.draw.ellipse(( 20, 20,  60, 60))
    lcd.draw.ellipse((118, 20, 158, 60))

    if smile:
        lcd.draw.arc((20, 80, 158, 100), 0, 180)
    else:
        lcd.draw.arc((20, 80, 158, 100), 180, 360)

    smile = not smile  # toggle between True and False

    # Atualiza o display lcd
    lcd.update() # Aplica alterações pendentes na tela.
    # Nada será desenhado na tela do lcd
    # até que esta função seja chamada.

    sleep(1)
```

Observe que o comando `screen.draw.ellipse()` possui parâmetros que são o canto superior esquerdo e o canto inferior direito de uma caixa delimitadora (uma caixa imaginária que conteria a elipse). O comando `screen.draw.arc()` precisa do mesmo par de coordenadas e também

precisa de um ângulo inicial e um ângulo final, ambos em graus. Os ângulos são medidos a partir das 3 horas, aumentando no sentido horário.

Exibir texto com a função print ()

A função `print()` é mais simples de usar do que a função `draw.text()`, uma vez que ela simplesmente funciona - ela não requer que você crie um objeto Screen e não requer que você atualize a tela para ver o texto. Você não precisa importar o módulo ev3dev.ev3 para usar print (). Ele envolve automaticamente o texto na borda direita da tela e rola automaticamente o texto conforme necessário. Você pode alterar a fonte como no exemplo a seguir. Quando você definir uma fonte, essa configuração será lembrada até que você desligue o bloco. Se você não definir a fonte como essa, o script usará a fonte usada na última vez em que foi definida.

O script abaixo mostra como definir a fonte desejada. Você pode obter uma lista de fontes disponíveis executando `ls/usr/share/consolefonts` no terminal SSH. Observe que as fontes usadas pela função `print()` e pelo procedimento para especificar uma, são diferentes das fontes usadas pela função `draw.text()` e pelo procedimento para especificar uma. O script abaixo também mostra como você pode impedir que a função `print()` inicie uma nova linha após a impressão.

```
#!/usr/bin/env python3
# from ev3dev.ev3 import *   # não necessário
from time import sleep
import os
os.system('setfont Lat15-TerminusBold14')
# os.system('setfont Lat15-TerminusBold32x16')  # Tente essa fonte
menor
# Uma lista complete de fonts pode ser encontrada em
#ls/usr/share/consolefonts`

print('EV3 Python rules!')
print()  # imprime uma linha em branco
print('EV3', 'Python rules!')  # vírgula significa continuar na mesma
linha

# Por padrão, o parâmetro invisível 'end' é o novo caractere de linha
print('EV3')    # uma nova linha será começada depois disso
print('Python rules!')
```

```
# Aqui o parâmetro 'end' é substituído
# com uma string vazia, então nenhuma nova linha é iniciada
print('EV3', end='')
print('Python rules!')

# Aqui o parâmetro 'end' é substituído
# com um caractere de espaço para que nenhuma nova linha seja
iniciadaprint('EV3', end=' ')
print('Python rules!')

sleep(15)  # exibir o texto por tempo suficiente para que seja visto
```

Exibir texto com a função draw.text ()

Exibir texto com a função `draw.text()` é mais difícil, mas oferece mais controle, pois você pode especificar onde o texto deve ser colocado e se o texto deve ser preto ou branco.

Este script exibirá "Hello, World" como texto branco pequeno em um retângulo preto com "ESTE TEXTO ESTÁ PRETO" exibido em preto na metade inferior da tela. As coordenadas no parâmetro da função `text()` referem-se ao canto superior esquerdo do texto.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep

lcd = Screen()
lcd.draw.rectangle((0,0,177,40), fill='black')
lcd.draw.text((48,13),'Hello, world.', fill='white')
lcd.draw.text((36,80),'THIS TEXT IS BLACK')
lcd.update()
sleep(6)
```

Centralizar texto horizontalmente

A função `textsize()` não é usada para definir o tamanho do texto, ela é usada para obter o tamanho (comprimento e altura) de uma string de texto em pixels. Por exemplo, pode retornar a tupla (111, 15). O primeiro elemento da tupla (elemento zero) pode ser usado para centralizar o texto horizontalmente, conforme mostrado abaixo. Para centralizar o texto verticalmente, basta

subtrair a metade da altura do texto de 64. O centro da tela pode ser considerado como (89, 64). Esse script também imprime o valor da tupla textsize.

```
#!/usr/bin/env python3
# Alinha o texto ao centro horizontalmente.
from ev3dev.ev3 import *
from time import sleep

lcd = Screen()
mystring = 'EV3 Python rules!'

lcd.clear()
size = lcd.draw.textsize(mystring) # returns a tuple
lcd.draw.text((20, 20), 'textsize = '+str(size))
# altura do ecrã = 128 pixels, então o centro vertical é
# em 64 pixels, então posicione o texto (height 11 pixels) a
# 5 pixels acima do centro, a 59
lcd.draw.text((89-size[0]/2, 59), mystring)
lcd.update()
sleep(10)
```

Embrulhe o texto

A função `print()` envolve o texto automaticamente, mas a função `draw.text()` não. O script a seguir envolve o texto em uma nova linha a cada 30 caracteres. Isso não é adequado 'quebra de linha', uma vez que não tenta detectar os espaços entre as palavras.

```
#!/usr/bin/env python3
# envolve o texto em uma nova linha a cada 30 caracteres
from ev3dev.ev3 import *
from time import sleep

lcd = Screen()

mystring = 'This is a very long string! '*7
newstring=''
for i in range(len(mystring)):
    # len (mystring) dá o número de caracteres
    newstring = newstring+mystring[i]
    if i%30 == 29:  # modulo, ou restante após a divisão
```

```
        newstring = newstring+'\n'
        # \n é o caracter nova linha

lcd.clear()
lcd.draw.text((0,0), newstring)
lcd.update()
sleep(10)
```

Use fontes maiores

O comando `draw.text()` usado acima sem nenhum argumento de 'fonte' desenha um texto muito pequeno. A especificação de um argumento "fonte" permite escolher entre uma ampla variedade de fontes e tamanhos que você pode visualizar em http://python-ev3dev.readthedocs.io/en/latest/other.html#ev3dev.fonts.load. As letras maiúsculas nos nomes das fontes têm os seguintes significados: B = Negrito, I = Itálico, O = Oblíquo (semelhante ao itálico), S = ?, R =? Como exemplo, usarei a fonte luBS24, que é, portanto, em negrito, tamanho 24. Como de costume, antes de usar o SSH para executar este script ou qualquer outro script que desenhe na tela LCD, siga as instruções na parte inferior desta página. Observe que a linha 'from PIL import ...' não é realmente necessária, pois a função `draw.text()` é importada com ev3dev.ev3.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
import ev3dev.fonts as fonts
from PIL import Image, ImageDraw, ImageFont  # this line is optional

lcd = Screen()
lcd.draw.text((10,10), 'Hello World!', font=fonts.load('luBS24'))
lcd.update()
sleep(8)
```

Instale fontes para usar texto ainda maior

Para usar fontes e tamanhos de fonte diferentes dos incluídos por padrão na versão 0.8.0 e posteriores, você pode instalar fontes adicionais. Denis D, um dos principais colaboradores do projeto ev3dev, sugeriu este procedimento para instalar e usar fontes adicionais:

1. Instale o pacote de fontes principais do MS:

```
sudo apt-get install ttf-mscorefonts-installer
```

2. Execute isto para ver quais fontes foram instaladas:

```
ls /usr/share/fonts/truetype/msttcorefonts/
```

Aqui está um exemplo de script usando uma das fontes recém-disponíveis, uma fonte TrueType (escalável) chamada Arial no tamanho 75 (a.k.a. huge):

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from PIL import Image, ImageDraw, ImageFont
from time import sleep

lcd = Screen()

f=ImageFont.truetype('/usr/share/fonts/truetype/msttcorefonts/Arial.ttf', 75)
lcd.draw.text((3,0), 'Hello', font=f)
lcd.draw.text((2,55), 'world', font=f)
lcd.update()
sleep(7)  # if run from Brickman, need time to see displayed image
```

Aqui está o resultado:

Exibir um arquivo de imagem

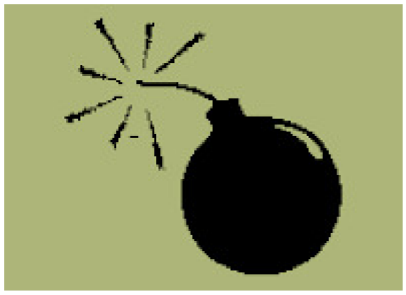
Antes de poder exibir um arquivo de imagem, você precisa ter um arquivo de imagem para exibir! A maioria dos formatos de imagem populares deve funcionar (jpg, bmp, gif, png etc). Eu sugiro que você comece fazendo o que eu fiz: copie todos os arquivos de imagem padrão do Lego EV3 BMP que estão incluídos com o software gratuito Lego EV3 em um diretório chamado 'pics' dentro do seu diretório 'robot'. Você pode ver os arquivos BMP listados nesta página e na parte inferior dessa página há um link para um arquivo zip (328KB) contendo os 107 arquivos BMP monocromáticos, cada um perfeitamente dimensionado para caber na tela EV3 (178 * 128 pixels). Se preferir, você pode encontrar os arquivos em seu PC em um local como este: C: \ Arquivos de Programas (x86) \ LEGO Software \ LEGO MINDSTORMS EV3 Home Edition \ Recursos \ BrickResources \ Retail \ Imagens \ arquivos. A pasta 'Arquivos de programas (x86)' também pode ser chamada simplesmente de 'Arquivos de programas'. Se você está recebendo os arquivos do seu PC, eu sugiro que você copie todos os arquivos BMP (não os arquivos RGF) em uma única pasta.

Descompacte o arquivo zip, crie um diretório dentro do diretório do robô, nomeie-o como 'pics' e carregue os arquivos BMP descompactados no diretório pics. Isso é muito fácil de fazer se você estiver usando o MobaXTerm como eu recomendo. Depois de fazer tudo isso, você deve estar pronto para executar o seguinte script, substituindo 'Bomb.bmp' pelo nome do arquivo de imagem de sua escolha.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
from PIL import Image

lcd = Screen()
logo = Image.open('pics/Bomb.bmp')
lcd.image.paste(logo, (0,0))
lcd.update()
sleep(5) # quando correr de Brickman, precisa de tempo para admirar a
imagem
```

Aqui está o resultado:



O script é executado facilmente a partir do Brickman (não se esqueça de tornar o arquivo do programa executável). Se você deseja executar seu script via SSH, não deixe de ler o parágrafo na parte inferior desta página. Note que a linha `from PIL import Image` não é realmente necessária, pois o módulo Image já é importado por `from ev3dev.ev3 import *`. No entanto, ainda é uma boa ideia incluir a imagem de importação PIL, pois isso torna mais óbvio que o módulo Image está sendo usado.

Como capturar uma captura de tela da tela do EV3

1. Se você estiver usando o código VS com a extensão EV3

Se você estiver usando o VS Code com a extensão EV3, fazer uma captura de tela é tão fácil quanto clicar com o botão direito do mouse no botão verde e escolher "Capturar captura de tela". Não esqueça de salvar a imagem.

2. Se você não estiver usando o código VS com a extensão EV3

Se você não estiver usando o código VS, mas em vez disso, estiver usando o SSH, a maneira mais simples de fazer uma captura de tela é executá-lo na linha de comando:

```
fbgrab "screenshot.png"
```

Isso salvará uma imagem png em preto e branco chamada 'screenshot.png' na pasta atual. Se você estiver usando o MobaXTerm, você pode clicar no botão 'Baixar arquivos selecionados' para baixar o arquivo no local de sua escolha no PC.

WasabiFan, um dos principais colaboradores do projeto ev3dev, me enviou este script mais avançado abaixo para tirar uma captura de tela da tela do EV3. O script cria um arquivo de imagem 'screenshot.png' no diretório a partir do qual o script é executado. A imagem é (opcionalmente) dimensionada para dobrar seu tamanho original e receber um fundo verde claro que se assemelha ao plano de fundo real da tela LCD EV3. Este script foi usado para criar a captura de tela "Hello World" que você viu mais acima nesta página. Você desejará executar este script a partir do SSH e não do Brickman, portanto veja o parágrafo na parte inferior desta página.

