

# Apostila Raspberry Pi: Básico

Matheus Ramos Soares

UTFPR - UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ  
LPSA - LABORATÓRIO DE PROCESSAMENTO DE SINAIS E APLICAÇÕES



CORNÉLIO PROCÓPIO - PR, 2018



## LISTA DE FIGURAS

1	Modelos de Raspberry Pi . . . . .	8
2	Pinos gpio . . . . .	10
3	Utilização do comando pinout no terminal do Raspberry Pi . . . . .	12
4	Pinos GPIO em diversos modelos de Raspberry Pi . . . . .	12
5	Download e descompactação da .ISO do Raspbian . . . . .	14
6	Abra a imagem do Raspbian no Win32DiskImager . . . . .	15
7	Formatação do cartão de memória FAT32 . . . . .	15
8	Escrever .ISO pelo Win32 Disk Imager . . . . .	16
9	Processo de escrita no cartão de memória . . . . .	16
10	Área inicial Raspbian . . . . .	17
11	LXTerminal Raspberry Pi . . . . .	17
12	Comando para desligar Raspberry Pi . . . . .	18
13	Comando para reiniciar Raspberry Pi . . . . .	18
14	Comando para atualizar Raspberry Pi . . . . .	18
15	Comando para atualiza python . . . . .	18
16	Gerenciador de arquivos Midnight Commander . . . . .	19
17	Área inicial Raspbian . . . . .	22
18	Terminal . . . . .	22
19	Criação diretório . . . . .	22
20	Visualização do conteúdo do diretório . . . . .	23
21	Direcionamento de diretório . . . . .	23
22	Criação do arquivo . . . . .	24
23	Utilização do comando nano . . . . .	24
24	Editor de texto . . . . .	25
25	Código Hello Word! . . . . .	25
26	Execução do código . . . . .	26
27	Círcuito projeto 2 . . . . .	27
28	Criação dos arquivos 2_on.py e 2_off.py . . . . .	27

29	Código do arquivo 2_on.py . . . . .	28
30	Código do arquivo 2_off.py . . . . .	28
31	Arquivo 2_on.py rodando . . . . .	28
32	Arquivo 2_on.py rodando . . . . .	29
33	Círculo projeto 3 . . . . .	30
34	Criação e acesso do arquivo pisca.py . . . . .	31
35	Código parte 1 . . . . .	31
36	Código parte 2 . . . . .	32
37	Código rodando . . . . .	32
38	Círculo projeto 4 . . . . .	33
39	Diretório de códigos . . . . .	34
40	Criação do arquivo em python . . . . .	34
41	Acessar o arquivo led_eterno.py . . . . .	34
42	Código led_eterno.py . . . . .	34
43	Código led_eterno.py rodando . . . . .	35
44	Círculo projeto 5 . . . . .	36
45	Diretório dos códigos . . . . .	37
46	Criação do arquivo em python . . . . .	37
47	Acessar o arquivo led_controle.py . . . . .	37
48	Código led_controle.py . . . . .	38
49	Inicialização do código . . . . .	39
50	Inicialização do código . . . . .	39
51	Código rodando - parte 1 . . . . .	39
52	Código rodando - parte 2 . . . . .	40
53	Círculo projeto 6 . . . . .	41
54	Diretório gpio_python_code . . . . .	41
55	Criar e abrir o arquivo botao.py . . . . .	41
56	Código para configurar botão . . . . .	42
57	Rodando o código botao.py . . . . .	42

58	Resposta ao apertar botão táctil . . . . .	43
59	Círculo projeto 7 . . . . .	44
60	Sensor de temperatura e umidade DHT11 . . . . .	44
61	Página inicial Github . . . . .	45
62	Página Github Adafruit Python DHT . . . . .	46
63	Download, instalação da biblioteca Adafruit DHT 11 . . . . .	46
64	Instalação da biblioteca . . . . .	47
65	Sair do diretório Adafruit_Python_DHT . . . . .	47
66	Entrar no diretório gpio_python_code e criação do arquivo dht11.py . . . . .	48
67	Abrir o arquivo dht11.py . . . . .	48
68	Código dht11.py . . . . .	49
69	Inicialização do código dht11.py . . . . .	50
70	Leitura de valores de temperatura e umidade . . . . .	50
71	Círculo de teste ADS115. . . . .	51
72	Configuração Software Raspberry versão mais atual. . . . .	52
73	Configuração Software Raspberry versão mais antiga. . . . .	52
74	Habilitação I2C. . . . .	53
75	Habilitar interface ARM I2C. . . . .	53
76	Habilitar módulo kernel I2C como padrão. . . . .	54
77	Instalação da biblioteca Adafruit ADS1115. . . . .	55
78	Tabela com canais do ADS e seus valores. . . . .	56
79	Círculo de teste ADS115 com potenciômetro. . . . .	56
80	Círculo projeto osciloscópio com gerador de função. . . . .	58
81	Código osciloscópio com gerador de função. . . . .	59
82	Código osciloscópio sem nenhuma entrada no canal 0. . . . .	60
83	Código osciloscópio com potênciometro. . . . .	61
84	Código osciloscópio com onda quadrada gerada pelo gerador de função. . . . .	61
85	Código osciloscópio com onda triangular gerada pelo gerador de função. . . . .	62

# Conteúdo

<b>1 INTRODUÇÃO</b>	<b>7</b>
<b>2 CARACTERÍSTICAS GERAIS</b>	<b>8</b>
2.1 Hardware . . . . .	9
2.2 Processador . . . . .	9
2.3 Overclocking . . . . .	9
2.4 RAM . . . . .	9
2.5 Networking . . . . .	10
2.6 Video . . . . .	10
2.7 GPIO . . . . .	10
2.8 Software . . . . .	12
2.9 Aplicações . . . . .	13
<b>3 PROJETOS</b>	<b>14</b>
3.1 Instalando Raspbian . . . . .	14
3.2 Comandos básicos . . . . .	17
3.3 Projeto 1 - Hello Word . . . . .	22
3.4 Projeto 2 - Acende e apaga LED . . . . .	27
3.5 Projeto 3 - LED pisca-pisca n vezes . . . . .	30
3.6 Projeto 4 - LED pisca-pisca eterno . . . . .	33
3.7 Projeto 5 - LED pisca-pisca controlado . . . . .	36
3.8 Projeto 6 - Configurar botão . . . . .	41
3.9 Projeto 7 - Monitoramento de temperatura e umidade com DHT11 . . . . .	44
3.10 Projeto 8 - Conversor ADS 1115 . . . . .	51
3.11 Projeto 9 - Osciloscópio . . . . .	58
<b>REFERÊNCIAS</b>	<b>63</b>

# 1 INTRODUÇÃO

O Raspberry Pi é um computador do tamanho de um cartão de crédito de baixo custo que se conecta a um monitor de computador ou TV, e usa um teclado e mouse padrão. É um pequeno dispositivo capaz que permite que pessoas de todas as idades explorem computação e aprenda a programar em linguagens como Scratch e Python. Ele é capaz de fazer tudo o que se espera de um computador desktop, desde navegar na Internet e reproduzir vídeo de alta definição, até fazer planilhas, processamento de texto e jogos.

Além disso, o Raspberry Pi tem a capacidade de interagir com o mundo exterior e tem sido usado em uma ampla gama de projetos de fabricantes digitais. Uma das principais aplicações tem sido ensinar programação para crianças, estimulando que elas aprendam mais sobre o universo dos computadores.

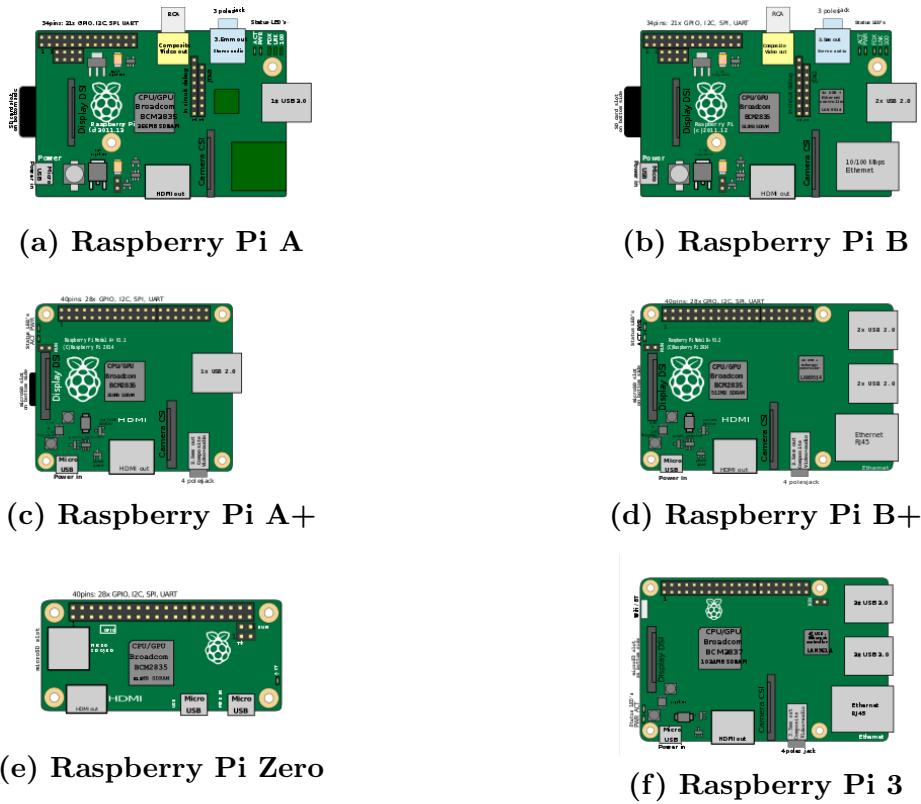
Nesta apostila será abordado alguns aspectos básicos do Raspberry Pi, trazendo uma visão do seu funcionamento e alguns projetos simples para ser implementados.

## 2 CARACTERÍSTICAS GERAIS

Em 2006, Eben Upton e sua equipe desenvolveram os primeiros conceitos para o Raspberry Pi, baseados no Atmel ATmega. A ideia de atrair jovens interessados no mini-computador já fazia parte do programa. Em 2009, os membros oficialmente estabeleceram a Raspberry Pi Foundation. Em agosto de 2011, a série alpha de aproximadamente 50 placas deixou a produção.

De 2011 para cá a fundação Raspberry Pi cresceu e upgrades para melhorar o minicomputador foram feitos, assim nasceu diversos modelos de Raspberry Pi como mostra a Figura 1

Figura 1: Modelos de Raspberry Pi



Apesar dos diversos modelos, eles possuem diversas características em comuns:

- Alimentação a partir de porta  $\mu$ USB;
- Saída de vídeo HDMI;
- Saída/entrada de som;
- Memória de massa: SD/ $\mu$ SD;
- Ligação de rede RJ-45;

- Portas USB;
- RAM 128-512MB.

## 2.1 Hardware

Existem atualmente dois modelos: Modelo A e Modelo B. A grande diferença entre os dois modelos é que o Modelo B possui um controlador Ethernet e duas portas USB, enquanto que o Modelo A possui apenas uma porta USB e nenhuma porta de Ethernet.

## 2.2 Processador

A Broadcom SoC usado no Raspberry Pi é equivalente a um chip usado em um smartphone antigo (Android ou iPhone). Ao operar em 700 MHz por padrão, o Raspberry Pi oferece um desempenho mais ou menos equivalente aos 0.041 GFLOPS. No nível CPU, o desempenho é semelhante a um Pentium II 200 MHz de 1997-1999. Os recursos gráficos do Raspberry Pi é aproximadamente equivalente ao nível de desempenho do Xbox de 2001, operando em 700 MHz por padrão, não aquece o suficiente para precisar de um dissipador de calor ou de frio especial. O SoC é empilhada embaixo do chip de memória RAM, portanto, apenas a sua borda é visível.

## 2.3 Overclocking

A maioria dos dispositivos Raspberry Pi podem fazer overclock para 800 MHz e alguns até mesmo superior a 1000 MHz. No Raspbian distro linux as opções de overclock na inicialização pode ser feita por um comando de software executando ”sudo Raspi-config” sem anular a garantia. Nestes casos, a PI transporta automaticamente o overclocking, chegando a 85 °C o chip. Nesse caso, pode-se tentar colocar um dissipador de calor de tamanho apropriado para manter nele o chip de aquecimento muito acima de 85 °C.

## 2.4 RAM

No primeiro modelo B foi liberado 256 MB (e Modelo A), três divisões diferentes eram possíveis. A divisão padrão era 192 MB (CPU RAM), que deve ser suficiente para decodificação de vídeo 1080p, ou para simples 3D, mas provavelmente não para os dois juntos. 224 MB era apenas em Linux, com apenas um framebuffer 1080p.

## 2.5 Networking

Apesar de não possuir a porta Ethernet, o Modelo A pode ser conectado a internet através de um adaptador USB de Ethernet ou Wi-Fi. No modelo B da porta Ethernet é fornecido por um adaptador Ethernet USB embutido.

## 2.6 Video

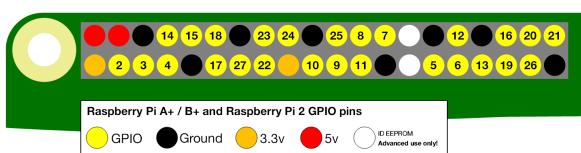
O controlador de vídeo é capaz de atribuir as seguintes resoluções de vídeo: 640x350 EGA; 640x480 VGA; 800x600 SVGA; 1024x768 XGA; 1280x720 720p HDTV; 1280x768 WXGA Variant; 1280x800 WXGA Variant; 1280x1024 SXGA; 1366x768 WXGA Variant; 1400x1050 SXGA +; 1600x1200 UXGA; 1680x1050 WXGA +; 1920x1080 1080p HDTV; 1920x1200 WUXGA. Ela também pode gerar 576i e 480i sinais de vídeo composto para PAL-BGHID, PAL-M, PAL-N, NTSC e NTSC-J.

## 2.7 GPIO

Um dos recursos do Raspberry Pi é a linha de pinos GPIO (entrada / saída de propósito geral) ao longo da borda superior da placa. Um cabeçalho GPIO de 40 pinos é encontrado em todas as placas Raspberry Pi atuais (não preenchidas no Pi Zero e Pi Zero W). Antes do Pi 1 Model B + (2014), as placas compreendiam um cabeçalho de 26 pinos mais curto.

Qualquer um dos pinos GPIO pode ser designado (em software) como um pino de entrada ou saída e usado para uma ampla gama de propósitos.

**Figura 2: Pinos gpio**



Dois pinos de 5V e dois pinos de 3V3 estão presentes na placa, bem como vários pinos de aterramento (0V), que não são configuráveis. Os pinos restantes são todos pinos 3V3 de propósito geral, o que significa que as saídas são definidas como 3V3 e as entradas são tolerantes a 3V3. Um pino GPIO designado como um pino de saída pode ser definido como alto (3V3) ou baixo (0V). Isso é facilitado com o uso de resistores pull-up ou pull-down internos. Os pinos GPIO2 e GPIO3 possuem resistores fixos, mas para outros pinos isso pode ser configurado no software.

Além dos dispositivos simples de entrada e saída, os pinos GPIO podem ser usados com uma variedade de funções alternativas, alguns estão disponíveis em todos os pinos, outros em pinos específicos.

- PWM (modulação por largura de pulso)
  - Software PWM disponível em todos os pinos
  - Hardware PWM disponível em GPIO12, GPIO13, GPIO18, GPIO19
- SPI
  - SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
  - SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)
- I2C
  - Dados: (GPIO2); Relógio (GPIO3)
  - Dados da EEPROM: (GPIO0); Relógio EEPROM (GPIO1)
- Serial
  - TX (GPIO14); RX (GPIO15)

É importante estar ciente de qual pino é qual. Uma referência útil pode ser acessada no Raspberry Pi, abrindo uma janela de terminal e executando o comando `pinout`. Essa ferramenta é fornecida pela biblioteca GPIO Zero Python, que é instalada por padrão na imagem da área de trabalho Raspbian, mas não no Raspbian Lite.

Figura 3: Utilização do comando pinout no terminal do Raspberry Pi

```

pi@raspberrypi:~ $ pinout
Pi Model B V1.2

Revision : a02082
SoC      : BCM2837
RAM      : 1024Mb
Storage   : MicroSD
USB ports : 4 (excluding power)
Ethernet ports : 1
Wi-fi    : True
Bluetooth : True
Camera ports (CSI) : 1
Display ports (DSI): 1

JB:
  3V3 (1) (2) 5V
  GPIO2 (3) (4) 5V
  GPIO3 (5) (6) GND
  GPIO4 (7) (8) GPIO14
  GND (9) (10) GPIO15
  GPIO17 (11) (12) GPIO18
  GPIO27 (13) (14) GND
  GPIO22 (15) (16) GPIO23
  3V3 (17) (18) GPIO24
  GPIO10 (19) (20) GND
  GPIO9 (21) (22) GPIO25
  GPIO11 (23) (24) GPIO18
  GND (25) (26) GPIO17
  GPIO10 (27) (28) GPIO101
  GPIO5 (29) (30) GND
  GPIO6 (31) (32) GPIO102
  GPIO13 (33) (34) GND
  GPIO19 (35) (36) GPIO106
  GPIO26 (37) (38) GPIO1020
  GND (39) (40) GPIO21

For further information, please refer to https://pinout.xyz/
pi@raspberrypi:~ $

```

Figura 4: Pinos GPIO em diversos modelos de Raspberry Pi

Raspberry Pi (Rev1)	Raspberry Pi (Rev.2)	Raspberry Pi B+, 2, 3 & Zero	Key
3V3	1 2	SV	+
GPIO0	3 4	SV	Ground
GPIO1	5 6	GND	UART
GPIO4	7 8	GPIO14	I2C
GND	9 10	GPIO15	SPI
GPIO17	11 12	GPIO18	GPIO
GPIO27	13 14	GND	Pin Number
GPIO2	15 16	GPIO23	
3V3	17 18	GPIO24	
GPIO10	19 20	GND	
GPIO9	21 22	GPIO25	
GPIO11	23 24	GPIO18	
GND	25 26	GND	

**Raspberry Pi (Rev1)**

**Raspberry Pi (Rev.2)**

**Raspberry Pi B+, 2, 3 & Zero**

**Key**

+  
Ground  
UART  
I2C  
SPI  
GPIO  
Pin Number

**PI My Life UP**

## 2.8 Software

A Fundação Raspberry Pi recomenda o uso do Raspbian , um sistema operacional Linux baseado no Debian. Outros sistemas operacionais de terceiros disponíveis através do site oficial incluem o Ubuntu MATE, o Windows 10 IoT Core, o RISC OS e distribuições especializadas para o centro de mídia Kodi e Sistemas operacionais não baseados em Linux.

## 2.9 Aplicações

O Raspberry Pi vem sendo aplicado em várias áreas, na educação a fundação Raspberry Pi vem trabalhando desde 2014 em capacitar professores para ensinarem professores programação e inspirar jovens nesse caminho.

Outra aplicação comum é a de automação industrial. Assim diversos aplicativos estão aproveitando o Raspberry Pi em uma solução econômica em monitoramento de energia e consumo de energia. Por causa do custo relativamente baixo do Raspberry Pi, isso se tornou uma solução popular e econômica para as alternativas comerciais mais caras.

Para as automações industriais, pode-se citar a TECHBASE, fabricante polonesa de automação industrial, projetou o primeiro computador industrial do mundo baseado no Raspberry Pi Compute Module, chamado ModBerry. O dispositivo possui várias interfaces, principalmente as portas seriais RS-485/232, entradas/saídas digitais e analógicas, CAN e barramentos econômicos de 1 fio, todos amplamente utilizados no setor de automação. O design permite o uso do Compute Module em ambientes industriais adversos, levando à conclusão de que o Raspberry Pi não está mais limitado a projetos domésticos e científicos, mas pode ser amplamente utilizado como uma solução Industrial IoT e atingir as metas da Indústria 4.0 .

### 3 PROJETOS

Nesta sessão serão apresentados alguns projetos simples para se implementar com o Raspberry Pi.

Será necessário além do Raspberry Pi + SD Card, teclado, mouse, monitor; será necessário uma Protoboard, Led's, resistores, sensor de temperatura LM35 e DHT 11 e fios para conexões.

#### 3.1 Instalando Raspbian

Raspbian é uma distribuição Linux desenvolvida a partir do Debian especificamente para o Raspberry Pi. Simples e funcional, o sistema possui uma interface gráfica, que o deixa muito agradável de usar; sendo assim o sistema operacional padrão do Raspberry.

O raspbian pode ser obtido gratuitamente através do seguinte link: <https://www.raspberrypi.org/downloads/raspbian/>.

Após fazer o download no site oficial é necessário instalar o .ISO do sistema no cartão de memória que será usado no Raspberry pi. Para isso pode-se utilizar diversos programas, recomenda-se a utilização Win32DiskImager, que se mostrou eficiente nos testes realizados.

Com a .ISO instalada no cartão de memória, basta conectar o Raspberry Pi em um mouse, teclado e monitor; ligar sua fonte de energia e o sistema operacional será instalado. A seguir, as Figuras 5, 6, 7, 8, 9, 10 demonstram passo a passo como instalar.

