Recentemente, estive trabalhando em um projeto de conversor de temperatura, de Celsius para Fahrenheit, para desenvolvimento de alguns programas com viés científico.

Lembrei da fórmula de conversão que me ensinaram na época de escola, na qual, sendo **F** a temperatura em Fahrenheit e **C** a temperatura em Celsius, tínhamos `F = 1,8C + 32` e `C = (F - 32) / 1,8`, e implementei o seguinte código:

```
## aluratemp.py

def celsius_para_fahrenheit(temp_em_celsius):
    temp_em_fahrenheit = 1.8 * temp_em_celsius + 32
    return temp_em_fahrenheit

def fahrenheit_para_celsius(temp_em_fahrenheit):
    temp_em_celsius = (temp_em_fahrenheit - 32) / 1.8
    return temp_em_celsius
```

O objetivo principal desse meu código era para ser usado em outros programas maiores, com um `import`. Apenas para testar, adicionei uma **verificação do escopo de execução do programa**:

```
if __name__ == '__main__':
    print('{} ºF'.format(celsius_para_fahrenheit(10)))
    print('{} ºC'.format(fahrenheit_para_celsius(212)))
```

Rodando o programa com `python aluratemp.py`, recebo o seguinte:

```
50.0 ºF
100.0 ºC
```

Certo, o código funciona bem! E se importarmos as funções pelo interpretador do Python?

```
>>> from aluratemp import *
>>>
>>> celsius_para_fahrenheit(10)
50.0
>>>
>>> fahrenheit_para_celsius(212)
100.0
```

> *Geralmente, evitamos importar tudo com o `*`, para não acabarmos importando algo que não era o objetivo. Nesse caso, não há problema, porque sabemos que há apenas duas funções no código e queremos, de fato, importar as duas.*

Também funciona perfeitamente!

O código estava legal e, com o tempo, alguns amigos meus começaram a se interessar por ele, me pedindo uma cópia.

Eu não vejo problema nenhum em compartilhar o código, mas vou ter que ficar mandando uma cópia por email para cada um? Como eu poderia fazer isso de uma forma mais eficiente?

Distribuição de pacotes Python com PyPI

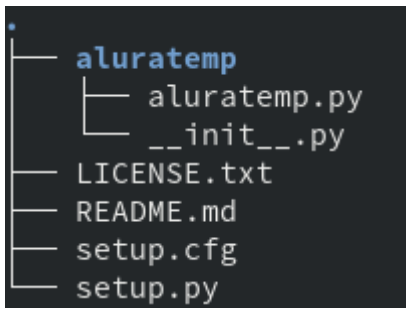
Quem está acostumado com a linguagem Python, provavelmente já se deparou com uma instrução no terminal do tipo **pip install**. Usamos ela para instalar programas, ou simplesmente códigos, feitos pela comunidade e armazenados como pacotes no **PyPI**.

O PyPI é, de fato, o maior repositório de código Python que temos hoje em dia. Nele, temos desde códigos amplamente usados, como o **numpy**, até códigos bem menores e menos conhecidos, como o **cidadaobr**, um gerador de dados de um brasileiro.

Mas como podemos usá-lo? Sabemos instalar pacotes guardados lá, mas dessa vez queremos enviar nosso próprio pacote. E agora, quais são os passos?

Configurando nosso projeto

Antes de empacotar nosso código e fazer o upload no PyPI, precisamos configurar alguns arquivos básicos em nosso projeto. Ao final dessa configuração, a árvore de arquivos na qual estamos trabalhando deve estar mais ou menos dessa forma:



Já temos pronto o `aluratemp.py`. Vamos, então, cuidar de cada um desses outros arquivos. Mas afinal, por que precisamos de tudo isso?

O PyPI é uma ferramenta para a comunidade, ou seja, uma ferramenta na qual todo mundo pode participar. Sendo assim, ele precisa de algumas regras e especificações, senão como ele vai saber como tratar o código que a gente mandar?

Em primeiro lugar, então, temos que entender que o PyPI trabalha com pacotes. Mais a frente, veremos como empacotar de fato nosso projeto, mas, antes disso, como o Python em si vai saber que a gente está trabalhando com pacotes?

Tratando o diretório como pacote com o `__init__.py`

Para o Python saber que queremos tratar o diretório de nosso código como um pacote, ele precisa de um arquivo em específico - o **init.py**.

A partir da versão **3** do Python esse arquivo torna-se, em parte, desnecessário. Nas versões anteriores, era realmente obrigatório, mesmo que estivesse vazio - ainda assim o Python reconheceria o diretório como pacote.