```
#!/usr/bin/env python3

# Import system utilities
import sys
from subprocess import call
# Import imaging library to resize and recolor the screenshot
from PIL import Image

# Choose the name for the screenshot to save.
# It defaults to "screenshot.png",
# but you can supply a custom name as a command-line argument.
out_name = sys.argv[1] if len(sys.argv) > 1 else "screenshot.png"

# Call the fbgrab utility to have it save a screenshot
call(["fbgrab", out_name]);

# Load the screenshot that fbgrab saved so that we can modify it
image = Image.open(out_name)
# Convert the black-and-white screenshot (where
# each pixel is just "black" or "white")
# to an image with red, green and blue channels.
# This lets us make it more colorful later.
image = image.convert("RGB")
```

```
# Resize the image to be twice the dimensions,
# making sure to preserve crisp edges
image = image.resize(tuple(i * 2 for i in image.size), Image.NEAREST)

# Get access to the underlying pixel data so that we can modify it
pixel_data = image.load()

# Loop through each pixel
for y in range(image.size[1]):
    for x in range(image.size[0]):
        # If the pixel is white, make it #adb578 (the color we
        # use as an approximation of the LCD screen) instead
        if pixel_data[x, y] == (255, 255, 255):
            pixel_data[x, y] = (173, 181, 120)
# Save the image again
image.save(out_name);
```

A captura de tela real é feita pelo método fbgrab em negrito. A linha destacada em amarelo dimensiona a imagem para dobrar seu tamanho original - você pode comentar a linha se preferir o tamanho original (178x128 pixels). O bloco destacado em verde altera cada pixel branco em um pixel verde claro.

Como executar a partir do terminal um script que grava no visor LCD do EV3

Isto é para pessoas que não estão usando o Código VS com a extensão EV3. Se você executar um código que aborda a tela de uma sessão SSH sem seguir essas instruções, você poderá ter dois problemas:

- a tela do Brickman pode reaparecer antes que o script pare de ser executado, mesmo depois de um segundo ou menos
- ou você pode ter o problema oposto: quando o script parar, pode levar um ou dois minutos antes que a tela seja revertida para a interface do Brickman

David L, um dos principais colaboradores do projeto ev3dev, me deu o seguinte conselho:

Maneira fácil, recomendado a menos que você seja um perfeccionista. Desvantagem: você verá um pequeno cursor piscando na parte inferior de um pequeno 'quadrado morto' em algum lugar da tela (você pode ver o cursor na imagem 'Hello World' mais acima nesta página). :

- Execute `sudo chvt 6`. Isso apenas muda a tela para mostrar um terminal vazio em vez de Brickman, para que Brickman não mexa com você. Você precisará digitar a senha do robô, que é 'maker'.
- Execute o seu programa. Você deve vê-lo aparecer no visor do EV3, como seria de esperar. Você pode interromper o script com Ctrl + C, se necessário. Você pode trabalhar a partir deste terminal como normal, executar o seu programa várias vezes, etc.
- Execute `sudo chvt 1` para obter Brickman de volta

O caminho do perfeccionista. Desvantagem: Mais etapas e você estará usando uma cópia do display minúsculo do EV3.

- Execute `sudo chvt 6`. Isso apenas muda a tela para mostrar um terminal vazio em vez de Brickman, para que Brickman não mexa com você. Você precisará digitar a senha do robô, que é 'maker'.
- Execute `sudo conspy`. Isso fará com que sua sessão SSH mostre o mesmo terminal que está no visor do EV3. Faça o login no prompt que mostra com login = robot e password = maker.
- Execute o seu programa. Você deve vê-lo aparecer no visor do EV3, como seria de esperar. Você pode interromper o script com Ctrl + C, se necessário. Você pode trabalhar a partir deste terminal como normal, executar o seu programa várias vezes, etc.
- Pressione a tecla ESC no PC 3 vezes para sair do terminal de exibição
- Execute `sudo chvt 1` para obter Brickman de volta

Tela de lcd (avançado)

Esta página tenta entender o tutorial oficial do LCD no ev3dev.org. O método apresentado parece terrivelmente complicado, então sugiro que você ignore esta página e foque nessa página de EV3 LCD, onde tenho certeza de que posso descrever um método muito mais simples.

O EV3 tem um LCD monocromático (preto e branco) de 178 x 128 pixels. Como o LCD só pode exibir preto e branco, cada pixel corresponde a 1 bit, com 0 = branco e 1 = preto. Cada linha contém 178 pixels, então são necessários 178 bits (22,25 bytes) para cada linha, mas na verdade o buffer contém 192 bits (24 bytes) para cada linha, de modo que alguns dos bits / bytes não são usados. O tamanho total do buffer é, portanto, 128 linhas * 24 bytes por linha = 3072 bytes.

Os bytes para cada linha estão na ordem que você espera, com o primeiro byte para os 8 pixels mais à esquerda, etc. No entanto, os bits dentro de cada byte estão na ordem oposta à que você poderia esperar, com o pixel mais à esquerda do oito representados pelo bit menos significativo, ou seja, o bit mais à direita! Portanto, a maneira mais simples de obter o padrão de pixel para cada byte pode ser inverter os bits dentro do byte. Por exemplo, se a linha zero começa com estes dois

bytes (2 * 8 = 16 bits) 10000100 00000000 então podemos obter o padrão de pixel invertendo os bits em cada byte para obter 00100001 00000000. Assim, podemos ver que os pixels 2 e 7 ser preto (não esquecendo que o primeiro pixel é o pixel zero).

O programa a seguir (uma versão modificada do exemplo nesta página) funciona para mim, mas acho difícil de seguir, embora tudo o que ele faça é desenhar linhas de grade horizontais e verticais (na coluna 100 e 64) e um círculo (centro em (40,40), raio 20). Por exemplo, eu tive que procurar o significado de << em Python - é o 'Operador de deslocamento à esquerda do Bitwise'. Como um exemplo (desta página) do que ele faz, se x = 10 (0000 1010 em binário), então x << 2 = 42 (0010 1000). Observe como o padrão de bits mudou dois lugares para a esquerda.

Um aviso antes de executar o programa abaixo: Depois de executar o programa, você sem dúvida desejará retornar à interface brickman, mas isso pode ser mais fácil de dizer do que fazer! A página do tutorial não indica como retornar à interface do Brickman e a única maneira que eu encontrei é manter pressionado os botões esquerdo, central e traseiro por alguns segundos - isso irá reiniciar o bloco.

```
#!/usr/bin/env python
"""
Hard coding these values is not a good idea because the values could
change. But, since this is an example, we want to keep it short.
Since the LCD only displays black and white, each pixel corresponds to
1 bit with 0=white and 1=black
"""
SCREEN_WIDTH = 178  # pixels
SCREEN_HEIGHT = 128 # pixels
LINE_LENGTH = 24    # 24 bytes = 192 bits, mas a tela tem apenas 178
# pixels de largura, portanto, nem todos os bits ou bytes são usados
SIZE = 3072         # tamanho do buffer será 128 * 24 = 3072 bytes

import os
import array

def main():
    buf = [0] * SIZE  # cria uma lista com 3072 elementos
    # cada um igual a zero (um zero corresponde a um pixel branco)

    # desenha uma linha vertical na coluna 100 (índice baseado em 0)
```

```
    for row in range(0, SCREEN_HEIGHT):
        buf[row * LINE_LENGTH + int(100 / 8)] = 1 << (100 % 8)

    # desenha uma linha horizontal na linha 64 (índice baseado em 0)
    for col in range(0, LINE_LENGTH):   # range(0, 24)
        buf[64 * LINE_LENGTH + col] = 0xff
      """
           0x seguido por um número significa que o número está em
           hexadecimal
           Um byte corresponde a 8 bits ou 2 caracteres hexadecimais
           O hexadecimal f é decimal 15 ou binário 1111
           Hexadecimal ff é binário 11111111 (um byte)
            Então a linha de código acima define cada elemento da lista
           buf que corresponde a um dos pixels
      """
# na linha 64 para hexadecimal ff ou binário 11111111 ou seja, todos
# pixels pretos
    import math
    # desenha um circulo, centro em (40,40), raio e 20
    for x in range(0, 20):
        y = math.sqrt(20 * 20 - x * x)
        buf[(40 + int(y)) * LINE_LENGTH + int((40 + x) / 8)] = 1 <<
((40 + x) % 8)
        buf[(40 - int(y)) * LINE_LENGTH + int((40 + x) / 8)] = 1 <<
((40 + x) % 8)
        buf[(40 + int(y)) * LINE_LENGTH + int((40 - x) / 8)] = 1 <<
((40 - x) % 8)
        buf[(40 - int(y)) * LINE_LENGTH + int((40 - x) / 8)] = 1 <<
((40 - x) % 8)

    f = os.open('/dev/fb0', os.O_RDWR)
    s = array.array('B', buf).tostring()
    os.write(f, s)
    os.close(f)

if __name__ == '__main__':
    main()
```

Eu estou supondo que as linhas em negrito copiem o conteúdo da lista buf para o buffer de quadros real dentro do EV3.

Este programa é tão complexo e obscuro! Só para desenhar 2 linhas e um círculo! Certamente deve haver uma maneira melhor de desenhar gráficos no LCD! E que tal escrever texto no LCD? E que tal poder usar os arquivos gráficos Lego RGF padrão em nossos programas EV3 em Python. É bastante fácil usar os arquivos RGF dentro dos programas EV3 Basic, então presumivelmente deve ser fácil no EV3 Python também? (Veja no final desta página)

BOTÕES

Aqui estão alguns scripts curtos demonstrando o uso da classe `Button()`.

- Se você executar os scripts a partir da linha de comando, a interface do Brickman continuará a ser exibida e a responder enquanto o script estiver em execução. O comando de impressão será impresso no terminal. Um script pode ser forçado a terminar com Ctrl-C.
- Se você executar os scripts do navegador de arquivos do Brickman, a tela do EV3 será apagada para que o script possa ser impresso nele. A interface do Brickman reaparecerá quando o script terminar. O script pode ser forçado a terminar pressionando o botão Backspace.

Atualmente, por razões desconhecidas, esses scripts funcionam somente para mim na linha de comando - eles são fechados imediatamente se forem executados no Brickman.

Pressione qualquer botão para sair

O loop verifica se algum botão está pressionado e, em caso afirmativo, emite um sinal sonoro e sai do script. O script verifica o estado do botão a cada 0,01 segundo.

```
#!/usr/bin/env python3

from ev3dev.ev3 import *
from time import sleep

btn = Button()

while True:
    if btn.any():     # checa se algum botão é pressionado.
        Sound.beep().wait()  # espera pelo beep para finalizar.
        exit()  # para o programa.
    else:
        sleep(0.01)  # espera 0.01 segundo
```

Aqui está uma versão muito mais limpa do script acima:

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
btn = Button()

while btn.any()==False: # enquanto nenhum botão for pressionado.
        sleep(0.01)  # espera 0.01 segundo
Sound.beep().wait()
```

Reagir ao botão pressionar e liberar

No loop deste script, o comando destacado `btn.process()` verifica se há alguma mudança no estado dos botões. Se detectar uma alteração, acionará os 'eventos' correspondentes. Por exemplo, se detectar que o botão esquerdo acabou de ser pressionado, ele acionará um evento "mudança de estado do botão esquerdo" e um evento "alteração de botão". Ele também atribui um valor True ao estado do parâmetro se o botão for pressionado e um valor False se o botão for liberado. Os manipuladores de eventos respondem aos eventos. Por exemplo, se o botão esquerdo for pressionado, o evento "mudança de estado do botão esquerdo" acionará o manipulador de eventos `on_left` destacado, que chamará a função "`left`" (a função não precisa ter esse nome).

```
#!/usr/bin/env python3

from ev3dev.ev3 import *
from time import sleep

btn = Button()

# faz alguma coisa quando o estado de algum botão mudar:

def left(state):
    if state:
        print('Left button pressed')
    else:
        print('Left button released')

def right(state):  # uso mais limpo de 'if' segue:
    print('Right button pressed' if state else 'Right button
released')

def up(state):
```

```
    print('Up button pressed' if state else 'Up button released')

def down(state):
    print('Down button pressed' if state else 'Down button released')

def enter(state):
    print('Enter button pressed' if state else 'Enter button
released')

def backspace(state):
    print('Backspace button pressed' if state else 'Backspace button
released')

btn.on_left = left
btn.on_right = right
btn.on_up = up
btn.on_down = down
btn.on_enter = enter
btn.on_backspace = backspace

while True:  # este loop checa continuamente o estado dos botões,
             # chama manipuladores de eventos apropriados
    btn.process() # Verifica os botões atualmente pressionados.
    # se o velho estado difere do velho estado,
    # chama os manipuladores de eventos de botão apropriados.
    sleep(0.01)  # o estado dos botões será checado a cada 0.01
segundo

# Se estiver executando este script via SSH, pressione Ctrl + C para
sair
# se estiver executando este script do Brickman, pressione longamente
o botão # de retrocesso para sair
```

Obter uma lista de estados de botão alterados

Esse script tem funcionalidade semelhante à anterior, mas usa o manipulador de eventos `on_change` (`changed_buttons`) para obter um script muito mais conciso. Pressione Ctrl-C (linha de comando) ou pressione e segure o botão de retrocesso (Brickman) para finalizar o script.

```
#!/usr/bin/env python3
```

```
from ev3dev.ev3 import *
from time import sleep

btn = Button()

def change(changed_buttons):   # changed_buttons é uma lista de
# tuple de nomes de changed button names e seus estados
    print('These buttons changed state: ' + str(changed_buttons))

btn.on_change = change

while True:  # este loop checa o estado dos botões
# continuamente e chama os manipuladores de eventos apropriados
    btn.process()
    sleep(0.01)
```

Verifica os estados dos botões

O script anterior mostrou como `process()` e seus manipuladores de eventos correspondentes podem ser usados para detectar e responder a mudanças no estado. Existem também funções (down, left, enter etc) que simplesmente detectam o estado dos botões. O script abaixo verifica e exibe o estado do botão esquerdo uma vez por segundo. Você pode finalizar o script com um pressionamento de um segundo no botão Backspace.

```
#!/usr/bin/env python3

from ev3dev.ev3 import *
from time import sleep

btn = Button()

while not btn.backspace:
    print(btn.left)
    sleep(1)
```

Outras funções de botão

O script a seguir demonstra mais duas funções: `buttons_pressed` e `check_buttons()`. A cada segundo, uma lista dos botões pressionados é impressa e você pode terminar o script pressionando SOMENTE os botões esquerdo e direito simultaneamente.

```
#!/usr/bin/env python3

from ev3dev.ev3 import *
from time import sleep

btn = Button()

while True:
    print(btn.buttons_pressed)
    if btn.check_buttons(buttons=['left','right']):
        exit()
    sleep(1)
```

USANDO MOTORES

No EV3 Python, você define a velocidade de destino definindo um valor para `speed_sp` ou 'speed setpoint'. Para os motores EV3, `speed_sp` é em graus por segundo, por isso, se você definir speed_sp para 360, o motor tentará girar 360 ° por segundo ou uma rotação por segundo. Para os motores EV3 padrão, um valor de `speed_sp` de 1000 (graus por segundo) é aproximadamente equivalente a um valor de potência de 100 no software padrão Lego EV3 (EV3-G). Portanto, com os motores grandes, você deve sempre usar valores `speed_sp` no intervalo de -1000 a +1000. Na verdade, eu recomendo que você use valores na faixa de -900 a +900 com os motores grandes, porque eles podem não ser capazes de atingir com precisão velocidades mais altas do que isso.

