Ambientes Virtuais

Até agora fizemos um script independente, o nosso `hello.py` que não possui dependências e pode rodar em qualquer ambiente com Python.

Porém conforme nossos programas ficam mais complexos acabamos utilizando soluções prontas do ecossistema Python, o PyPI, repositório de pacotes oferece mais 300 mil pacotes para reutilizar.

Qual o problema?

Se instalarmos as bibliotecas do PyPI diretamente no Python principal do sistema podemos criar conflitos e deixar o ambiente muito cheio de biblitecas que podem se tornar obsoletas.

Para resolver esse problemas o recomendado é criarmos um **sandbox**, um ambiente separado onde podemos ter uma cópia do ambiente Python isolada onde não corremos o risco de criar conflitos.

Ambiente real

Para saber qual é o ambiente **real** do Python utilize:

```
python3 -m site
```

Este comando retorna os cominhos de instalação do Python e suas bibliotecas, para criar um **ambiente virtual** a partir destes arquivos basta copiarmos tudo isso para um novo local isolado.

Como este processo é algo bastante comum de ser feito, o próprio Python já vem com uma ferramenta que faz essa cópia automaticamente.

Ambiente Virtual

O ambiente virtual é um **sandbox**, é uma cópia de todo o ambiente Python e a recomendação é que você tenha um ambiente virtual em cada um dos seus projetos cada projeto deve usar seu próprio conjunto de bibliotecas isoladamente.

Criando o ambiente virtual

Na pasta do seu projeto usamos o módulop `venv` e informamos um nome para a pasta do ambiente ser criada, é comum que o nome seja `.venv` pois o `.` torna a pasta oculta no seu sistema e esse nome é comumente adotado.

```
cd python-base
python3 -m venv .venv
```

Ao executar esse comando irá notar que foi criada uma nova pasta chamada `.venv` e dentro dela tem a cópia de todos os arquivos do Python.

```
$ ls -a .venv
.  ..  bin  include  lib  lib64  pyvenv.cfg  share
```

Ali dentro da pasta `bin` é onde encontramos o `python` e também outras ferramentas como o `pip` e a apartir de agora todos os módulos que instalarmos vão para dentro da pasta `lib`.

Mas para usar o ambiente virtual sempre será necessário efetuar a ativação, no linux isso é feito com o comando abaixo:

```
source .venv/bin/activate
```

Ao rodar o activate o seu prompt passa a exibir `(.venv)` que é o nome do ambiente virtual, e para se certificar execute novamente o módulo `site`.

```
python3 -m site
```

Repare que agora os caminhos de bibliotecas (1 e 4 da lista) apontam para a pasta isolada do seu projeto.

Outra forma de verificar qual Ambiente Python está ativado é usando o comando `which`

```
which python
```

O retorno deve ser algo como `~/Projects/python-base/.venv/bin/python`

> **IMPORTANTE** sempre que abrir um terminal, antes de executar os comandos você deverá ativar o ambiente virtual do seu projeto. Existem ferramentas que podem fazer isso automaticamente para você como o zsh ou o poetry mas durante o aprendizado eu recomendo manter os comandos todos manuais para você fixar a idéia da necessidade deles e no futuro automatize.

Git Ignore

Em nosso projeto agora temos uma nova pasta `.venv` com centenas de arquivos e se fizermos um commit + push usando o git iremos mandar essa pasta toda para o repositório remoto do github e queremos evitar isso, a `.venv` é apenas para uso local, se outra pessoa precisar executar seu código ela terá que criar o ambiente virtual diretamente lá no ambiente que precisar.

Crie um arquivo chamado `.gitignore` na raiz do projeto

```
cd python-base
touch .gitignore
```

O comando `touch` do Linux cria um arquivo vazio e então você pode abrir ele com o seu editor para adicionar as pastar que queremos que fiquem de fora do controle do git.

Basta abrir o `.gitignore` e adicionar a linha:

```
.venv
```

Uma outra forma mais fácil de fazer isso é com este comando:

```
echo ".venv" >> .gitignore
```

O comando acima adiciona o texto `.venv` no final do arquivo `.gitignore`

Desta forma evitamos que a pasta venv vá para o git mas agora você precisa fazer um commit para adicionar o `.gitignore`.

```
git add .gitignore
git commit -m "adicionado git ignore"
git push
```

Instalando pacotes

Agora sim podemos instalar pacotes dentro do nosso ambiente virtual :)

Primeiro certifique-se de que `(.venv)` aparece em seu terminal ou que `which python` mostra o Python de dentro da pasta `.venv`.

Agora a primeira coisa a fazer é usar o `pip` que é o gerenciador de pacotes do Python e através dele podemos instalar novas bibliotecas e ferramentas.

Atualize o próprio pip

```
python3 -m pip install --upgrade pip
```

Com o pip atualizado vamos instalar nosso primeiro pacote e ele se chama **IPython**

```
python3 -m pip install ipython
```

O terminal Python que usamos até agora é muito bom e serve para fazer tudo o que precisamos mas faltam algumas coisas como output colorido que facilita a leitura das mensagens e também ferramentas mais fáceis para obter ajuda.

O Ipython é uma versão do interpretadotr Python que possui mais funcionalidades.

Digite `ipython` e perceba como ele é um pouco diferente do terminal que usamos anteriormente.

```
$ ipython

Python 3.10.2 (main, Jan 15 2022, 19:56:27) [GCC 11.1.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.0.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

É bastante similar, a única grande diferença que vai notar é que ao invés de exibir o prompt padrão `>>>` ele agora mostra `In [1]:` e fica a espera de alguma instrução, experimente digitar `1 + 1`

```
In [1]: 1 + 1
Out[1]: 2
```

Repare que a resposta vem em um novo prompt contento `Out [1]:` e isso é bastante poderoso pois ele grava um histórico de todos os seus comandos e você pode por exemplo digitar `_1` ou `_2` etc para acessar o retorno de algum comando que digitou anteriormente.

Além disso o Ipython possui uma ajuda mais completa usando `?` ou `??` ao invés de usar `help()`

```
In [4]: print?
Docstring:
print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

Prints the values to a stream, or to sys.stdout by default.
Optional keyword arguments:
file:  a file-like object (stream); defaults to the current sys.stdout.
sep:   string inserted between values, default a space.
```

```
end:   string appended after the last value, default a newline.
flush: whether to forcibly flush the stream.
Type:      builtin_function_or_method
```

E também oferece auto-complete, experimente começar a digitar por exemplo a letra `p` e depois pressionar `tab` e ele vai te mostrar um seletor com todos os objetos começados pela letra `p`.

```
In [5]: p<tab>
 pass       %page      %pdef      %pinfo     %pprint    %psearch   %pycat     %%python2
 pow()      %paste     %pdoc      %pinfo2    %precision %psource   %pylab     %%python3
 print()    %pastebin  %%perl     %pip       %prun      %pushd     %%pypy
 property   %pdb       %pfile     %popd      %%prun     %pwd       %%python
```

Recomendo que em todos os seus ambientes virtuais você adicione o Ipython :)