Insper

10 - Distribuição de programas escritos em Python

Desenvolvimento Aberto - 2020/2

Igor Montagner

Na última aula trabalhamos com qualidade de software e documentação. Nesta aula completaremos essa dinâmica criando um pequeno pacote Python instalável via `pip`. Com isto teremos visto

1. Documentação de software para usuários e desenvolvedores
2. Tradução de interface de usuário
3. Distribuição de software

O único tópico restante é **Licenças de Software**, que será abordado na segunda parte do curso.

Distribuindo software para desenvolvedores

Parte 1 - nosso pacote

Nosso módulo se chamará `dev_aberto` e disponibilizará um programa executável `hello.py` (disponível nessa pasta). Crie a seguinte estrutura de pastas para nosso pacote.

```
pacote_exemplo/
    dev_aberto/
        __init__.py
        dev_aberto.py
    scripts/
        hello.py
    README.md
    LICENSE
```

Exercício: com a estrutura acima, qual seria o `import` a ser feito para usar a função `hello` do arquivo *dev_aberto.py*?

Exercício: pesquise para que serve o arquivo `__init__.py` e use-o para permitir importar `hello` usando somente `import dev_aberto`.

Exercício: crie um projeto no github para esta atividade. Faça um primeiro commit nele com o conteúdo "zerado" do projeto.

- Um arquivo *README* contendo uma frase de descrição do pacote e um link para o repositório da disciplina.
- Um arquivo *LICENSE* com a licença MIT.

Parte 2 - o arquivo `setup.py`

A descrição de um pacote Python é feita usando um arquivo setup.py Veja abaixo uma versão inicial deste arquivo:

```
from setuptools import setup

setup(name='dev_aberto_seunome',
      version='0.1',
      packages=['dev_aberto']
      )
```

Exercício: crie o arquivo acima no seu projeto, substituindo *seunome* por …. seu nome. Instale o seu próprio pacote usando

```
> pip install .
```

Exercício: em outra pasta, abra um console Python e tente importar seu módulo.

Exercício: pesquise quais argumentos são usados para especificar o autor do pacote, as versões de Python e sistemas operacionais suportados. Preencha estes valores com suas informações. Note que o `pip` leva estas informações em conta e só instalará um pacote se ele estiver em um ambiente suportado.

Dependências

Para adicionar pacotes que são automaticamente instalados quando instalamos nosso pacote precisamos identificá-los no nosso arquivo *setup.py*. Para adicionar uma dependência de instalação basta adicionar o seguinte argumento:

```
...
install_requires=[
    'pacote>=1.0',
    'pacote2'
],
...
```

Exercício: Verifique as dependências do código e adicione-as no `setup.py`.

requirements.txt

Muitos softwares usam também um arquivo *requirements.txt* para listar **todas** as dependências do software de modo a obter uma instalação idêntica à do desenvolvedor. Isto é importante para uniformizar os ambientes de desenvolvimento. Ou seja, este arquivo nunca será usado por usuários finais.

Exercício: crie um *requirements.txt* para seu projeto com as mesmas dependências listadas no seu *setup.py*.

Scripts executáveis

Além de instalar o nosso módulo para uso via `import` desejamos também disponibilizar o arquivo *hello.py* como um executável para todo o sistema. Isto pode ser feito adicionando a seguinte linha no nosso *setup.py* indicando que *scripts/hello.py* deverá ser instalado como um executável.

```
...
scripts=['scripts/hello.py'],
...
```

Não se esqueça de adicionar a seguinte linha no topo de seu arquivo para que ele possa ser executado diretamente do shell:

```
#!/usr/bin/env python3
```

No Windows é criado um executável que chama nosso script, de modo que as chamdas do executável continuarão funcionando normalmente. Note que isto não cria menus em nenhum tipo de interface gráfica.

Parte 3 - criando arquivos de distribuição

Dois tipos de arquivos de distribuição podem ser usados:

- sdist: é um arquivo contendo os fontes do projeto, incluindo arquivos adicionais especificados usando o argumento `data_files`. Usado se seu projeto for Python-puro.
- wheel: é um formato pré-compilado e específico para cada plataforma. Mais usado quando o projeto contém extensões em *C*.

A criação de um arquivo de distribuição de fontes é bem simples:

```
> python setup.py sdist
```

A instalação deste pacote pode ser feita via `pip`.

Parte 4 - envio para o PyPI

Vamos agora enviar nosso pacote para o *Python Package Index* para que ele possa ser instalado diretamente via `pip`. Para não poluir o repositório com pacotes temporários e de teste, podemos usar o *TestPyPI*. Toda sua infraestrutura é igual ao oficial, mas ele é limpo de maneira regular.

Visite https://test.pypi.org/account/register/ e registre-se no *TestPyPI*.

Após o registro, usaremos o pacote *twine* (instalável via *pip*) para fazer o upload:

```
> twine upload --repository-url https://test.pypi.org/legacy/ dist/*
```

Você poderá, então, instalar seu pacote a partir do test PyPI usando o seguinte comando:

```
> pip install --index-url https://test.pypi.org/simple/ my_hello_nome
```

Distribuindo software para usuários

Apesar do método acima ser muito útil para desenvolvedores Python, nossos usuários gostariam somente de receber um pacote que eles possam simplesmente executar em seus computadores. Ou seja, a preocupação com dependências e instalações em linha de comando não deveria existir.

Neste caso podemos usar outras soluções de distribuição, desta vez focando usuários finais. Para Python nossa opção neste roteiro será o PyInstaller.

Exercício: Visite o site desta ferramenta e aprenda a usá-la.

Exercício: O Github suporta adicionar *Releases* em um repositório. Adicione uma nova release com o arquivo executável criado pelo PyInstaller.

Entrega

Faça a entrega de sua atividade adicionando a skill *Pacote Python* e inclua nela a url do seu repositório no github.

Figure 1: Skill Pacote Python

Objetivo: Primeira experiência distribuindo software Python.

"metadata": {"url": "repo-seu-pacote"}