**Figura 5: Download e descompactação da .ISO do Raspbian**

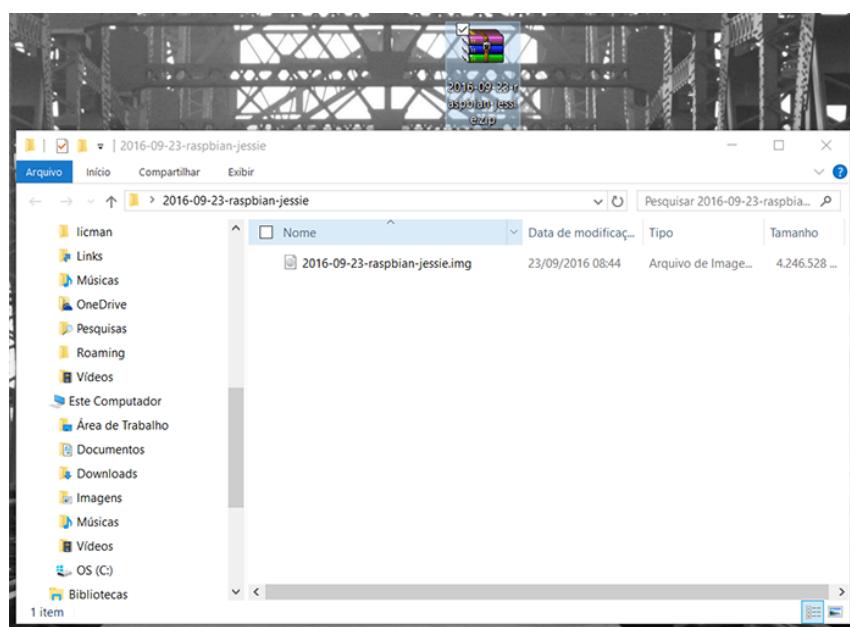


Figura 6: Abra a imagem do Raspbian no Win32DiskImager

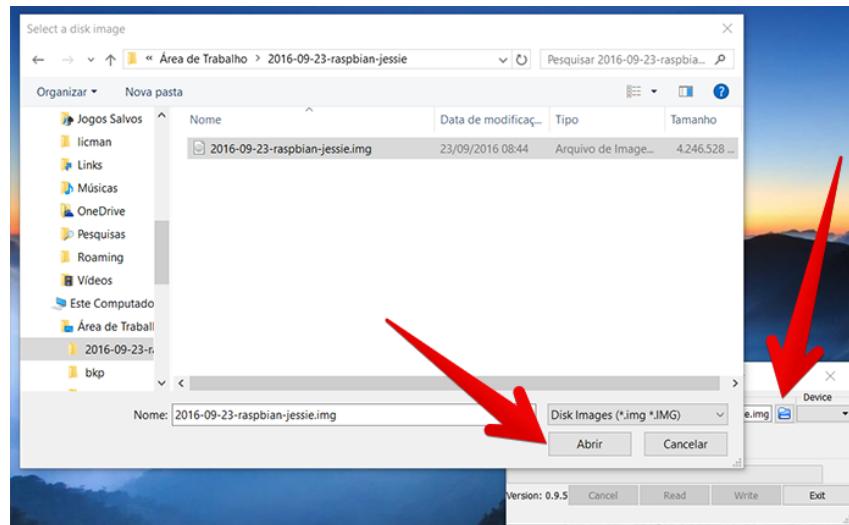


Figura 7: Formatação do cartão de memória FAT32

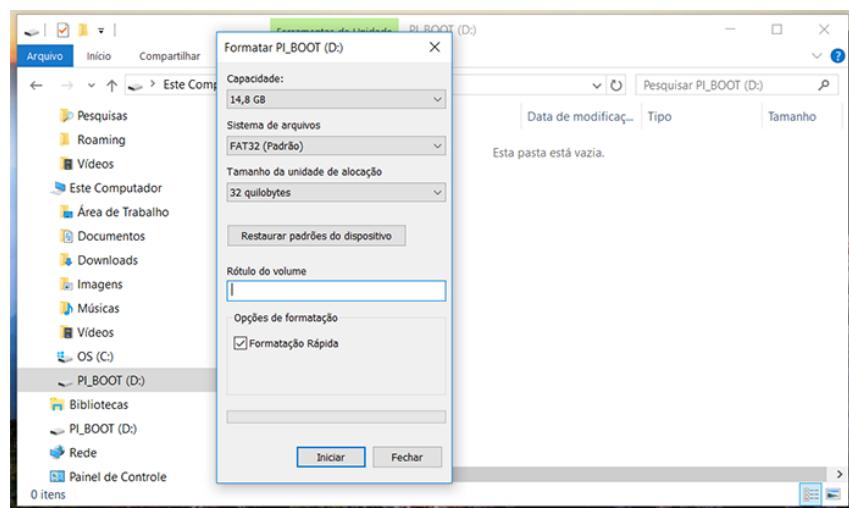


Figura 8: Escrever .ISO pelo Win32 Disk Imager

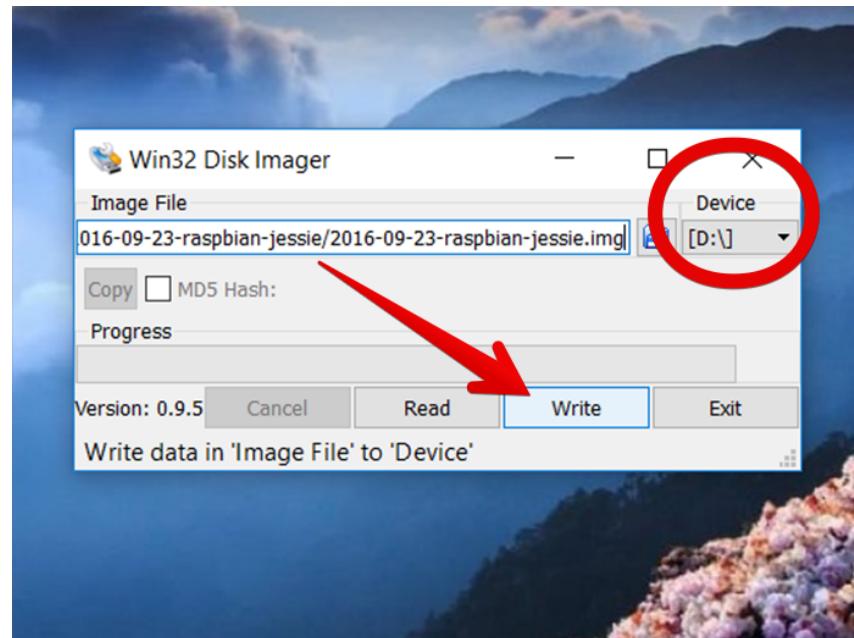
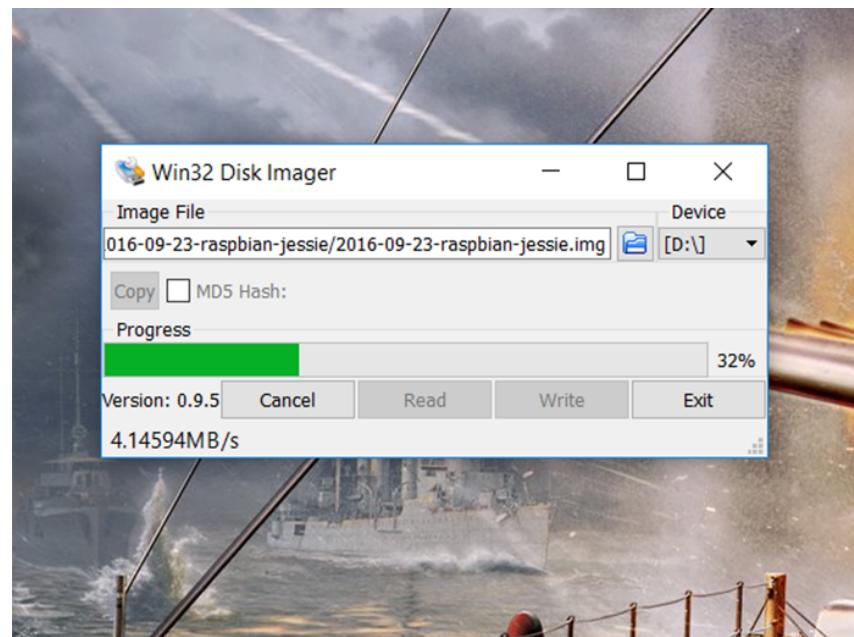
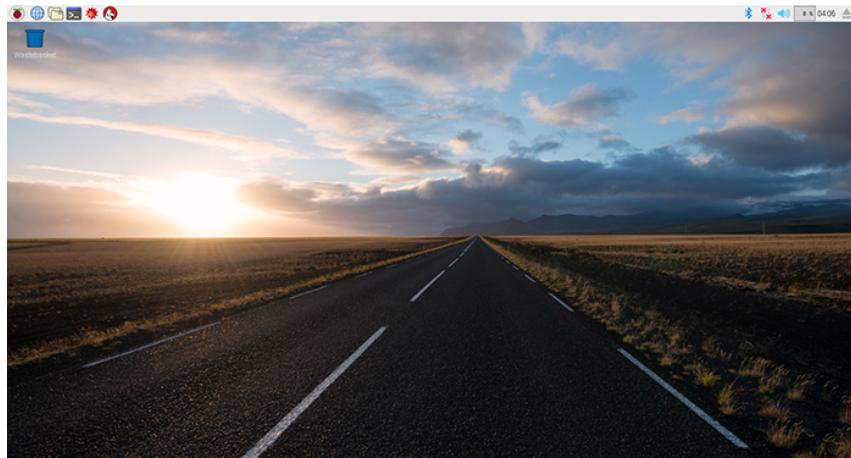


Figura 9: Processo de escrita no cartão de memória



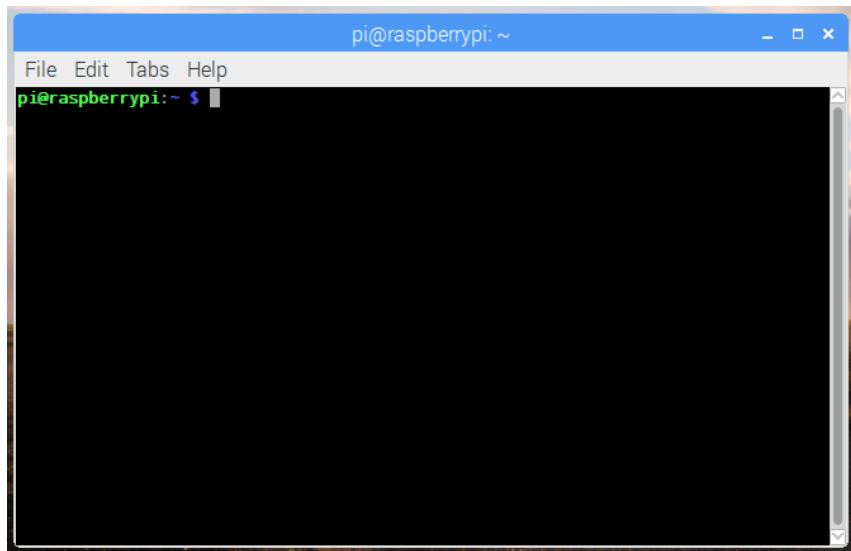
**Figura 10:** Área inicial Raspbian



### 3.2 Comandos básicos

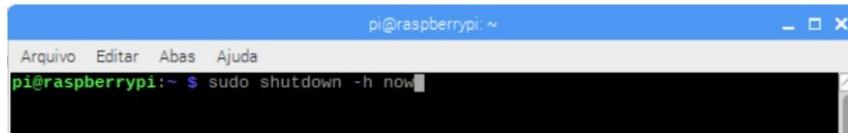
É interessante de antes de iniciar alguns projetos ter algumas noções dos comandos básicos utilizados. Para executar comandos e códigos pode-se utilizar-se do LXTerminal como mostra a figura 11, análogo ao Prompt de Comando do Windows.

**Figura 11:** LXTerminal Raspberry Pi



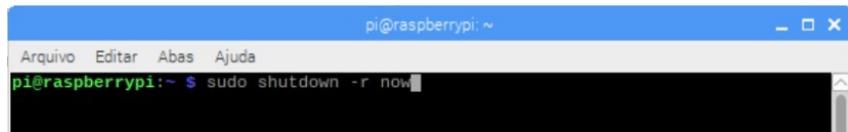
Outros comandos interessantes são os de desligamento e reinicialização como mostrado nas figuras 12 e 13; apesar da interface gráfica possuir um botão específico para isso, porém em determinadas aplicações é interessante não utilizar a interface gráfica para que o processamento do código seja exclusivo.

**Figura 12:** Comando para desligar Raspberry Pi



A screenshot of a terminal window titled "pi@raspberrypi: ~". The menu bar includes "Arquivo", "Editar", "Abas", and "Ajuda". The command "pi@raspberrypi:~ \$ sudo shutdown -h now" is typed into the terminal.

**Figura 13:** Comando para reiniciar Raspberry Pi

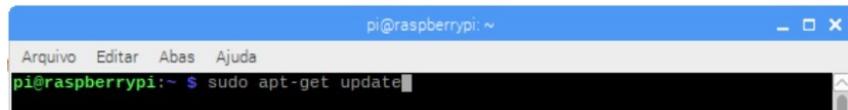


A screenshot of a terminal window titled "pi@raspberrypi: ~". The menu bar includes "Arquivo", "Editar", "Abas", and "Ajuda". The command "pi@raspberrypi:~ \$ sudo shutdown -r now" is typed into the terminal.

Interessante notar que a primeira palavra utilizada na linha de comando é o "sudo", que mostra ao raspberry pi que o usuário que está dando o comando tem prioridade nas ações, seria análogo ao executar como administrador do Windows.

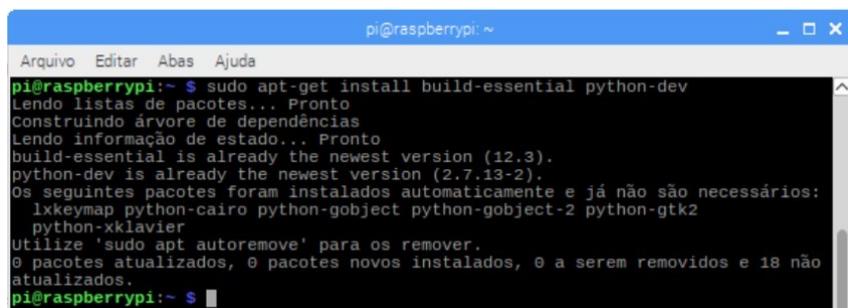
Outros comandos importantes para um bom desenvolvimento de códigos são os de atualização tanto do sistema operacional quanto do python.

**Figura 14:** Comando para atualizar Raspberry Pi



A screenshot of a terminal window titled "pi@raspberrypi: ~". The menu bar includes "Arquivo", "Editar", "Abas", and "Ajuda". The command "pi@raspberrypi:~ \$ sudo apt-get update" is typed into the terminal.

**Figura 15:** Comando para atualiza python



A screenshot of a terminal window titled "pi@raspberrypi: ~". The menu bar includes "Arquivo", "Editar", "Abas", and "Ajuda". The command "pi@raspberrypi:~ \$ sudo apt-get install build-essential python-dev" is typed into the terminal. The output shows the package list and dependency resolution process.

Para criar um novo diretório utiliza-se o comando "mkdir" seguido do nome do diretório e para visualizar todos os diretórios contidos utiliza-se o comando "ls-l". Para direcionar o diretório utiliza-se o comando "cd" e o nome do diretório, desta forma somente haverá alterações dentro do diretório. As figuras 19, 20 e 21 mostram a utilização desses comandos.

A seguir será apresentados mais uma série de comandos muito utilizados ao programar para Raspberry Pi e utilização do terminal.

## - Listar arquivos e pastas

Para listar arquivos e pastas, digite:

```
$ ls
```

Para mostrar mais informações sobre diretórios, tamanho e data:

```
$ ls -la
```

Para alterar o tipo de diretório:

```
$ cd seudiretorio
```

Para sair de um diretório:

```
/seudiretorio $ cd
```

## - Direitos de superusuário

Esta função permite executar aplicativos com direitos elevados de administrador como no Windows; assim ao digitar sudo é possível executar programas com os privilégios.

```
$sudo
```

## Entre na área de trabalho

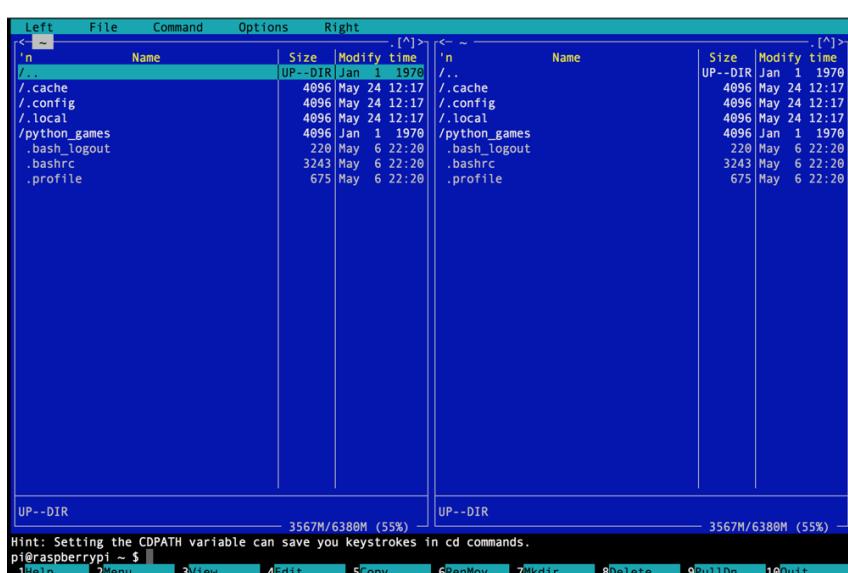
Entre na área de trabalho da interface gráfica do usuário.

```
$startx
```

## Gerenciar arquivos (copiar / mover etc.)

Gerenciar arquivos no console pode ser tedioso ao lidar com grandes quantidades de arquivos e diretórios diferentes. Considere checar "Midnight Commander" (site oficial) que é um gerenciador de arquivos visual similar ao "Total Commander" para windows.

**Figura 16: Gerenciador de arquivos Midnight Commander**



Instale o Midnight Commander:

```
$sudo apt-get install mc
```

Para fazer o gerenciamento manualmente utiliza-se os seguintes códigos:

Mover/ renomear arquivo/ diretório

```
$mv sourcefile destinationfile
```

Excluir

```
$rm filename
```

Cópia de

```
$mv sourcefile destinationfile
```

Excluir pastas

```
$rmdir /folder/
```

Excluir pasta e conteúdo

```
$rm -rf /folder/
```

Obter diretório atual

```
$pwd
```

### Informação da Cpu e número de série

Mostrar informações como nome do modelo da cpu, BogoMIPS, recursos, implementador da CPU, arquitetura da CPU, variante do processador, parte da CPU, revisão da CPU, etc. Ele também mostrará o número de série do pi.

```
$& cat /proc/cpuinfo
```

### Mostrar versão do Linux

Para mostrar qual versão do Linux (Raspbian) você está executando, digite:

```
$uname -a
```

### editor de texto (nano)

Existem vários editores de texto. Nano é um deles. Edite o arquivo atual ou crie um novo.

```
$nano nomeArquivo.TipoArquivo
```

Para criar um arquivo, utiliza-se o comando touch

```
$touch nomeArquivo.TipoArquivo
```

### Limpar tela

Para limpar a tela digite no terminal:

```
$clear
```

### Visão geral do disco

Use a opção '-h' (human-readable) para dados de impressão em formato human readable format (e.g., 1K 234M 2G)

```
$df -h
```

### Forçar desistir (matar)

Para forçar a saída (kill) de um tipo de aplicativo:

```
$ sudo killall applicationName.bin
```

### Visão geral de processos em execução

Para ter uma visão contínua da atividade do processador em tempo real usa-se o comando 'top'. Ele exibe uma listagem das tarefas mais intensivas de CPU no sistema e pode fornecer uma interface interativa para manipular processos. Ele pode classificar as tarefas por uso da CPU, uso de memória e tempo de execução.

```
$ top
```

### Torne o arquivo executável

Altere os direitos de um arquivo para que ele possa ser executado como um programa.

```
$ sudo chmod +x filename
```

### Execute o arquivo de programa

Para executar um programa, você deve colocar um './' antes do nome do arquivo, se você estiver no mesmo diretório que o arquivo de programa:

```
./myProgram.bin
```

Caso contrário, basta digitar o caminho para o arquivo:

```
/home/pi/myProgram.bin
```

### Roteiro de bash

Crie o arquivo como comando nano:

```
$sudo nano NOMEDOSCRIP
```

Inicie o script assim:

```
#!/bin/bash
```

### 3.3 Projeto 1 - Hello Word

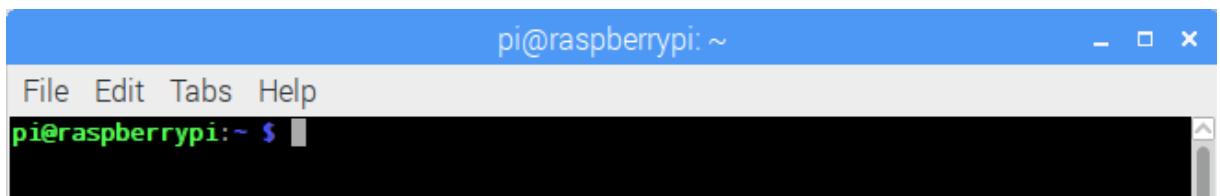
Este é um projeto clássico de quem está começando a programar. Para familiarizar-se com a forma de programar para o Raspberry Pi, será utilizado esse projeto; que tem a finalidade de imprimir na tela "Hello Word!"