Com o motor médio EV3 padrão, deve ser seguro usar valores de `speed_sp` até 1400, mas é muito conveniente assumir que o valor máximo recomendável é 1000, de modo que o motor médio possa ser considerado como se fosse o motor grande.

Note que `speed_sp` representa a velocidade de TARGET em graus por segundo, mas os motores estão sujeitos às leis da física, então a velocidade real pode não corresponder à velocidade requerida. Por exemplo, se você executar este código `mB.run_timed (time_sp = 600, speed_sp = 600)`, então você pode esperar que o motor gire 0.6s * 600 ° / s = 360 ° = 1 rotação, mas na realidade ele se tornará significativamente menor (talvez 15% menos), porque a inércia do motor impede que o motor atinja instantaneamente sua velocidade alvo e, portanto, por um curto período no início do movimento, o motor gira menos rápido do que o solicitado.

Quando valores baixos de `speed_sp` são usados, os movimentos podem diferir novamente do que foi solicitado, mas neste caso o motor tende a se mover mais rápido do que o solicitado, por razões que ainda não são bem compreendidas. Por exemplo, ao usar `speed_sp = 100`, o motor pode girar em torno de 23% mais rápido do que o solicitado.

Para a documentação oficial do motor, acesse http://python-ev3dev.readthedocs.io/en/stable/motors.html.

Você pode querer fazer um ou mais motores girar a uma determinada velocidade

- através de um determinado ângulo ou número de rotações
- por um tempo determinado
- 'para sempre' (até que o motor seja parado por um comando stop () mais tarde durante a execução do programa)

Gire o motor através de um determinado ângulo

Use `run_to_rel_pos (position_sp = <ângulo em graus>, speed_sp = <valor>)`. Se você quiser que o motor funcione para trás, use um valor negativo para `position_sp` em vez de speed_sp. Usar um valor negativo para `speed_sp` não funcionará porque o sinal de `speed_sp` é ignorado por esse comando (mas não por outros). Use valores de `speed_sp` entre 0 e 1000 e tente evitar o uso de valores acima de 900, pois o seu motor pode não ser capaz de fornecer a velocidade solicitada acima de 900.

Exemplo:

Para fazer um grande motor na porta B girar 360 ° na velocidade 900 e, opcionalmente, aplicar um 'hold' (como um freio forte - veja adiante):

```
#!/usr/bin/env python3
# esse script pode rodar a partir do Brickman

from ev3dev.ev3 import *
from time import sleep

m = LargeMotor('outB')

m.run_to_rel_pos(position_sp=360, speed_sp=900, stop_action="hold")

sleep(5)   # dá ao motor o tempo para se mover
```

Ligue o motor por um tempo determinado

Use `run_timed (time_sp = <tempo em milissegundos>, speed_sp = <valor>)`

Observe que qualquer sinal negativo para `time_sp` é ignorado.

Exemplo

Este exemplo executa um motor grande conectado à porta B para trás por 3 segundos com um 'setpoint de velocidade' configurado para -750 (equivalente a uma configuração de energia de -75 no software padrão EV3). Ele funcionará mesmo se um segundo motor grande também estiver conectado a outra porta do motor, desde que a letra da porta seja especificada. Sem a última linha, o programa terminaria assim que o motor começasse a girar, então você provavelmente não o veria se mover. Esperar por 5 segundos (1 + 4) garante que o motor possa girar por 3 segundos antes que o programa termine.

```
#!/usr/bin/env python3
# esse programa roda a partir do Brickman

from ev3dev.ev3 import *
from time import sleep
m = LargeMotor('outB')
m.run_timed(time_sp=3000, speed_sp=-750)
print("set speed (speed_sp) = " + str(m.speed_sp))
sleep(1)  # dá um tempo para o motor iniciar seu movimento
print("actual speed = " + str(m.speed))
sleep(4)
```

O exemplo acima imprime o valor de `speed_sp` na janela do terminal ou no console. Eu usei um operador '+' para concatenar (juntar) a sequência de texto à esquerda com o valor `speed_sp` à direita, mas o valor `speed_sp` é um número inteiro, então tive que convertê-lo em uma sequência de texto com `str()` função antes que a concatenação possa ocorrer. Eu acho que é razoável para iniciantes usar o operador '+' dessa maneira, mas há outro método que é frequentemente preferido por programadores Python mais avançados. Note que ao executar o programa acima, a instrução de impressão é executada assim que o motor começa a se mover - o programa não espera até que o motor tenha terminado de se mover antes de executar o próximo comando.

O exemplo também fornece um valor (`m.speed`) para a velocidade real do motor, um segundo depois que o motor foi instruído a começar a girar. O atraso de um segundo é necessário porque

leva um momento para o motor continuar - sem o atraso, a velocidade real provavelmente seria dada como zero porque o motor ainda não teria começado a girar.

Curiosamente, o programa acima também funciona se a terceira linha for alterada para
`m = Motor ('outB')`
mas não funciona (com um motor grande ainda conectado à porta B) se esta linha for usada:
`m = MediumMotor ('outB')`

Acione o motor 'para sempre'

Use `run_forever (speed_sp = <value>)` e pare o motor com `stop()` ou `stop (stop_action = <value>)` onde <value> pode ser "coast", "brake" ou "hold". Ao usar o `run_forever` você precisa estar ciente de que a mudança de `speed_sp` enquanto o `run_forever` está sendo executado NÃO tem efeito imediato

```
#!/usr/bin/env python3
# esse programa roda a partir do Brickman

from ev3dev.ev3 import *
from time import sleep
m = LargeMotor('outB')

m.run_forever(speed_sp=900)
sleep(5)
m.stop(stop_action="hold")
sleep(5)
```

Eu mostro este código porque é breve - na realidade você nunca usaria este código porque o motor rodaria por um tempo determinado, então seria melhor usar `run_timed`. O uso real de `run_forever` poderia ser, por exemplo, iniciar um robô em movimento e interromper o movimento quando um obstáculo é detectado.

No meu exemplo, o `stop_action` é definido como 'hold', o que faz um esforço ativo e vigoroso para parar o giro do motor. Você pode alternativamente usar `stop_action = "brake"`, que passivamente e com menos força tenta parar o giro do motor, ou você pode usar `stop_action = "coast"`, que permite que o motor continue girando até parar por fricção. Se desejar, você também pode especificar um valor para `stop_action` ao executar outros comandos do motor,

como `run_timed()` ou `run_to_rel_pos()`. No software padrão Lego EV3, as únicas opções são 'coast' e 'brake'.

O valor de `stop_action` será lembrado no código a seguir, por exemplo, o segundo comando `stop()` usará `stop_action = "coast"`. Além disso, o segundo comando `run_forever` usará o valor lembrado de `speed_sp`, uma vez que nenhum foi especificado.

```
#!/usr/bin/env python3
# esse programa roda a partir do Brickman
from ev3dev.ev3 import *
from time   import sleep
m = LargeMotor('outB')

m.run_forever(speed_sp=200)   # equivalente a power=20 em EV3-G
sleep(5)
m.stop(stop_action="coast")
sleep(4)

m.run_forever()
sleep(5)
m.stop()
sleep(4)
```

Aguarde a conclusão

Quando um comando faz com que um motor inicie uma execução do programa de movimento, não faz uma pausa para que o movimento seja concluído antes de continuar. Muitas vezes, queremos que a execução do programa seja pausada. Geralmente é possível usar um comando `sleep()` para isso, mas isso não é ideal - o script abaixo mostra uma maneira melhor de fazer o programa pausar até que o movimento seja concluído.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *

motor = LargeMotor('outC')
motor.run_timed(time_sp=3000, speed_sp=-750, stop_action='brake')
motor.wait_while('running')

Sound.beep()
```

Note que o método acima funciona com todos os três valores `stop_action` possíveis, 'coast', 'brake' e 'hold'. O comando `beep()` é incluído para que você possa ouvir que a execução do programa foi de fato pausada até que o movimento do motor esteja completo antes de tocar o bipe. O comando `wait_while()` só funciona no EV3 Python v0.8.0 ou posterior, portanto, tenha a versão mais recente. A versão 0.8.0 foi lançada em outubro de 2016.

Às vezes, você pode ver o seguinte fragmento de código:

```
While any(motor.state):
    sleep(0,1)
```

este código também é projetado para fazer a execução do programa pausar até que o motor complete seu movimento, mas é menos satisfatório, principalmente porque ele não funciona com `stop_action = 'hold'`. Este fragmento de código acima deve, portanto, ser considerado obsoleto.

Parando os motores

Você já sabe que os motores podem ser parados com o comando `stop()` e que a parada pode ser feita de três maneiras: coast, brake ou hold (por exemplo, para parar um motor chamado `mB`: `mB.stop (stop_action = 'brake')` Eu acho que o `stop_action = 'coast'` provavelmente é raramente usado. Eu também acho que seria muito raro mudar para o modo `stop_action` durante um script, por exemplo, de brake para hold. Portanto, eu acho que em programas onde o modo `stop_action` não vai mudar o modo `stop_action` deve ser definido apenas uma vez e o mais cedo possível no script. No entanto, não defina o modo `stop_action` na mesma linha que atribui o motor a uma classe de motor, uma vez que isso não funcionará como esperado. faça isso (usei strikethrough para enfatizar que isso está errado) `mB = LargeMotor ('outB', stop_action = 'brake').`

Eu recomendo que você escreva e execute seus scripts do EV3 Python usando o Visual Studio Code com a extensão EV3. Se você não estiver usando o fluxo de trabalho do Código VS, poderá descobrir que às vezes os motores continuarão a funcionar depois que um script (mal escrito?) Terminar. Como você pararia os motores em tal caso? Você poderia iniciar o interpretador Python3 e emitir alguns comandos, ou você poderia fazer o que eu faço: manter um script especial `'stop-motors.py'` em mãos para esses casos. Meu script funciona apenas para dois grandes motores chamados mB e mC conectados às portas B e C, mas para mim isso é sempre o caso. Modifique o script, se necessário, para corresponder à sua situação.

