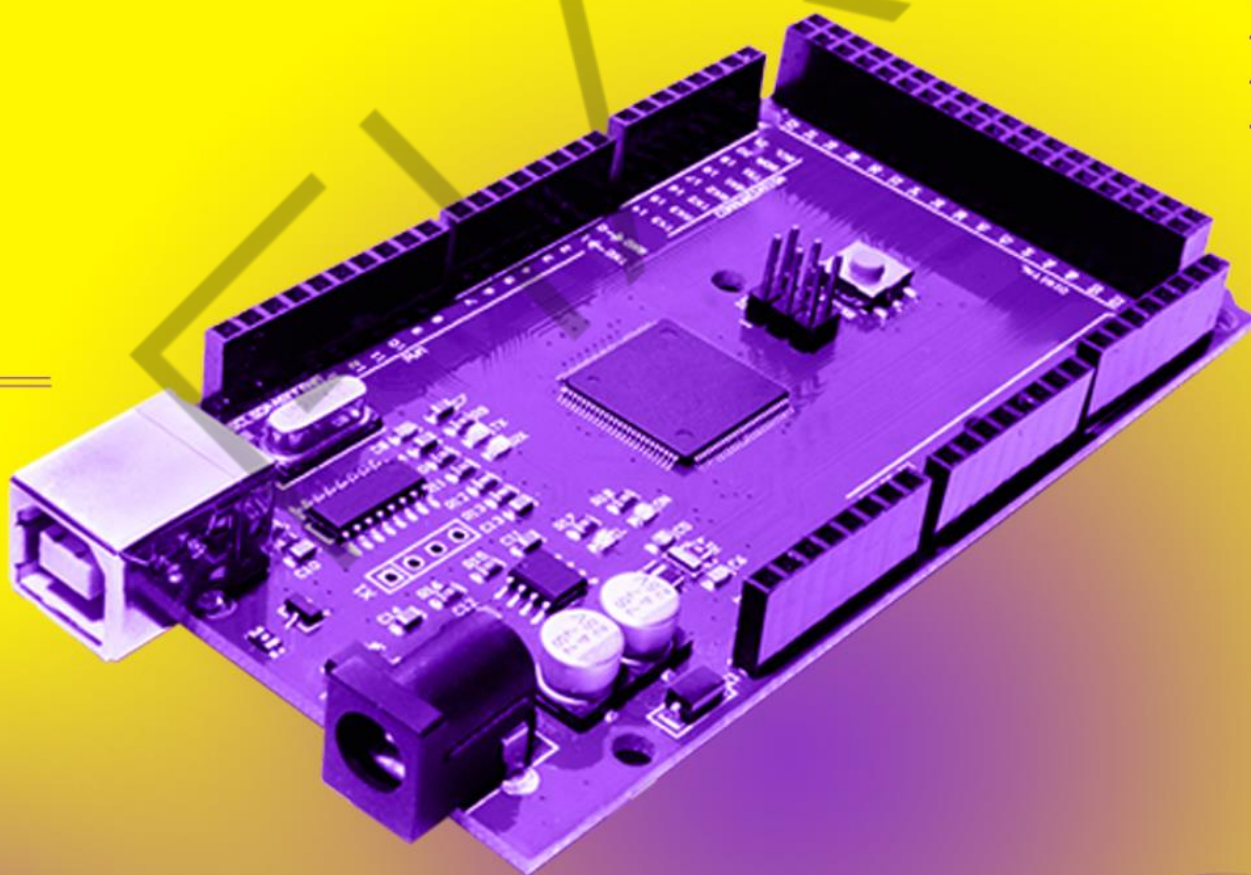


PYTHON

Phython x IOT

HUMBERTO D. DE SOUSA



8

LISTA DE FIGURAS

Figura 8.1 – Arduino (esquerda) e Raspberry (direita).....	7
Figura 8.2 – Tela inicial do Arduino Sketch.	8
Figura 8.3 – Arduino.....	9
Figura 8.4 – Porta serial não disponível.	11
Figura 8.5 – Led e Resistor.	14
Figura 8.6 – Imagem da ligação proposta.	15
Figura 8.7 – Código do Arduino para receber dados.....	16
Figura 8.8 – Compilando um código Arduino.	17
Figura 8.9 – Transferindo código para Arduino.	18
Figura 8.10 – Sensor de luminosidade LDR.....	23
Figura 8.11 – Protoboard.	23
Figura 8.12 – Cabos (jumper) machos.....	24
Figura 8.13 – Resistores marrom, preto e laranja.	24
Figura 8.14 – Esquema de ligação do sensor de luminosidade (fotorresistor).....	25

LISTA DE CÓDIGOS-FONTE

Código-fonte 8.1 – Utilizando o pacote “serial”	10
Código-fonte 8.2 – Listando as portas.....	11
Código-fonte 8.3 – Código Python para enviar dados.....	18
Código-fonte 8.4 – Código Arduino Sketch para enviar dados.....	20
Código-fonte 8.5 – Código Python para receber dados do Arduino.	21
Código-fonte 8.6 – Código para retornar valores capturados pelo sensor de luminosidade.	25
Código-fonte 8.7 – Recebendo no Python os dados devolvidos pelo sensor de luminosidade.	26
Código-fonte 8.8 – Persistindo dados do sensor.....	28

SUMÁRIO

8 PYTHON X IOT	5
8.1 Momento “flash back”	5
8.2 O lado do Python.....	9
8.3 Enviando dados do Python para o Arduino	14
8.3.1 Recebendo dados – Lado IoT	14
8.3.2 Enviando dados – Lado Python.....	18
8.4 Enviando dados do Arduino para o Python	20
8.4.1 Enviando dados – Lado IoT.....	20
8.4.2 Recebendo dados – Lado Python	21
8.5 Persistindo os dados recolhidos.....	22
REFERÊNCIAS	30

8 PYTHON X IOT

No capítulo anterior, pudemos aprender como o Python pode ser importante dentro de um ambiente de redes de computadores e como ele se integra aos protocolos de transmissão de dados ou arquivos. Continuando a ideia de apresentar para você as diversas áreas nas quais ele pode ser aplicado, chegou o momento de interligarmos o Python aos dispositivos IoT.

Com o crescimento exponencial da “Internet das Coisas”, cresce também a necessidade de gerenciá-las, protegê-las e desenvolver ainda mais aplicações voltadas para esse mercado. O dispositivo fará o papel de capturar o que se deseja, como: som, imagem, temperatura, áudio, tremores, umidade, luminosidade, entre tantos outros dados que podem ser capturados por meio de sensores.

Mas, então, o que fazer com todos esses dados? Como transformá-los em informações, para que possam auxiliar na tomada de decisão, seja ela gerencial, executiva ou até mesmo legislativa? Pois bem, iremos fazer a persistência dos dados coletados por dispositivos IoT para que, posteriormente, você utilize o que aprendeu em *Machine Learning* ou ainda o que irá aprender em *Data Science*, para que possa converter todos esses dados persistidos em informações relevantes, seja para a empresa, para um projeto pessoal ou até mesmo para um projeto governamental. Então, vamos lá!

8.1 Momento “flash back”

Antes de começarmos a montar um novo projeto, vamos alinhar alguns itens importantes para melhor compreensão e aproveitamento de todos neste capítulo.

Primeiramente, é válido falar que todas as imagens aqui apresentadas, sobre IoT, são imagens pautadas no Arduino, ou seja, se ainda não adquiriu um kit Arduino, recomendo que o faça, para que possa praticar e criar soluções para automação de tarefas do seu dia a dia, sem se preocupar com escalabilidade ou com o negócio em si.

Procure pensar em soluções para você, siga avançando nas ideias até desenvolver um projeto que talvez possa gerar uma startup e, quem sabe, você

possa ter em mãos o novo “unicórnio” brasileiro, assim como a “99 Taxi”. Também é importante dizer que a ideia central deste capítulo é estabelecer a comunicação entre Python e um dispositivo, seja passando ou recebendo dados. Informações mais detalhadas sobre IoT você encontrará nos capítulos específicos sobre o assunto.