1. Comece abrindo um Terminal, clique no ícone na barra de tarefas

**Figura 17:** Área inicial Raspbian

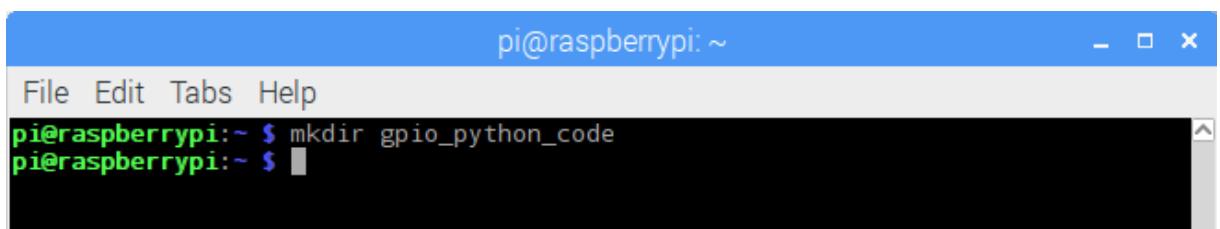


**Figura 18:** Terminal



2. Comece criando um novo diretório, aqui será armazenado todos os scripts realizados, crie um diretório chamado gpio\_python\_code:

**Figura 19:** Criação diretório



3. Pode-se usar o comando ls -l para visualizar o conteúdo do diretório atual:

Figura 20: Visualização do conteúdo do diretório

```
pi@raspberrypi:~ $ mkdir gpio_python_code
pi@raspberrypi:~ $ ls -l
total 44
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Nov 25 17:55 Documents
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Downloads
drwxr-xr-x 2 pi pi 4096 Feb 20 14:49 gpio_python_code
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 MUSIC
drwxr-xr-x 4 pi pi 4096 Feb 7 12:03 oldconffiles
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Pictures
```

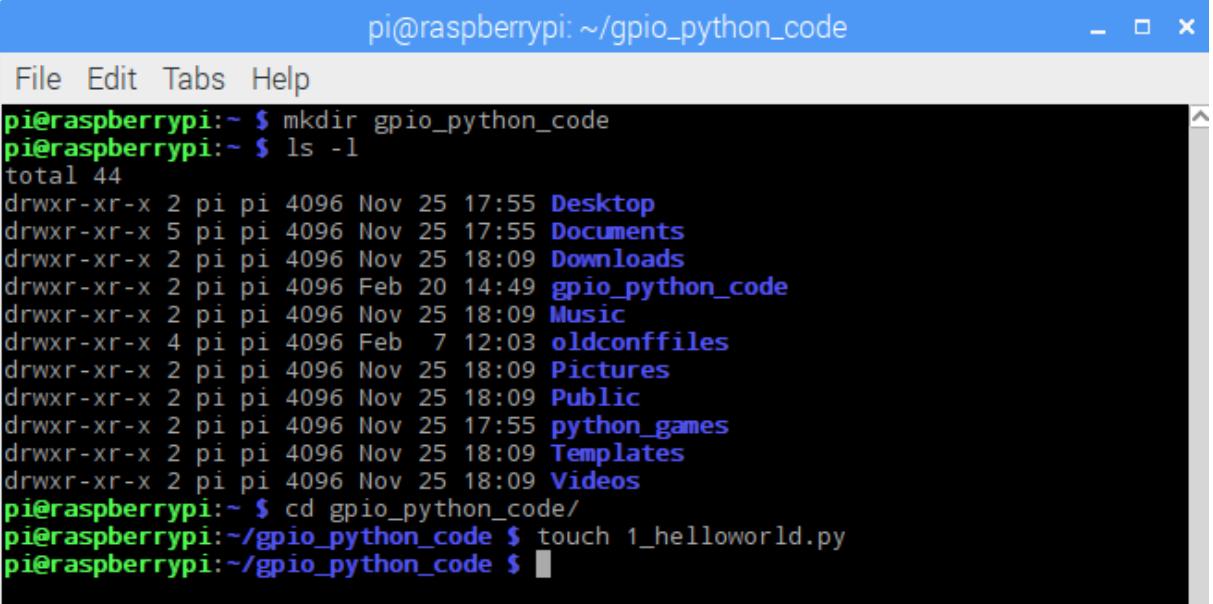
4. Alterar nosso diretório atual para o diretório gpio\_python\_code recém-criado:

Figura 21: Direcionamento de diretório

```
pi@raspberrypi:~ $ cd gpio_python_code/
pi@raspberrypi:~/gpio_python_code $
```

Usando o comando touch, pode-se criar um arquivo vazio. Este será o local onde será codificado o script Hello World.

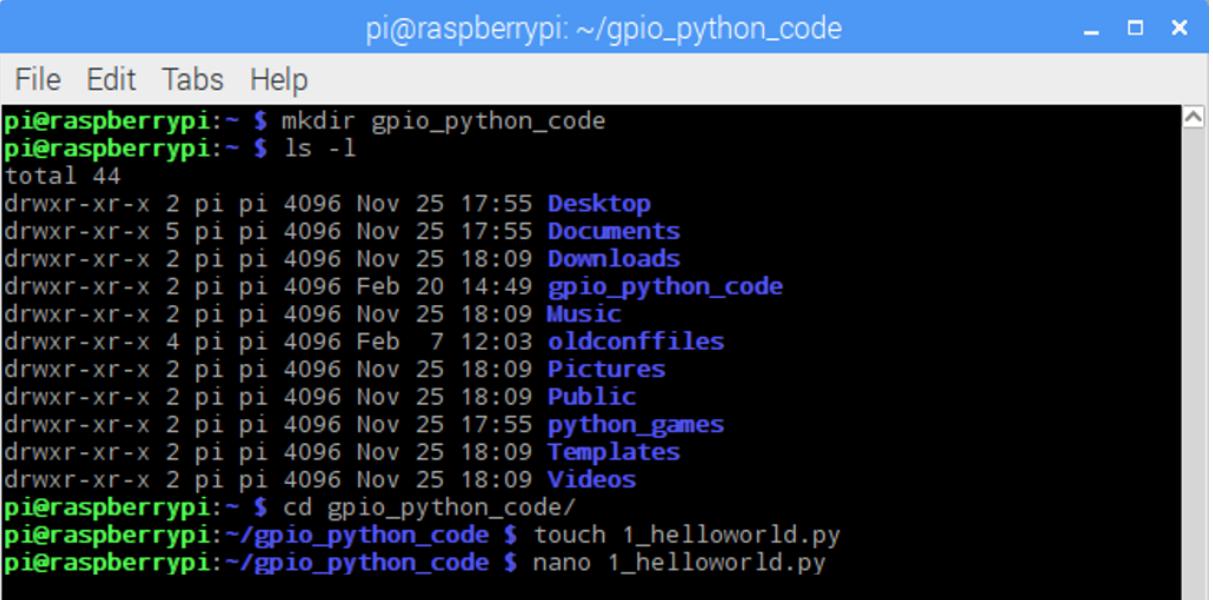
Figura 22: Criação do arquivo



```
pi@raspberrypi: ~	gpio_python_code
File Edit Tabs Help
pi@raspberrypi:~ $ mkdir gpio_python_code
pi@raspberrypi:~ $ ls -l
total 44
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Nov 25 17:55 Documents
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Downloads
drwxr-xr-x 2 pi pi 4096 Feb 20 14:49 gpio_python_code
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Music
drwxr-xr-x 4 pi pi 4096 Feb 7 12:03 oldconffiles
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Pictures
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Public
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 python_games
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Templates
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Videos
pi@raspberrypi:~ $ cd gpio_python_code/
pi@raspberrypi:~/gpio_python_code $ touch 1_helloworld.py
pi@raspberrypi:~/gpio_python_code $
```

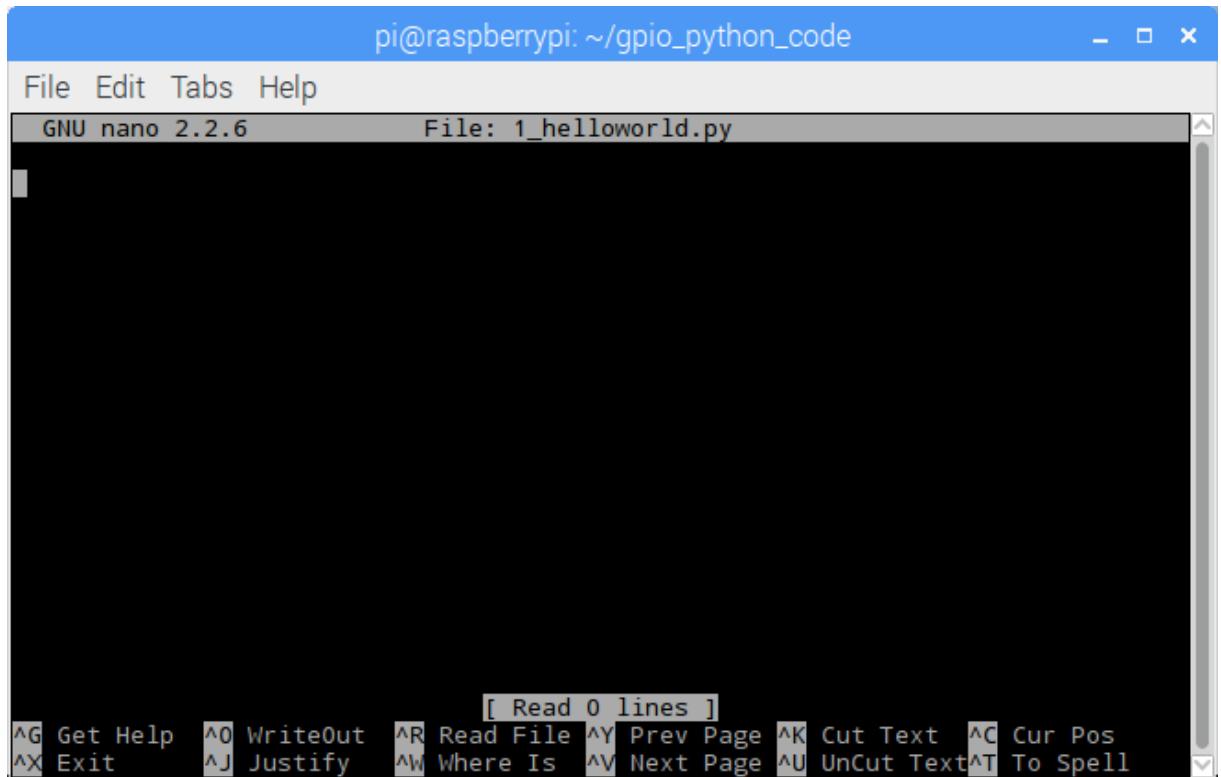
6. Depois de criar o arquivo vazio, utiliza-se o comando nano para abri-lo em um editor de texto.

Figura 23: Utilização do comando nano



```
pi@raspberrypi: ~	gpio_python_code
File Edit Tabs Help
pi@raspberrypi:~ $ mkdir gpio_python_code
pi@raspberrypi:~ $ ls -l
total 44
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Nov 25 17:55 Documents
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Downloads
drwxr-xr-x 2 pi pi 4096 Feb 20 14:49 gpio_python_code
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Music
drwxr-xr-x 4 pi pi 4096 Feb 7 12:03 oldconffiles
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Pictures
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Public
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 python_games
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Templates
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Videos
pi@raspberrypi:~ $ cd gpio_python_code/
pi@raspberrypi:~/gpio_python_code $ touch 1_helloworld.py
pi@raspberrypi:~/gpio_python_code $ nano 1_helloworld.py
```

**Figura 24: Editor de texto**



7. Agora digite o seguinte código:

**Figura 25: Código Hello Word!**

A screenshot of the nano text editor window showing the code for "Hello World". The title bar says "pi@raspberrypi: ~/gpio\_python\_code". The menu bar includes "File", "Edit", "Tabs", and "Help". The status bar shows "GNU nano 2.2.6", "File: 1\_helloworld.py", and "Modified". The main area contains the following Python code:

```
#!/usr/bin/python
#print Hello world
print "Hello World!"
```

8. Pressione **ctrl + X** para sair, você será solicitado a salvar o arquivo, basta pressionar **Y** e, em seguida, você será solicitado a dar um nome ao arquivo. Ele usará automaticamente seu nome atual **1.helloworld.py**, então apenas pressione **enter** para aceitar.

9. Para executar o script, basta digitar o seguinte comando:

Figura 26: Execução do código

A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window has a blue header bar with the title and standard window controls (minimize, maximize, close). Below the header is a menu bar with "File", "Edit", "Tabs", and "Help". The main area of the terminal shows the following command-line session:

```
pi@raspberrypi:~ $ mkdir gpio_python_code
pi@raspberrypi:~ $ ls -l
total 44
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 Desktop
drwxr-xr-x 5 pi pi 4096 Nov 25 17:55 Documents
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Downloads
drwxr-xr-x 2 pi pi 4096 Feb 20 14:49 gpio_python_code
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Music
drwxr-xr-x 4 pi pi 4096 Feb 7 12:03 oldconffiles
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Pictures
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Public
drwxr-xr-x 2 pi pi 4096 Nov 25 17:55 python_games
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Templates
drwxr-xr-x 2 pi pi 4096 Nov 25 18:09 Videos
pi@raspberrypi:~ $ cd gpio_python_code/
pi@raspberrypi:~/gpio_python_code $ touch 1_helloworld.py
pi@raspberrypi:~/gpio_python_code $ nano 1_helloworld.py
pi@raspberrypi:~/gpio_python_code $ sudo python 1_helloworld.py
Hello World!
pi@raspberrypi:~/gpio_python_code $
```

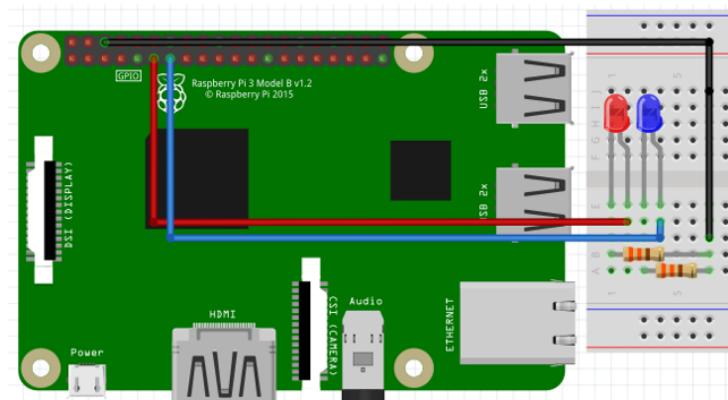
### 3.4 Projeto 2 - Acende e apaga LED

#### Materiais:

- 1x LED vermelho
- 1x LED azul
- 2x Resistor 330Ω
- 3x Jumper

#### Circuito:

Figura 27: Circuito projeto 2



Neste projeto, será executado separadamente dois códigos, um para acender LED's e outro para apagar os LED's. Assim será criado com o comando touch o arquivo 2\_on.py e 2\_off.py.

Para acessar o editor utiliza-se o comando nano seguido do nome do arquivo.

Figura 28: Criação dos arquivos 2\_on.py e 2\_off.py

```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
pi@raspberrypi ~ $ cd gpio_python_code
pi@raspberrypi ~/gpio_python_code $ touch 2_on.py
pi@raspberrypi ~/gpio_python_code $ touch 2_off.py
pi@raspberrypi ~/gpio_python_code $ nano 2_on.py
```

Com o editor para 2\_on.py, será escrito o seguinte código. Os comentários do código na Figura 29 mostram o que cada linha do código irá executar.

Figura 29: Código do arquivo 2\_on.py

```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
GNU nano 2.2.6           File: 2_on.py

#!/usr/bin/python
import RPi.GPIO as GPIO    #importa biblioteca

GPIO.setmode(GPIO.BCM)      #configura o sistema de numeracao para o BCM
GPIO.setup(17,GPIO.OUT)     #define o GPIO17 como um OUTPUT
GPIO.setup(27,GPIO.OUT)     #define o GPIO27 como um OUTPUT
print "Luzes acesas"       #imprime na tela a mensagem
GPIO.output(17,GPIO.HIGH)   #define o GPIO17 como HIGH
GPIO.output(27,GPIO.HIGH)   #define o GPIO27 como HIGH
```

De forma análoga, será repetido o processo para 2\_off.py.

Figura 30: Código do arquivo 2\_off.py

```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
GNU nano 2.2.6           File: 2_off.py

#!/bin/python
import RPi.GPIO as GPIO    #importa biblioteca

GPIO.setmode(GPIO.BCM)      #configura o sistema de numeracao de pinos para bcm
GPIO.setup(17,GPIO.OUT)     #define o GPIO17 como output
GPIO.setup(27,GPIO.OUT)     #define o GPIO27 como output

print "Luzes apagadas"

GPIO.output(17,GPIO.LOW)    #define o GPIO17 como LOW
GPIO.output(27,GPIO.LOW)    #define o GPIO27 como LOW
```

Ao executar sudo 2\_on.py, os LED's irão acender e uma mensagem na tela aparecerá para o usuário.

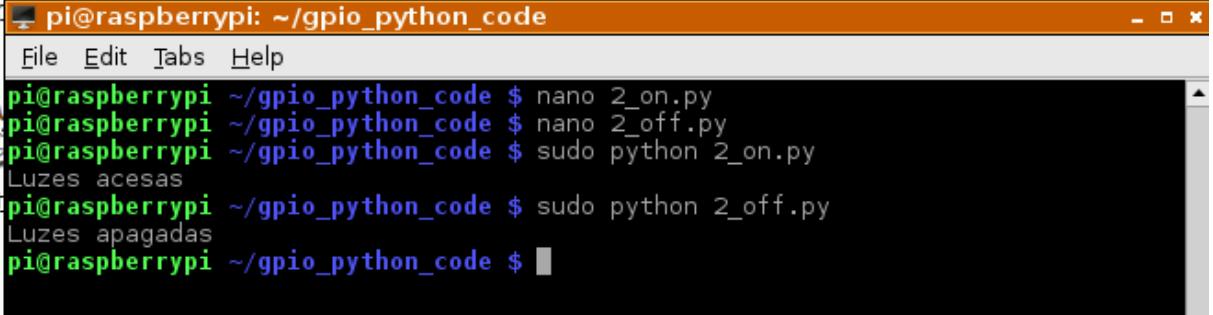
Figura 31: Arquivo 2\_on.py rodando

```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
pi@raspberrypi:~/gpio_python_code$ nano 2_on.py
pi@raspberrypi:~/gpio_python_code$ nano 2_off.py
pi@raspberrypi:~/gpio_python_code$ sudo python 2_on.py
Luzes acesas
pi@raspberrypi:~/gpio_python_code$
```

Ao executar sudo 2\_off.py, os LED's irão apagar e uma mensagem na tela aparecerá

para o usuário.

Figura 32: Arquivo 2\_on.py rodando



A screenshot of a terminal window titled "pi@raspberrypi: ~/gpio\_python\_code". The window has a standard Linux-style title bar with icons for close, minimize, and maximize. Below the title bar is a menu bar with "File", "Edit", "Tabs", and "Help". The main area of the terminal shows the following command-line session:

```
pi@raspberrypi:~/gpio_python_code $ nano 2_on.py
pi@raspberrypi:~/gpio_python_code $ nano 2_off.py
pi@raspberrypi:~/gpio_python_code $ sudo python 2_on.py
Luzes acesas
pi@raspberrypi:~/gpio_python_code $ sudo python 2_off.py
Luzes apagadas
pi@raspberrypi:~/gpio_python_code $
```

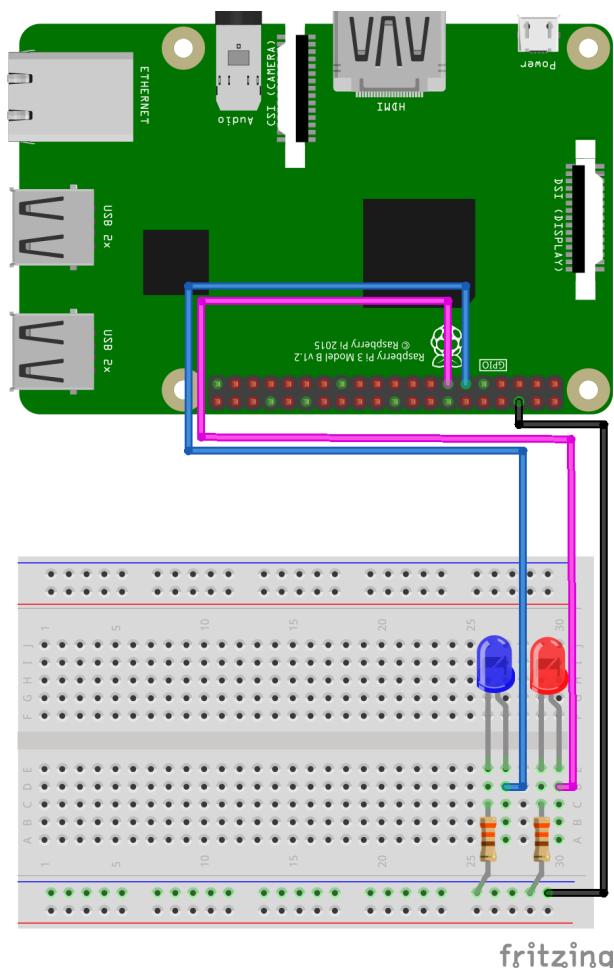
### 3.5 Projeto 3 - LED pisca-pisca n vezes

Materiais:

- 1x LED vermelho
- 1x LED azul
- 2x Resistor 330Ω
- 3x Jumper

Circuito:

Figura 33: Circuito projeto 3



Neste projeto o mesmo código fará o controle de acender e apagar os LED's. Primeiramente será criado o arquivo pisca.py e depois acessado para poder escrever o código como mostra a Figura 34.