```
#!/usr/bin/env python3
```

```
from ev3dev.ev3 import *

mB = LargeMotor('outB')
mC = LargeMotor('outC')

mB.stop()
mC.stop()

# to make extra sure the motors have stopped:
mB.run_forever(speed_sp=0)
mC.run_forever(speed_sp=0)
```

Se você usar o fluxo de trabalho do Código VS, não precisará se preocupar com a continuidade dos motores quando o programa for interrompido. Esta é uma das várias razões para adotar o maravilhoso fluxo de trabalho do VS Code.

Regulação de velocidade

O EV3 Python geralmente usa um recurso chamado 'regulação de velocidade' ao usar motores. Quando a regulagem de velocidade está ativada, o controlador do motor irá variar a potência fornecida ao motor para tentar manter a velocidade especificada em speed_sp, que é em graus por segundo. É comparável à maneira como o controle de cruzeiro do seu carro tentará manter uma velocidade constante, mesmo se o carro subir ou descer as colinas. Se a regulação de velocidade estiver desligada, o que, desde a versão do kernel 10 e posterior, é apenas o caso de `run_direct()`, o controlador usará a potência especificada em `duty_cycle_sp`. Não esqueça que para os motores EV3 de grande porte, um valor de potência de 100 no software Lego EV3 padrão (EV3-G) é aproximadamente equivalente a um valor `speed_sp` de 1000.

Observe que a regulação de velocidade está sempre ativada (exceto pelo comando `run_direct`), o que significa que os motores agora são sempre controlados pelo parâmetro `speed_sp` (exceto pelo comando `run_direct`). Basicamente, então, você deve evitar o uso de `run_direct` e, portanto, nunca use `duty_cycle_sp` e sempre use `speed_sp`.

Funções motoras semelhantes a EV3G

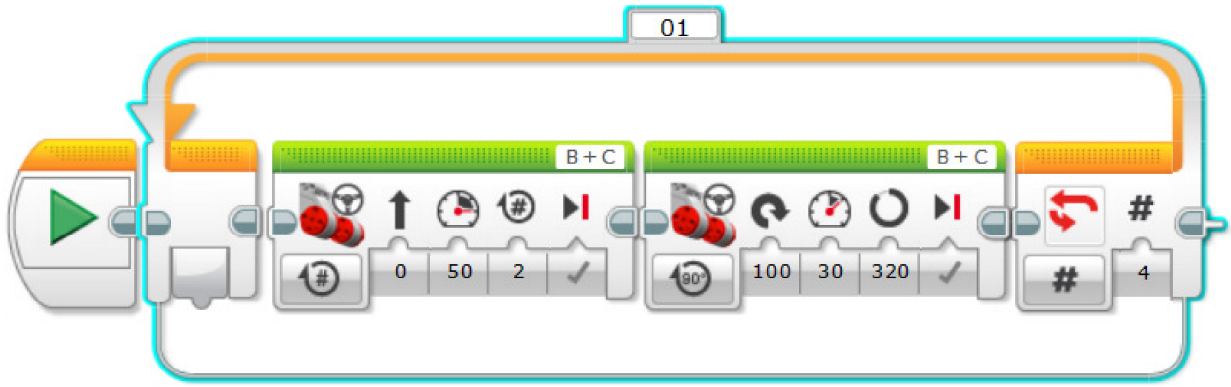
Este material existe porque eu quero tornar mais fácil para os iniciantes começarem com o EV3 Python. A maioria das pessoas que começam a aprender o EV3 Python terá usado pela primeira vez o software padrão baseado em blocos do Lego EV3, comumente chamado de EV3-G. Eles estarão muito familiarizados com os quatro blocos de motor do EV3-G: motor médio, motor

grande, mover direção e mover tanque. Fazer a mudança para EV3 Python será bastante difícil para esses usuários, porque o EV3 Python não possui funções que espelhem de perto as funções do EV3-G. Por exemplo, duas das funções do motor EV3-G são usadas para controlar dois motores simultaneamente, enquanto o EV3 Python não possui funções que possam fazer isso. Pior ainda, isso significa que o EV3 Python não tem capacidade de sincronizar o movimento de dois motores para garantir que eles mantenham uma taxa de velocidade constante - essa é uma característica de quase todas as linguagens compatíveis com EV3, exceto EV3 Python e eu argumento no problema 853 do EV3dev que uma função deve ser adicionada ao EV3dev e EV3 Python para permitir o movimento sincronizado de dois motores.

De qualquer forma, acho que seria muito útil se houvesse funções disponíveis no EV3 Python que imitassem as funções do motor EV3-G. Isso não apenas tornaria mais fácil para os usuários trocarem de EV3-G para EV3 Python, mas também permitiria que o código Python fosse muito mais curto e legível.

Eu criei quatro novas funções para essa finalidade: `movemedium()`, `movelarge()`, `movesteer()` e `movetank()`. Essas funções já estão funcionando quando são fornecidos valores sensíveis, mas precisam de polimento, como a validação de argumentos fornecidos. Contudo, estou retendo a publicação de minhas funções na esperança de que a equipe do ev3dev desenvolva suas funções para imitar, pelo menos, os blocos de Mover Direção e Mover Tanque. A equipe do ev3dev é capaz de fazer um trabalho melhor do que eu poderia fazer e seria muito melhor ter funções que fazem parte do pacote oficial do ev3dev para Python.

Aqui, por exemplo, é um programa EV3-G clássico que faz com que um robô rover siga um caminho quadrado:



Dentro deste loop estão dois blocos de 'movimentação de direção'. Para aqueles de vocês que não estão familiarizados com esses blocos, a ideia é especificar um valor de direção entre -100 e +100 de forma que:

- um valor de 0 faz com que o rover se mova para frente (motores girando com as mesmas velocidades)

- um valor de + -50 faz o rover girar com um motor não girando
- um valor de + -100 faz com que o rover ligue no local (motores girando com velocidades iguais e opostas)
- valores positivos têm o motor mais rápido girando para a frente para que o movimento geral seja para frente

O primeiro bloco de mover direção tem o valor de direção ajustado em 0, de modo que o rover se moverá para frente. O segundo bloco de mover direção destina-se a fazer o rover girar 90 °. O ângulo através do qual os motores têm que girar para que isso aconteça depende do diâmetro e do espaçamento das duas rodas motrizes do rover, de modo que o ângulo de 320 graus mostrado aqui precisaria ser modificado para se adequar ao rover em questão.

No padrão EV3 Python, o equivalente ao programa acima seria algo assim:

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
mB = LargeMotor('outB')
mC = LargeMotor('outC')

for i in range(0,4): # i=0, 1, 2, 3
    mB.run_to_rel_pos(position_sp=720, speed_sp=500,
stop_action="brake")
    mC.run_to_rel_pos(position_sp=720, speed_sp=500,
stop_action="brake")
    mB.wait_while('running')
    mC.wait_while('running')

    mB.run_to_rel_pos(position_sp=300, speed_sp=320)
    mC.run_to_rel_pos(position_sp=300, speed_sp=-320)
    mB.wait_while('running')
    mC.wait_while('running')
```

Eu não acho que isso seja particularmente fácil para um novato em Python EV3 entender. Com minha nova função `'movesteer()'` o script pode ser escrito assim:

```
#!/usr/bin/env python3
from motorfunctions import *

for i in range(0,4): # i=0, 1, 2, 3
```

```
    movesteer('rotations', ports='BC', steer=0, pwr=50, rots=1,
stop_action='brake')
    movesteer('degrees', ports='BC', steer=100, pwr=30,
degs=320 stop_action='brake')
```

Como você pode ver, o script encolheu de 13 linhas para 5 linhas, e o conteúdo do loop 'for' diminuiu de 8 linhas para 2. Não apenas isso, mas a função `movesteer()` combina com o bloco mover direção:

- Como o bloco de mover direção, ele tem um parâmetro de direção.
- Como o bloco de mover direção, ele usa valores de potência na faixa de -100 a +100, facilmente interpretados como porcentagens da velocidade máxima possível. A nova função é configurada por padrão para que um valor de potência de 100 nas novas funções forneça o mesmo valor de potência de velocidade de 100 no EV3-G, tanto para os motores médios quanto para os grandes.
- Como o bloco de mover direção, ele pode aceitar um ângulo expresso em rotações ou graus.
- Como o bloco de mover direção, ele tem esses modos possíveis: ligado, desligado, rotações, graus, cronometrado.

O primeiro script (tradicional) do EV3 Python acima é problemático não apenas em extensão, falta de similaridade com o EV3-G e, portanto, redução da legibilidade, mas também porque:

- Ele não contém nada que se pareça com o recurso de 'direção' do bloco de mover direção EV3-G.
- Ele usa valores de velocidade de 'unidades tacho' por segundo (para os motores EV3 isto é equivalente a graus por segundo). O intervalo permitido de valores nunca foi declarado, penso eu, embora pareça que é seguro usar valores no intervalo de -900 a +900 para o motor grande EV3 (não muito conveniente). Para o motor médio, a faixa permitida é diferente (provavelmente -1400 a +1400), o que é ainda mais confuso!
- Não pode aceitar ângulos nas rotações, por isso é necessário converter as rotações em graus.

Há notícias ainda melhores sobre as novas funções! Com exceção do primeiro argumento, que deve sempre ser incluído para maior clareza, os outros argumentos são "argumentos-chave", o que significa que eles precisam ser incluídos apenas se seus valores diferirem dos valores padrão ou dos últimos valores usados. Os valores padrão para `movesteer()` e `movetank()` incluem o uso das portas B e C, o que é uma convenção forte para robôs rover EV3. E os padrões também incluem um valor de direção de 0 e o `stop_action` de 'brake', assim como o EV3-G, então o script pode ser escrito ainda mais curto:

```
#!/usr/bin/env python3
from motorfunctions import *

for i in range(0,4): # i=0, 1, 2, 3
    movesteer('rotations', pwr=50, rots=1)
    movesteer('degrees', steer=100, pwr=30, degs=320)
```

Compare isso com o script original do EV3 Python!

Como você deve ter adivinhado, as novas funções são definidas em um módulo chamado 'motorfunctions' que é importado após o 'shebang'. (A primeira linha, o 'shebang', é necessária para que o script seja inicializável de dentro do Brickman.) O módulo de funções motoras inclui uma linha de ev3dev.ev3 import *, portanto, não é necessário que o usuário inclua isso em seu ou seus scripts, além de importar o módulo de funções motoras. As operações internas do módulo de funções motoras poderiam permanecer uma 'caixa preta' para usuários iniciantes, mas idealmente eles deveriam aprender como o módulo funciona - não é particularmente difícil, mas a necessidade de incluir a validação de argumentos torna o código um pouco difícil de ler. Outro fator que dificulta o código de funções motoras é a necessidade de nunca ignorar o sinal do argumento da velocidade. Os blocos de motor EV3-G nunca ignoram os sinais negativos nos argumentos, mas a função `run_to_rel_pos()` do EV3 Python ignora qualquer sinal negativo para `speed_sp`.

Aqui estão mais alguns exemplos das novas funções:

```
# gira o motor médio a velocidade máxima por 4 seg (4000ms)
movemedium('timed', port='D', pwr=100, time_sp=4000)

# começa a se movimentar para frente
movesteer('on', ports='BC', steer=0, pwr=50)

# para os motores
movesteer('off', ports='BC')

#  gira no ponto
movetank('rotations', ports='BC', pwrL=50, pwrR=50, rots=-2.84,
stop_action='hold')
```

Observações

1) Embora eu visasse um alto nível de similaridade entre as funções de motorfunctions e os blocos de motor EV3-G, decidi disponibilizar o `stop_action 'hold'` nas novas funções, mesmo

que esta ação de parada não esteja disponível no EV3- Blocos do motor G (eles só oferecem 'coast' ou 'brake').

2) Usando as novas funções, é muito fácil tornar o código compatível com diferentes tamanhos de roda (as rodas no kit educacional têm diâmetros 30% maiores do que as rodas no kit de varejo). Veja como um 'fator de pneu' chamado tf foi incluído no código abaixo:

```
#!/usr/bin/env python3
from motorfunctions import *
tf=1 # tire factor = 1 for educational, 1.3 for retail

for i in range(0,4): # i=0, 1, 2, 3
    movesteer('rotations', pwr=50, rots=1*tf)
    movesteer('degrees', steer=100, pwr=30, degs=320*tf)
```

Isso significa que se o código der a rotação correta com os pneus grandes do kit de treinamento, alguém usando os pneus menores do kit de varejo teria que alterar o valor de tf para obter as rotações corretas. Eu também poderia ter usado o fator do pneu como multiplicador para o poder, mas optei por não simplificar - isso significa que um rover usando os pneus menores do kit de varejo se moverá um pouco mais lento que um rover usando os pneus maiores do kit educacional mas isso normalmente não é um problema. Outra razão importante para não aplicar sistematicamente o multiplicador de pneus aos valores de potência seria que isso poderia facilmente fazer com que o valor de potência excedesse a faixa permitida. Por exemplo, se o código original de um rover com pneus de kit de educação incluir um valor de potência de 100, multiplicá-lo por um fator de pneu de 1,3 forneceria um valor de potência de 130 que está fora da faixa permitida (-100 a +100) as funções de funções motoras.

3) Alguns dirão que as novas funções que estou propondo não são necessárias, uma vez que não introduzem nenhuma nova funcionalidade. Espero que esta página forneça um argumento convincente de que as novas funções podem ser úteis para novos usuários do EV3 Python.

4) Alguns dirão que é muito confuso para as novas funções usar intervalos de argumentos e unidades que são diferentes daquelas usadas pelas funções motoras existentes do EV3 Python, como usar valores 'power' no intervalo de -100 a +100 em vez de valores speed_sp no intervalo de -900 a +900 para o motor grande e -1400 a +1400 para o motor médio (muito complicado para iniciantes!). Acho que os usuários iniciantes do EV3 Python raramente precisariam usar as funções motoras padrão do EV3 Python, o que significa que não deve haver muita confusão. Menos, talvez,

do que seria causado por usuários do EV3-G terem que se acostumar a trabalhar com valores de velocidade que tenham um alcance tão estranho, às vezes excedendo 1000! Em qualquer caso, os usuários do EV3 Python já precisam lidar com unidades inconsistentes, pois, por exemplo, o sleep () usa segundos e as funções do motor usam milissegundos. Dito isto, decidi usar milissegundos em vez de segundos para as novas funções motoras.

5) Estou inclinado a fazer com que as novas funções sempre esperem que o movimento do motor seja concluído antes de permitir que o programa continue, assim como o EV3-G. Este será certamente o comportamento padrão. Mas pode ser possível incluir outro argumento de palavra-chave para que a espera pela conclusão do movimento possa ser ativada ou desativada.

Outras notas

Observe que, se você tiver apenas um motor grande conectado ao EV3, não será necessário especificar em qual porta do motor ele está conectado. O mesmo vale para o motor médio e até para cada tipo de sensor.

Esta página introduziu alguns mas não todos os comandos do motor. Por exemplo, ele introduziu `run_to_rel_pos()`, mas não `run_to_abs_pos()`. Para aprender sobre outros comandos, consulte a documentação oficial.