Sobre os Arduinos, vale lembrar que se trata de uma plataforma de hardware para a rápida execução de projetos eletrônicos que, em conjunto com vários sensores disponíveis no mercado, permite interagir com o meio. É um projeto *open source*, tanto no que tange ao hardware quanto ao software, ou seja, caso tenha conhecimentos em eletrônica, você mesmo pode desenvolver a sua placa de Arduino. Pelo simples fato de ser *open source*, o seu custo é tão reduzido, assim como os vários componentes que podemos conectar à placa, como sensores de luminosidade, umidade, temperatura, leds, gases, sons e tantos outros.

Esse funcionamento é diferente de quando o comparamos com Raspberry, que possui uma aplicação bem diversa, partindo do princípio de que o Raspberry nada mais é do que um microcomputador extremamente portátil, ou seja, possui sistema operacional e todos os requisitos que um microcomputador precisa possuir. Para esclarecer melhor a diferença de aplicação entre esses dois dispositivos, “Arduino” e “Raspberry”, podemos pensar no seguinte exemplo: desejamos montar uma placa que terá a função de identificar a umidade da terra que está no vaso da sua planta favorita, e quando a umidade estiver baixa, o dispositivo deverá enviar um sinal para um aplicativo que está no seu smartphone. Qual a melhor plataforma?

Primeiro, a proposta pode ser resolvida com qualquer uma das duas plataformas, a diferença está, primeiramente, no custo, com Raspberry você investirá quatro ou cinco vezes mais. A segunda diferença é que como o Raspberry é um microcomputador, ele consome muito mais energia que um Arduino. Se você optasse por um Raspberry teria que mantê-lo conectado em uma fonte de energia ou se preocupar frequentemente se a bateria dele não está precisando de carga. Diferente de um Arduino que, dependendo da aplicação, pode até mesmo se manter por meio de uma bateria simples por anos. Ou seja, para um projeto como esse, o ideal seria desenvolver sobre o Arduino mesmo.

Vamos para outro exemplo: caso eu queira desenvolver uma central de videoconferência para a empresa, por meio de uma central VoIP, a fim de evitar

custos de transmissão. Qual seria a melhor plataforma? Sem dúvida, o Raspberry; percebe o nível de processamento que será exigido (transmissão de imagem e som em uma rede) e que não é uma solução que ficará 24 horas no ar. Esses já são motivos suficientes para eleger o Raspberry como plataforma ideal. Seria inviável fazer com que um Arduino realizasse todo o processamento de áudio e vídeo para o trânsito em uma rede de computadores.

O poder de processamento do Arduino é bem limitado quando comparado ao Raspberry. Se ainda restaram dúvidas, procure pesquisar na Internet projetos realizados com Raspberry e projetos realizados com Arduino, o que irá ampliar bastante a sua criatividade para o desenvolvimento de soluções envolvendo IoT.

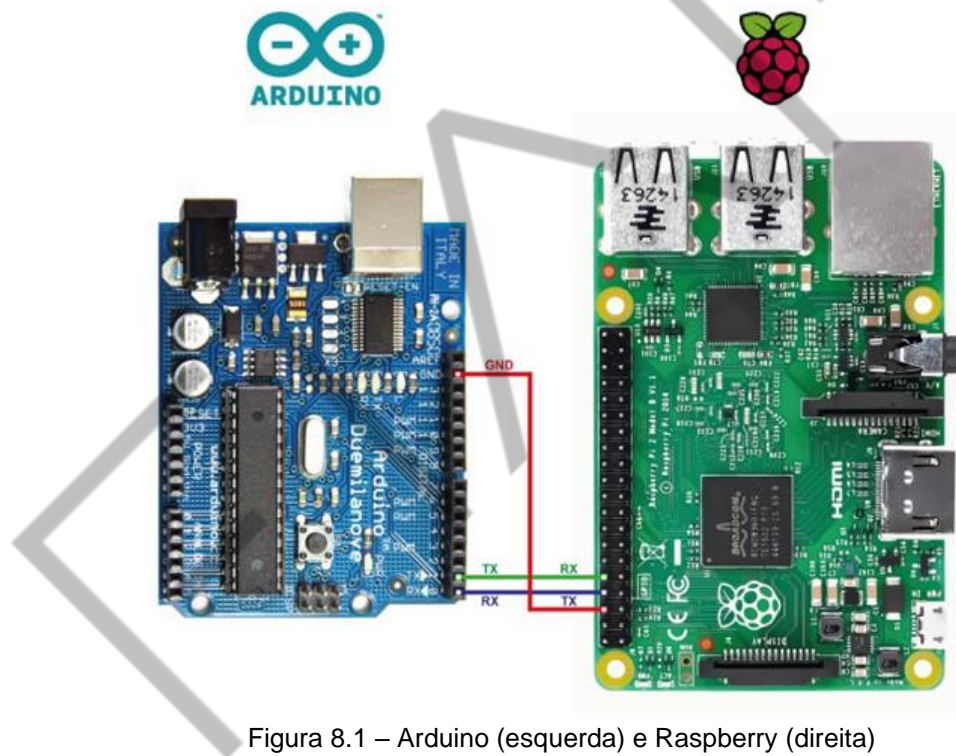


Figura 8.1 – Arduino (esquerda) e Raspberry (direita)

Fonte: <https://www.pddnet.com> (2018)

Outro fator importante para lembrarmos é que toda a programação que gerencia a plataforma Arduino é baseada em C. Mas, conforme você já observou em capítulos anteriores, ele possui uma IDE (ambiente de desenvolvimento) simples que facilita a transmissão do código para a placa. E os códigos utilizados para esse fim, normalmente, são bem simples e não requerem grandes conhecimentos em programação; com o básico sobre tomadas de decisões, laços de repetições, variáveis e funções, já será possível desenvolver diversas aplicações. Caso ainda

não tenha baixado o Arduino Sketch, utilize o link: <https://www.arduino.cc/en/Main/Software> .

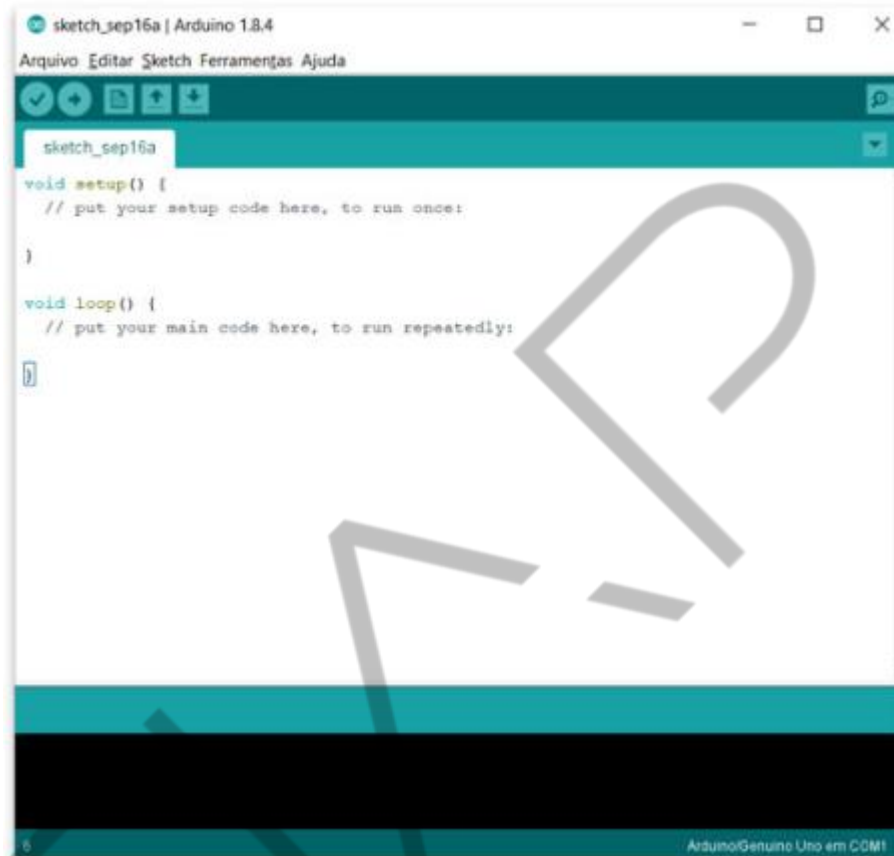


Figura 8.2 – Tela inicial do Arduino Sketch
Fonte: Desenvolvida pelo autor (2018)

Conforme podemos observar na figura apresentada, encontraremos duas funções básicas:

- `void setup()` => é executada apenas ao ligar o Arduino ou quando pressionamos o botão “reset”; e
- `void loop()` => é executada sequencialmente infinitas vezes.