Figura 34: Criação e acesso do arquivo pisca.py

```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
pi@raspberrypi ~/gpio_python_code $ touch pisca.py
pi@raspberrypi ~/gpio_python_code $ nano touch pisca.py
```

Ao acessar o script do arquivo pisca.py será feito a importação das bibliotecas necessárias configuração do sistema de numeração para BCM e defini-se os pinos que farão o controle dos LED's. Posteriormente será feito dois blocos de códigos, um para definir os pinos como alto (HIGH) para acender os LED's e baixo(LOW) para apagar os LED's. a função sleep determina o tempo de delay que haverá entre um bloco e outro, onde cada bloco será repetido quatro vezes.

Figura 35: Código parte 1

```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
GNU nano 2.2.6 File: pisca.py
#!/usr/bin/python

from time import sleep
import RPi.GPIO as GPIO #importa biblioteca

GPIO.setmode(GPIO.BCM) #configura o sistema de numeracao para o BCM
GPIO.setwarnings(False) #desativa mensagens para os pinos

GPIO.setup(17,GPIO.OUT) #define o GPIO17 como um OUTPUT
GPIO.setup(27,GPIO.OUT) #define o GPIO027 como um OUTPUT

#acende os LED's
print "Luzes acesas" #imprime na tela a mensagem
GPIO.output(17,GPIO.HIGH) #define o GPIO17 como HIGH
GPIO.output(27,GPIO.HIGH) #define o GPIO027 como HIGH
sleep(1)

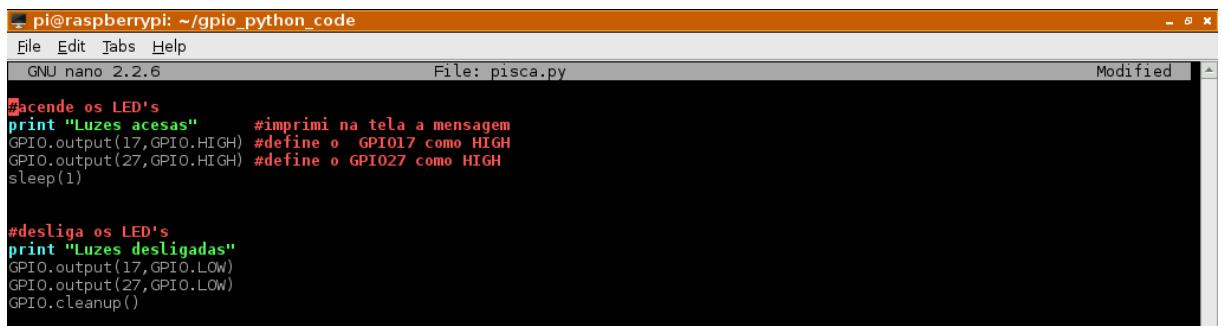
#desliga os LED's
print "Luzes desligadas"
GPIO.output(17,GPIO.LOW)
GPIO.output(27,GPIO.LOW)
sleep(1)

#acende os LED's
print "Luzes acesas" #imprime na tela a mensagem
GPIO.output(17,GPIO.HIGH) #define o GPIO17 como HIGH
GPIO.output(27,GPIO.HIGH) #define o GPIO027 como HIGH
sleep(1)

#desliga os LED's
print "Luzes desligadas"
GPIO.output(17,GPIO.LOW)
GPIO.output(27,GPIO.LOW)
sleep(1)

#acende os LED's
print "Luzes acesas" #imprime na tela a mensagem
GPIO.output(17,GPIO.HIGH) #define o GPIO17 como HIGH
GPIO.output(27,GPIO.HIGH) #define o GPIO027 como HIGH
sleep(1)
```

Figura 36: Código parte 2



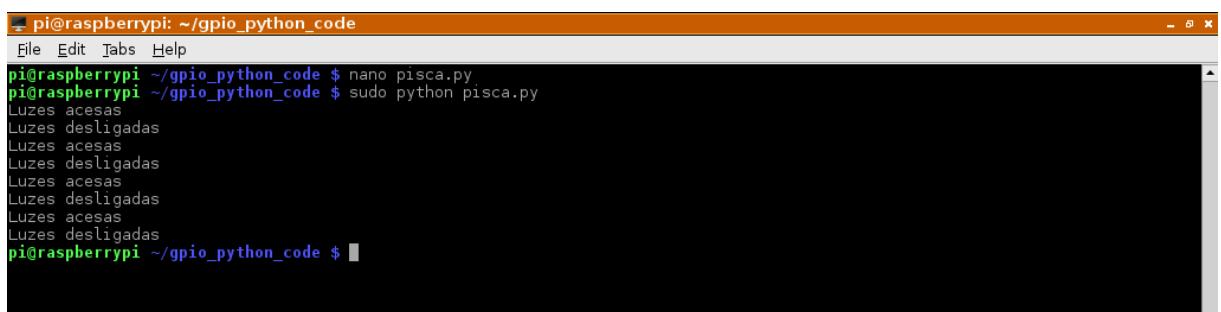
```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
GNU nano 2.2.6          File: pisca.py
Modified ▲

#acende os LED's
print "Luzes acesas"    #imprime na tela a mensagem
GPIO.output(17,GPIO.HIGH) #define o GPIO17 como HIGH
GPIO.output(27,GPIO.HIGH) #define o GPIO027 como HIGH
sleep(1)

#desliga os LED's
print "Luzes desligadas"
GPIO.output(17,GPIO.LOW)
GPIO.output(27,GPIO.LOW)
GPIO.cleanup()
```

Ao rodar o código na tela será apresentado ”Luzes acessas” e ”Luzes apagadas” como mostra a Figura 37 e na protoboard os LED’s piscaram quatro vezes.

Figura 37: Código rodando



```
pi@raspberrypi: ~/gpio_python_code
File Edit Tabs Help
pi@raspberrypi: ~/gpio_python_code $ nano pisca.py
pi@raspberrypi: ~/gpio_python_code $ sudo python pisca.py
Luzes acesas
Luzes desligadas
Luzes acesas
Luzes desligadas
Luzes acesas
Luzes desligadas
Luzes acesas
Luzes desligadas
pi@raspberrypi: ~/gpio_python_code $
```

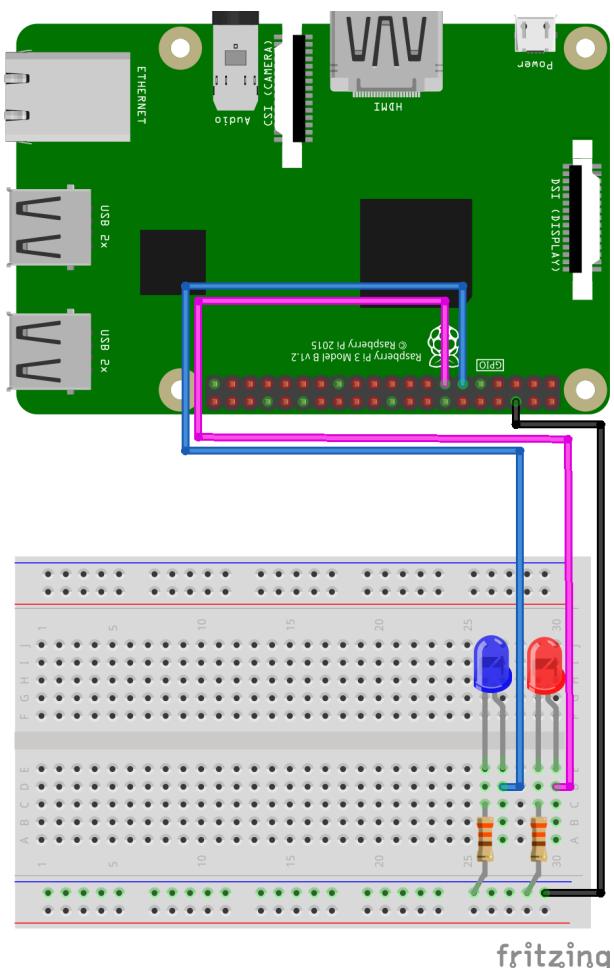
### 3.6 Projeto 4 - LED pisca-pisca eterno

Materiais:

- 1x LED vermelho
- 1x LED azul
- 2x Resistor 330Ω
- 3x Jumper

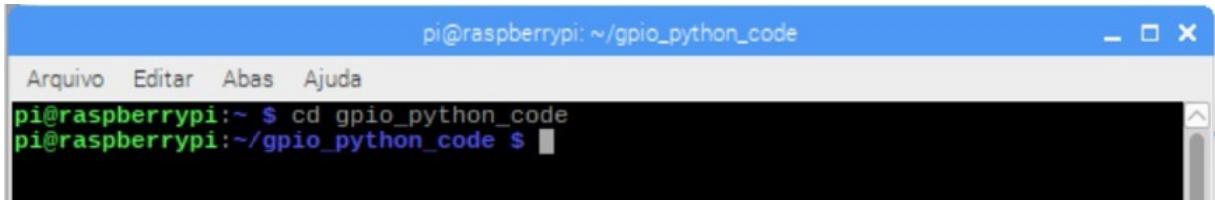
Circuito:

Figura 38: Circuito projeto 4



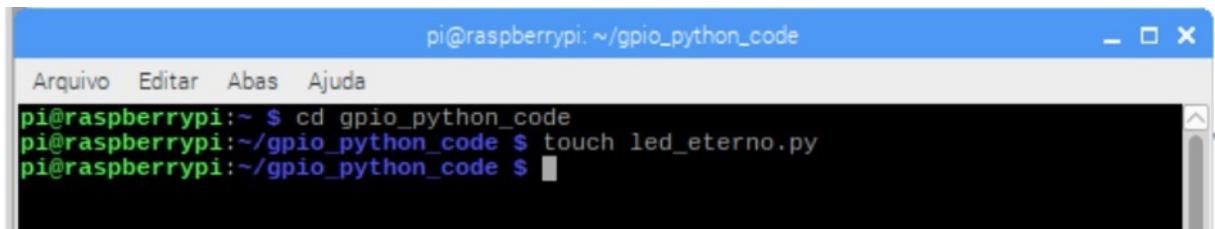
Este projeto será muito parecido com o projeto anterior, porém em vez de fazer os LED's piscarem n vezes, ele ficará piscando até que seja dado um comando para que eles parem.

Figura 39: Diretório de códigos



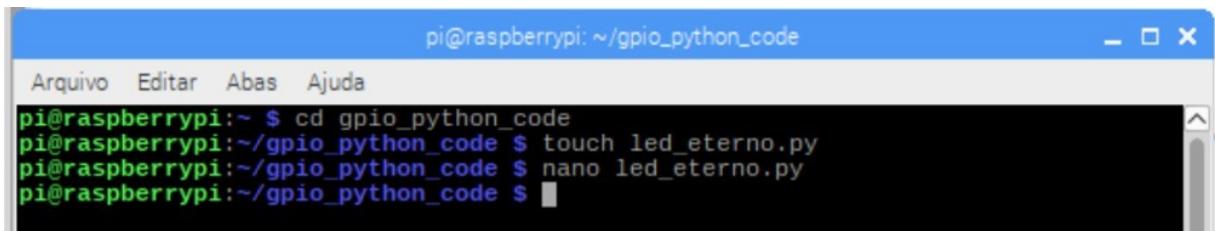
```
pi@raspberrypi: ~	gpio_python_code
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $
```

Figura 40: Criação do arquivo em python



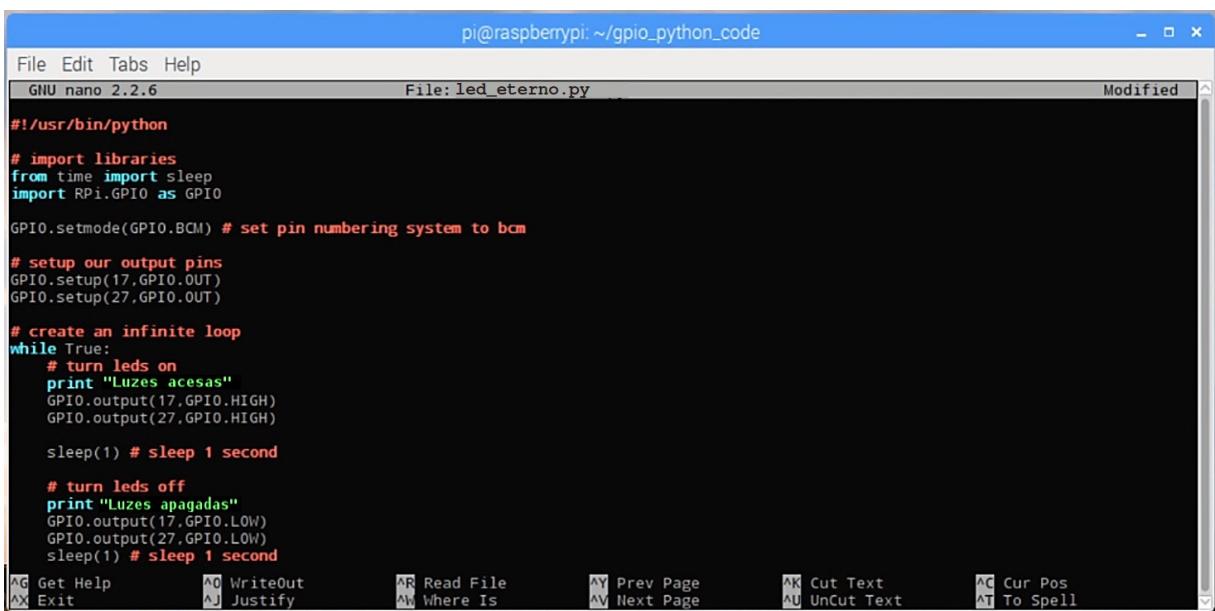
```
pi@raspberrypi: ~	gpio_python_code
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ touch led_eterno.py
pi@raspberrypi:~/gpio_python_code $
```

Figura 41: Acessar o arquivo led\_eterno.py



```
pi@raspberrypi: ~	gpio_python_code
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ touch led_eterno.py
pi@raspberrypi:~/gpio_python_code $ nano led_eterno.py
pi@raspberrypi:~/gpio_python_code $
```

Figura 42: Código led\_eterno.py



```
File Edit Tabs Help
GNU nano 2.2.6          File: led_eterno.py          Modified
#!/usr/bin/python

# import libraries
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM) # set pin numbering system to bcm

# setup our output pins
GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)

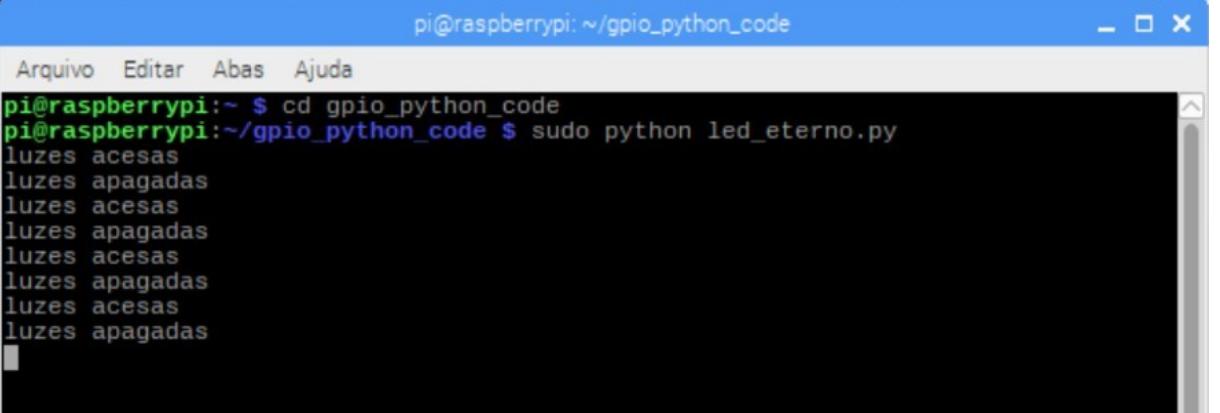
# create an infinite loop
while True:
    # turn leds on
    print "Luzes acesas"
    GPIO.output(17,GPIO.HIGH)
    GPIO.output(27,GPIO.HIGH)

    sleep(1) # sleep 1 second

    # turn leds off
    print "Luzes apagadas"
    GPIO.output(17,GPIO.LOW)
    GPIO.output(27,GPIO.LOW)
    sleep(1) # sleep 1 second
```

O código acima é bem similar aos outros já realizados, porém haverá um novo comando o `while True:`; esse comando é utilizado para repetir seções do código, no caso desse código, para ligar e desligar os LED's infinitamente.

**Figura 43: Código led\_eterno.py rodando**



A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window has a blue header bar with the title and standard window controls. Below the title bar is a menu bar with options "Arquivo", "Editar", "Abas", and "Ajuda". The main area of the terminal shows the command line and its output. The command entered was "sudo python led\_eterno.py". The output consists of repeated lines of text: "luzes acesas" followed by "luzes apagadas", which indicates the script is alternating the state of the LED. The terminal window has a dark background and light-colored text.

```
pi@raspberrypi:~$ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ sudo python led_eterno.py
luzes acesas
luzes apagadas
luzes acesas
luzes apagadas
luzes acesas
luzes apagadas
luzes acesas
luzes apagadas
```

Para parar o código deve-se pressionar **ctrl + c**.

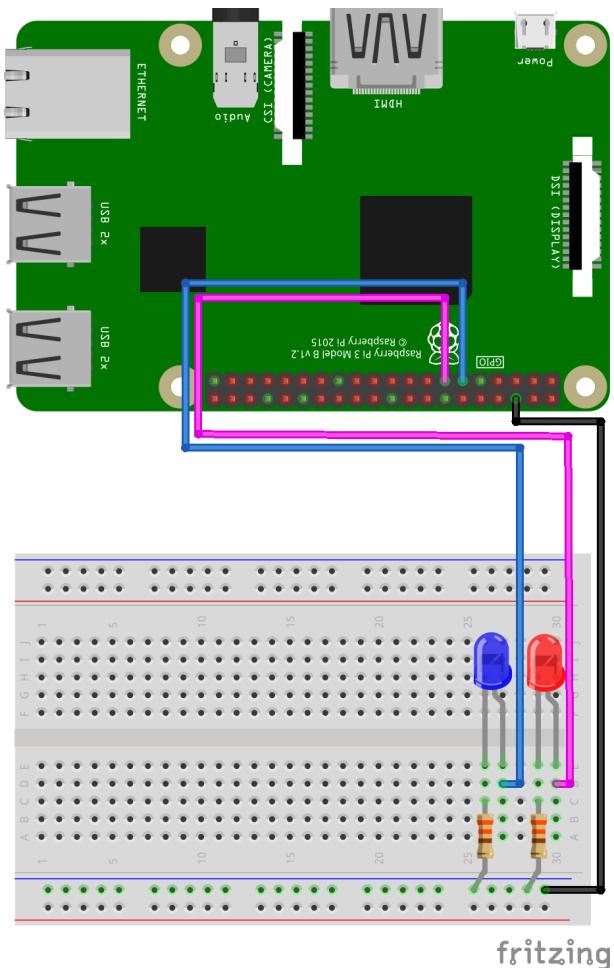
### 3.7 Projeto 5 - LED pisca-pisca controlado

Materiais:

- 1x LED vermelho
- 1x LED azul
- 2x Resistor 330Ω
- 3x Jumper

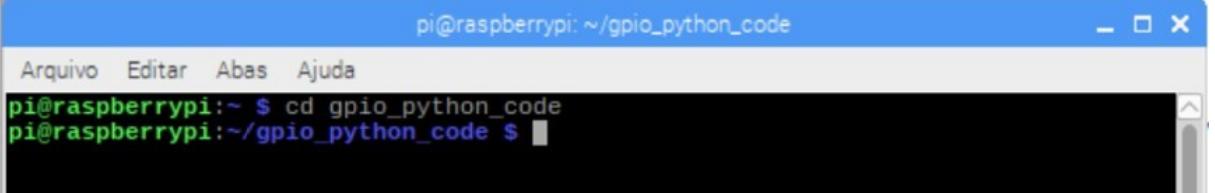
Circuito:

Figura 44: Circuito projeto 5



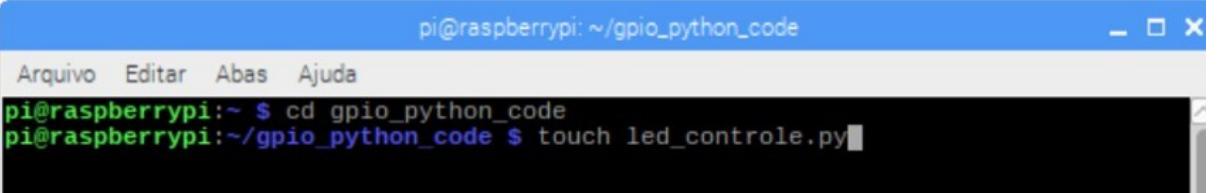
Este projeto será um pouco mais elaborado que os outros. Será o primeiro projeto que haverá interação com o usuário. Ao compilar o projeto o usuário irá escolher qual LED irá piscar e quantas vezes ele irá piscar.

