



# Curso Programação Backend com Python

Guia de instalação

© Cleuton Sampaio 2021

<b>Instalação do python</b>	<b>3</b>
Python 2.7	3
Sem python 3	3
<b>Microsoft Windows</b>	<b>4</b>
<b>O código de exemplo</b>	<b>4</b>
Fazendo um “test drive” com o projeto	5
Ambiente virtual	7
Rodando	10

## Instalação do python

Se você utiliza plataforma Linux atualizada, como o Ubuntu, por exemplo, já deve possuir o python 3 instalado. Se o seu sistema operacional for Microsoft Windows, pule para o tópico mais adiante.

Para verificar é fácil:

1. Abra o terminal;
2. Digite: `python --version`

Se aparecer versão 3.x então está ok.

### Python 2.7

As versões mais antigas de Linux, como o Ubuntu 16 ou anteriores, utilizam o python 2.7 como padrão do sistema e podem não possuir o python 3 instalado. Ao digitar: `python --version` você verá uma versão como 2.7.x. Se isto acontecer, tente digitar: `python3 --version` e veja se está instalado. Se estiver, é só usar o comando “python3” em vez de “python” e pronto.

### Sem python 3

Se tem Linux e não tem python 3, então você pode instalar seguindo as instruções de instalação de acordo com a sua distribuição:

**Ubuntu:**

<https://www.digitalocean.com/community/tutorials/how-to-install-python-3-and-set-up-a-programming-environment-on-an-ubuntu-20-04-server-pt>

### Outros:

<https://python.org.br/instalacao-linux/>

## Microsoft Windows

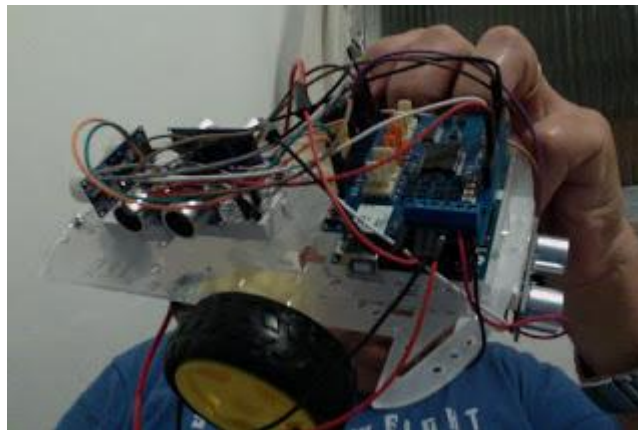
Estou assumindo que você tenha a versão 10 do Windows. Instalar o python é muito fácil, bastando baixar o instalador diretamente do site python.org:

<https://www.python.org/downloads/windows/>

Procure o instalador “microsoft installer” para a sua plataforma (32 ou 64 bits).

## O código de exemplo

Para esta seção eu uso um projeto denominado **maze** (labirinto), um programa que usei para controlar meu carro robô por um labirinto artificial:



Este robô é controlado por um Arduino e um Raspberry e utilizando o protocolo Firmata eu conseguia fazer o python controlar os movimentos e sensores do robô.

Ele conseguia medir distâncias, detectar paredes e corredores e, utilizando algoritmo de [backtracking](#), eu consegui fazê-lo resolver labirintos e achar a saída.

É uma versão adaptada deste código que vamos utilizar como exemplo agora.

O projeto está em:

<https://github.com/cleuton/pythondrops/tree/master/maze>

Você pode clonar todo o pythondrops, assim terá um repositório de código ao seu dispor.

Caso não saiba utilizar o Github, siga estes passos:

1. Acesse o site: <https://github.com/cleuton/pythondrops>
2. Clique no botão verde “Code” e selecione “Download zip”;

## Fazendo um “test drive” com o projeto

Ok, a animação deve estar no auge! Você quer executar o maze e ver como funciona! Vamos lá! Abra um terminal (ou prompt de comandos), vá para a pasta onde descompactou o pythondrops.zip, procure uma subpasta “maze”. Dentro dela, digite:

```
python maze.py 10 10
```

## OOPS!

```
$ python maze.py
```

```
Traceback (most recent call last):
```

```
File "maze.py", line 1, in <module>
```

```
    from model.labirinto import Labirinto
```