A ideia principal do Arduino é facilitar e baratear a criação de projetos que envolvam Internet das Coisas. Ele pode estar conectado diretamente com um PC, via USB no modo FTDI, sendo mapeado em uma porta serial ou ainda, conforme falamos anteriormente, ter como fonte de energia uma bateria. Precisaremos, para este capítulo, basicamente identificar os seguintes itens:



Figura 8.3 – Arduino
Fonte: Imagem do Google (2018)

As portas de entradas e saídas, tanto analógicas quanto digitais, servirão para conectarmos os sensores e/ou atuadores, e as portas para alimentação serão utilizadas para enviar energia para o funcionamento dos sensores e/ou atuadores.

8.2 O lado do Python

Agora, que estabelecemos pontos importantes de conhecimentos sobre a IoT que iremos utilizar, vamos começar a verificar como o Python entrará nesse cenário.

Começaremos pela biblioteca “serial”, que será a responsável por realizar a troca de dados entre o Arduino e o Python por meio de uma comunicação que irá se estabelecer através de uma porta serial. Caso você não tenha essa biblioteca instalada, abra o console (terminal) do seu sistema operacional e digite:

Se for Windows:

- `pip install pyserial`

Se for Linux:

- apt-get install python-serial

Após concluir a instalação, vamos criar um novo “Directory” em nosso projeto e, dentro dele, adicionar um “Python File”, chamado “**Pacote_Serial.py**”, no qual será digitado o seguinte código (para Windows):

```
import serial
conexao = serial.Serial('COM3', 115200, timeout=0.5)
```

Código-fonte 8.1 – Utilizando o pacote “serial”
Fonte: Elaborado pelo autor (2018)

A partir do código apresentado, observamos a importação da biblioteca “serial”, seguida da criação de um objeto, denominado “conexao”, que será responsável por representar a saída serial (por meio do método “Serial()”) “COM3”, sendo essa saída serial a que você está utilizando no Arduino Sketch.

O segundo parâmetro do método “serial()” indica o “Baud Rate”, que está associado à capacidade de transmissão de bits por segundo. O fator mais importante sobre o “baud rate” é que os dispositivos conectados devem estar com a mesma faixa de transmissão, ou seja, se você definir 9600 dentro do Python, o seu Arduino deverá estar operando também na faixa de 9600; com valores diferentes, dificilmente a transmissão de dados ocorrerá de maneira satisfatória. Para definição dessa taxa, existem alguns valores padrões: 75, 300, 600, 1200, 2400, 4800, 9600, 19200, 28880, 38400, 57600 e 115200. O mais comumente utilizado é o 9600.

Quanto maior o valor, maior será a capacidade de transmissão, entretanto, observe com cuidado dois pontos: quanto maior o valor, menor poderá ser a distância entre os dispositivos interligados para que consigam realizar a transmissão de modo satisfatório e, principalmente, se utilizar um valor muito alto, será que sua aplicação conseguirá processar todos os dados que ela receberá? Por isso, defina sempre dentro dos valores padrões e comuns, a não ser que tenha certeza de que o valor utilizado atenda perfeitamente à sua solução.

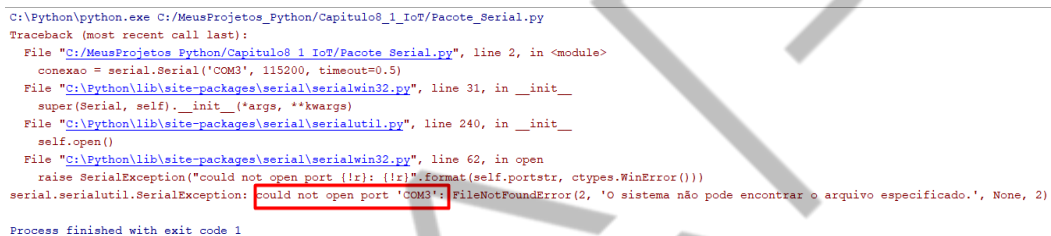
O último parâmetro refere-se ao tempo (em segundos) máximo que sua aplicação irá esperar até recolher os dados enviados pelo dispositivo. Caso o dispositivo envie os dados antes do tempo, sua aplicação irá recolhê-los. Se o dispositivo não enviar os dados dentro do tempo definido, a sua aplicação irá recolher os dados que já estiverem disponíveis. De acordo com o exemplo apresentado, nossa aplicação irá recolher os dados a cada meio segundo, ou

quando todos os dados já estiverem prontos. Se você utiliza uma distribuição Linux, a segunda linha deveria ser alterada para algo como:

```
conexao = serial.Serial('/dev/ttyUSB0', 115200)
```

Nesse caso, a principal mudança está relacionada à forma como o sistema operacional identifica a porta serial, no Linux, as portas são identificadas por: **/dev/ttyUSB0**, **/dev/ttyUSB1** e assim sucessivamente. Já no Windows, as portas seriais são identificadas por **COM1**, **COM2** e assim sucessivamente.

Se você executar o seu código, irá verificar uma mensagem de erro, como a apresentada na imagem seguinte:



```
C:\Python\python.exe C:/MeusProjetos/Python/Capitulo8_1_IoT/Pacote_Serial.py
Traceback (most recent call last):
  File "C:/MeusProjetos/Python/Capitulo8_1_IoT/Pacote_Serial.py", line 2, in <module>
    conexao = serial.Serial('COM3', 115200, timeout=0.5)
  File "C:\Python\lib\site-packages\serial\serialwin32.py", line 31, in __init__
    super(Serial, self).__init__(*args, **kwargs)
  File "C:\Python\lib\site-packages\serial\serialutil.py", line 240, in __init__
    self.open()
  File "C:\Python\lib\site-packages\serial\serialwin32.py", line 62, in open
    raise SerialException("could not open port {}: {}".format(self.portstr, ctypes.WinError()))
serial.serialutil.SerialException: could not open port 'COM3': FileNotFoundEr
```

Figura 8.4 – Porta serial não disponível
Fonte: Imagem do Google (2018)

O erro acima representa que a porta “COM3” não está disponível. Esse erro somente não será apresentado para você, caso já esteja com o Arduino conectado no seu computador e também se o Arduino estiver conectado na porta COM3. E quando não sabemos exatamente em qual porta o nosso dispositivo está conectado? Poderíamos abrir o Arduino Sketch e verificar atrás dele, mas essa ação ficaria pouco prática. Podemos fazer isso dentro do nosso próprio código Python, acrescentando as seguintes linhas:

```
import serial
conexao=""
for porta in range(10):
    try:
        conexao = serial.Serial("COM"+str(porta), 115200, timeout=0.5)
        print("Conectado na porta: ", conexao.portstr)
        break
    except serial.SerialException:
        pass
if conexao!="":
    conexao.close()
    print("Conexão encerrada")
else:
    print("Sem portas disponíveis")
```

Código-fonte 8.2 – Listando as portas
Fonte: Elaborado pelo autor (2018)

Esse código está próprio para Windows, caso utilize Linux deverá alterar a linha:

- `conexao = serial.Serial("COM"+str(porta), 115200, timeout=0.5)`

Por:

- `conexao = serial.Serial("/dev/ttyUSB"+str(porta), 115200)`

Com a execução do código, deverá ocorrer o seguinte resultado: se o seu Arduino estiver devidamente conectado ao seu computador, deverá aparecer a porta que está conectada e depois outra mensagem avisando que a conexão foi encerrada. Por outro lado, caso não tenha nenhum Arduino conectado ao seu computador, ele deverá exibir a mensagem “Sem portas disponíveis”.