Figura 45: Diretório dos códigos



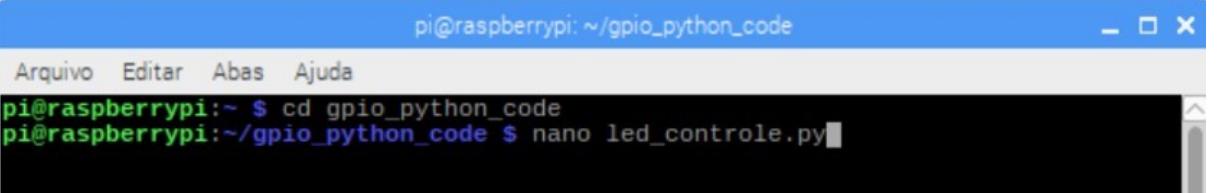
```
pi@raspberrypi: ~	gpio_python_code
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $
```

Figura 46: Criação do arquivo em python



```
pi@raspberrypi: ~	gpio_python_code
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ touch led_controle.py
```

Figura 47: Acessar o arquivo led\_controle.py



```
pi@raspberrypi: ~	gpio_python_code
Arquivo Editar Abas Ajuda
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ nano led_controle.py
```

Figura 48: Código led\_controle.py

The screenshot shows a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window contains the Python script "led\_controle.py". The script imports os, time, and RPi.GPIO, sets up GPIO pins 17 and 27 as outputs, and initializes variables led\_choice and count. It prints a question about which LED to flash and asks for input. If the choice is 1, it flashes the blue LED; if choice is 2, it flashes the red LED. Both loops ask for the number of times to flash and then execute the flashing logic. The terminal window has a menu bar with "Arquivo", "Editar", "Abas", and "Ajuda". The status bar at the bottom shows keyboard shortcuts for various functions like Help, Save, Find, Justify, Paste, Exit, and Read file.

```
#!/usr/bin/python

import os
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup(17,GPIO.OUT)
GPIO.setup(27,GPIO.OUT)

#Setup variaveis entrada
led_choice = 0
count = 0

os.system('clear')

print "Qual LED você quer piscar: "
print "1: VERMELHO?"
print "2: AZUK?"

led_choice = input("Faca sua escolha: ") # ask for an input

if led_choice == 1:
    os.system('clear')
    print "Voce escolheu LED Azul"
    count = input("Quantas vezes voce quer que ele pisque?: ")
    while count > 0:
        GPIO.output(27,GPIO.HIGH)
        sleep(1)
        GPIO.output(27,GPIO.LOW)
        sleep(1)
        count = count - 1

if led_choice == 2:
    os.system('clear')
    print "Voce escolheu LED Vermelho"
    count = input("Quantas vezes voce quer que ele pisque?: ")
    while count > 0:
        GPIO.output(17,GPIO.HIGH)
        sleep(1)
        GPIO.output(17,GPIO.LOW)
        sleep(1)
        count = count - 1

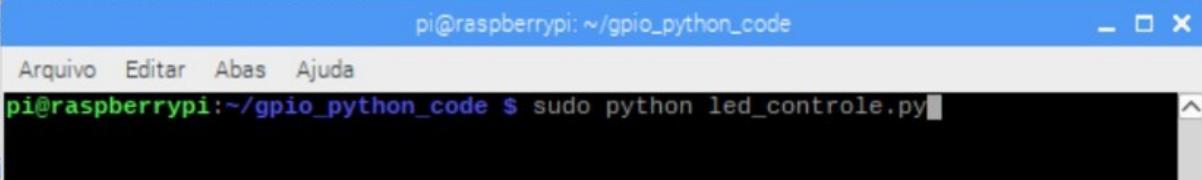
^G Ajuda      ^O Gravar      ^W Onde está? ^K Recort txt ^J Justificar ^C Pos atual
^X Sair       ^R Ler o arq  ^\ Substituir ^U Colar txt ^T Lintar   ^_ Ir p/ linha
```

A estrutura desse código não difere muito dos anteriores, porém é um pouco mais elaborada sua programação. A princípio o usuário escolherá entre 1 ou 2, para escolher qual LED irá piscar, para o código o valor 1 ou 2 será atribuído a variável "led\_choice".

Com o comando `if` que é uma estrutura de condição que permite avaliar uma expressão e, de acordo com seu resultado, executar uma determinada ação; assim quando o usuário escolhe 1 será executado o código para o LED azul piscar e idem para 2, porém com o LED vermelho.

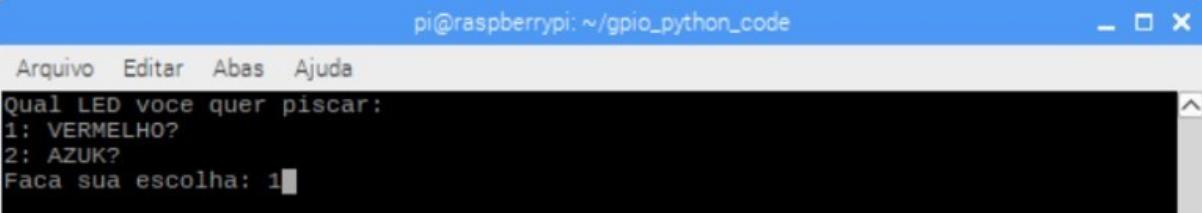
A segunda parte do código deverá receber a quantidade de vezes que o usuário quer que o LED pisque e executar. A variável "count" receberá o valor das n vezes que o LED irá piscar. Assim será usado o laço de repetição `while`, que repete um bloco de instrução enquanto a condição definida em seu cabeçalho for verdadeiro. No caso do código a condição é que count deva ser maior que 0. Assim a estrutura acende o LED, apaga o LED e diminui em 1 a variável "count". Quando "count" chega a 0, o código é finalizado.

**Figura 49: Inicialização do código**



```
pi@raspberrypi:~/gpio_python_code$ sudo python led_controle.py
```

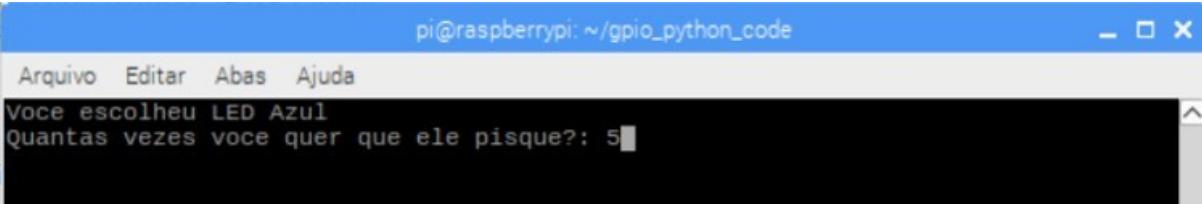
**Figura 50: Inicialização do código**



```
pi@raspberrypi:~/gpio_python_code$ Qual LED voce quer piscar:  
1: VERMELHO?  
2: AZUL?  
Faca sua escolha: 1
```

Escolha de qual LED irá piscar.

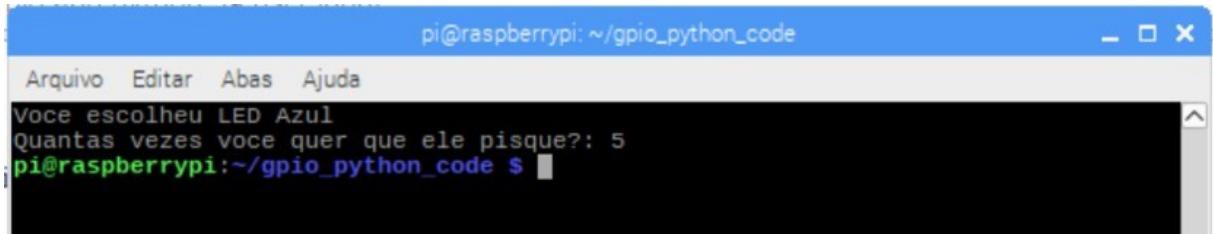
**Figura 51: Código rodando - parte 1**



```
pi@raspberrypi:~/gpio_python_code$ Voce escolheu LED Azul  
Quantas vezes voce quer que ele pisque?: 5
```

Escolha de n vezes que os LED's irão piscar.

**Figura 52:** Código rodando - parte 2



A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window has a blue header bar with the title and standard window controls (minimize, maximize, close). Below the header is a menu bar with options "Arquivo", "Editar", "Abas", and "Ajuda". The main area of the terminal shows the following text:  
Voce escolheu LED Azul  
Quantas vezes voce quer que ele pisque?: 5  
pi@raspberrypi:~/gpio\_python\_code \$ █

Após piscar a quantidade de vezes que o usuário escolheu o código para de rodar.

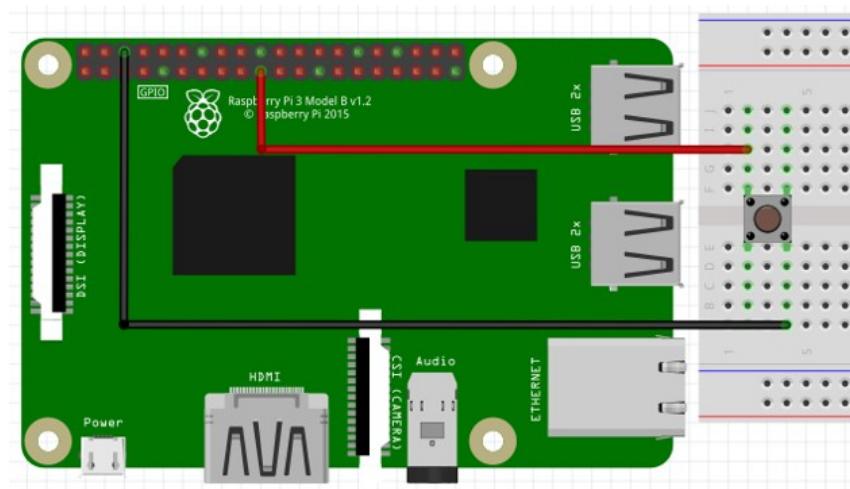
### 3.8 Projeto 6 - Configurar botão

Materiais:

- 1x botão táctil
- 2x Jumper

Circuito:

Figura 53: Circuito projeto 6



Neste projeto será configurado o botão táctil para executar uma determinada tarefa. Neste exemplo o botão imprimirá na tela dia da semana, mês, dia, horário e ano.

Figura 54: Diretório gpio\_python\_code

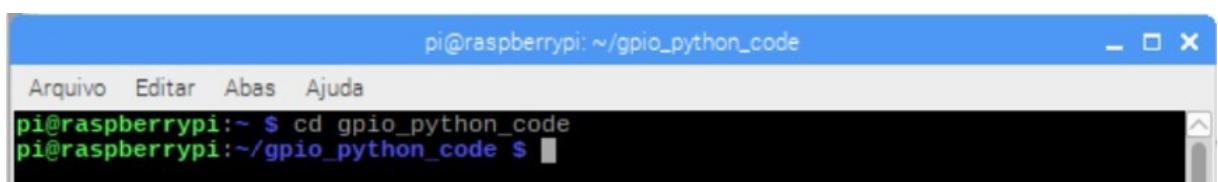


Figura 55: Criar e abrir o arquivo botao.py

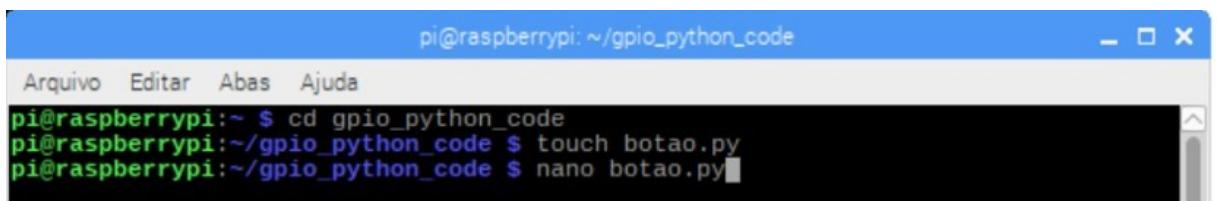


Figura 56: Código para configurar botão

A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window shows a Python script named "botao.py" with the following code:

```
#!/usr/bin/python

import os
from time import sleep
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

# setup pino de entrada
# resistor pull up interno para segurar o pino em 3v3
GPIO.setup (10, GPIO.IN, pull_up_down = GPIO.PUD_UP)

while True:
    if (GPIO.input (10) == False):
        print ("Botao pressionado")
        os.system ('date') # imprime a data e a hora do sistema
        print GPIO.input(10)
        sleep (5)
    else:
        os.system ('clear') # limpa o texto das telas
        print ("Esperando voce apertar um botao")
        sleep (0.1)

Salvar buffer modificado? (Responder "Não" vai DESCARTAR alterações.)
```

The terminal window has a modal dialog at the bottom asking "Salvar buffer modificado? (Responder "Não" vai DESCARTAR alterações.)" with options "S Sim", "N Não", and "Cancel".

No código desse projeto, observa-se uma nova biblioteca chamada `os`. Este módulo fornece uma maneira portátil de usar uma funcionalidade dependente do sistema operacional.

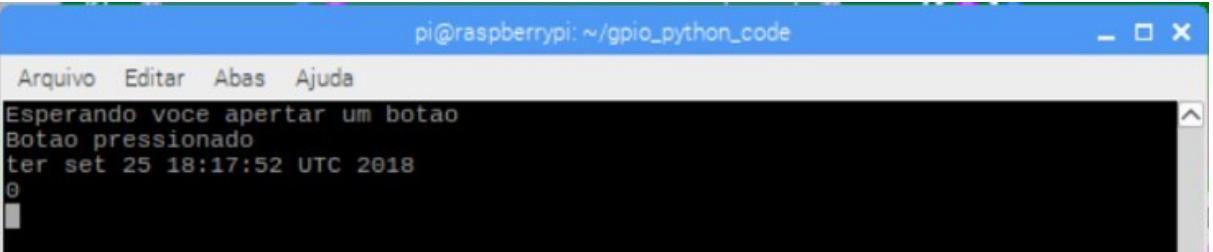
O código é simples, habilita-se a porta 10 do GPIO que está que está ligada a uma das pernas do botão. O laço de repetição `while` irá executar os laços condicionais `if` e `else`. Se o botão for pressionado será impresso na tela `Botão pressionado` e através do módulo da biblioteca `os`, o `os.system` que irá executar uma sting, no exemplo a string `date` que imprimi valor de dia da semana, mês, dia, horário e ano. Caso não aperte o botão, a tela é limpada e a mensagem `Esperando voce apertar um botao` é impressa.

Figura 57: Rodando o código botao.py

A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window shows the command `sudo python botao.py` being run, with the output:

```
pi@raspberrypi:~/gpio_python_code $ sudo python botao.py
```

Figura 58: Resposta ao apertar botão táctil



A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window has a blue header bar with the title and standard window controls (minimize, maximize, close). Below the title bar is a menu bar with options "Arquivo", "Editar", "Abas", and "Ajuda". The main area of the terminal displays the following text:

```
Esperando voce apertar um botao
Botao pressionado
ter set 25 18:17:52 UTC 2018
0
```

### 3.9 Projeto 7 - Monitoramento de temperatura e umidade com DHT11

#### Materiais:

- 1x Sensor de temperatura DHT11
- 1x Resistor 4,7K  $\Omega$
- 4x Jumper

#### Circuito:

Figura 59: Circuito projeto 7

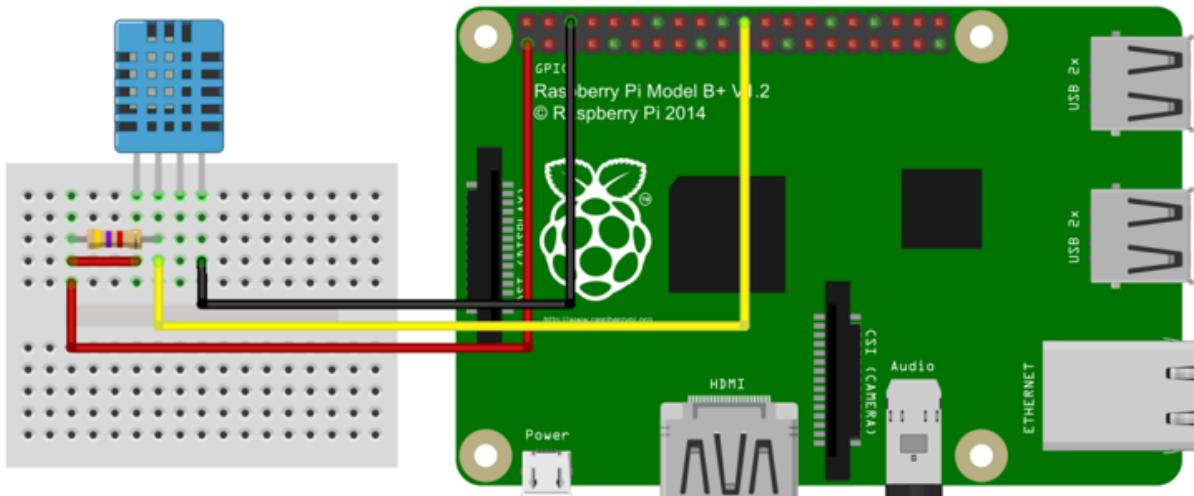


Figura 60: Sensor de temperatura e umidade DHT11

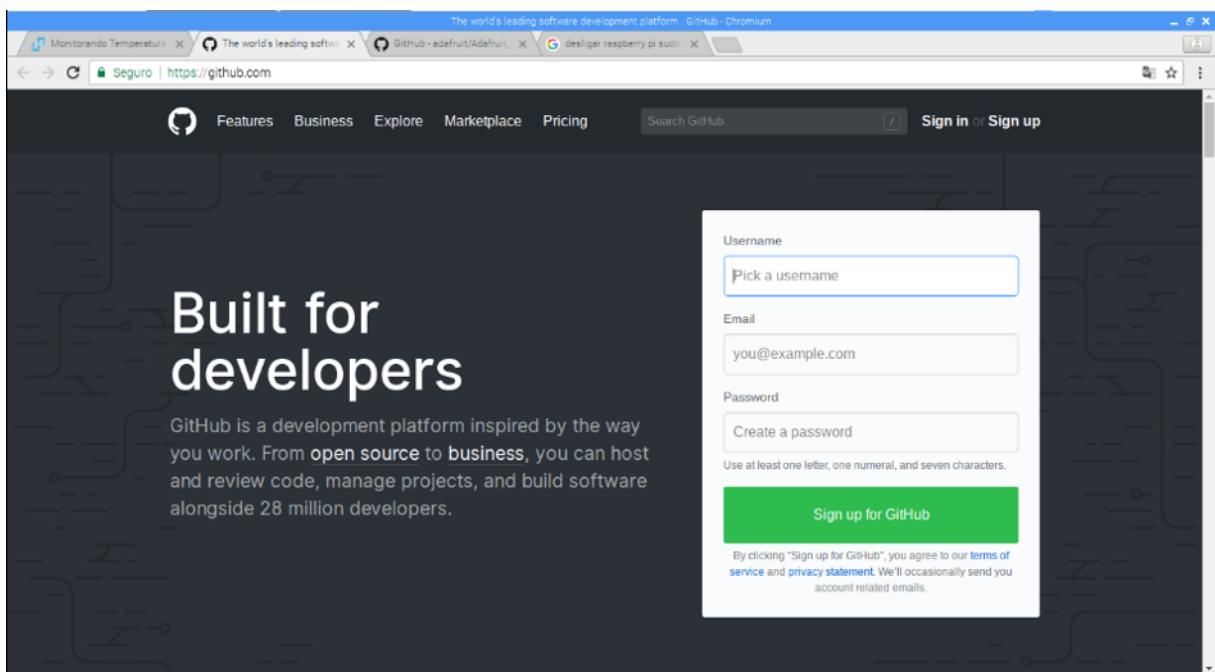


Este projeto será implementado um sistema de monitoramento de temperatura e umidade. O sensor DHT11 mostrado na figura 60 possui 4 pinos, o primeiro é de alimentação, o segundo pino é o que envia os dados, terceiro não tem nenhuma função e o quarto é o terra do sensor. Algumas características do sensor:

- Alimentação : 3 à 5,5 V
- Faixa de leitura – Umidade : 20 à 80%
- Precisão umidade : 5%
- Faixa de leitura – Temperatura : 0 – 50°C
- Precisão temperatura : +/- 2°C

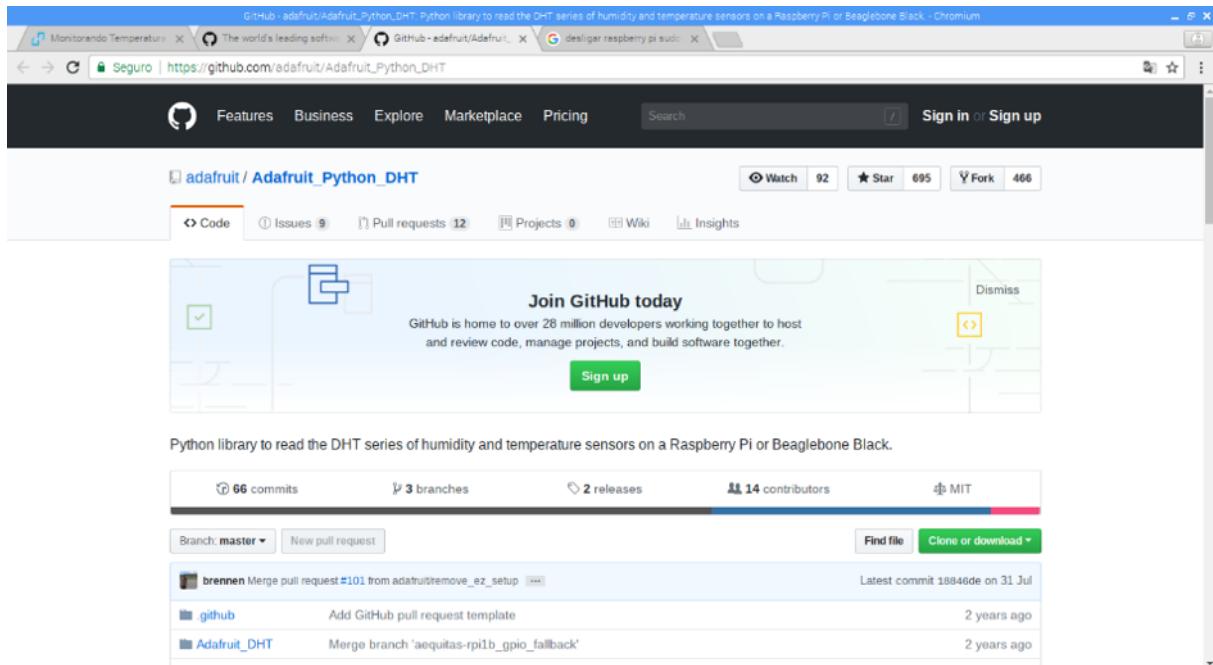
Para o projeto será necessário utilizar uma biblioteca deste sensor. Um dos locais onde se encontra muitas bibliotecas em Python e diversas outras linguagens de programação é a plataforma GitHub (<https://github.com/>) como mostra a figura 61.

