



PYTHON
ACADEMY

AMBIENTES VIRTUAIS COM PYTHON E VIRTUALENV

Vamos entender o funcionamento do incrível virtualenv, uma ferramenta simples e poderosa que permite criar ambientes isolados de desenvolvimento Python.

[PYTHONACADEMY.COM.BR](https://pythonacademy.com.br)

Gere ebooks como este com



em <https://ebookr.ai>

Crie ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

E deixe que nossa IA faça o trabalho pesado!



Capas gerados por IA



Infográficos feitos por IA



Edite em Markdown em Tempo Real



Adicione Banners Promocionais

TESTE AGORA

PRIMEIRO CAPÍTULO 100% GRÁTIS

Olá pessoal! Nesse post, vamos entender o funcionamento do incrível **virtualenv**, uma ferramenta simples e poderosa que permite criar ambientes isolados de desenvolvimento Python

Dessa forma é possível a utilização de diversas bibliotecas em um mesmo ambiente sem que haja conflitos entre elas.

Neste post vamos aprender:

- As vantagens ao se isolar seu ambiente de desenvolvimento Python
- Porquê escolher o virtualenv
- Como programar utilizando Python 2 e Python 3 no mesmo PC!

Portanto, se você está cansado de misturar pacotes Python ou ver aquele incômodo erro `ImportError: No module named Foo`, vamos nessa!

Introdução

Quando desenvolvemos em Python, é comum ficarmos maravilhados com o poder de suas bibliotecas.

Tudo se torna mais fácil ainda quando podemos baixá-las com um simples comando (`pip install <pacote>` por exemplo).

Dessa forma, se não tomarmos alguns cuidados, podemos deixar nosso ambiente de desenvolvimento lotado de bibliotecas.

Isso quando não as utilizamos algumas vezes e depois nunca mais.

Não apenas isso! Imagine o seguinte cenário: você está desenvolvendo uma aplicação, chamada **CoolApp** que utiliza a biblioteca **powerlib** em sua versão **1.0**.

Por enquanto tudo ok...

Daí, algum tempo se passa e você acaba largando essa aplicação, pois cansou-se dela e ela nem era tão promissora assim.

Você começa então a desenvolver uma aplicação ultra mega powerfull, chamada **TheApp**, que por coincidência utiliza a mesma biblioteca (**powerlib**), mas em sua versão **2.0**.

Bom, você inocentemente realiza o *upgrade* (`sudo pip install --upgrade powerlib`) dessa biblioteca para sua versão 2.0 para poder continuar seu desenvolvimento.

Então me diga: o que acontecerá caso você tente retomar o desenvolvimento da aplicação **CoolApp** (juntamente com o desenvolvimento da aplicação **TheApp**)?

Resposta: Caso você tenha instalado todas as suas bibliotecas no diretório padrão (`/usr/lib/pythonX.X/site-packages`), apenas uma aplicação irá funcionar... **NOOOOOOOOOOOO!!!!**

Ou seja, quando desenvolvemos Python “globalmente” e não isolamos cada ambiente de desenvolvimento, podemos ter conflitos entre versões de bibliotecas.

Outro ponto, e se quisermos desenvolver um projeto em Python 2 e outro em Python 3? Como fazer?

E ainda mais: e se você não puder instalar suas dependências no diretório global `site-packages` (seu programa pode estar sendo executado em um *host* remoto, por exemplo)?

Mas calma, a solução para todos os seus problemas está aqui, e chama-se **virtualenv**. Portanto, vamos começar entendendo melhor do que se trata!

Como o Virtualenv Funciona

Funcionamento do virtualenv

O funcionamento do **virtualenv** é realmente simples. Ele basicamente cria uma cópia de todos os diretórios necessários para que um programa Python seja executado, isto inclui:

- As bibliotecas comuns do Python (*standard library*);
- O gerenciador de pacotes `pip`;
- O próprio binário do Python (Python 2.x/3.x);
- As dependências que estiverem no diretório `site-packages`;
- Seu código fonte descrevendo sua aplicação.

Assim, ao instalar uma nova dependência dentro do ambiente criado pelo **virtualenv**, ele será colocado no diretório `site-packages` relativo à esse ambiente, e não mais globalmente.

Assim, só esse ambiente enxergará essa dependência. Show, né?!

Primeiros Passos

Instalação

Para a instalação do **virtualenv** vamos precisar do `pip`.