Vamos explicar as linhas adicionadas:

- **Nas linhas 1 e 2:** estamos importando a classe serial e criando uma variável “conexao” sem qualquer conteúdo do tipo *string*.
- **Na linha 3:** abrimos um for que irá caracterizar um laço de repetição de 0 a 10. Esse número irá representar a porta à qual o Arduino estará conectado. Se o seu Arduino estiver na porta 11, ela não será encontrada por esse for, você deverá aumentar o número de repetições do laço, entretanto, dificilmente seu dispositivo estará em uma porta serial maior que 9.
- **Na linha 4:** abrimos um bloco “try”, que sempre virá acompanhado de um bloco “except”. Dentro do “try”, colocamos as linhas que eventualmente podem disparar uma exceção. O que é uma exceção? É um evento que o programador não consegue controlar, por exemplo, não consigo controlar, quando esse código for executado, se o usuário irá ter conectado o Arduino ou não. E, conforme fizemos anteriormente, no primeiro teste com a saída serial, caso o Arduino não esteja conectado, será exibida uma mensagem de erro e o programa será abortado. Não desejamos que isso ocorra, se a porta não estiver disponível, quero que ele procure na próxima

e assim sucessivamente, até a porta “9” e, caso não encontre qualquer porta ativa, que simplesmente dê uma mensagem para o usuário e não aborte o sistema. Por isso, colocamos as linhas 5, 6 e 7 dentro do “try”, indicando que, se houver qualquer exceção, ou seja, se a linha 5 não conseguir estabelecer conexão com a saída serial, o código irá pular para a linha “8”. Então, executará a linha “9” (que está dentro do bloco except) e irá ignorar a exceção (comando “pass”), como consequência, não abortará o sistema e permanecerá no laço executando o sistema normalmente.

- **Linha 5:** linha que tenta estabelecer a conexão com a porta serial, de acordo com o valor que estiver valendo para o laço (valor da variável “porta” que irá variar entre 0 e 9). Se a conexão for estabelecida com sucesso, passará para a linha 6, caso dê a exceção, ou seja, se não conseguir conexão, irá direto para a linha 8 (bloco except), que executará a linha 9 e, então, irá ignorar a exceção e continuar a execução do laço.
- **Linha 6:** somente será executada se a linha anterior ocorrer com sucesso, então, irá mostrar para o usuário o nome da porta que conseguiu conectar, por meio do valor “conexao.portstr”, que retorna a porta no formato *string*.
- **Linha 7:** irá abandonar o laço, uma vez que a conexão já foi estabelecida com sucesso.
- **Linha 8:** somente será executada se a conexão não for bem-sucedida e lançar uma exceção do tipo “serial.SerialException”, ou seja, a conexão não ocorreu.
- **Linha 9:** esta linha faz com que o sistema não seja abortado, caso ocorra a exceção “serial.SerialException”.
- **Linhas 10, 11 e 12:** se o conteúdo da variável não for vazio (significa que uma conexão ocorreu com sucesso), irá fechar a conexão e exibir a mensagem “Conexao encerrada”.
- **Linhas 13 e 14:** caso contrário, ou seja, se nenhuma conexão foi realizada com sucesso, irá exibir a mensagem “Sem portas disponíveis”.

8.3 Enviando dados do Python para o Arduino

Vamos preparar um pequeno exemplo para entendermos como podemos mandar dados do Python para um dispositivo Arduino. Vários seriam os exemplos a ser explorados, como seu Python poderia monitorar um tráfego de dados em uma rede e, caso o tráfego em uma estação saísse de um padrão, você enviaria um dado para que um dispositivo Arduino disparasse uma sirene, por exemplo. Iremos fazer com que o nosso código envie um sinal para ligar ou desligar um led. Começaremos preparando o lado IoT.

8.3.1 Recebendo dados – Lado IoT

Separe uma *protoboard*, dois conectores macho (um verde e um branco), um *led*, um resistor (marrom, laranja, laranja); na imagem a seguir, temos o resistor e o led:

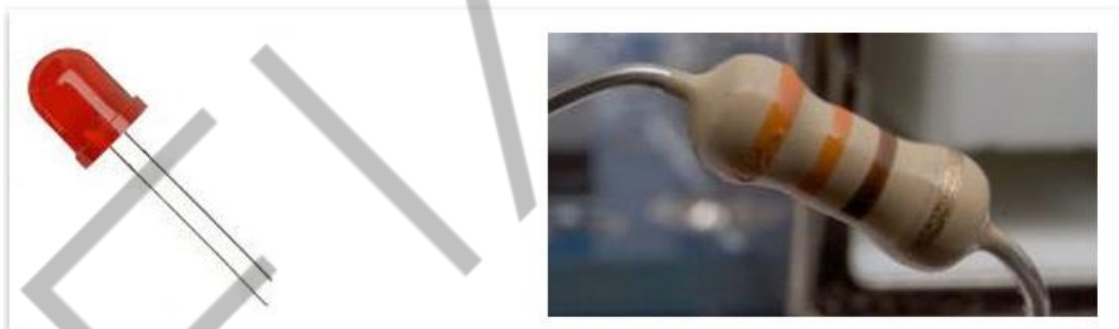


Figura 8.5 – Led e Resistor
Fonte: Imagem do Google (2018)

Agora, seguiremos as seguintes etapas:

- Insira o led na *protoboard*;
- Na mesma fileira da perna maior do led, ligue uma das pontas do resistor;
- Ligue a outra ponta do resistor em outra fileira vazia da *protoboard* e, nessa mesma fileira, conecte uma das pontas do fio branco;
- A outra extremidade do fio branco irá na porta 10 do Arduino; e

- Ligue o fio verde na fileira da perna menor do led, e a outra extremidade do fio verde, você deverá ligar em uma porta GND do Arduino.

Você terá um cenário, como o apresentado pela imagem a seguir:

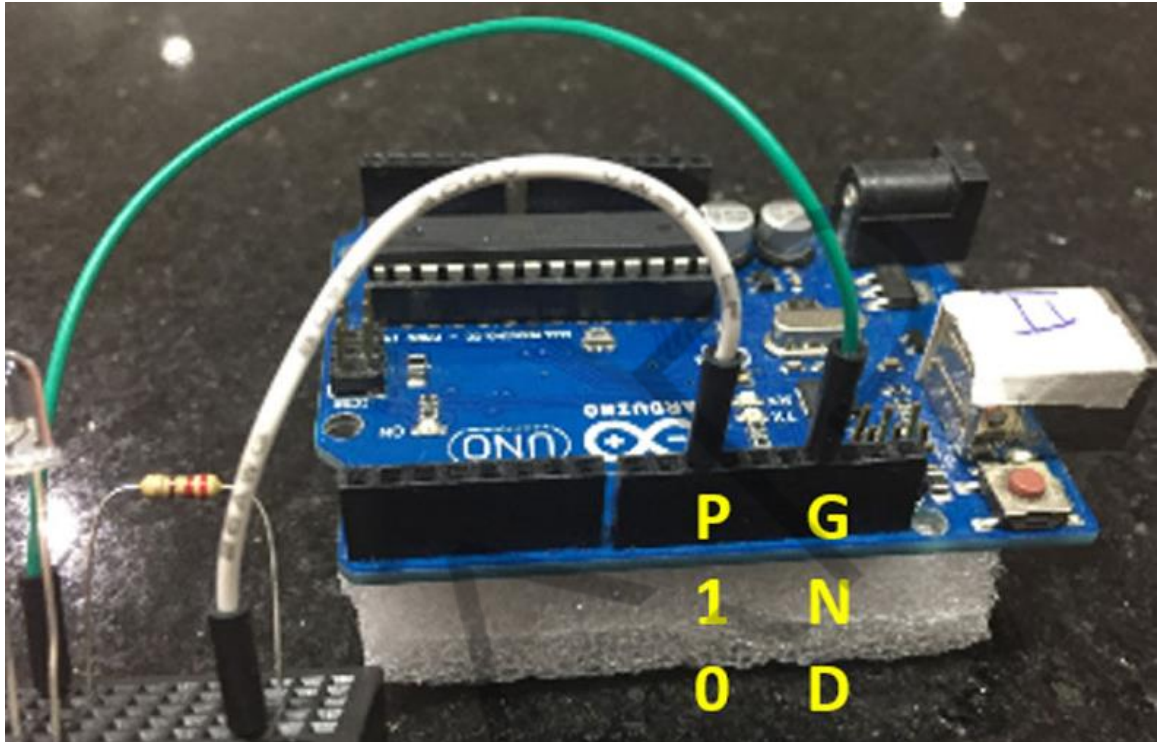


Figura 8.6 – Imagem da ligação proposta
Fonte: Imagem do Google (2018)

Agora que a estrutura está pronta, vamos desenvolver o código que o nosso Arduino deverá receber, abra o “Arduino Sketch” e digite o código como apresentado na imagem:

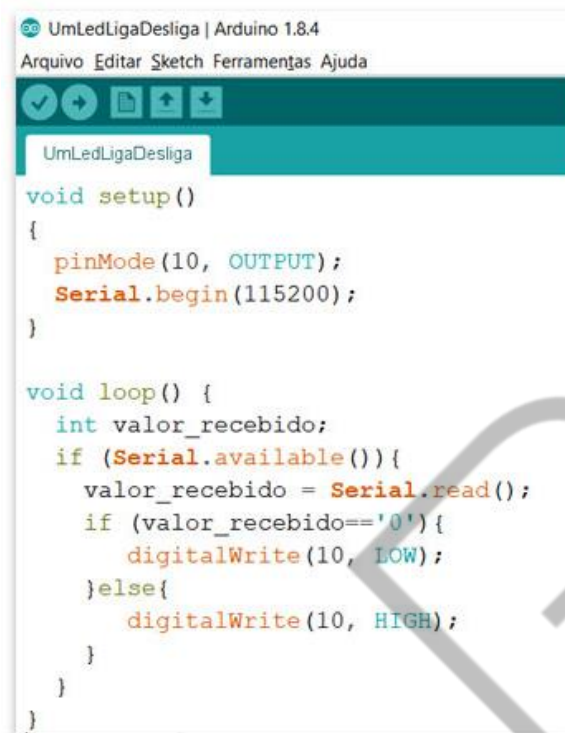


Figura 8.7 – Código do Arduino para receber dados
Fonte: Imagem do Google (2018)

Observando esse código, podemos notar que, na definição do “setup()”, configuramos a porta 10 para um dispositivo somente de saída (*output*), no nosso caso, o “led”. Também definimos a taxa de transmissão para “115200”, o mesmo valor deverá ser determinado no nosso código do Python, para que possam ficar sincronizados. Na função “loop()”, criamos uma variável para receber um valor inteiro, chamada “valor_recebido”, e abrimos uma decisão que somente retornará “True” (verdadeiro) se a saída serial estiver disponível para leitura.

Dentro da primeira tomada de decisão, recebemos o valor por meio da função “read()”, que irá realizar a leitura do dado que estiver na saída serial. Depois, verificaremos o valor que foi recebido. Se for igual a “0”, iremos passar para o nosso dispositivo que está na porta 10 do Arduino (no nosso caso, o “led”) o valor “LOW”, ou seja, iremos desligá-lo, caso contrário, iremos passar, para o mesmo dispositivo que está na porta 10, o valor “HIGH”, isso significa que iremos ligá-lo. Em resumo, “0” desliga e “1” liga o nosso led. Antes de passarmos para o nosso código em Python, compile o seu código do Arduino, clicando sobre o botão destacado na imagem a seguir:



Figura 8.8 – Compilando um código Arduino
Fonte: Imagem do Google (2018)

Clicando sobre o botão em destaque nessa imagem, deverá aparecer, após a compilação, uma mensagem como a que está na parte inferior da imagem. Caso apareça uma mensagem com destaque em vermelho, significa que seu código apresenta erro, procure corrigi-lo e clicar novamente, até que o código esteja completamente correto, então, terá chegado o momento de passar o código para a placa Arduino. Verifique se ela está devidamente conectada ao seu computador e clique no botão que está em destaque na imagem a seguir:



Figura 8.9 – Transferindo código para Arduino
Fonte: Imagem do Google (2018)

Após a transferência do Arduino, poderemos iniciar o nosso código no Python. Vamos lá.

8.3.2 Enviando dados – Lado Python

Vamos começar criando um novo arquivo do tipo “Python File”, chamado “Enviar_Dados.py”. Digite os dados a seguir:

```

import serial
conexao=""
for porta in range(10):
    try:
        conexao = serial.Serial("COM"+str(porta), 115200, timeout=0.5)
        print("Conectado na porta: ", conexao.portstr)
        break
    except serial.SerialException:
        pass
if conexao!="":
    acao=input("Digite:\n<L> para Ligar\n<D> para Desligar: ").upper()
    while acao=="L" or acao=="D":
        if acao=="L":
            conexao.write(b'1')
        else:
            conexao.write(b'0')
        acao = input("Digite:\n<L> para Ligar\n<D> para Desligar: ").upper()
    conexao.close()
    print("Conexao encerrada")
else:
    print("Sem portas disponíveis")

```

Código-fonte 8.3 – Código Python para enviar dados
Fonte: Elaborado pelo autor (2018)

Com base no código apresentado anteriormente, acrescentamos a esse novo arquivo apenas as linhas que estão com a fonte em vermelho. Vamos à sua explicação:

- Na primeira linha vermelha, criamos uma variável chamada “acao”, a qual receberá as letras “L” ou “D” (convertendo o conteúdo digitado para letras maiúsculas, por meio da função “**upper()**”), que representarão a ação de ligar e desligar o led, respectivamente.
- Na segunda linha, iniciamos um laço de repetição que irá permanecer perguntando ao usuário enquanto ele digitar “L” ou “D”, se ele digitar qualquer outro caractere, o laço será encerrado.
- Dentro do laço, temos uma tomada de decisão composta, na qual estamos avaliando, se o valor da variável “acao” for igual a “L”, iremos escrever na porta serial (por meio da função “**write()**”), o caractere “1”. Perceba que passaremos o dado no formato *byte*, pois, conforme vimos em capítulos anteriores, o formato de *bytes* é o utilizado para a transmissão de dados via protocolos de transporte. Caso contrário, o dado enviado para a saída serial será “0”.
- Fora da tomada de decisão, será perguntado ao usuário o que deseja realizar, lembrando que, se digitar algo diferente de “L” ou “D”, o laço será encerrado, a conexão fechada e exibirá a mensagem “Conexão encerrada.”.

Dessa forma, com o dispositivo devidamente conectado ao seu computador, rode sua aplicação Python e verificará que quando digitar “L”, o led irá acender, e quando digitar “D”, o led irá apagar. Você pode procurar adicionar mais leds à sua *proto board*, a fim de treinar um pouco mais, sendo um vermelho, um amarelo e um verde. Conforme o valor recebido, ele irá acender apenas um dos três leds, como se fosse um semáforo, por exemplo, se enviar “1”, o semáforo será fechado; se enviar “2”, acenderá o amarelo; e se enviar o “3”, o semáforo será aberto. Procure praticar e dar asas à sua imaginação.