**Figura 61:** Página inicial Github



Para o sensor trabalhado nesse projeto, escolheu-se a biblioteca criada pela Adafruit. A figura 62 mostra a página da biblioteca ([https://github.com/adafruit/Adafruit\\_Python\\_DHT](https://github.com/adafruit/Adafruit_Python_DHT))

**Figura 62: Página Github Adafruit Python DHT**



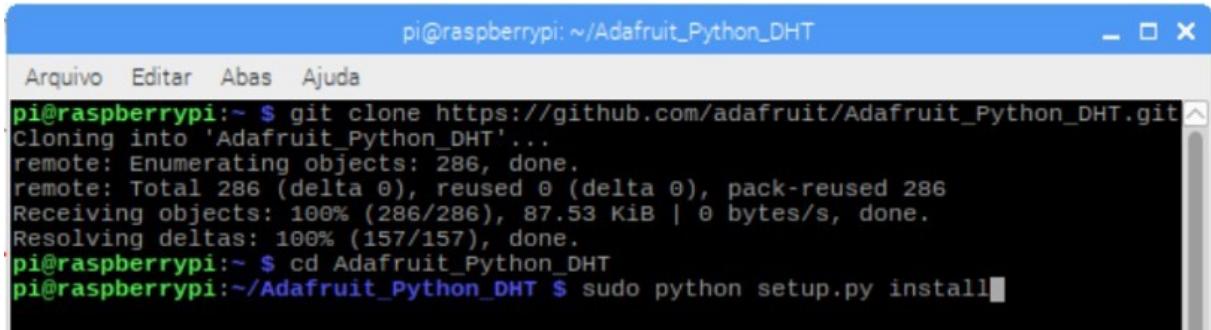
Para poder utilizar essa biblioteca, no terminal do raspberry utiliza-se o comando `git clone` seguindo da URL da página como mostra a figura.

**Figura 63: Download, instalação da biblioteca Adafruit DHT 11**

A screenshot of a terminal window titled 'pi@raspberrypi: ~/Adafruit\_Python\_DHT'. The user has run the command `git clone https://github.com/adafruit/Adafruit_Python_DHT.git`. The output shows the cloning process, including object enumeration, receiving objects, and resolving deltas. After cloning, the user changes directory to the newly created 'Adafruit\_Python\_DHT' folder.

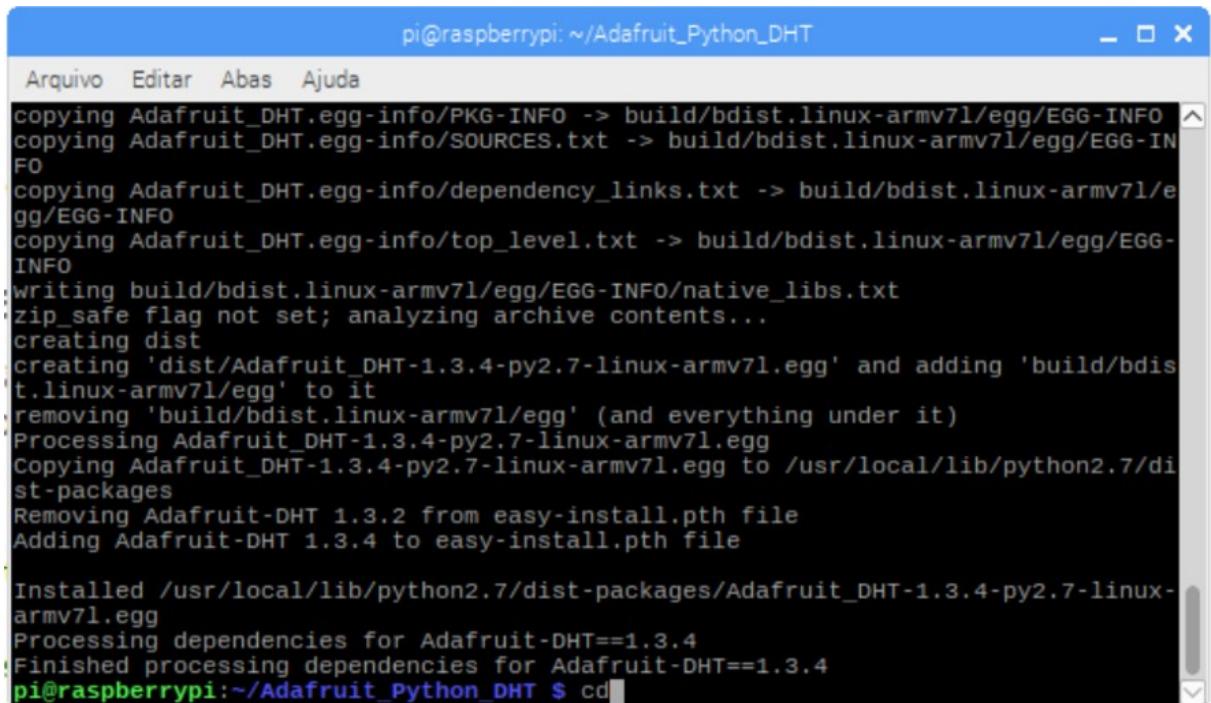
O próximo passo é a instalação da biblioteca, assim deve-se entrar no diretório que foi feito o download, sendo o padrão `$cd Adafruit_Python_DHT`. Com o diretório aberto inicializa-se o `setup.py` da biblioteca como superusuário, como mostra a figura 64.

Figura 64: Instalação da biblioteca



```
pi@raspberrypi:~$ git clone https://github.com/adafruit/Adafruit_Python_DHT.git
Cloning into 'Adafruit_Python_DHT'...
remote: Enumerating objects: 286, done.
remote: Total 286 (delta 0), reused 0 (delta 0), pack-reused 286
Receiving objects: 100% (286/286), 87.53 KiB | 0 bytes/s, done.
Resolving deltas: 100% (157/157), done.
pi@raspberrypi:~$ cd Adafruit_Python_DHT
pi@raspberrypi:~/Adafruit_Python_DHT$ sudo python setup.py install
```

Figura 65: Sair do diretório Adafruit\_Python\_DHT



```
pi@raspberrypi:~$ cd
Arquivo Editar Abas Ajuda
copying Adafruit_DHT.egg-info/PKG-INFO -> build/bdist.linux-armv7l/egg/EGG-INFO
copying Adafruit_DHT.egg-info/SOURCES.txt -> build/bdist.linux-armv7l/egg/EGG-INFO
copying Adafruit_DHT.egg-info/dependency_links.txt -> build/bdist.linux-armv7l/egg/EGG-INFO
copying Adafruit_DHT.egg-info/top_level.txt -> build/bdist.linux-armv7l/egg/EGG-INFO
writing build/bdist.linux-armv7l/egg/EGG-INFO/native_libs.txt
zip_safe flag not set; analyzing archive contents...
creating dist
creating 'dist/Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg' and adding 'build/bdist.linux-armv7l/egg' to it
removing 'build/bdist.linux-armv7l/egg' (and everything under it)
Processing Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg
Copying Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg to /usr/local/lib/python2.7/dist-packages
Removing Adafruit-DHT 1.3.2 from easy-install.pth file
Adding Adafruit-DHT 1.3.4 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg
Processing dependencies for Adafruit-DHT==1.3.4
Finished processing dependencies for Adafruit-DHT==1.3.4
pi@raspberrypi:~/Adafruit_Python_DHT$ cd
```

Após a instalação completa da biblioteca, deve-se sair do diretório da biblioteca e voltar para o diretório padrão que está sendo utilizado como mostra as figuras 65 e 66. A figura 67 mostra a criação e abertura do editor de texto para dht11.py.

Figura 66: Entrar no diretório gpio\_python\_code e criação do arquivo dht11.py

A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window has a menu bar with "Arquivo", "Editar", "Abas", and "Ajuda". The terminal output shows the following steps:

```
FO
copying Adafruit_DHT.egg-info/dependency_links.txt -> build/bdist.linux-armv7l/e
gg/EGG-INFO
copying Adafruit_DHT.egg-info/top_level.txt -> build/bdist.linux-armv7l/egg/EGG-
INFO
writing build/bdist.linux-armv7l/egg/EGG-INFO/native_libs.txt
zip_safe flag not set; analyzing archive contents...
creating dist
creating 'dist/Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg' and adding 'build/bdis
t.linux-armv7l/egg' to it
removing 'build/bdist.linux-armv7l/egg' (and everything under it)
Processing Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg
Copying Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg to /usr/local/lib/python2.7/di
st-packages
Removing Adafruit-DHT 1.3.2 from easy-install.pth file
Adding Adafruit-DHT 1.3.4 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_DHT-1.3.4-py2.7-linux-
armv7l.egg
Processing dependencies for Adafruit-DHT==1.3.4
Finished processing dependencies for Adafruit-DHT==1.3.4
pi@raspberrypi:~/Adafruit_Python_DHT $ cd
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ touch dht11.py
```

Figura 67: Abrir o arquivo dht11.py

A screenshot of a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window has a menu bar with "Arquivo", "Editar", "Abas", and "Ajuda". The terminal output shows the following steps:

```
copying Adafruit_DHT.egg-info/dependency_links.txt -> build/bdist.linux-armv7l/e
gg/EGG-INFO
copying Adafruit_DHT.egg-info/top_level.txt -> build/bdist.linux-armv7l/egg/EGG-
INFO
writing build/bdist.linux-armv7l/egg/EGG-INFO/native_libs.txt
zip_safe flag not set; analyzing archive contents...
creating dist
creating 'dist/Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg' and adding 'build/bdis
t.linux-armv7l/egg' to it
removing 'build/bdist.linux-armv7l/egg' (and everything under it)
Processing Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg
Copying Adafruit_DHT-1.3.4-py2.7-linux-armv7l.egg to /usr/local/lib/python2.7/di
st-packages
Removing Adafruit-DHT 1.3.2 from easy-install.pth file
Adding Adafruit-DHT 1.3.4 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_DHT-1.3.4-py2.7-linux-
armv7l.egg
Processing dependencies for Adafruit-DHT==1.3.4
Finished processing dependencies for Adafruit-DHT==1.3.4
pi@raspberrypi:~/Adafruit_Python_DHT $ cd
pi@raspberrypi:~ $ cd gpio_python_code
pi@raspberrypi:~/gpio_python_code $ touch dht11.py
pi@raspberrypi:~/gpio_python_code $ nano dht11.py
```

Figura 68: Código dht11.py

The screenshot shows a terminal window titled "pi@raspberrypi: ~ gpio\_python\_code". The window contains the Python script "dht11.py". The script imports Adafruit\_DHT and RPi.GPIO, defines a sensor type (DHT11), sets the GPIO mode to BOARD, and initializes a pin (25) for the data line. It prints a message indicating it's reading temperature and humidity. A while loop reads from the sensor, prints the values if successful, or an error message if there's a communication error. The terminal window has a menu bar with "Arquivo", "Editar", "Abas", and "Ajuda". The status bar at the bottom shows keyboard shortcuts for various functions like Help, Save, and Exit.

```
# Carrega as bibliotecas
import Adafruit_DHT
import RPi.GPIO as GPIO
import time

# Define o tipo de sensor
sensor = Adafruit_DHT.DHT11
#sensor = Adafruit_DHT.DHT22

GPIO.setmode(GPIO.BOARD)

# Define a GPIO conectada ao pino de dados do sensor
pin_sensor = 25

# Informações iniciais
print ("*** Lendo os valores de temperatura e umidade");

while(1):
    # Efetua a leitura do sensor
    umid, temp = Adafruit_DHT.read_retry(sensor, pin_sensor);
    # Caso leitura esteja ok, mostra os valores na tela
    if umid is not None and temp is not None:
        print ("Temperatura = {0:0.1f} Umidade = {1:0.1f}\n").format(temp, umid);
        print ("Aguarda 5 segundos para efetuar nova leitura...\n");
        time.sleep(5)
    else:
        # Mensagem de erro de comunicação com o sensor
        print("Falha ao ler dados do DHT11 !!!")
```

O código deste projeto trará algumas mudanças. Nos demais códigos ao importar uma biblioteca foi utilizado o comando `from` + biblioteca + `import` + módulo da biblioteca; com essa nomenclatura somente é importado aquele módulo específico, assim o código fica mais leve e quando necessário aquele módulo pode-se utilizar o comando redigindo somente o nome do módulo. Quando é importado toda a biblioteca, como nesse exemplo; deve-se utilizar o nome da biblioteca seguido do módulo, como a linha 25 do código mostra. Outra função interessante é a `as` que modifica o nome da biblioteca, pois em algumas situações o nome é muito comprido, e ao utilizar o `as` pode-se dar um nome mais curto, facilitando escrever o código.

O funcionamento desse código também é simples, o terceiro pino está conectado a porta 25 do GPIO e ele envia os dados por essa porta para o Raspberry. Enquanto o usuário desejar, será impresso duas mensagens na tela, a dos valores de temperatura e umidade e caso aja um erro na leitura, irá apresentar que houve falha na leitura.

Para a leitura de dados as variáveis `umid` e `temp` recebem respectivamente os

valores de leitura sensor e pino\_sensor que recebem as leituras de umidade e temperatura do sensor. Atribuir esses valores a cada uma das variáveis dá a possibilidade de apresentar ela na tela de forma mais bonita, como é feito na linha 23 do código.

Figura 69: Inicialização do código dht11.py

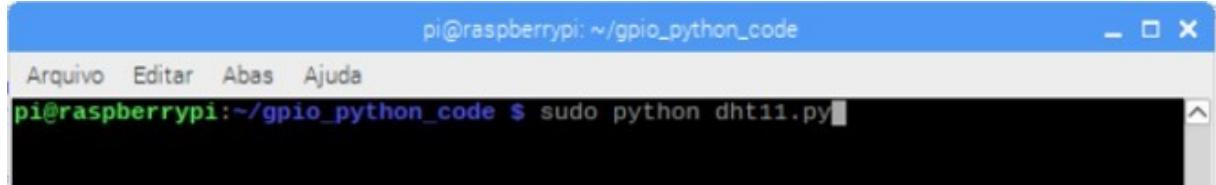
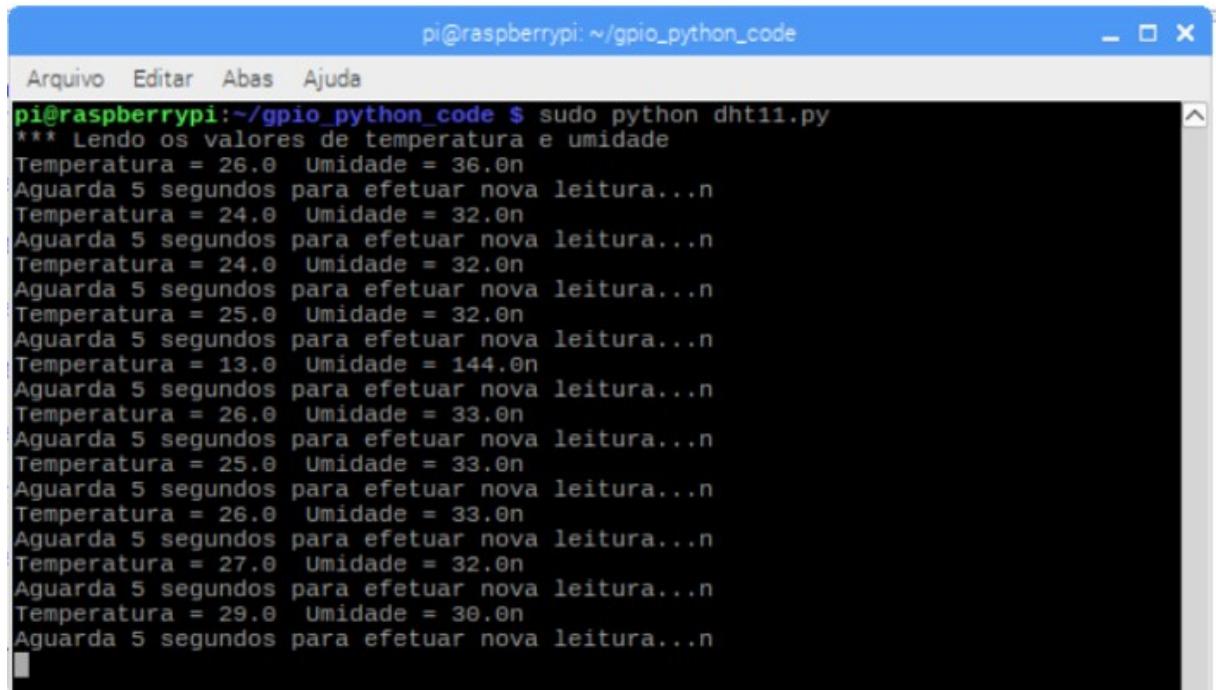


Figura 70: Leitura de valores de temperatura e umidade



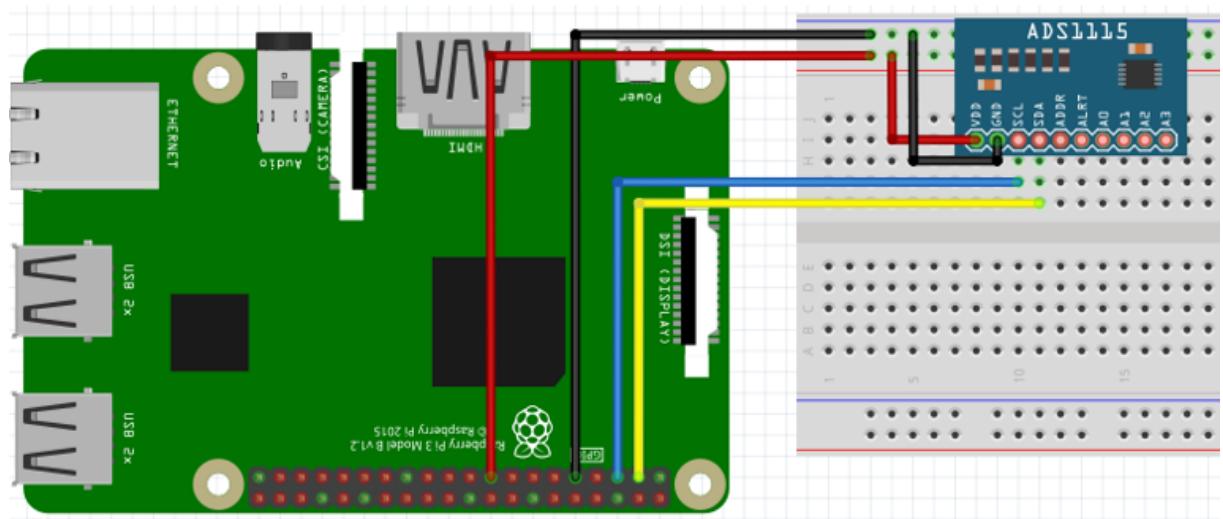
### 3.10 Projeto 8 - Conversor ADS 1115

Materiais:

- 1x ADS 1115
- 4x Jumper

Circuito:

Figura 71: Circuito de teste ADS115.



O ADS1115 é um conversor digital analógico do tipo Sigma/Delta de 16 bits e comunicação I2C. Ele possui quatro canais de entrada que podem ser configurados para medições Single Ended, Diferencial ou Comparador. Esse modelo possui 16 bits de resolução abrindo um leque de 65536 valores possíveis em uma leitura, consegue uma taxa de amostragem de até 860 amostras consumido até  $150\mu\text{A}$ .

Antes de conectar o ADS 1115, deve-se habilitar as portas I2C do Raspberry Pi. Para isso dois passos devem ser feitos, o primeiro instalar o utilitário I2C e o segundo instalar o suporte a I2C para o core ARM e o kernel do Linux.

Para instalar o utilitário I2C, deve-se executar os seguintes comandos no Terminal.