CONTROLE REMOTO

Note que a versão educacional do kit EV3 não inclui o dispositivo de controle remoto, mas pode ser comprado separadamente.

Eu tive que modificar o programa suprimindo as referências aos botões do bloco EV3, já que o código não funciona para mim (usando o kernel 3.16.7-ckt21-9-ev3dev-ev3 conforme relatado pelo item 'About' na parte inferior da tela). Menu Brickman). Eu também removi as linhas que fizeram o robô voltar e falar se o sensor de toque detectou uma colisão, já que isso não tem nada a ver com o controle remoto. O programa modificado é mostrado abaixo, com as linhas que eu modifiquei destacadas em amarelo. Eu também destaquei em ciano as linhas que dizem respeito ao controle remoto. Já que não é mais possível parar o programa pressionando um botão no bloco, você deve encerrar o programa com Ctrl-C se estiver usando o terminal ou pressionar o botão de volta se estiver iniciando o programa a partir do Brickman. interface no bloco.

```
#!/usr/bin/env python3
# o 'shebang' acima está incluído para que o programa
```

```
# pode ser lançado no Python3 a partir do Brickman

from time import sleep
from ev3dev.ev3 import *

# conecta 2 motores grandes às portas A e B
lmotor = LargeMotor('outB')
rmotor = LargeMotor('outC')

# checa se os motores estão, de fato, conectados
assert lmotor.connected
assert rmotor.connected

# conecta o controle remoto
rc = RemoteControl(); assert rc.connected

# inicializa o manipulador de botões
# button = Button()   # não funcional, logo, desabilitado
# desliga os leds
Leds.all_off()

def roll(motor, led_group, direction):
    """
   Gerar manipulador de eventos de controle remoto. Ele rola dado
motor em dado direção (1 para frente, -1 para trás). Quando o motor
rola para a frente,dado grupo led pisca verde, quando para trás -
vermelho. Quando o motor pára,leds estão desligados.
   A função on_press possui assinatura exigida pela classe
RemoteControl.
     É necessário um parâmetro de estado booleano; Verdade quando o
botão é pressionado, False
     de outra forma.
    """
    def on_press(state):
        if state:
            # Rolar quando o botão é pressionado
            motor.run_forever(speed_sp=90*direction)
            Leds.set_color(led_group, direction > 0 and Leds.GREEN or
Leds.RED)
        else:
```

```
            # Pare de outra forma
            motor.stop(stop_action='brake')
            Leds.set(led_group, brightness_pct=0)

    return on_press

# Atribuir manipulador de eventos a cada um dos botões do controle
remotorc.on_red_up    = roll(lmotor, Leds.LEFT,   1)
rc.on_red_down  = roll(lmotor, Leds.LEFT,  -1)
rc.on_blue_up   = roll(rmotor, Leds.RIGHT,  1)
rc.on_blue_down = roll(rmotor, Leds.RIGHT, -1)

# entra no loop de processamento de eventos
#while not button.any():   #não functional, por isso comentário.
while True:   #substitui linha prévia, então , use Ctrl +C para sair
    rc.process()
    sleep(0.01)
    # Precione Ctrl +C para sair
```

USANDO SENSORES

O EV3 Python é compatível com todos os sensores padrão EV3 e NXT, exceto que o sensor de luz NXT não pode ser usado para detectar cores (a partir de setembro de 2016).

Os sensores EV3 e / ou NXT podem ser conectados a qualquer uma das quatro portas do sensor do EV3, mas vou aderir à convenção útil de que eles devem ser anexados da seguinte forma:

porta 1 = toque, porta 2 = giroscópio, porta 3 = cor, porta 4 = infravermelho ou ultrassônico.

Essa convenção é útil porque significa que você não precisa reconectar os sensores ao passar de um tutorial do EV3 para outro.

Se você não anexar mais de um sensor de um tipo específico, não importará em qual porta um sensor está conectado. Você não precisa incluir nenhuma referência em seu código ao número da porta do sensor - seu programa simplesmente funcionará. No entanto, se você conectar mais de um sensor de um determinado tipo, será necessário especificar o número da porta com 'in1', 'in2', etc. ou observar o atributo `port_name`.

Como de costume, eu só mencionei o que eu acho que é mais provável que seja útil para você como um iniciante. Para sensores, isso inclui sensores de `InfraredSensor()`, `TouchSensor` (),

`UltrasonicSensor()`, `ColorSensor()`, `LightSensor()`, `GyroSensor()`, `RemoteControl()`, `SoundSensor()`, `value()`, `mode()` e `units`. Note que o sensor 'color' do EV3 corresponde ao sensor 'light' do NXT.

Como no software padrão Lego EV3, muitos sensores podem ser usados em diferentes modos. Para definir o modo desejado para o sensor que você está usando, use o modo como no primeiro exemplo abaixo. Por exemplo, se um único sensor IR estiver conectado a qualquer porta de sensor e nomeado ir, você poderá configurá-lo para o modo de proximidade com `ir.mode = 'IR-PROX'`. O sensor de toque não tem vários modos, é claro, mas para os outros sensores é importante sempre definir o modo do sensor antes de usá-lo.

A leitura de todos os sensores em todos os modos (exceto o sensor de cor no modo RGB-RAW) é obtida com o método `value()` (um 'método' é um tipo especial de função). Não esqueça de incluir os parênteses.

Exemplo 1 (sensor de toque e sensor IR)

O exemplo abaixo usa o sensor de toque e o sensor de infravermelho. Ele exibe continuamente o valor da distância detectado pelo sensor IR, junto com as unidades correspondentes (pct, significando porcentagem, para o sensor IR). Acende o LED esquerdo de verde para vermelho quando um objeto é trazido para perto do sensor de infravermelho e o transforma em verde quando nenhum objeto está próximo. O loop while é executado enquanto o botão do sensor de toque NÃO estiver pressionado. Quando o sensor de toque é pressionado, o loop é encerrado, um bipe é emitido e o LED esquerdo é colocado em verde (pode ter sido vermelho quando o loop foi encerrado).

O exemplo usa a instrução `assert` para verificar se os sensores estão conectados. Se a asserção falhar, o programa fecha com uma mensagem de erro útil (neste caso, uma mensagem de texto). Assert é uma declaração e não uma função, portanto deve ser usada sem parênteses como aqui.

```
#!/usr/bin/env python3
# esse script roda a partir do Brickman

from ev3dev.ev3 import *

# Conecta infravermelho e toque nas portas de sensores
# e checa se estão conectados.

ir = InfraredSensor()
```

```
assert ir.connected, "Connect a single infrared sensor to any sensor
port"

ts = TouchSensor();    assert ts.connected, "Connect a touch sensor to
any port"
# pode haver 2 statements na mesma linha com o uso do ponto e vírgula

# Põe o sensor infravermelho em modo de proximidade.
ir.mode = 'IR-PROX'

while not ts.value():    # Para o programa acionando o sensor de toque
# O sensor infravermelho no modo de proximidade medirá
#a distância até o mais próximo em frente a ele.
    distance = ir.value()

    if distance < 60:
        Leds.set_color(Leds.LEFT, Leds.RED)
    else:
        Leds.set_color(Leds.LEFT, Leds.GREEN)

Sound.beep()
Leds.set_color(Leds.LEFT, Leds.GREEN)
#certifica que o led esquerdo esteja verde anters de sair
```

Exemplo 2 (sensor de toque e sensor ultrassônico EV3)

Um sensor ultrassônico está incluído na versão educacional do kit EV3, mas não na versão caseira (pode ser comprado separadamente). Se você tiver um único sensor ultrassônico EV3 conectado ao EV3, você pode modificar o código do Exemplo 1 para fazê-lo funcionar com o sensor EV3 US como mostrado abaixo. Quando você ajusta o modo do sensor EV3 US para `US-DIST-CM` conforme abaixo, você espera que o sensor retorne um valor em cm, é claro, e em certo sentido, mas é um dos vários modos (consulte a página de modos do sensor) que também inclui uma primeira casa decimal (mas sem ponto decimal) no valor retornado. Isso significa que o valor retornado representa realmente mm, em vez de cm. Por exemplo, se o sensor retornar o valor 90 cm, na verdade significa 9,0 cm (ou 90 mm), pois o valor inclui uma casa decimal! Obviamente, você precisa tomar muito cuidado ao manipular os valores retornados por esses modos. Para lidar com esses problemas, incluí uma divisão por dez no código abaixo para converter a distância de mm para cm. Assim, o código abaixo fará com que a cor do LED esquerdo mude para vermelho quando um objeto estiver a 60,0 cm (600 mm) do sensor US.

Da mesma forma, se você definir o modo como `US-DIST-IN`, você espera que o valor seja em polegadas, mas na verdade ele é em décimos de uma polegada, então você precisa dividir o valor por 10 para obter polegadas. Ou, em outras palavras, o valor está em polegadas, mas inclui a primeira casa decimal do valor. Confuso, não é? Se você usar polegadas, provavelmente desejará usar uma distância menor que 60,0 polegadas para o led mudar para vermelho. Observe que o problema com o sensor ultrassônico EV3 NÃO é compartilhado pelo sensor ultrassônico NXT, que simplesmente fornece a leitura como um número inteiro (casas decimais zero).

```
#!/usr/bin/env python3
# esse script roda no Brickman

from ev3dev.ev3 import *

# conecta os sensores ultassonico e toque nas portas
# e checa se estao conectados.
us = UltrasonicSensor()
assert us.connected, "Connect a single US sensor to any sensor port"
ts = TouchSensor();    assert ts.connected, "Connect a touch sensor to
any port"
# pode haver 2 statements na mesma linha com o uso do ponto e virgula

# poe o sensor US no modo distancia.
us.mode='US-DIST-CM'

units = us.units
# reporta 'cm' apesar de o sensor medir 'mm'

while not ts.value():    # Stop program by pressing touch sensor
button
# O sensor infravermelho no modo de proximidade medirá
# a distância até o mais próximo em frente a ele.
    distance = us.value()/10  # converte mm para cm
    print(str(distance) + " " + units)

    if distance < 60:  #
        Leds.set_color(Leds.LEFT, Leds.RED)
    else:
        Leds.set_color(Leds.LEFT, Leds.GREEN)
```

```
Sound.beep()
Leds.set_color(Leds.LEFT, Leds.GREEN)
```

Exemplo 3 (dois sensores de toque)

Normalmente você não terá dois sensores do mesmo tipo conectados ao EV3 e você não terá que especificar ou determinar onde os sensores estão conectados. Mas neste exemplo, vamos supor que temos dois sensores de toque conectados ao EV3. Especificaremos em quais portas os sensores são conectados usando 'in1' e 'in2' para as portas 1 e 2 do sensor (entrada). Neste programa, o LED esquerdo está verde se o sensor de toque na porta 1 não estiver pressionado e vermelho se for pressionado (mais especificamente, se o sensor de toque não for pressionado, ele retornará um valor zero e o código usará o item zero da tupla). De maneira semelhante, o sensor de toque na porta 2 controla o led direito. Para parar este programa, você precisará pressionar Ctrl-C para interromper o loop while.

```
#!/usr/bin/env python3
# esse programa roda a partir do Brickman
from ev3dev.ev3 import *
from time   import sleep

# conecta 2 sensores de toque nas portas 1 e 2
# e checa se estão conectados.

ts1 = TouchSensor('in1')
assert ts1.connected, "Connect a touch sensor to sensor port 1"

ts2 = TouchSensor('in2')
assert ts2.connected, "Connect a touch sensor to sensor port 2"

while True:    # Para esse programa com Ctrl-C
    Leds.set_color(Leds.LEFT, (Leds.GREEN, Leds.RED)[ts1.value()])
    Leds.set_color(Leds.RIGHT, (Leds.GREEN, Leds.RED)[ts2.value()])

# Para o programa com Ctrl-C
```

Exemplo 4 (sensor giroscópio e sensor táctil)

O sensor giroscópio está incluído na versão educacional do kit EV3, mas não na versão caseira (pode ser comprado separadamente). Ele mede o ângulo em graus, dando o valor de zero graus à orientação do sensor quando o programa é iniciado. Note que é vital que o sensor giroscente fique

absolutamente imóvel quando o programa for lançado, caso contrário a leitura passará mais tarde, mesmo quando o sensor estiver parado.

O sensor também pode medir a taxa de mudança do ângulo, em graus por segundo, mas, como iniciante, você provavelmente achará isso menos útil.