Caso você ainda não tenha o gerenciador de pacotes `pip`, [instale-o clicando aqui](#) e executando o script baixado com `python get-pip.py`.

Agora com o pip, precisamos executar apenas um comando para instalar o **virtualenv**:

```
$ sudo pip install virtualenv
```

Pronto! virtualenv instalado e funcionando! Agora vamos começar a utilizá-lo!

Inicializando um Ambiente Virtual

virtualenv possui apenas um comando! Olha que simples:

```
$ virtualenv [opções] <nome_da_pasta>
```

Esse comando cria um novo ambiente de desenvolvimento totalmente isolado!

No nosso caso, vamos chamar nossa pasta de **ENV**.

Nessa pasta são criados alguns diretórios importantes, como:

- **ENV/lib** e **ENV/include** : contêm as bibliotecas de suporte ao ambiente virtual. Os pacotes baixados via **pip** por exemplo serão instalados em **ENV/lib/pythonX.X/site-packages/** (não mais globalmente).
- **ENV/bin** : residem os binários necessários para executar o próprio Python, entre outros executáveis (como o **pip** ou **setuptools**). Dessa forma, os comandos **python script.py** e **ENV/bin/python script.py** produzem exatamente o mesmo resultado.

Script de Ativação do Ambiente Virtual

Após criar o seu novo ambiente, podemos ativá-lo com o comando:

```
$ source ENV/bin/activate
```

O comando `source` lê um arquivo e executa os comandos contidos ali.

Ao ativar o **virtualenv**, o diretório `ENV/bin` será adicionado como primeiro registro do caminho `$PATH` do seu sistema operacional.

Ele também altera o padrão do seu prompt, adicionando o prefixo `(ENV)` para indicar que você está em um ambiente controlado pelo **virtualenv**.

Como Desativar seu Ambiente Virtual

Para desativar um ambiente virtual e remover tudo que foi feito e instalado, retornando ao estado anterior do sistema (sem o **virtualenv**) basta executar:

```
$ deactivate  
$ rm -r ENV
```

Fazendo isso, seu ambiente virtual é desfeito e tudo que foi copiado para este ambiente é apagado (comando `rm`).



Estou desenvolvendo o [Ebookr.ai](https://ebookr.ai), uma plataforma que transforma suas ideias em ebooks profissionais usando IA — com geração de capa, infográficos e exportação em PDF. Te convido a conhecer!


Crie Ebooks profissionais incríveis em minutos com IA



Chega de formatar texto no Google Docs, Word ou ferramentas que só te fazem perder tempo...

... e deixe que nossa IA faça o trabalho pesado!

TESTE AGORA! PRIMEIRO CAPÍTULO 100% GRÁTIS 

 Capas gerados por IA

 Adicione Banners Promocionais

 Edite em Markdown em Tempo Real

 Infográficos feitos por IA

Opção --python

Para criarmos um novo ambiente contendo a versão do Python de sua escolha (Python 2.x ou Python 3.x), devemos informar ao virtualenv, no momento da criação de um novo ambiente, a localização do binário do Python.

Caso você não saiba o caminho do binário do Python que você está procurando, utilize o comando `which`.

Por exemplo, para saber onde estava o binário do Python 3 em minha máquina, eu executei o comando `which python3` que me respondeu com `/usr/local/bin/python3`.

Com isso, podemos criar nosso novo ambiente utilizando o `python3` da seguinte forma:

```
virtualenv --python='/usr/local/bin/python3' ENV
```

Com seu ambiente ativado, verifique qual versão está sendo utilizada, com o comando `python --version`.

Pronto! Python 2 e Python 3 no sistema operacional de maneira simples!

Outras Opções Interessantes

De acordo com sua documentação, o comando possui algumas opções e é utilizado da seguinte maneira: `virtualenv [opções] <nome_da_pasta>`.

Para a lista completa de valores possíveis de `[opções]`, acesse sua [documentação](#).

Separei algumas opções interessantes para vocês:

- `--system-site-packages` : Essa opção cria um ambiente com todas as dependências já instaladas globalmente. Ou seja, não será um ambiente limpo.
- `--python=/path/to/pythonX.X` : Usado para definir o diretório de uma versão alternativa do Python que será utilizado nesse ambiente virtual.
- `--no-pip` : Cria o ambiente sem o `pip`.
- `--no-setuptools` : Cria o ambiente sem o `setuptools`.