---

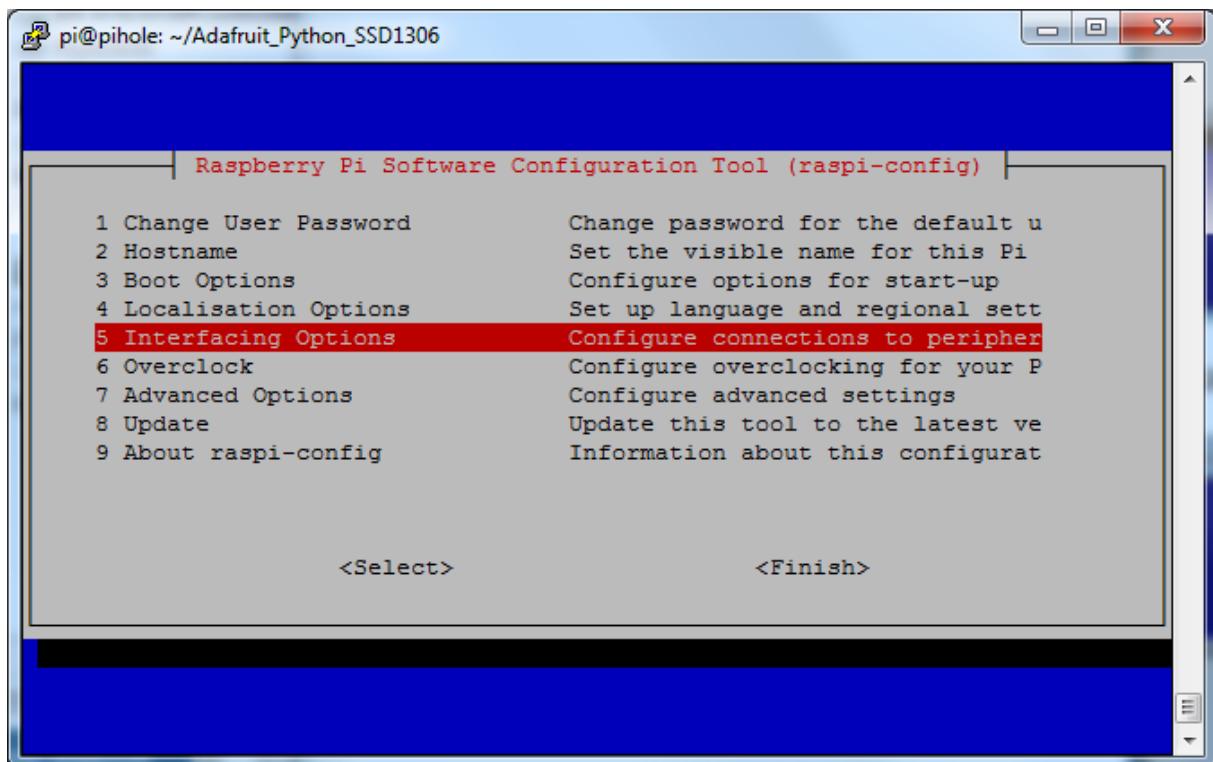
```
sudo apt-get install -y python-smbus  
sudo apt-get install -y i2c-tools
```

---

Para instalar o suporte kernel execute sudo raspi-config e siga as instruções para instalar o suporte a i2c para o core ARM e o kernel do Linux.

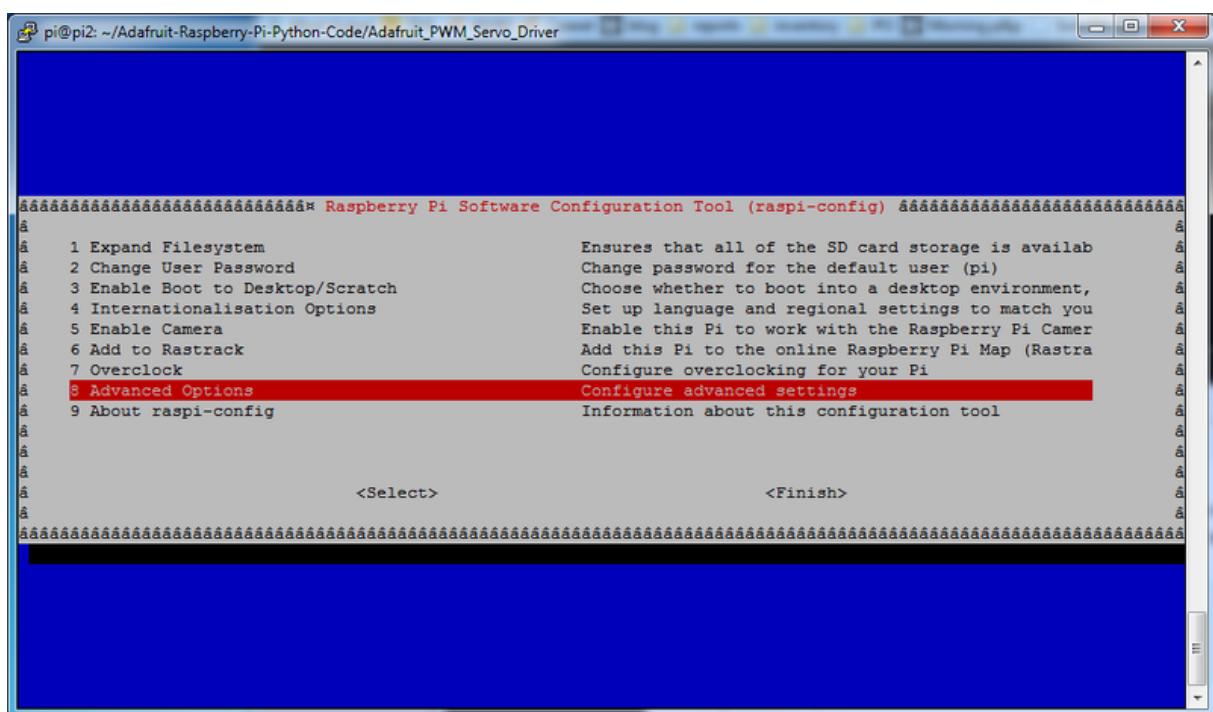
Vá para Opções de Interfaces.

Figura 72: Configuração Software Raspberry versão mais atual.



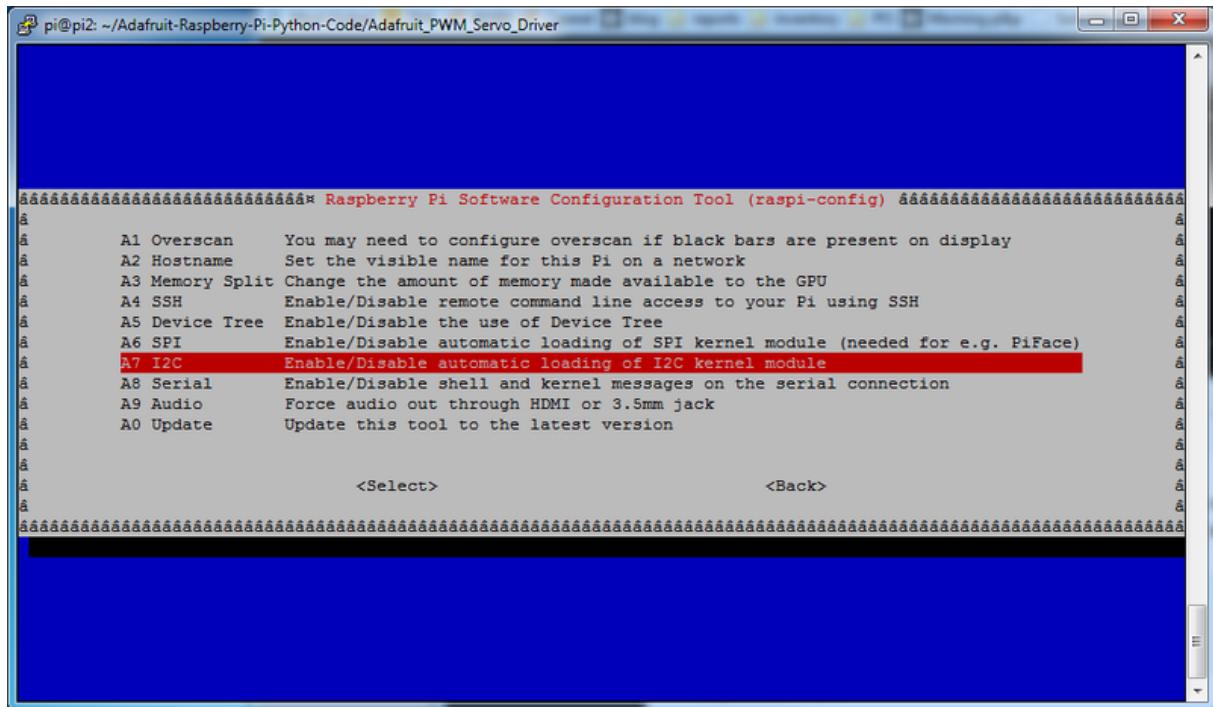
Em versões mais antigas, veja em Avançado.

Figura 73: Configuração Software Raspberry versão mais antiga.



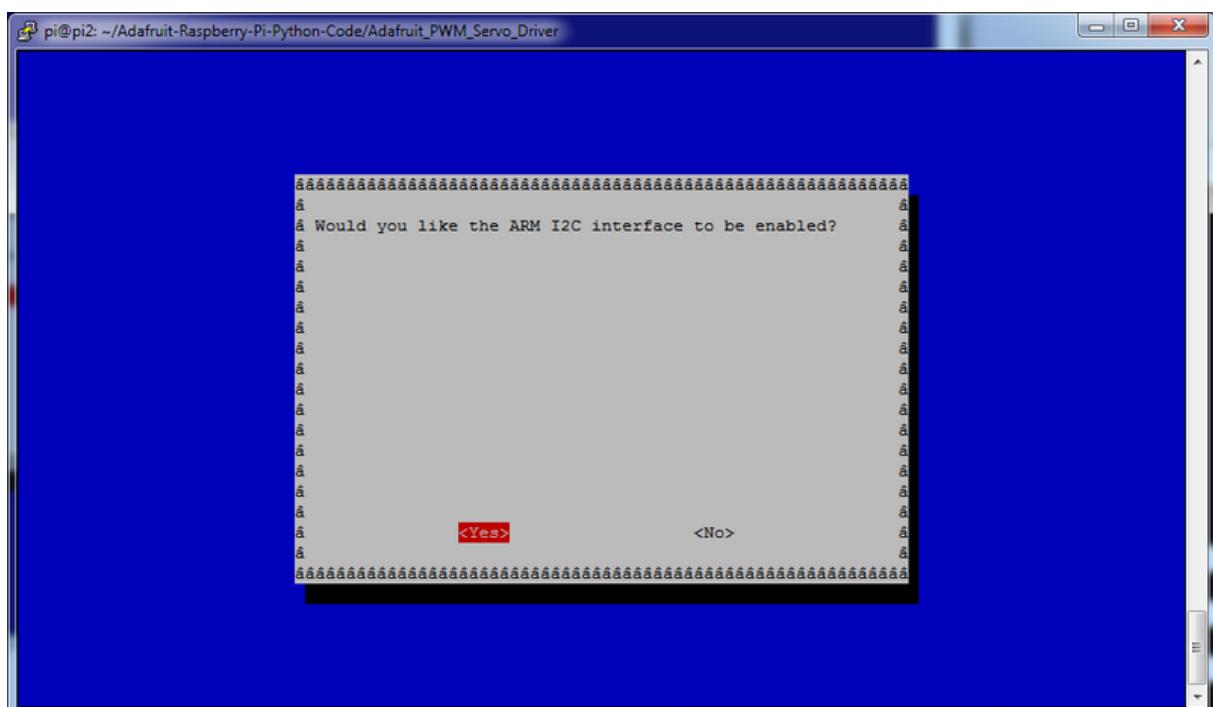
Ir em I2C.

Figura 74: Habilitação I2C.

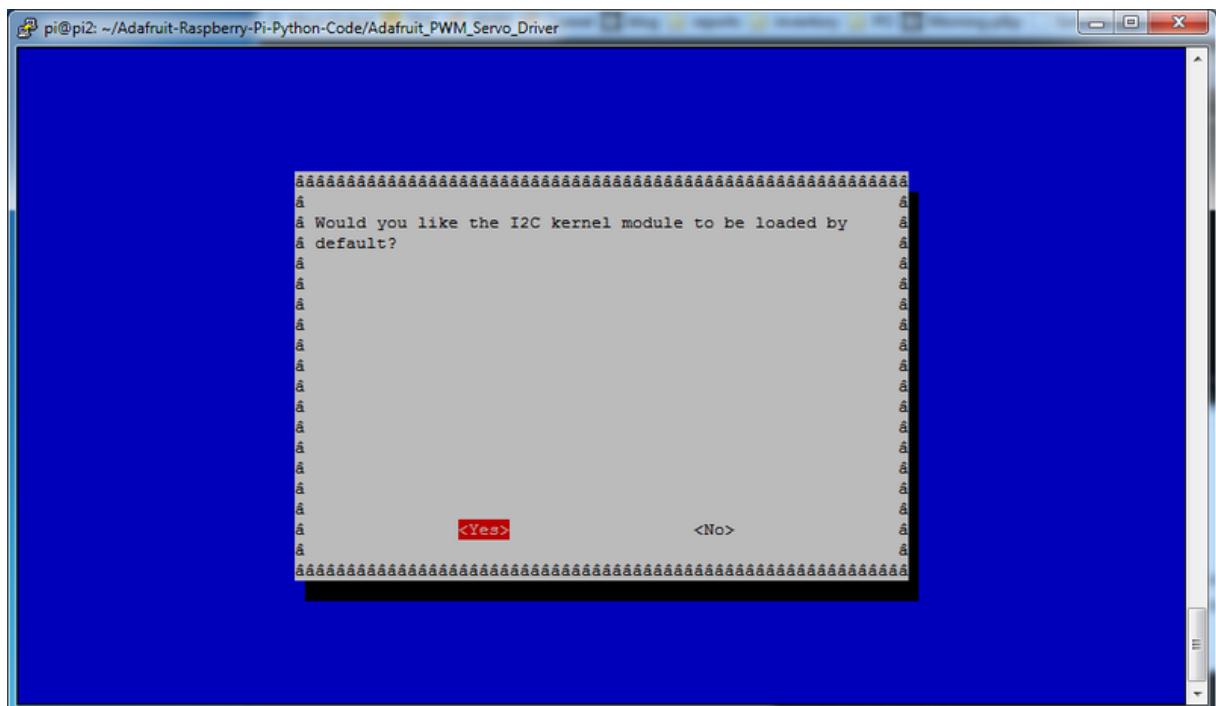


Habilitar.

Figura 75: Habilitar interface ARM I2C.



**Figura 76: Habilitar módulo kernel I2C como padrão.**



Depois disso tudo, reinicie.

Com as portas I2C habilitadas, deve-se fazer a seguinte ligação com ADS1115 e o Raspberry Pi.

Depois de ter conectado o ADS1x15 ao Raspberry Pi, você será instalada a biblioteca Adafruit ADS1x15 Python. Recomenda-se instalar a biblioteca a partir do código-fonte no GitHub, pois ele também baixará exemplos para usar a biblioteca.

Para instalar a partir da fonte no Github, execute os seguintes comandos no terminal do Raspberry Pi.

---

```
sudo apt-get update
sudo apt-get install build-essential python-dev python-smbus git
cd ~
git clone https://github.com/adafruit/Adafruit_Python_ADS1x15.git
cd Adafruit_Python_ADS1x15
sudo python setup.py install
```

---

Figura 77: Instalação da biblioteca Adafruit ADS1115.

```
pi@raspberrypi: ~/Adafruit_Python_ADS1x15
- □ ×

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_GPIO-1.0.3-py2.7.egg
Searching for adafruit-pureio
Reading https://pypi.python.org/simple/adafruit-pureio/
Downloading https://pypi.python.org/packages/55/fa/99b1006fb4bb356762357b297d8db
6ec9ffa13af480692ab72aa4a0dd0c4/Adafruit_PureIO-0.2.1.tar.gz#md5=5b3276059eb55d6
c37429a8413a92029
Best match: Adafruit-PureIO 0.2.1
Processing Adafruit_PureIO-0.2.1.tar.gz
Writing /tmp/easy_install-0wCISv/Adafruit_PureIO-0.2.1/setup.cfg
Running Adafruit_PureIO-0.2.1/setup.py -q bdist_egg --dist-dir /tmp/easy_install
-0wCISv/Adafruit_PureIO-0.2.1/egg-dist-tmp-F0OKpv
zip_safe flag not set; analyzing archive contents...
Moving Adafruit_PureIO-0.2.1-py2.7.egg to /usr/local/lib/python2.7/dist-packages
Adding Adafruit-PureIO 0.2.1 to easy-install.pth file

Installed /usr/local/lib/python2.7/dist-packages/Adafruit_PureIO-0.2.1-py2.7.egg
Searching for spidev==3.3
Best match: spidev 3.3
Adding spidev 3.3 to easy-install.pth file

Using /usr/lib/python2.7/dist-packages
Finished processing dependencies for Adafruit-ADS1x15==1.0.2
pi@raspberrypi:~/Adafruit_Python_ADS1x15 $
```

Para aprender a usar a biblioteca, serão utilizados os códigos de exemplo incluídos nela. Esses exemplos estão na pasta de [examples](#). Mude para essa pasta executando no Pi:

```
cd ~/Adafruit_Python_ADS1x15/examples
```

Será examinando primeiramente o exemplo simpletest.py, que é um exemplo básico de leitura e exibição dos valores do canal ADC. O arquivo deve ser configurado para qual chip estiver usando, pois a biblioteca pode ser utilizada tanto para o ADS 1115, tanto para o ADS 1015. Execute o seguinte comando para abrir o arquivo no editor de texto nano:

```
cd ~/Adafruit_Python_ADS1x15/examples
```

Agora, role até o seguinte bloco de código próximo ao topo:

```
# Create an ADS1115 ADC (16-bit) instance.  
adc = Adafruit_ADS1x15.ADS1115()
```

```
# Or create an ADS1015 ADC (12-bit) instance.  
#adc = Adafruit_ADS1x15.ADS1015()
```

*# Note you can change the I2C address from its default (0x48), and/or the I2C bus by passing in these optional parameters:*

```
#adc = Adafruit_ADS1x15.ADS1015(address=0x49, bus=1)
```

A última linha comentada mostra um uso mais avançado, como a escolha de um endereço personalizado I<sub>2</sub>C ou número de barramento.

Depois deste passo, deve-se salvar o arquivo e executar o código:

```
sudo python simpletest.py
```

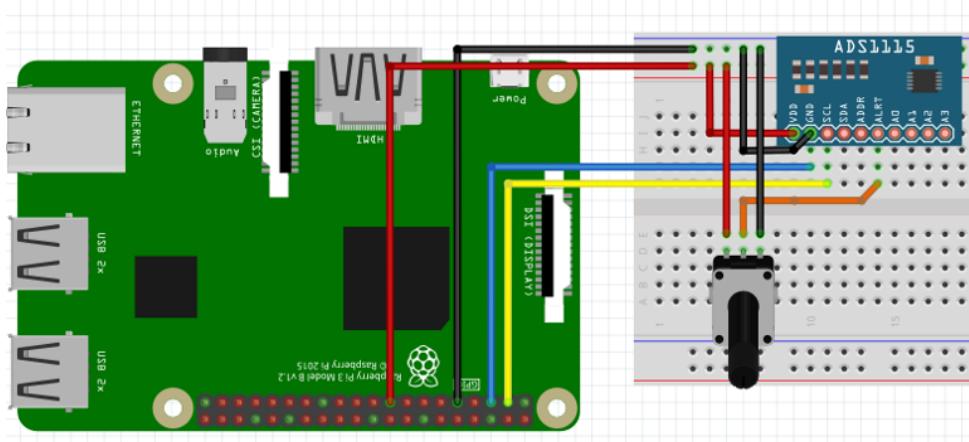
O exemplo imprimirá uma tabela com todos os canais do ADC e seus valores. A cada meio segundo, uma nova linha será impressa com os valores mais recentes do canal.

Figura 78: Tabela com canais do ADS e seus valores.

0	1	2	3
4699	4584	4625	4665
4583	4587	4601	4614
4563	4604	4600	4612
4601	4630	4609	4585
4614	4606	4577	4636
4616	4580	4621	4630
4566	4630	4618	4631
4614	4619	4615	4620
4577	4622	4609	4625
4624	4615	4626	4648
4636	4660	4656	4607
4609	4616	4629	4651

Outra coisa legal de fazer é incrementar o circuito da figura 71 com um potenciômetro, como mostra a figura 79 em um dos canais analógicos. Ao variar o potenciômetro, o valor impresso do canal será alterado.

Figura 79: Circuito de teste ADS115 com potenciômetro.



Cada coluna representa um canal diferente e o cabeçalho na primeira linha mostra o número do canal (de 0 a 3, 4 canais no total). O valor de cada canal é o valor ADC para esse canal. Este é um número que varia de -32768 a 32767 no ADS1115 de 16 bits. Um valor de 0 significa que o sinal está no nível terra (referência), 32767 significa que é igual ou maior que o valor máximo de tensão para o ganho atual (4.096V por padrão) e -32768 significa que é uma tensão negativa abaixo da tensão de referência (por exemplo, se estiver no modo diferencial). Pressione Ctrl-c para parar o exemplo.