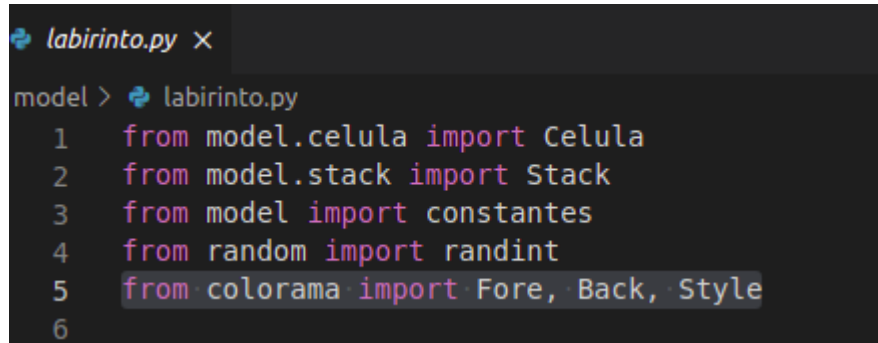
```
File "/home/cleuton/Documentos/projetos/pythondrops/maze/model/labirinto.py",  
line 5, in <module>
```

```
    from colorama import Fore, Back, Style
```

```
ImportError: No module named colorama
```

`ModuleNotFoundError: No module named 'colorama'`

`ModuleNotFoundError: No module named 'colorama'!` Parece que está faltando um módulo... Se olharmos a linha que deu erro (linha 5 dentro de “labirinto.py”, dentro da pasta “model”):

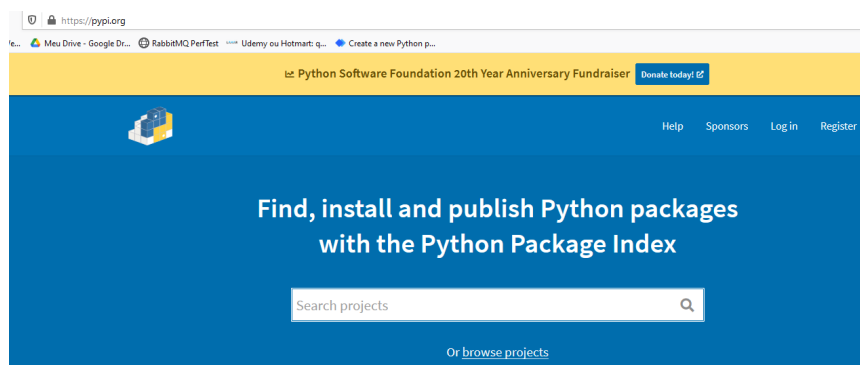


```
labirinto.py x
model > labirinto.py
1  from model.celula import Celula
2  from model.stack import Stack
3  from model import constantes
4  from random import randint
5  from colorama import Fore, Back, Style
6
```

Temos uma instrução para importar alguns objetos do módulo **colorama**! E este não é um módulo padrão do python ou um módulo que eu construí. Temos alguns exemplos aqui que não deram erro:

- **model.celula**: Eu criei e está dentro da pasta **model** no módulo **celula.py**;
- **random** e **randint**: O módulo **random** é padrão do python e possui a função **randint**;

Mas **colorama** parece ser um módulo externo, uma dependência externa. O python pode utilizar repositórios externos para procurar dependências. O mais comum é o Pypi - Python Package Index: <https://pypi.org/>



Se você digitar **colorama** nesse campo, verá que o pacote existe e que pode ser baixado pelo utilitário **pip**.

Você pode usar um dos comandos:

- `pip install colorama`
- `pip3 install colorama`

Dependendo da versão do executável python. Se for Linux e já possuir o python 2.7 instalado, provavelmente será “python3 / pip3”.

O projeto maze possui em sua pasta raiz um arquivo chamado requirements.txt, que pode ser utilizado com o pip para instalar as dependências externas de uma só vez:

```
pip install -r requirements.txt
```

Este arquivo é uma lista das dependências externas devidamente marcadas com suas versões.

**Atenção:** *Evite “poluir” o seu ambiente python com dependências de projetos! Use um ambiente virtual!*

## Ambiente virtual

O python vem com um utilitário chamado **venv** - Virtual Environment, que cria ambientes virtuais com seu próprio binário python e pacotes, evitando poluir sua instalação padrão.

Com o venv é possível criar ambientes com diferentes de suas dependências. Para saber mais sobre o **venv**, leia esta documentação:

<https://docs.python.org/pt-br/3/library/venv.html>

Há alternativas mais robustas, como o Anaconda (<https://www.anaconda.com/>), muito bom, porém mais complexo.