8.4 Enviando dados do Arduino para o Python

Agora, montaremos outro exemplo, no qual realizaremos o processo inverso. Iremos definir a mudança no dispositivo, no nosso caso, o led, e os dados serão enviados para o Python. Vamos lá!

8.4.1 Enviando dados – Lado IoT

Utilizaremos a mesma estrutura apresentada anteriormente, mas, dessa vez, o Arduino irá ligar e desligar o “led” por meio de um intervalo de tempo e o Python irá receber o estado do led, ou seja, será informado se ele está ligado ou desligado. Mudaremos o nosso código no Arduino Sketch para:

```
void setup()
{
  pinMode(10, OUTPUT);
  Serial.begin(115200);
}

void loop() {
  int intervalo_pisca;
  intervalo_pisca=4000;
  digitalWrite(10, LOW);
  Serial.write('0');
  delay(intervalo_pisca);
  digitalWrite(10, HIGH);
  Serial.write('1');
  delay(intervalo_pisca);
}
```

Código-fonte 8.4 – Código Arduino Sketch para enviar dados
Fonte: Elaborado pelo autor (2018)

Com base no código apresentado, primeiramente definimos a função “setup()”, assim como no tópico anterior, mas dentro da função “loop()”, o código foi alterado significativamente. Vejamos:

- Nas duas primeiras linhas, criamos uma variável para controlar o intervalo de tempo entre ligar e desligar o led. A unidade de medida utilizada é em milissegundos, por isso, com 4000 estamos representando de quatro em quatro segundos.
- Na terceira linha, desligamos o nosso led.
- Na quarta linha, escrevemos na porta serial o valor “0”, por meio da função “write()”.
- Na quinta linha, chamamos a função “delay()” e passamos para ela o valor dado ao intervalo que está dentro da variável “intervalo_pisca”.
- Nas três últimas linhas, ligamos o led, escrevemos na saída serial o valor “1” e ativamos a espera.

Verifique se digitou tudo corretamente e transfira para o seu Arduino.

8.4.2 Recebendo dados – Lado Python

Crie um novo arquivo do tipo “Python File”, com o nome “Receber_Dados.py”, e digite o código a seguir:

```
import serial
conexao=""
for porta in range(10):
    try:
        conexao = serial.Serial("COM"+str(porta), 115200)
        print("Conectado na porta: ", conexao.portstr)
        break
    except serial.SerialException:
        pass
if conexao!="":
    while True:
        resposta = conexao.read()
        if resposta==b'1':
            print("LED Ligado")
        else:
            print("LED Desligado")
    conexao.close()
    print("Conexão encerrada")
else:
    print("Sem portas disponíveis")
```

Código-fonte 8.5 – Código Python para receber dados do Arduino
Fonte: Elaborado pelo autor (2018)

As linhas destacadas na cor vermelha foram as linhas alteradas de acordo com o primeiro código, no qual estabelecemos a primeira conexão com o Arduino. Vejamos as linhas inseridas, com mais detalhes:

- Na primeira linha, na qual definimos o objeto de conexão, retiramos o parâmetro “timeout”. Isso porque a função do código será receber o valor, bastando que o valor esteja disponível na porta serial para que o código realize a leitura, não precisamos de qualquer intervalo de tempo.
- Já dentro da tomada de decisão para a conexão válida, criamos um laço infinito, que receberá o dado da porta serial por meio do método “read()” e armazenará na variável “resposta”. Sempre é bom lembrar que o dado recebido estará no formato “byte”.
- Finalizamos com uma tomada de decisão: se o valor recebido for igual a “1”, iremos exibir a mensagem “LED Ligado”, caso contrário, “LED Desligado”.

Dessa forma, conecte o seu Arduino, caso não tenha realizado a transferência do novo código, faça-a nesse momento e execute o seu código Python. Você verá que, quando o led acender, o Python irá receber o valor e imprimir a mensagem de que o led está “ligado”. Quando receber o valor “0”, irá imprimir a mensagem de que o led foi “desligado” e assim ficará, infinitamente, até que você interrompa o processo.

8.5 Persistindo os dados recolhidos

Agora, vamos montar mais um exemplo no qual a nossa aplicação Python irá receber os dados e gravá-los fisicamente em um arquivo “json”. Para isso, vamos utilizar um sensor de luminosidade LDR (Light Dependent Resistor) que funciona variando sua resistência elétrica por meio da luminosidade recebida. Isso significa que quanto mais luz, menor a resistência elétrica e vice-versa. Precisaremos, para o desenvolvimento desse exemplo, de: um sensor de luminosidade LDR, uma *protoboard*, um resistor (marrom, preto, laranja), três cabos machos e a nossa placa Arduino.

A seguir, as imagens dos itens que utilizaremos:

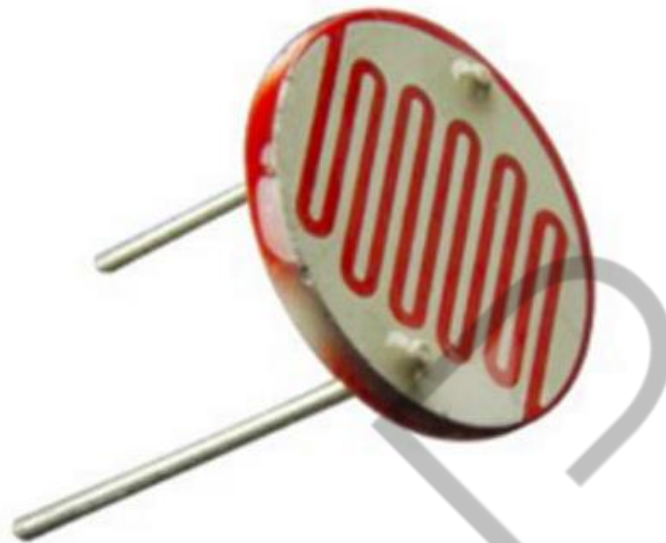


Figura 8.10 – Sensor de luminosidade LDR
Fonte: Imagem do Google (2018)

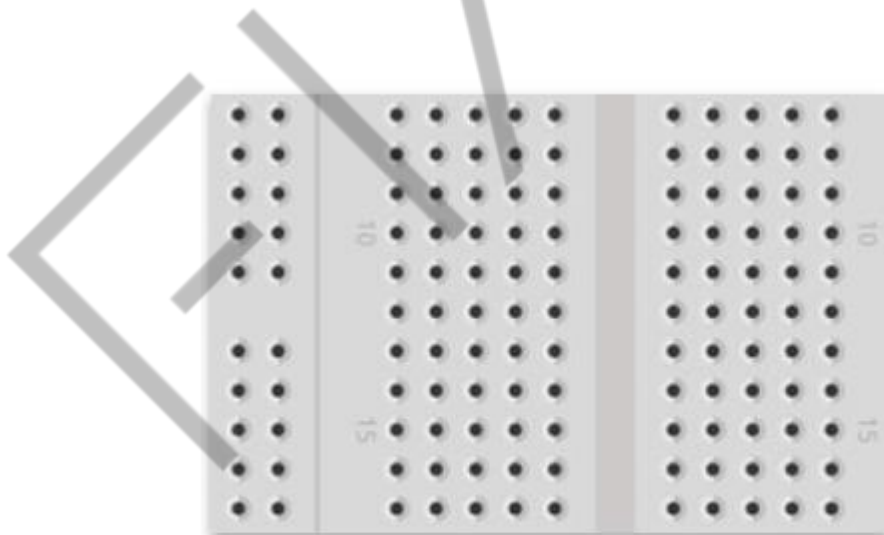


Figura 8.11 – *Protoboard*
Fonte: Imagem do Google (2018)