Arquivo de Configuração virtualenv.ini

Ao criar um novo ambiente, o **virtualenv** busca parâmetros no arquivo `virtualenv.ini` presente no caminho `$HOME/.virtualenv/` `virtualenv.ini` (Mac OS/Linux) ou `%APPDATA%\virtualenv\virtualenv.ini` (Windows).

Nesse arquivo, podemos adicionar parâmetros do **virtualenv** que serão utilizados em todos os ambientes criados.

Por exemplo, podemos definir opções da seguinte forma:

```
[virtualenv]
no-pip
no-setuptools
system-site-packages
python = /opt/python-3.3/bin/python
```

Assim, todo ambiente que criarmos terão essas opções por padrão (sem termos que digitar os parâmetros toda vez).

Conclusão

O uso do **virtualenv** traz muitas vantagens e facilidades quando desenvolvemos em Python, principalmente na separação de ambientes de desenvolvimento.

Também nos ajuda muito na solução de conflitos entre versões de uma biblioteca e é essencial para o desenvolvimento de aplicações utilizando diferentes versões do Python em uma mesma máquina (sem ter que recorrer à máquinas virtuais, por exemplo)

Caso você ainda não o utilize, eu aconselho a baixar o mais rápido possível e aproveitar ao máximo essa ferramenta poderosa e extremamente simples que eu acabei de apresentar a vocês.

Bom... Espero que tenham gostado! E se você achou interessante este post, e acha que ele facilitou a sua vida organizando seu ambiente Python, não se esqueça de deixar aqui seu comentário ou sugestão!

Assim como você, eles são sempre bem vindos!

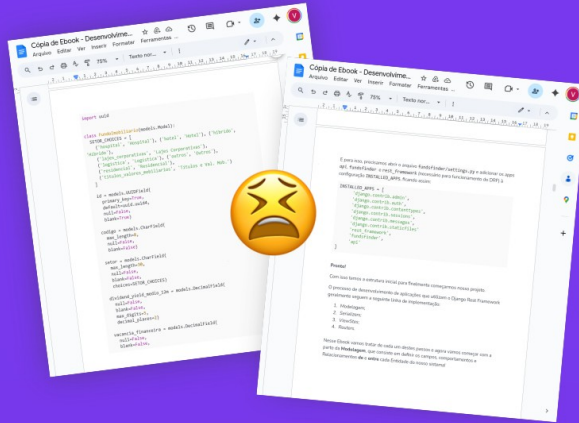
Até a próxima!

Não se esqueça de conferir!

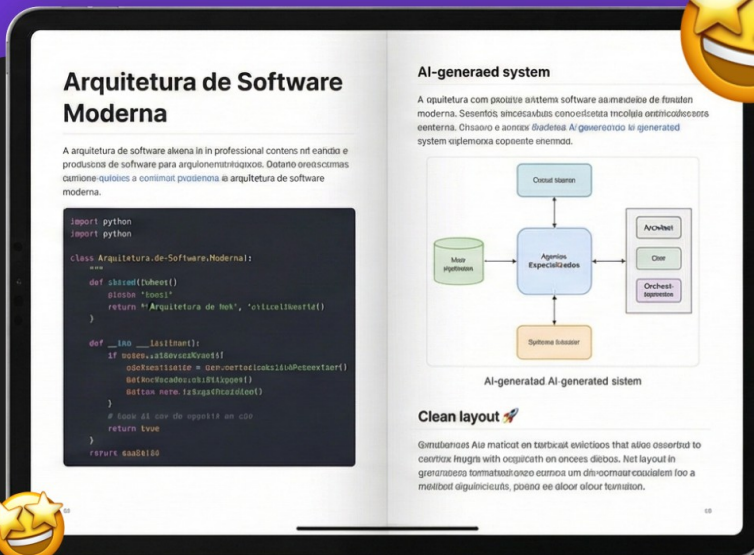


Ebookr

Crie Ebooks profissionais em minutos com IA



Chega de formatar código no Google Docs ou Word



Capas gerados por IA



Infográficos feitos por IA



Adicione Banners Promocionais

Deixe que nossa IA faça o trabalho pesado



Edite em Markdown em Tempo Real

TESTE AGORA



PRIMEIRO CAPÍTULO 100% GRÁTIS