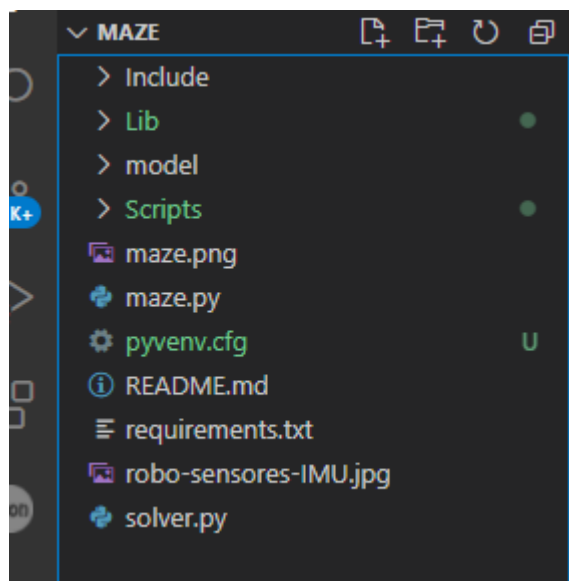
Vamos criar um ambiente virtual para este projeto rapidamente. Para criar um ambiente basta usar o comando:

```
python -m venv /<pasta onde quer criar o ambiente python>
```

Normalmente usamos a pasta onde está o projeto. Entre na pasta do projeto maze e digite:

```
python -m venv .
```

Ele criará várias pastas e arquivos:



Como podemos ver em verde, ele criou as pastas **Lib** e **Scripts** (ou **bin** no linux) e o arquivo **pyvenv.cfg**.

O arquivo pyvenv.cfg (no seu computador pode estar diferente) tem informações sobre a versão do python:



```
home = C:\Program Files\Python39
include-system-site-packages = false
version = 3.9.2
```

Você pode até apontar seu ambiente virtual para uma versão diferente do python, que esteja instalada em seu computador.

A pasta **Lib** guarda os pacotes instalados pelo **pip** e a pasta **Scripts** (ou **bin**, no Linux) guarda os executáveis do python e do pip, além de um script para ativação do ambiente (“activate”, se for Linux, e “activate.bat”, se for MS Windows).

Ativar o ambiente virtual significa redirecionar todo o ambiente python para o executável e as bibliotecas do ambiente. Para ativar:

- “Scripts\activate” se for MS Windows;
- “source bin/activate” se for Linux com bash;

```
C:\Users\Cleuton Sampaio\Documents\projetos\pythondrops\maze>Scripts\activate
```

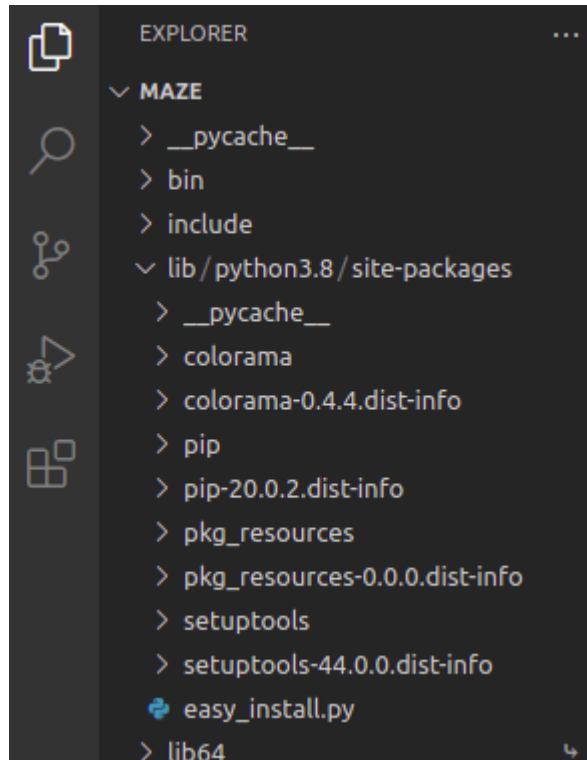
```
(maze) C:\Users\Cleuton Sampaio\Documents\projetos\pythondrops\maze>
```

Ele muda o prompt do comando para indicar qual ambiente está ativo. A princípio, o nome do ambiente é o nome da pasta onde foi criado.

Agora, você pode instalar dependências do projeto com o comando:

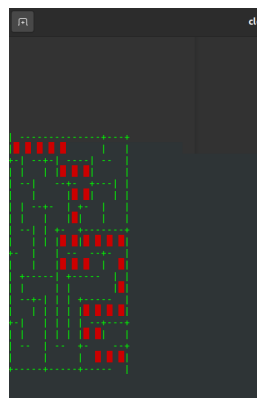
```
pip install -r requirements.txt
```

Se você olhar a pasta Lib/site-packages (no Linux é “lib/pythonX.Y/site-packages”) verá o **colorama** instalado:



Rodando

Agora o programa maze deve rodar sem problemas:



Ele imprime um labirinto vazio e depois o resolve, percorrendo em vermelho.