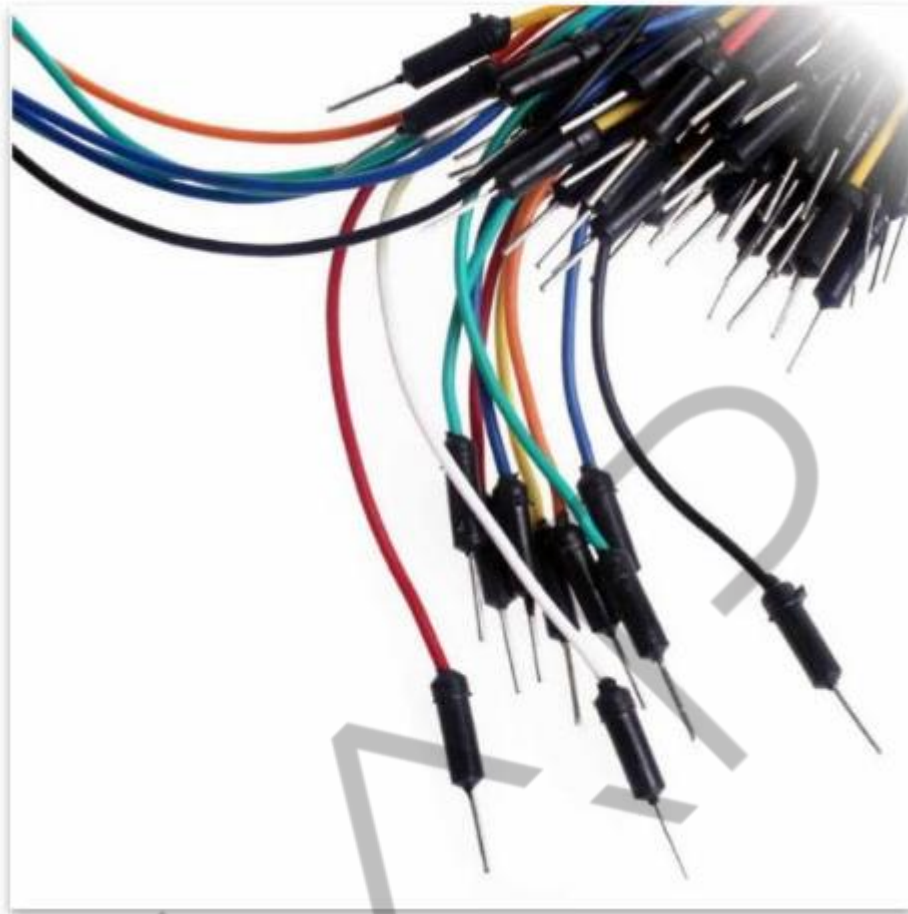


Figura 8.12 – Cabos (jumper) machos
Fonte: Imagem do Google (2018)

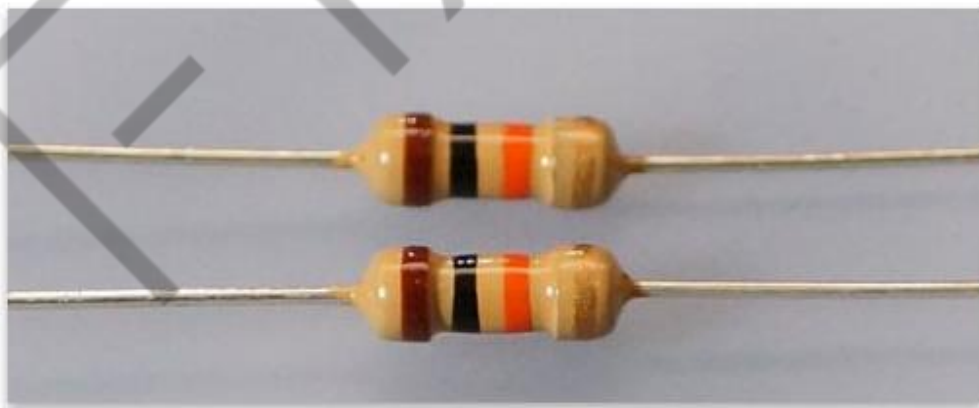


Figura 8.13 – Resistores marrom, preto e laranja
Fonte: Imagem do Google (2018)

A montagem deve seguir as instruções a seguir:

- Encaixe o sensor de luminosidade na *protoboard*, na mesma fileira de uma das pernas do sensor, ligue uma ponta de um cabo vermelho; e a outra extremidade desse cabo, ligue na porta de 5V do Arduino.

- Na fileira da outra perna do sensor, encaixe um cabo verde, a outra extremidade do cabo verde, você deverá encaixar em uma porta analógica do Arduino, utilizaremos a porta “A1”. Ainda nessa fileira da *protoboard*, encaixe uma das extremidades do resistor à outra perna do resistor e deixe em uma fileira livre da *protoboard*.
- Na fileira onde está somente a perna do resistor, conecte um cabo preto, e a outra extremidade desse cabo, conecte em uma porta “GND” da placa Arduino.

Você deverá estar com uma montagem, como a apresentada na figura a seguir:

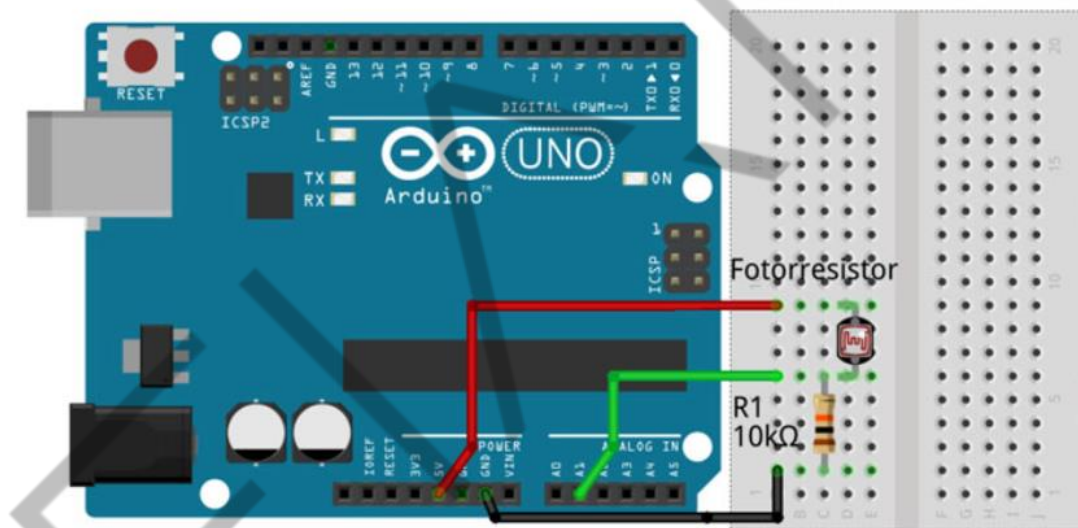


Figura 8.14 – Esquema de ligação do sensor de luminosidade (fotorresistor)
Fonte: Imagem do Google (2018)

Agora que a ligação está pronta, conecte a sua placa Arduino ao seu computador e, no Arduino Sketch, digite o seguinte código:

```
void setup() {  
  Serial.begin(115200);  
}  
void loop() {  
  int luz=analogRead(1);  
  Serial.println(luz);  
  delay(5000);  
}
```

Código-fonte 8.6 – Código para retornar valores capturados pelo sensor de luminosidade
Fonte: Elaborado pelo autor (2018)

Podemos observar, nesse código, que definimos a taxa de transmissão para 115200 e, na função “loop()”, criamos uma variável chamada de “luz”, que irá receber os dados capturados pelo sensor (por meio do método “analogRead()”) conectado na porta “A1”. Caso você tenha utilizado outra porta analógica, terá também que mudar o valor da função “analogRead()”. Caso tenha conectado na porta “A2”, terá que colocar o número “2” dentro dos parênteses da função “analogRead()”.