O programa abaixo imprime o ângulo medido para o console, em seguida, reproduz por um segundo um tom cuja frequência depende do ângulo e aguarda por 0,5 segundo antes de repetir o ciclo. Pressione o botão do sensor de toque por pelo menos um segundo para parar o programa.

```
#!/usr/bin/env python3
# esse script roda a partir do Brickman

from ev3dev.ev3 import *
from time import sleep

# Conecta o sensor gyro and toque em qualquer porta
# e checa se estao conectados.
gy = GyroSensor()
assert gy.connected, "Connect a single gyro sensor to any sensor port"
ts = TouchSensor();    assert ts.connected, "Connect a touch sensor to
any port"
# pode haver 2 statements emu ma linha com o uso do ponto e virgula

# Poe o sensor gyro em modo ANGLE.
gy.mode='GYRO-ANG'

units = gy.units
#reporta 'deg' significando graus

while not ts.value():    # para o programa pressionando o sensor de
toque
    angle = gy.value()
    print(str(angle) + " " + units)
    Sound.tone(1000+angle*10, 1000).wait()
    sleep(0.5)

Sound.beep()
```

Exemplo 5: Sensor de cor no modo COL-REFLECT

Neste script, o sensor de cor emite luz e mede a intensidade da luz refletida, retornando duas vezes por segundo um valor teoricamente entre 0 e 100. Para obter melhores resultados, coloque o sensor a cerca de 3 mm da superfície refletora. Usando essa separação, obtenho um valor de cerca de 80 com papel branco normal e cerca de 5 com uma superfície preta típica.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep

# Conecta o sensor cor EV3 and checa se está conectado.

cl = ColorSensor()
assert cl.connected, "Connect a color sensor to any sensor port"
# Põe o sensor de cor em modo COL-REFLECT
# para medir a intensidade da luz refletida.
# nesse modo o sensor retornará um valor entre 0 e 100
cl.mode='COL-REFLECT'

while True:
    print(cl.value())
    sleep(0.5)

# Consegui o máximo de 80 em papel branco com 3mm de sepaeação
# e 5 com plástico preto com  a mesma separação
```

Exemplo 6: Sensor de cor EV3 no modo COL-AMBIENT

Neste programa, o sensor de cor mede a intensidade da luz ambiente, retornando um valor entre 0 e 100. Esse valor é usado para controlar a velocidade de um grande motor conectado à porta B, então quanto mais brilhante a luz ambiente, mais rápido o motor girará. Pressione o botão do sensor de toque para parar o programa. Para fazer este programa funcionar com intensidade de luz refletida em vez de intensidade de luz ambiente, basta substituir `COL-AMBIENT` por `COL-REFLECT`.

```
#!/usr/bin/env python3
# O script roda a partir do brickman

from ev3dev.ev3 import *
```

```
from time import sleep

# Conecta o sensor de cor EV3 e o de toque
# e checa se estão conectados.

cl = ColorSensor()
assert cl.connected, "Connect an EV3 color sensor to any sensor port"

ts = TouchSensor()
assert ts.connected, "Connect a touch sensor to any sensor port"

# Conecta um motor grande na porta B e checa a conexão.
m = LargeMotor('outB')
assert m.connected, "Connect a large motor to port B"

# poe o sensor de cor em modo COL-AMBIENT
# para medir a intensidade da luz ambiente.
# Nesse modo o sensor retorna um valor entre 0 e 100
cl.mode='COL-AMBIENT'

# o commando run_forever command nos permitira variar a performance
# do motor emu so ajustando o atributo speed_sp.
m.run_forever(speed_sp = 0)

while not ts.value():    # para o programa acionando o sensor de toque
# poe a velocidade do motor para ser igual
# ao valor medido da luz ambiente
    m.speed_sp = cl.value()

Sound.beep()
```

Exemplo 7: Sensor de cor EV3 no modo COL-COLOR e sensor de toque

Quando o sensor de cor EV3 está no modo `COL-COLOR`, ele tenta reconhecer a cor dos blocos de Lego padrão posicionados a cerca de 5-6mm na frente do sensor (a distância é crítica) e retorna um valor inteiro correspondente entre 0 (desconhecido) e 7 (Castanho). O programa abaixo lê o inteiro uma vez por segundo, converte-o na string de texto correspondente usando uma tupla e exibe a string no console. Pressione o botão do sensor de toque por pelo menos um segundo para parar o programa. Para que o programa fale as cores e exiba suas cadeias de texto, descomente a

linha destacada em azul. Isso aumentará ligeiramente o tempo entre as medições, pois incluí uma função `wait ()` para garantir que a fala não seja interrompida.

```
#!/usr/bin/env python3
# O script roda no brickman

from ev3dev.ev3 import *
from time   import sleep

# Conecta o sensor de cor EV3 a alguma porta
# e checa a conexão.

cl = ColorSensor()
assert cl.connected, "Connect a single EV3 color sensor to any sensor
port"

# conecta o sensor de toque a alguma porta
# e checa a conexão.
ts = TouchSensor();    assert ts.connected, "Connect a touch sensor to
any port"
# pode-se ter 2 statements na mesma linha usando o ponto e virgula

# Poe o sensor de cor em modo COL-COLOR.
cl.mode='COL-COLOR'

colors=('unknown','black','blue','green','yellow','red','white','brown
')
while not ts.value():    # para o programa acionando o sensor de toque
    print(colors[cl.value()])
    Sound.speak(colors[cl.value()]).wait()
    sleep(1)
Sound.beep()
```

Exemplo 8: Sensor de cor EV3 no modo RGB-RAW e sensor de toque

Este modo é especial, pois retorna 3 valores simultaneamente, representando as quantidades de luz refletida vermelha, verde e azul (qualquer cor pode ser obtida misturando as quantidades corretas de luz vermelha, verde e azul). Esses valores podem ser referenciados no código como valor (0), valor (1) e valor (2) respectivamente. O programa a seguir imprime os três valores detectados a

cada segundo. Pressione o botão do sensor de toque por pelo menos um segundo para parar o programa.

```
#!/usr/bin/env python3
# O script roda no Brickman

from ev3dev.ev3 import *
from time   import sleep

# Conecta o sensor de cor EV3 a alguma porta
# e checa a conexao.

cl = ColorSensor()
assert cl.connected, "Connect a single EV3 color sensor to any sensor
port"

# conecta o sensor de toque a alguma porta
# e checa a conexão.
ts = TouchSensor();  assert ts.connected, "Connect a touch sensor to
any port"
# tem-se 2 statements na mesma linha usando o ponto e virgula

# Poe o sensor de cor em modo RGB.
cl.mode='RGB-RAW'

while not ts.value():    # para o programa pressionando o sensor de
#toque
    red = cl.value(0)
    green=cl.value(1)
    blue=cl.value(2)
    print("Red: " + str(red) + ", Green: " + str(green) + ", Blue: " +
str(blue))
    sleep(1)
Sound.beep()
```

Modos de Sensor

Esta página fornece informações sobre os modos de sensor que acredito serem mais úteis para iniciantes. Para outros modos, consulte a documentação oficial na seção Lego da seção de recursos específicos do sensor . Os sensores EV3 são listados primeiro e depois os sensores NXT. Deve ser

óbvio que tipo de sensor está sendo referido (infravermelho, ultrassom, som, toque etc). Na tabela, pct significa percentual e + -2550 significa de -2550 a +2550. Dec significa casas decimais - veja o próximo parágrafo.

Aviso: Alguns modos do sensor retornam valores que incluem uma casa decimal (Dec = 1 nas tabelas abaixo, destacadas em amarelo). Esses valores incluem uma casa decimal, mas não um ponto decimal (!), Portanto, você deve tomar muito cuidado para lidar com esses valores corretamente. O sensor ultrassônico EV3 no modo US-DIST-CM, por exemplo, pode retornar o valor 184, o que na verdade não significa 184 cm como seria de esperar, mas 18,4 cm. Alternativamente, você pode fazer o que eu fiz com frequência neste site e simplesmente considerar que o sensor dos EUA mede em mm em vez de cm. Outro exemplo: se o sensor de luz NXT no modo REFLECT retorna o valor 50 (pct), isso não significa 50 por cento, isso significa que 5,0 por cento desde que o valor inclua uma casa decimal! Portanto, ao usar qualquer um desses modos, eu recomendo que você divida o valor por 10 conforme ele está sendo lido para obter o valor verdadeiro nas unidades corretas.

Sensores EV3

Modo	Descrição	Units	Dec	Values
TOQUE	Estado do botão	-	0	0 or 1
COL-REFLECT	Luz refletida. LED cor: vermelho	pct	0	0-100
COL-AMBIENT	Luz ambiente. LED cor: azul	pct	0	0-100
COL-COLOR	Cor. LED cor: branca	col	0	0-7*
RGB-RAW	Componentes de cor crua. LED cor: branca	-	0	**
US-DIST-CM	Medição contínua.	cm	1	0-2550
US-DIST-IN	Medição contínua.	inch	1	0-1003
GYRO-ANG	Angulo	deg	0	+-32767
GYRO-RATE	Velocidade rotacional	d/s	0	+-440
IR-PROX	Distancia (100% é aproximadamente 70cm/27in)	pct	0	100

* 0 = desconhecido, 1 = preto, 2 = azul, 3 = verde, 4 = amarelo, 5 = vermelho, 6 = branco, 7 = marrom

** retorna 3 valores, cada um 0-1023: valor (0): Vermelho, valor (1): Verde, valor (2): Azul.

Sensores NXT

Modo	Descrição	Units	Dec	Values
TOQUE	Estado do botão	-	0	0 or 1
REFLECT AMBIENTE	Intensidade de luz refletida. LED on. Intensidade de luz ambiente. LED off.	pct pct	1 1	0-1000 0-1000
DB	Nível de pressão sonora (flat weighting)	pct	1	0-1000
US-DIST-CM US-DIST-IN	Medição contínua Medição contínua	cm in	0 1	0-255 0-1000

Modo RGB do Sensor de Cor EV3

Qualquer cor pode ser obtida misturando as quantidades certas de luz vermelha, verde e azul. Assim, uma medição RGB retornará, na verdade, três valores (intensidade da luz vermelha, intensidade da luz verde, intensidade da luz azul). Isso significa que você pode medir qualquer cor, a partir de uma paleta de milhões, e não se limita a reconhecer uma das sete cores padrão, como no modo COL-COLOR. Especificamente, o modo RGB-RAW retorna 3 valores: value0: Red (0 a 1023), value1: Green (0 a 1023), value2: Blue (0 a 1023). Estes valores máximos são hipotéticos - os leds vermelho, verde e azul no sensor de cor não são brilhantes o suficiente para dar algo parecido com esses valores.

Aviso: A documentação oficial indica que o sensor de cor EV3 retorna valores chamados value0, value1 e value2, mas acredito que em seu código você precisa usar valor (0), valor (1) e valor (2). Aqui estão os valores reais que obtive usando blocos padrão de Lego a cerca de 5 mm do sensor (não tenho nenhum bloco marrom para testar o valor 7)

		R	G	B
1	**Preto**	29	40	7
2	**Azul**	38	87	61
3	**Verde**	34	122	12
4	**Amarelo**	238	163	15
5	**Vermelho**	192	44	8
6	**Branco**	306	324	110
7	**Marrom**	?	?	?

É claro que você tem que testar um intervalo de valores para cada cor, pois nunca obterá exatamente os mesmos valores que eu. Depois de saber como obter valores de RBG, você pode escrever código para definir um intervalo de valores RGB que corresponderia a, por exemplo, 'laranja' ou 50 tons de cinza ...

SOM

Reproduzir um sinal sonoro padrão

Use `sound.beep()`

Exemplos:

```
#!/usr/bin/env python3
# O script roda no Brickman
from ev3dev.ev3 import *  # necessario para som
Sound.beep()
```

Ou, para fazer com que o programa espere até que o bipe termine antes de continuar:

```
#!/usr/bin/env python3
# O script roda no Brickman
from ev3dev.ev3 import *
Sound.beep().wait()
```

Reproduzir um som único

Para tocar um som SINGLE de freqüência (Hz) e duração (milissegundos), use

`Sound.tone(`*`frequency, duration`*`)`

Exemplo:

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
# toca um unico tom de 1500Hz por 2 segundos
# espera finalizar o tom antes de continuar
Sound.tone(1500, 2000).wait()
```

Toque uma SEQUÊNCIA de tons

Para tocar uma SEQUÊNCIA de tons, use

`Sound.tone(tom_sequence)`

O parâmetro `tone_sequence` é uma lista de tuplas, onde cada tupla contém até três números. Você deve saber agora que uma tupla é como uma lista, exceto que uma lista pode ser modificada

e uma tupla não pode. O primeiro número é a frequência em Hz, o segundo é a duração em milissegundos e o terceiro é o atraso em milissegundos entre este e o próximo tom da sequência.

```
#!/usr/bin/env python3
# O script roda no Brickman

from ev3dev.ev3 import *  # necessario para som
# toca um tom de 3000Hz por 2 segundos e espera 0.4 segundos
# antes de tocar o proximo tom, 800Hz por 1.8 segundos
# seguidos por 2 segundos de delay
# espera pelo fim da sequencia tocando antes de continuar
Sound.tone([(3000, 2000, 400),(800, 1800, 2000)]).wait()
```

Reproduzir um arquivo de som WAV

Para reproduzir um arquivo de som WAV, use `Sound.play(wav_file)` Você não pode usar arquivos mp3 com o comando `Sound.play`, nem pode usar arquivos de som Lego no formato RSF ou RSO (mas veja abaixo).