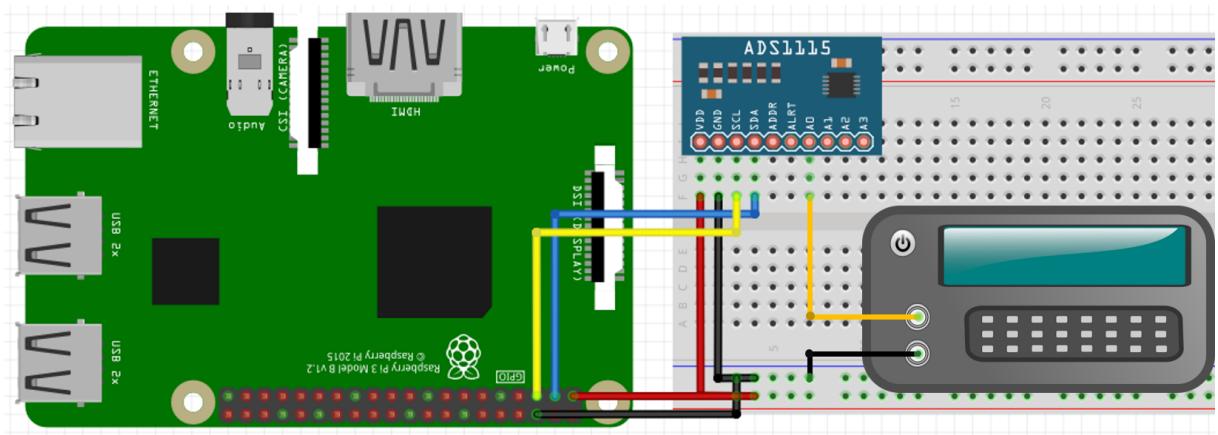
### 3.11 Projeto 9 - Osciloscópio

#### Materiais:

- 1x ADS 1115
- 4x Jumper

#### Círculo:

Figura 80: Circuito projeto osciloscópio com gerador de função.



Um osciloscópio é um instrumento de laboratório comumente usado para exibir e analisar a forma de onda dos sinais eletrônicos. Com efeito, o dispositivo desenha um gráfico da tensão instantânea do sinal em função do tempo. Neste projeto, será replicado as capacidades de visualização de sinal do osciloscópio usando o Raspberry Pi e um módulo conversor analógico para digital.

Como a biblioteca do ADS1115 já foi instalada e as portas I2C já foram habilitadas no projeto 8, o projeto 9 iniciará instalando a biblioteca Matplotlib que proporcionará a geração de gráficos e animações que serão utilizadas para leitura e apresentação gráfica em tempo real. Para instalar a biblioteca basta utilizar o código abaixo.

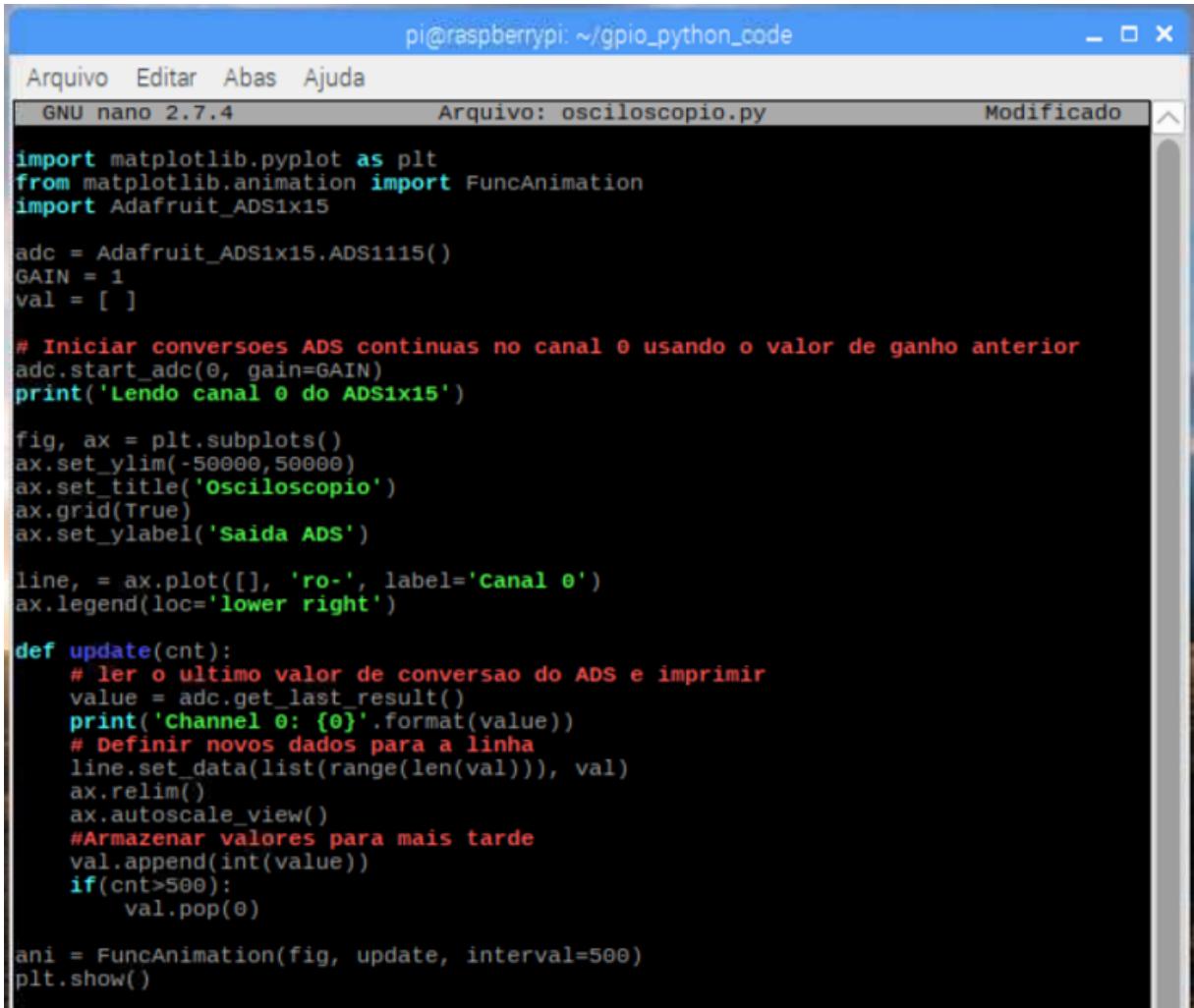
---

```
sudo apt-get install python-matplotlib
```

---

A seguir será apresentado o código para implementação desse projeto e posteriormente algumas análises do código.

Figura 81: Código osciloscópio com gerador de função.



The screenshot shows a terminal window titled "pi@raspberrypi: ~/gpio\_python\_code". The window contains Python code for an oscilloscope. The code imports matplotlib.pyplot and FuncAnimation from matplotlib.animation, and Adafruit\_ADS1x15. It initializes an ADC object, sets the gain to 1, and creates an empty list for values. A comment indicates starting continuous conversion on channel 0. The code then prints a message and starts reading from the ADC. It creates a plot with a single red line for channel 0, setting the y-axis limits from -50000 to 50000. The plot title is 'osciloscopio' and the y-axis label is 'Saida ADS'. The update function reads the last result from the ADC, prints it, defines new data for the line, and appends the value to a list. If the count exceeds 500, the first value is popped from the list. An animation is created with an interval of 500ms, and the plot is shown.

```
pi@raspberrypi: ~/gpio_python_code
Arquivo Editar Abas Ajuda
GNU nano 2.7.4 Arquivo: osciloscopio.py Modificado
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import Adafruit_ADS1x15

adc = Adafruit_ADS1x15.ADS1115()
GAIN = 1
val = []

# Iniciar conversões ADS contínuas no canal 0 usando o valor de ganho anterior
adc.start_adc(0, gain=GAIN)
print('Lendo canal 0 do ADS1x15')

fig, ax = plt.subplots()
ax.set_ylim(-50000,50000)
ax.set_title('osciloscopio')
ax.grid(True)
ax.set_ylabel('Saida ADS')

line, = ax.plot([], 'ro-', label='canal 0')
ax.legend(loc='lower right')

def update(cnt):
    # ler o último valor de conversão do ADS e imprimir
    value = adc.get_last_result()
    print('channel 0: {}'.format(value))
    # Definir novos dados para a linha
    line.set_data(list(range(len(val))), val)
    ax.relim()
    ax.autoscale_view()
    #Armazenar valores para mais tarde
    val.append(int(value))
    if(cnt>500):
        val.pop(0)

ani = FuncAnimation(fig, update, interval=500)
plt.show()
```

Após importar as bibliotecas necessárias e criar uma instância da biblioteca ADS1x15 especificando o ADS1115 ADC. Depois defini-se o ganho do ADC, pois existem diferentes faixas de ganho e devem ser escolhidas com base na tensão esperada na entrada do ADC. Para este projeto, estimando uma faixa de tensão de 0 - 4.09V, será usado ganho de 1. Para obter mais informações sobre o ganho, verifique a folha de dados do ADS1115. Em seguida, deve-se criar as variáveis de matriz que serão usadas para armazenar os dados a serem plotados e outra para servir como contagem.

Na linha 10 será iniciado a conversão ADC contínua especificando o canal usado no ADC, neste caso, o canal 0 e também especifica-se o ganho. Deve-se notar que todos os quatro canais ADC no ADS1115 podem ser lidos ao mesmo tempo, mas 1 canal é suficiente para esta demonstração.

Em seguida, a função plt.subplots(), para criar e definir os atributos do gráfico que manterá a plotagem ao vivo. Primeiro, defini-se os limites do eixo y usando ylim,

após é inserido o título da plotagem. É possível também indicar o canal (como o canal 0 foi indicado) para que possa ser identificado cada sinal quando os quatro canais do ADC estiverem sendo usados e `ax.legend` é usado para especificar que as informações sobre esse sinal (por exemplo, o canal 0) serão exibidas na figura.

Na linha 22 é definida uma função, onde a palavra reservada `def`, explicita a definição da função naquele ponto. Em seguida, entre parênteses, temos o parâmetro `cnt`. Ainda na mesma linha, observe a utilização dos dois pontos (:), que indicam que o código identado nas linhas abaixo faz parte da função que está sendo criada. `Value` receberá os valores da conversão e em seguida, os valores são impressos no terminal apenas para dar outra maneira de confirmar os dados plotados. Após a impressão e, em seguida, é anexado os dados à lista (`val`) criada para armazenar os dados desse canal. Para garantir que os dados mais recentes estejam disponíveis na plotagem, excluímos os dados no índice 0 após cada 500 contagens de dados. O código termina com a `FuncAnimation` que fará como que o gráfico tenha movimento e a plotagem do gráfico com `plt.show()`.

A seguir serão mostrados alguns gráficos gerados com o osciloscópio, sendo o primeiro sem nenhum sinal no canal 0; o segundo com um potenciômetro como na figura 79 e o terceiro com uma onda quadrada e triangular geradas por um gerador de função como na figura 80.

**Figura 82:** Código osciloscópio sem nenhuma entrada no canal 0.

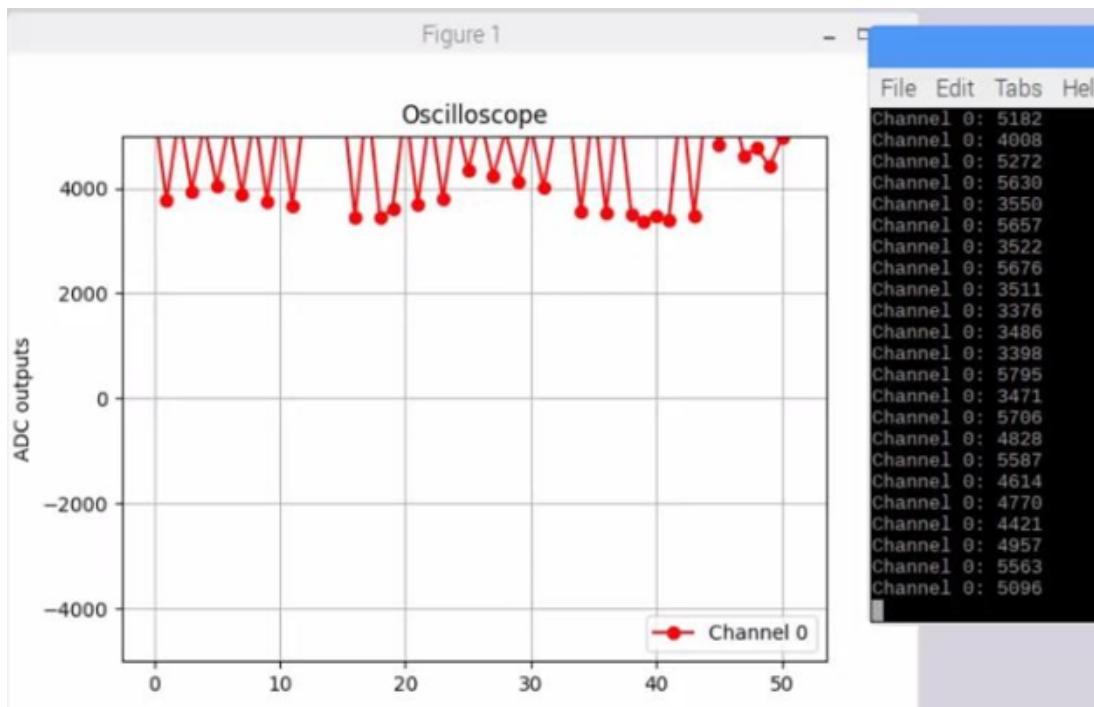


Figura 83: Código osciloscópio com potômetro.

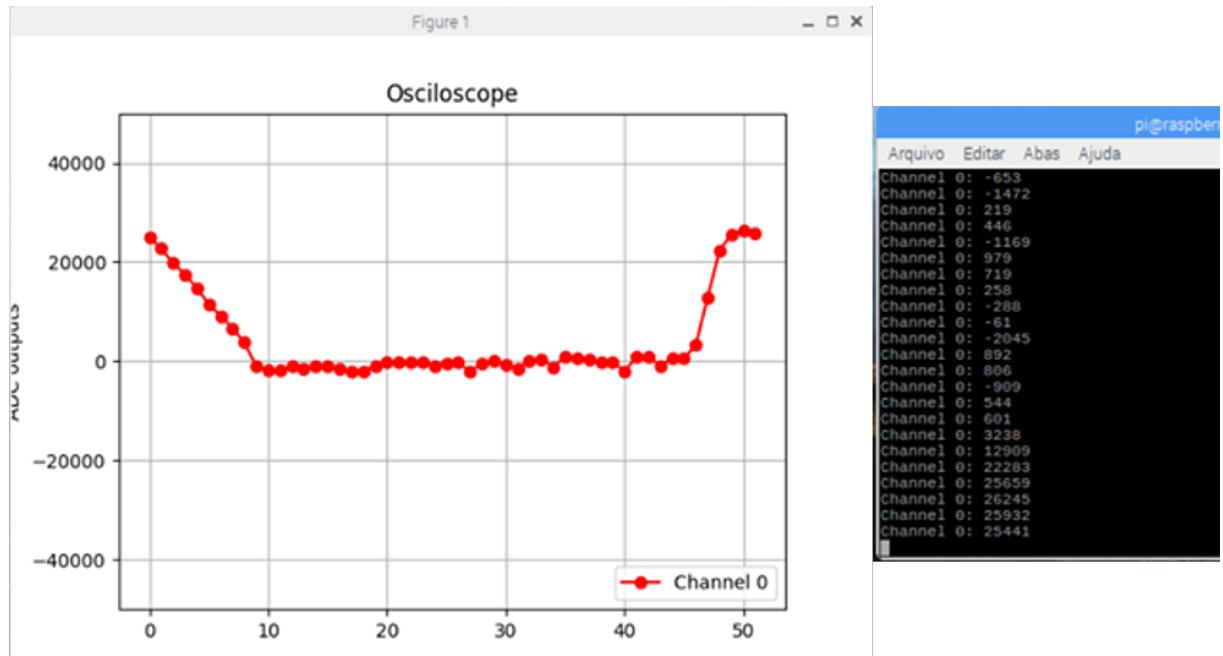


Figura 84: Código osciloscópio com onda quadrada gerada pelo gerador de função.

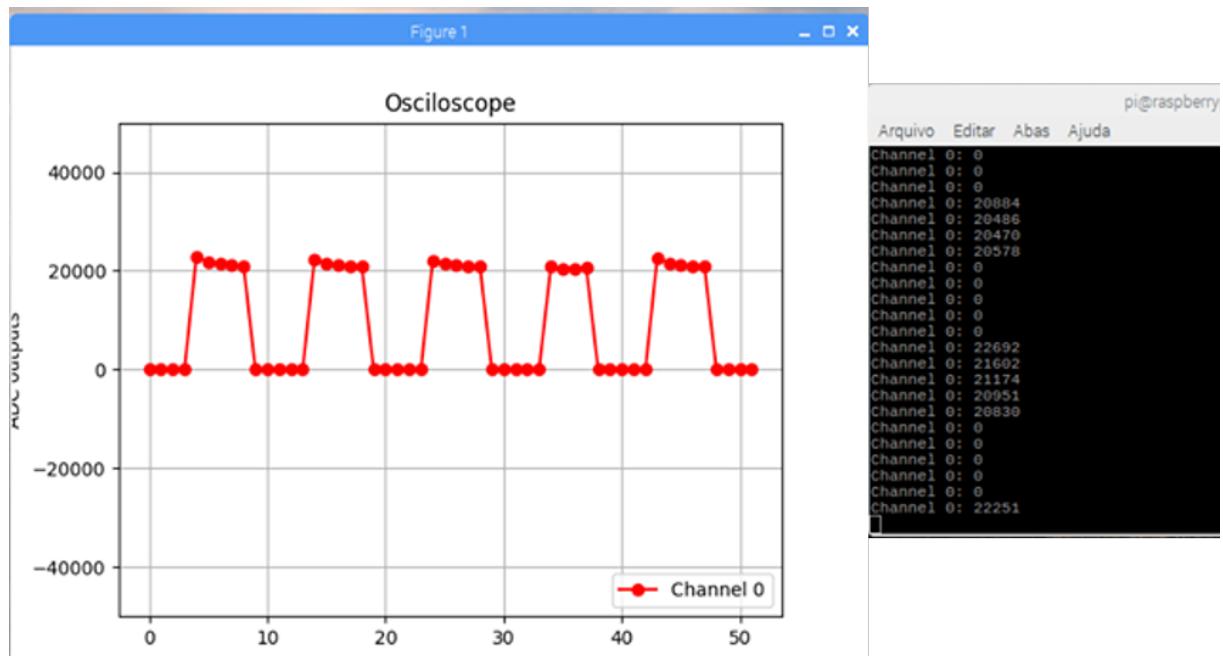
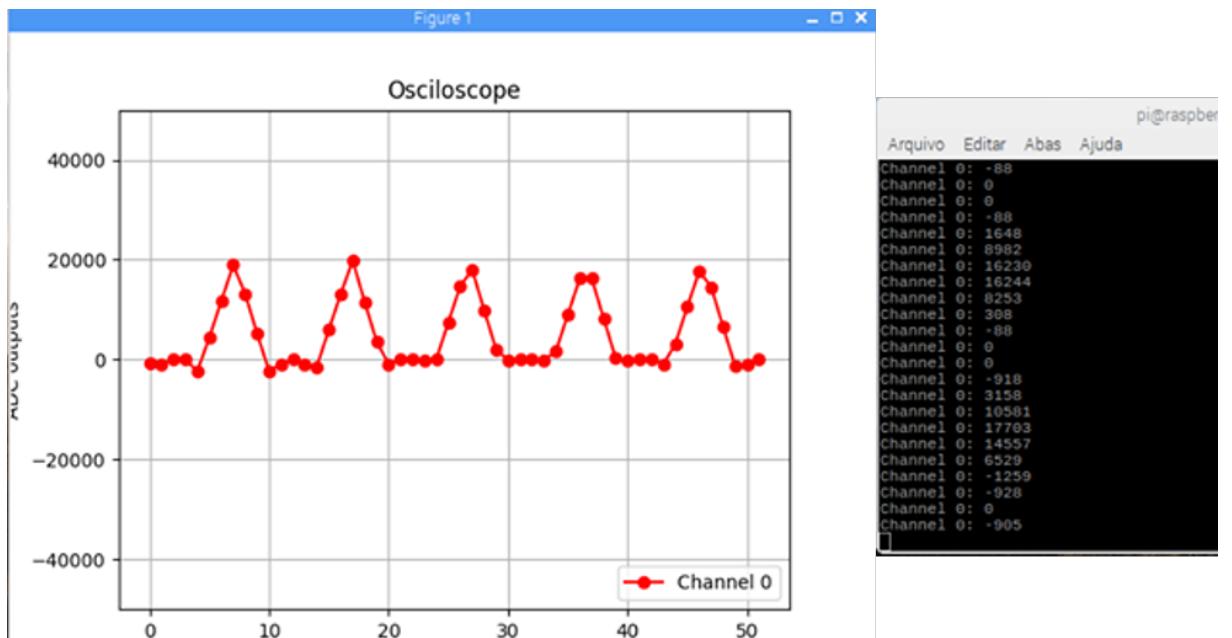


Figura 85: Código osciloscópio com onda triangular gerada pelo gerador de função.



## Referências

- [1] COOK, M.; CRAFT, B.; EVANS, J.; SHAW, A. *Raspberry Pi Projects For Dummies*. 1.ed. New Jersey:John Wiley & Sons, 2015.
- [2] MONK, S.; ZANOLLI, R. *Programando o Raspberry Pi: Primeiros passos com Python*. 1.ed. São Paulo:Novatec, 2013.
- [3] RICHARDSON, M.; WALLACE, S.; WALLACE, S.P *Getting Started with Raspberry Pi*. 1.ed. Massachusetts:O'Reilly Media, 2012.
- [4] ROBINSON,A.; COOK, M.; *Raspberry Pi Projects*. 1.ed. New Jersey:John Wiley & Sons, 2013.
- [5] ZANETTI, H.A.P.; OLIVEIRA, C.L.V.; NABARRO, C.B.M. *Raspberry Pi Descomplicado*. 1.ed. São Paulo:Érica, 2018.