Em seguida, iremos transmitir para a saída Serial o valor que foi retornado pelo sensor. Detalhe para a utilização do método “**println()**” e não o “write()”, devido ao retorno do dado ser dinâmico em relação ao seu tamanho, ou seja, não conseguimos precisar a quantidade de caracteres que será retornada pelo sensor. Assim, utilizaremos “println()”, que é um método mais flexível, e, por fim, acrescentaremos um tempo de 5 segundos para a próxima captura. Compile e, em caso de sucesso, transmita o código para a sua placa Arduino.

Agora, vamos para o PyCharm. Crie um novo arquivo do tipo “Python File”, denominado “**Capturar_Temperatura.py**”, no qual irá digitar o seguinte código:

```
import serial
conexao=""
for porta in range(10):
    try:
        conexao = serial.Serial("COM"+str(porta), 115200)
        print("Conectado na porta: ", conexao.portstr)
        break
    except serial.SerialException:
        pass
if conexao!="":
    while True:
        resposta = conexao.readline()
        print(resposta.decode())
        conexao.close()
        print("Conexão encerrada")
else:
    print("Sem portas disponíveis")
```

Código-fonte 8.7 – Recebendo no Python os dados devolvidos pelo sensor de luminosidade
Fonte: Elaborado pelo autor (2018)

Observe as funções das linhas com mais detalhes:

- **Linhas 1 e 2:** importamos a biblioteca e criamos uma variável para receber a conexão.

- **Linha 3 a 9:** montamos um laço para que o programa procure a porta na qual o Arduino está conectado.
- **Linhas 10 e 11:** se a conexão estiver diferente de vazio, ou seja, se a conexão foi estabelecida, então, teremos um laço infinito.
- **Linha 12:** dentro do laço, armazenaremos a resposta por meio do método “**readline()**”. Repare que, dessa vez, não utilizamos o método “**read()**”, porque será informada uma quantidade de dados indeterminada e podem ser retornados um caractere, dois caracteres e assim sucessivamente.
- **Linha 13:** imprimimos o valor capturado pela variável “resposta”, convertendo o seu conteúdo de “byte” para “string”, por meio da função “**decode()**”, conforme já vimos anteriormente.
- Em seguida, as linhas são responsáveis por encerrar a conexão ou avisar ao usuário que não foi possível realizar a conexão. Isso normalmente ocorre quando a porta já está em uso, ou seja, caso tente executar o código duas vezes simultaneamente, você verá uma mensagem de erro falando que a porta já está em uso, portas seriais suportam apenas uma conexão.

Agora, basta executar a sua aplicação Python. Você verá algum número provavelmente entre “100 e 999”, quanto menor, menos luminosidade, e vice-versa. Procure colocar o seu dedo sobre o sensor e observe a alteração do valor que será exibida. Procure também posicionar a lanterna do celular sobre o sensor e verá que o número irá aumentar. Agora, iremos realizar uma pequena alteração para que possamos armazenar os dados em um arquivo, posteriormente. Poderíamos utilizá-los para realizar médias, montar relatórios com histórico, entre tantas outras ações com esses dados. Dentro do código anterior, realize as seguintes alterações:

```
import serial
import json
import time
from datetime import datetime
conexao=""
for porta in range(10):
    try:
        conexao = serial.Serial("COM"+str(porta), 115200)
        print("Conectado na porta: ", conexao.portstr)
        break
    except serial.SerialException:
        pass
```

```
if conexao!="":
    dicionario={}
    cont=0
    while cont<10:
        resposta=conexao.readline()
        dicionario[str(datetime.now())]=[resposta.decode('utf-8')[0:3]]
        print(resposta.decode('utf-8')[0:3])
        cont+=1
    with open('Temperatura.json', "w") as arq:
        json.dump(dicionario, arq)
    conexao.close()
    print("Conexão encerrada")
else:
    print("Sem portas disponíveis")
```

Código-fonte 8.8 – Persistindo dados do sensor
Fonte: Elaborado pelo autor (2018)

Avaliando as linhas do nosso código acima:

- **Linhas 1, 2 e 3:** realizamos os *imports* das bibliotecas *serial* (para recuperar os dados do Arduino), *json* (para permitir a gravação dos dados) e *datetime* (para que possamos retornar a data e/ou hora do sistema).
- **Linha 4:** criamos um objeto para receber a conexão com o Arduino.
- **Entre as linhas 5 e 11:** montamos o *for* para buscar a porta disponível para a conexão.
- **Linha 12:** a tomada de decisão, caso a conexão seja estabelecida com sucesso ou, se não houver conexão, será exibida a mensagem “Sem portas disponíveis”.
- **Linhas 13 e 14:** criamos um dicionário para armazenar os dados e depois serem persistidos no arquivo “.json”.
- **Linha 15:** iniciamos um laço que irá repetir o código por dez vezes, o laço será controlado por meio da variável “cont”.
- **Linha 16:** retorna o dado que está na porta “serial” e armazenaremos na variável “resposta”.
- **Linha 17:** iremos adicionar no “dicionario” um novo elemento que terá como chave a data e hora (“now()” retorna, data e hora com milissegundos) e o valor será o dado retornado por meio da porta serial, decodificado para string e consideraremos apenas os três primeiros caracteres ([0:3]), que representam efetivamente o nível de luminosidade. Por exemplo: { "2018-06-16 20:33:14.022459": ["958"] }.

- **Linha 18:** estamos exibindo apenas o dado que será persistido no dicionário como valor.
- **Linha 19:** adicionamos mais uma na variável “cont”.
- **Linhas 20 e 21:** estamos definindo que o nome do arquivo será “Temperatura.json” em modo de escrita (“w”), e que, dentro do “with”, o arquivo será representado pela palavra “arq”. Dentro do “with”, invocamos o método “dump()” para realizarmos a gravação do arquivo **json**.
- Finalizamos o código após a gravação.

Quando executar o código, e o laço “while” for concluído, um arquivo “json” será criado. A partir dele, muitas ações poderão ser utilizadas, como armazenar um banco de dados, realizar uma análise sobre os dados, determinar médias, enfim, uma infinidade de possibilidades.

Agora, cabe a você, com toda a sua criatividade, utilizar diversos sensores para que possa desenvolver soluções aplicadas para a sua área de atuação profissional. Por exemplo, você pode desenvolver uma solução que ficará responsável por verificar a resposta de um IP responsável por algum serviço dentro da rede, e assim que ele não responder, o seu componente IoT pode disparar uma sirene, gerar um log, piscar uma luz ou passar um sinal para um aplicativo no smartphone ou qualquer outra forma de aviso de que o serviço saiu do ar.

REFERÊNCIAS

FORBELLONE, André Luiz Villar. **Lógica de programação**. 3. ed. São Paulo: Prentice Hall, 2005.

JET BRAINS. **PyCharm Community, version 2017.2.4**: IDE para programação. Disponível em: <<https://www.jetbrains.com/pycharm/download/#section=windows>>. Último acesso: nov. 2017.

KUROSE, James F. **Redes de computadores e a Internet**: uma abordagem top-down. 6. ed. São Paulo: Pearson Education do Brasil, 2013.

MAXIMIANO, Antonio Cesar Amaru. **Empreendedorismo**. São Paulo: Pearson Prentice Hall Brasil, 2012.

PIVA, Dilermando Jr. **Algoritmos e programação de computadores**. São Paulo: Elsevier, 2012.

PUGA, Sandra. **Lógica de programação e estruturas de dados**. São Paulo: Prentice Hall, 2003.

RAMEZ, Elmasri; SHAMKANT B. Navathe. **Sistemas de banco de dados**. 6. ed. São Paulo: Pearson Addison Wesley, 2011.

RHODES, Brandon. **Programação de redes com Python**. São Paulo: Novatec, 2015.

STALLINGS, W. **Arquitetura e organização de computadores**. 8. ed. São Paulo: Prentice Hall Brasil, 2010.