Uma boa fonte para arquivos de som WAV gratuitos é o Wavsource.com. Eu tive um problema com um arquivo WAV que eu baixei de lá embora - eu recebi uma mensagem de erro 'underrun' quando eu corri meu script, o que sugere que o arquivo de som foi danificado. Se você receber a mesma mensagem, exclua esse arquivo de som e tente outro. Um arquivo wav de amostra 'cat_yowl.wav' de wavsource.com pode ser baixado usando o link na parte inferior desta página. Eu sugiro que você crie um diretório chamado 'sons' no diretório 'robô'. Se você está usando o MobaXTerm free edition, é fácil criar um novo diretório e fazer o upload de um arquivo usando os botões apropriados. Faça o upload do arquivo 'cat_yowl.wav' em seu diretório de sons e execute o seguinte script (supostamente no diretório do robô):

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep

Sound.play('sounds/cat_yowl.wav').wait()
Sound.beep()

sleep(4)

Sound.play('sounds/cat_yowl.wav')
```

```
Sound.beep()

sleep(4)
```

Observe que o primeiro yowl de gato toca antes do som do bipe, mas o segundo toca ao mesmo tempo (ou mesmo depois) do bipe porque não há `.wait()` para fazer a execução do programa pausar até que o som tenha terminado de tocar antes de continuar.

Eu disse acima que `Sound.play()` só pode reproduzir sons que estão no formato WAV e, portanto, não pode reproduzir sons Lego no formato RSF ou RSO. Eu converti todos os sons do Lego para o formato WAV e os disponibilizei através de um link na parte inferior desta página. Todos os sons estão na mesma pasta, não separados em pastas diferentes como no EV3-G. Todos os scripts neste site que fazem uso desses sons assumem que você colocou todos eles em uma pasta 'sons' dentro da pasta 'robot'. Portanto, o endereço dos sons será / home / robot / sounds /. Isso é semelhante à maneira como os meus scripts assumem que você colocou todas as imagens Lego no formato BMP em uma pasta chamada 'fotos' na pasta 'robô'.

Texto para fala

Use `Sound.speak(text)`

Exemplo na parte inferior do exemplo combinado na próxima página.

Exemplo combinado

Um exemplo de um programa para demonstrar toques de reprodução, tons e conversão de texto em fala:

```
#!/usr/bin/env python3
# O script roda no Brickman

from time import sleep   # needed for sleep
from ev3dev.ev3 import *  # needed for Sound

#toca um beep padrao
Sound.beep()

sleep(2)  # pausa por 2 segundos

# toca um unico tom de 1500 Hz por 2 segundos
```

```
# espera finalizar o toque do tom antes de continuar
Sound.tone(2000, 1500).wait()

sleep(2)  # pausa de 2 segundos

# para tocar uma sequencia de tons, use uma LISTA de TUPLAS
# O terceiro argumento em cada tuple é um delay em ms
# antes que seja tocado o proxino tom.
# toca um tom de 3000 Hz por 2 segundos, então espera 0.4 segundos
# antes de tocar o proximo tom, 800Hz por 1.8 segundos
# seguido por um delay de 2 segundos
# espera finalizar a sequencia antes de continuar
Sound.tone([(3000, 2000, 400),(800, 1800, 2000)]).wait()

# toca um tom de 500 Hz por um Segundo e entao espera 0.4 segundos
# antes de tocar o proximo tom
# toca o tom 3 vezes
Sound.tone([(500, 1000, 400)] * 3).wait()

#texto para fala
Sound.speak('Hello, my name is E V 3!').wait()
```

MATEMÁTICA E TEMPO

Abs() and Round()

Ao contrário das outras funções matemáticas mencionadas mais abaixo na página, estas duas são construídas no interpretador Python e, portanto, estão sempre disponíveis, portanto, você não precisa importar a biblioteca matemática para poder usar essas duas funções. (Você precisa importar a biblioteca de matemática antes de usar as outras funções matemáticas descritas nesta página.)

`abs(x)` retorna o valor absoluto de um número. O argumento pode ser um número inteiro ou um número de ponto flutuante. Exemplo: `print(abs(-34.6))` irá imprimir 34.6.

`round(number)` retorna (como um inteiro) o inteiro mais próximo de sua entrada.

`round(number, ndigits)` retorna o número do valor do ponto flutuante arredondado para ndigits dígitos após o ponto decimal. Exemplo: `print(redondo(2.78, 1))` irá imprimir 2.8. O valor de retorno é do mesmo tipo que o número.

Divisão e Módulo

Além da divisão 'normal' com '/', o Python pode fazer uma divisão inteira com '//'. A divisão inteira arredonda o resultado para o inteiro mais próximo. Ao arredondar para baixo, quero dizer arredondar na direção negativa, o que nem sempre é o mesmo que truncar o resultado. O arredondamento também é chamado de 'piso'. Veja o script abaixo.

O Python também possui um operador de módulo '%' que, para números positivos, fornece o restante. Por exemplo, 15% 4 = 3. Observe que o Python calcula o módulo de maneira diferente de C.

```
#!/usr/bin/env python3
from time import sleep

print(11/3)       # 3.6666666666666665
print(11//3)      # 3
print(-11//3)     # -4
print(11//-3)     # -4
print(11%3)       # 2
sleep(6) # da tempo suficiente para o rsultado ser lido
```

Funções trigonométricas

É importante lembrar que as funções trigonométricas do Python usam radianos, não graus. Uma revolução = 2 * pi radianos. Um radiano = aprox. 57,3 ° Você pode converter graus em radianos com `math.radians(x)` e radianos para graus com `math.degrees(x)`. Por exemplo, isso imprimirá o co-seno de 45 °:

```
#!/usr/bin/env python3

from time import sleep
import math

print (math.cos(math.radians(45)))
sleep(6) # give enough time for result to be read
```

Números aleatórios

A biblioteca `random` deve ser importada antes que as funções de números aleatórios possam ser usadas. No Python, inteiros aleatórios entre a e b inclusive podem ser gerados com `randint(a,`

b). Por exemplo, para gerar um número inteiro aleatório entre 0 e 6 inclusive, use `randint(0,6)`. Veja um exemplo de um script usando `randint()` nesta página.

Um float aleatório entre 0 e 1 (maior ou igual a zero, mas sempre menor que 1) pode ser gerado com `random()`.

Este script irá imprimir um número inteiro aleatório entre -100 e +100 e uma flutuação aleatória entre 0 e 1.

```
#!/usr/bin/env python3

from time import sleep
import random

print (random.randint(-100,100)
print (random.random())
sleep(6) # da tempo suficiente para o resultado ser lido no Brickman
```

Tempo

Se você quiser medir quanto tempo leva para concluir uma ação, talvez a função `time.time()` seja útil. Ele retorna a hora como um número de ponto flutuante expresso em segundos desde a "época". A "época" está em algum momento da história que não nos interessa porque estaríamos interessados apenas em mudanças no tempo. Não esqueça de importar a biblioteca de tempo se quiser usar esta função.

O exemplo a seguir calcula quanto tempo o usuário leva para adicionar dois inteiros aleatórios e também indica se a resposta foi certa ou errada. Devemos ter cuidado para não escrever código que irá 'falhar' se um usuário inserir uma resposta que não seja um inteiro. O script abaixo demonstra uma técnica muito boa (usando `try: ... except ValueError: ... else: ...`) para validar a entrada do usuário, neste caso verificando se o usuário digitou um inteiro. O script abaixo provavelmente não pode ser executado a partir do Brickman, pois ele precisa de entrada de teclado, portanto, a primeira e a última linha provavelmente são supérfluas.

```
#!/usr/bin/env python3
from time import sleep, time
import random

starttime=time()
int1=random.randint(20,100)
```

```
int2=random.randint(20,100)
sum=int1+int2
# Adquire entrada do usuario
while True:
    try:
        ans = int(input('What is '+str(int1)+ ' + ' + str(int2) + ' ?
'))
    except ValueError:
        print("Sorry, I didn't understand that.")
        #melhor tentar de novo... retorna ao começo do loop
        continue
    else:
    # entrada foi analisada com sucesso para que possamos sair do loop
        break
timetaken=round(time()-starttime,1)
verdict=['wrong.', 'correct.'][ans==sum]
print('You took '+str(timetaken)+'s and your answer is '+verdict)
if verdict=='wrong.':
    print('The correct answer is '+str(sum)+'.')

sleep(5) # da tempo para o resultado ser lido no Brickman
```

A linha mais interessante pode ser a linha destacada em amarelo. Nessa linha, uma lista é definida contendo duas cadeias de texto "wrong". e 'correct'. Uma dessas cadeias de texto será copiada para a variável `veredict` de acordo com o valor do índice [`ans == sum`]. Se `ans` é igual a `sum`, isso dá `True`, que é considerado equivalente a 1, então a string de texto com índice 1 ('`correct`') será colocada em veredicto. Se `ans` não é igual a `sum`, então temos `False` que é equivalente a zero, então a string de texto com o índice 0 ('`wrong`') será colocada em veredicto.

MULTITHREADING

Um 'thread' é uma parte do código do programa que pode ser executado independentemente e ao mesmo tempo que outras partes do programa (algumas linguagens de programação chamam 'tarefas' de threads). Todo programa tem pelo menos um thread, o thread principal. Se o programa 'spawns' outros threads que são executados simultaneamente com o thread principal, isso é conhecido como 'multithreading'. Por exemplo, você poderia criar um thread principal que controla os motores, enquanto um thread diferente pode observar os sensores ou a entrada do usuário.

No primeiro exemplo abaixo, um tópico chamado '`playtone`' é iniciado - ele toca vinte tons enquanto o script principal espera que o botão do sensor de toque seja 'batido' (pressionado e liberado) 5 vezes. A ideia é que bater cinco vezes fará com que o tom de execução seja interrompido. (Se os tons pararem antes que os cinco solavancos tenham sido concluídos, a execução do programa pára depois que os solavancos (e o sinal sonoro) forem concluídos, como seria de se esperar.) Mas se você tentar executar o script, observará que neste caso dos tons NÃO é interrompido. As linhas-chave para configurar o encadeamento estão em negrito. Observe que na linha

```
t = Thread(target = playtone)
```

não há parênteses após o nome do thread 'playtone'.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
from threading import Thread
ts = TouchSensor()
assert ts.connected, "Connect a touch sensor to any sensor port"

def playtone():
    for j in range(0,20):             # Do twenty times.
        Sound.tone(1000, 200).wait()  #1000Hz for 0.2s
        sleep(0.5)

t = Thread(target=playtone)
t.start()
for i in range(0,5):        # faz 5 vezes.
    while ts.value() ==0:   # enquanto o botão não e pressionado
        sleep(0.01)         # faz nada alem de esperar
    while ts.value() ==1:   # enquanto o botao e pressionado
        sleep(0.01)         # faz nada alem de esperar
Sound.beep()
```

No script acima, o programa termina somente depois que a sequência de sons e o pressionamento dos botões tiverem terminado. Você pode passar um ou mais argumentos para o encadeamento como mostrado abaixo. Observe que, no caso de uma tupla de elemento único, a vírgula final é necessária.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
from threading import Thread
ts = TouchSensor()
assert ts.connected, "Connect a touch sensor to any sensor port"

def playtone(frequency):
    for j in range(0, 20):            # faz 20 vezes.
        Sound.tone(frequency, 200).wait()  # toca um tom por 0.2s
        sleep(0.5)

t = Thread(target=playtone, args=(1000,))
t.start()

for i in range(0, 5):        # faz 5 vezes.
    while ts.value() == 0:   # enquanto o botão não e pressionado
        sleep(0.01)          # faz nada alem de esperar
    while ts.value() == 1:   # enquanto o botão não e pressionado
        sleep(0.01)          # faz nada alem de esperar
Sound.beep()
```

Confira o próximo exemplo em que os tons tocam em um loop infinito e os cinco pressionamentos de botão destinam-se a interromper a reprodução dos tons. Como você aprendeu acima, o script abaixo não permitirá que o botão pressione para interromper o programa, pois o programa só será interrompido quando as duas partes forem concluídas e a sequência de tons nunca for concluída. (Você pode forçar o programa a sair pressionando Ctrl-C se o script foi executado a partir do SSH ou pressionando o botão Voltar se o script foi executado no Brickman).

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
from threading import Thread
ts = TouchSensor()
assert ts.connected, "Connect a touch sensor to any sensor port"

def playtone():
    while True:              # Do forever.
        Sound.tone(1000, 200).wait()  #1000Hz por 0.2s
```

```
        sleep(0.5)

t = Thread(target=playtone)
# t.setDaemon(True)   # see notes below
t.start()
for i in range(0,5):        # faz 5 vezes.
    while ts.value() ==0:   # enquanto o botão não e pressionado
        sleep(0.01)         # faz nada alem de esperar
    while ts.value() ==1:   # enquanto o botão não e pressionado
        sleep(0.01)         # faz nada alem de esperar
Sound.beep()
```

Então qual é a solução? Eu procurei maneiras de incluir algo no script principal que terminaria o encadeamento do playtone - acontece que isso não pode ser feito em Python (e isso é provavelmente uma coisa boa, porque pode ter consequências confusas, mas veja a nota na parte inferior desta página).

Mas é possível em python marcar (flag) um encadeamento como um encadeamento 'daemon' (background), o que é muito útil aqui porque todo o programa Python sai quando apenas encadeamentos do daemon são deixados. Para sinalizar um thread chamado 't' como um daemon thread do `t.setDaemon (True)`. Assim, todos os scripts acima poderiam ser feitos para se comportarem como desejado, incluindo esta linha logo após a linha t = Thread (target = playtone). Se você não comentar a linha destacada em verde no script acima, verá o script se comportando corretamente, com o encadeamento do daemon interrompido assim que o encadeamento principal parar.

Eu encontrei uma solução alternativa, mas muito menos boa depois de ter assistido a este vídeo(https://www.youtube.com/watch?v=XndHx3EaDgA&feature=youtu.be). No script abaixo, os toques serão reproduzidos indefinidamente até que o sensor de toque seja batido cinco vezes, após o qual o encadeamento do tom de reprodução terminará nitidamente. Preste atenção nas linhas em negrito.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
from threading import Thread
ts = TouchSensor()
assert ts.connected, "Connect a touch sensor to any sensor port"
```

```
def playtone():
    global run # podemos usar a mesma variavel global 'run'
    # que criamos no script principal acima
    while run:
        Sound.tone(1000, 200).wait()  #1000Hz por 0.2s
        sleep(0.5)

global run
run=True
t = Thread(target=playtone)
t.start()
for i in range(0,5):
    while ts.value() ==0:  # enquanto o botao nao e pressionado
        sleep(0.01)        # faz nada alem de esperar
    while ts.value() ==1:  # while button is pressed
        sleep(0.01)        # faz nada alem de esperar
Sound.beep()
run=False # Para que o loop no encadeamento do playtone pare o loop
```

No script acima, a parte principal cria uma variável global chamada '`run`' que recebe o valor `True`. No encadeamento do tom de reprodução, o laço se repete contanto que a variável ' `run` ' seja igual a `True`. A última linha da parte principal dos conjuntos de scripts é executada para ser igual a False, de forma que o loop no encadeamento do `playtone` também pare de ser executado. É um pouco chato ter que parar o tópico de maneira tão complexa, mas é necessário. Os programadores profissionais preferem não "desperdiçar" uma variável global da maneira mostrada no script acima - o vídeo sugere como eles podem lidar com o problema.

Note que também é possível fazer o script acima '`backwards`', com o 'thread' parando um loop na parte principal do script, como mostrado abaixo. O thread `'buttonStop'` é iniciado - ele espera que o sensor de toque seja pressionado 5 vezes e, uma vez que isso tenha ocorrido, ele define a variável global '`run`' como False, o que faz com que o loop na parte principal do script pare de fazer loop.

```
#!/usr/bin/env python3
from ev3dev.ev3 import *
from time import sleep
from threading import Thread
ts = TouchSensor()
assert ts.connected, "Connect a touch sensor to any sensor port"
```

```
def buttonStop():
    global run
    for i in range(0,5):
        while ts.value() ==0:  # enquanto o botao nao e pressionado
            sleep(0.01)  # faz nada alem de esperar
        while ts.value() ==1:  # enquanto o botao e pressionado
            sleep(0.01)  # faz nada alem de esperar
    Sound.beep()
    run=False # Para que o loop na parte principal pare
# Cria uma variável global 'run' que será definida como
# False quando queremos que o loop abaixo pare de fazer loop.
global run
run=True
t = Thread(target=buttonStop)
t.start()

while run:  # o mesmo que while run == True
    Sound.tone(1000, 200).wait()  #1000Hz for 0.2s
    sleep(0.5)
```

É claro que o script acima pode ser simplificado, de modo que apenas um único ressalto do sensor de toque seja necessário para interromper o loop de tom. Seria até mesmo possível parar o loop com um simples toque no sensor de toque, em vez de um '`bump`' (pressionar e soltar). Mas por que usar um sensor quando podemos usar os botões embutidos no bloco EV3? O script abaixo tem uma parte principal que usa um loop para fazer o robô avançar somente quando o nível de luz está acima de 15 e um thread que espera que qualquer botão EV3 seja pressionado e então emite um bipe e faz com que o loop '`move-when-bright`' para parar o loop. Se você brincar com este script, ficará surpreso ao notar que, mesmo que o bipe aconteça, assim que você pressionar qualquer botão, pode levar mais alguns segundos até que o script realmente pare. Isso é lógico se você pensar nisso - não estamos emitindo uma instrução para terminar instantaneamente a parte principal do código, apenas para fazer com que o loop não se repita, então o programa tem que alcançar a parte inferior do loop antes que o programa termine e Nesse caso, de acordo com a variação do nível de luz, isso pode levar muitos segundos. Para evitar esse problema, você poderia, neste caso, mudar

`while cl.value()<=15:` em `while cl.value()<=15 and run:`

e mudar

`while cl.value()>15:` em `while cl.value()>15 and run:`

para que o script espere por uma mudança no nível de luz somente quando 'run' for True. Observe que

`while cl.value()<=15 and run:`

é o mesmo que

`while (cl.value()<=15) and (run==True):`

```
#!/usr/bin/env python3

from ev3dev.ev3 import *
from time import sleep
from threading import Thread
mB = LargeMotor('outB')
mC = LargeMotor('outC')
cl = ColorSensor()
assert cl.connected, "Connect a color sensor to any sensor port"
cl.mode='COL-AMBIENT'
btn=Button()

def buttonStop():
    global run
    while btn.any()==False:  # enquanto nenhum botao e pressionado
        sleep(0.01)  # faz nada alem de esperar
    run=False  # passa run para False para parar o loop acima.
    Sound.beep()

# Cria uma variável global 'run' que será definida como
# False quando queremos que o loop abaixo pare de fazer loop.
global run
run=True
t = Thread(target=buttonStop)
t.start()

while run:  # Loop until run é definido como False pelo buttonStop
#thread.
    # Aguarda até que o nível de luz ambiente suba acima de 15
    # então inicia o robo.
    while cl.value()<=15:
        sleep(1)  # Não faz nada enquanto esperamos.
```

```
    mB.run_forever(speed_sp=500)
    mC.run_forever(speed_sp=500)

    # Aguarda até que o nível de luz ambiente seja menor ou igual a
    # 15,entao para o robo.
    while cl.value()>15:
        sleep(1)  # faz anada enquanto esperamos.
    mB.stop()
    mC.stop()
```

Como os threads do Python são complicados, vale sempre a pena considerar se eles podem ser evitados. O script abaixo tem a mesma funcionalidade que o acima, mas evita o uso de um thread. Ele provavelmente pode ser simplificado ainda mais usando `if .... else` em vez dos dois loops `while` que verificam o nível de luz - deixarei isso para você.

```
#!/usr/bin/env python3

# O script abaixo faz o robô avançar quando a luz ambiente
# é brilhante. Pode ser parado pressionando qualquer botão EV3 por
# pelo menos 0,5 segundos e isso é possível sem o uso de. Um thread
# Em vez disso, o código verifica se há um botão pressionado sempre
# verifica o nível de luz e se um botão for pressionado é detectado
# então os comandos de suspensão não são executados e
# o loop principal chega rapidamente ao fim. O loop principal não
# repita porque pode repetir somente quando nenhum botão é
# pressionado.

from ev3dev.ev3 import *
from time import sleep
mB = LargeMotor('outB')
mC = LargeMotor('outC')
cl = ColorSensor()
assert cl.connected, "Connect a color sensor to any sensor port"
cl.mode='COL-AMBIENT'  # define o modo para medir o nível de luz
# ambiente
btn=Button()

while btn.any()==False:  # Loop while no button is pressed.
    # Aguarda até que o nível de luz ambiente suba acima de 15
```

```
    # então, inicia o robô.
    while cl.value()<=15 and btn.any()==False:
        sleep(0.5)
    mB.run_forever(speed_sp=500)
    mC.run_forever(speed_sp=500)

    # Aguarda até que o nível de luz ambiente seja <= a 15,
    # então para o robô.
    while cl.value()>15 and btn.any()==False:
        sleep(0.5)
    mB.stop()
    mC.stop()
Sound.beep()
```

De um modo geral, multi-threading é um tema bastante complexo. Para realmente entender o multithreading, recomenda-se um estudo extra. Para dominar completamente o multithreading de Python, você precisará entender os conceitos de bloqueios, filas, daemons e `join ()`, nem todos explicados neste site. Existem muitos vídeos sobre o assunto na internet que podem ajudar bastante no entendimento desse assunto.

CONTROLE DO TECLADO

Definir 'robô'. Não é fácil, é? Mas provavelmente podemos concordar que esperamos que os robôs sejam capazes de realizar tarefas físicas com limitada intervenção humana e, portanto, um bom grau de autonomia. Se houver uma intervenção humana muito significativa, como quando um piloto no solo controla um plano de modelo controlado por rádio em tempo real, você provavelmente se recusaria a chamar esse avião de robô. Mas se um carro do Google ou um drone equipado com GPS pode encontrar seu caminho para seu destino de forma autônoma depois de ter sido dado o destino antes da viagem começar, então você aceitaria que isso é um comportamento robótico, não é? Mas a intervenção humana limitada não significa necessariamente zero intervenção humana. Certamente podemos dizer que gostaríamos que nossos robôs escravos domésticos do futuro respondessem a todos os tipos de comandos humanos durante o dia: "Retire o lixo", "Troque as fraldas do bebê" etc., então surge a pergunta: como podemos interagir com um robô enquanto está executando um programa? Já sabemos que podemos interagir com ele pressionando os botões do bloco, por exemplo, mas como podemos interagir com ele remotamente? Seria muito útil poder interagir com o robô a partir do PC, por exemplo, usando o teclado do PC. Isso é exatamente o que este primeiro programa faz: o programa rodando no brick pode detectar, através da conexão SSH entre o brick e o PC, quais caracteres do teclado do PC

estão sendo pressionados, e eles são impressos na janela do terminal no PC. Não é muito impressionante ver o programa rodando desde que o teclado pressiona e a exibição dos personagens acontece no PC, mas é impressionante perceber que essas teclas estão sendo detectadas por um programa rodando no bloco, e que o bloco pode estar do outro lado da sala, conectado ao PC somente por meio de uma conexão sem fio.

'Get character' Exemplo de interação de teclado

Este programa em execução no tijolo pode detectar pressionamentos de teclado no PC, fazendo uso da conexão SSH entre o PC e o bloco. Você pode fazer com que o programa saia a qualquer momento pressionando a tecla 'q'.

```
#!/usr/bin/env python3
# O script roda no Brickman

import termios, tty, sys

#==============================================

def getch():
   fd = sys.stdin.fileno()
   old_settings = termios.tcgetattr(fd)
   try:
      tty.setraw(fd)
      ch = sys.stdin.read(1)
   finally:
      termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)

   return ch
#==============================================

while True:
   k = getch()
   print(k)
   if k == 'q':
      exit()
```

Outro exemplo:

```
import termios, tty, sys

#==============================================

def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    tty.setcbreak(fd)
    ch = sys.stdin.read(1)
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)

    return ch

#==============================================
while True:
   k = getch()
   print(k)
   if k == 'q':
      exit()
```

Controle de teclado de um robô

Agora para o programa completo da página que o primeiro programa foi extraído, que você certamente deve visitar. O programa usa a mesma função `getch()` que o programa mais curto acima para detectar quais caracteres do teclado do PC foram pressionados e usa certos caracteres detectados para controlar um robô móvel. Supõe-se que as rodas motrizes do robô estejam ligadas a grandes motores conectados às portas B e C e que um motor médio esteja conectado à porta A - esse motor médio pode ser conectado a um mecanismo de disparo por bola, por exemplo. Deveria ser óbvio, estudando o final do programa, quais teclas estão ativas e o que cada uma delas faz. Como antes, pressionar a tecla 'q' encerrará o programa.

```
#!/usr/bin/env python3
# O script roda no Brickman

import termios, tty, sys
from ev3dev.ev3 import *

# conecta os motores grandes às portas B e C e um médio à porta A
motor_left = LargeMotor('outB')
```

```
motor_right = LargeMotor('outC')
motor_a = MediumMotor('outA')

#==============================================

def getch():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    tty.setcbreak(fd)
    ch = sys.stdin.read(1)
    termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)

    return ch

#==============================================

def fire():
   motor_a.run_timed(time_sp=3000, speed_sp=600)

#==============================================

def forward():
   motor_left.run_forever(speed_sp=450)
   motor_right.run_forever(speed_sp=450)

#==============================================

def back():
   motor_left.run_forever(speed_sp=-450)
   motor_right.run_forever(speed_sp=-450)

#==============================================

def left():
   motor_left.run_forever(speed_sp=-450)
   motor_right.run_forever(speed_sp=450)

#==============================================

def right():
```

```
   motor_left.run_forever(speed_sp=450)
   motor_right.run_forever(speed_sp=-450)

#==============================================

def stop():
   motor_left.run_forever(speed_sp=0)
   motor_right.run_forever(speed_sp=0)

#==============================================
while True:
   k = getch()
   print(k)
   if k == 'w':
      forward()
   if k == 's':
      back()
   if k == 'a':
      left()
   if k == 'd':
      right()
   if k == 'f':
      fire()
   if k == ' ':
      stop()
   if k == 'q':
      exit()
```

O programa 'Controle de Teclado' acima funciona corretamente, sem erros, somente se eu usar todos os motores declarados antes de sair do programa pressionando 'q', caso contrário, os erros são reportados e às vezes os motores continuam girando depois que o programa foi encerrado. Por exemplo, se eu iniciar o programa, pressione 'f' para executar o motor médio por 3 segundos e, em seguida, pressione 'q' para sair do programa. Recebo erros porque os motores B e C foram declarados e não utilizados. Existe um bug? Talvez...