O arquivo `__init__.py` será executado logo quando alguém importar o pacote. Sabendo disso, será que queremos deixá-lo vazio? Se ficasse vazio, olha como seria para alguém importar as funções de `aluratemp.py`:

```
from aluratemp.aluratemp import *
```

Isso até funciona, mas não fica muito legal… Assim, há algumas técnicas que podemos usa no `__init__.py` para facilitar o uso de todo o nosso projeto, permitindo que a importação fique só assim:

```
from aluratemp import *
```

Para isso, podemos simplesmente inserir o código inicial de importação do arquivo no próprio arquivo de inicialização, assim:

```
from .aluratemp import *
```

> *O `.` antes do `aluratemp` é necessário para esse import no `__init__.py`, por tratar-se de um pacote.*

E agora já garantimos um fácil import de nossas funções!

Agora, o objetivo disso tudo é compartilhar nosso código - eu quero que as outras pessoas possam utilizar meu código como quiserem, afinal sou um grande entusiasta do **software livre** e do **open source**.

Mas como as pessoas vão saber se podem usar meu código ou não, com tantas leis a respeito de *copyright* hoje em dia? Em algumas jurisdições, chega a ser proibido o uso de um software que não tenha licença definida por alguém sem autorização direta do dono deste software!

Nem o próprio PyPI vai saber como tratar as permissões que eu quero passar sobre meu programa, se eu não explicitar. Como podemos deixar isso claro?

Definindo uma licença de uso em *LICENSE.txt*

É importante, tanto para o PyPI, quanto para a comunidade, que lembremos de criar um arquivo, geralmente *LICENSE.txt*, especificando a licença que queremos para nosso projeto.

Nesse caso, como quero algo bem simples e permissivo, usarei a **Licença MIT**, mas existem várias outras. Em caso de dúvida, temos alguns guias na Internet que podem nos ajudar, como o ***choosealicence*** do GitHub. Se for algo mais sério, pode ser bom procurar um apoio legal mais formal.

Meu arquivo *LICENSE.txt* ficou assim, então:

```
MIT License

Copyright (c) 2018 Yan Orestes

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documer
...
```

Legal, agora qualquer usuário vai saber se (e em quais condições) ele pode usar nosso software! Mas… espera, como as pessoas vão saber de que forma usar nosso código? Não tem nada em nenhum lugar ensinando a usar, nem dando nenhum exemplo… Será que tem algum padrão para isso?

Explicando nosso projeto no *README.md*

Repare que, da maneira que estamos agora, alguém que baixasse meu software precisaria ler todo o código para saber como funciona, ou mesmo sobre o que se trata.

Assim, além de uma licença, é comum os projetos conterem um arquivo **README**, explicando o funcionamento do software, dentre outras especificações. Esse arquivo pode ter extensões diversas, como a `.md`, de **Markdown**, que usaremos para simplificar a escrita do arquivo.

Faremos um README bem objetivo e curto, mas o ideal é que ele tenha informações suficientes para o usuário conseguir instalar, usar e entender seu projeto. Uma abordagem simples pode ser, também, usar uma template pronta com algum padrão de README.

No caso do **aluratemp**, deixei o *README.md* simples assim:

```
# aluratemp

**aluratemp** é um simples conversor de temperatura
Celsius-Fahrenheit (e vice-versa) escrito em Python.

## Funções

* `celsius_para_fahrenheit(temp_em_celsius)` - Recebe valor float em Celsius e
retorna valor float em Fahrenheit

* `fahrenheit_para_celsius(temp_em_fahrenheit)` - Recebe valor float em Fahrenheit
e retorna valor em Celsius
```

Olha como fica visualmente:

Legal! Mas como o PyPI vai saber que vamos chamar nosso arquivo de `README.md`? O mesmo vale para a licença, como ele vai encontrá-la se não sabe seu nome? Precisamos indicar isso, de alguma maneira…

Definindo padrões de opção no *setup.cfg*

Para essa questão, temos o arquivo *setup.cfg*. Nele, podemos definir algumas opções padrões para o *setup* do projeto. No nosso caso, precisamos especificar a localização do README e do LICENSE, dessa forma:

```
[metadata]
description-file = README.md
license_file = LICENSE.txt
```

Agora o PyPI já vai saber onde está cada coisa!

E aí? O que falta? Bem, precisamos de algo para empacotar de fato nosso projeto, para conseguirmos mandar lá pro PyPI. Com o que vamos fazer isso?

setup.py - O arquivo chave

Para empacotarmos e colocarmos mais algumas informações importantes no pacote, usamos um arquivo Python chamado de `setup.py`. Ele talvez seja o mais importante dos arquivos de configuração de projeto.

A estrutura básica do arquivo `setup.py` é a execução da função `setup()` com seus devidos argumentos. Essa função pode ser importada, principalmente, de dois módulos - `setuptools` e `distutils.core`.

A própria documentação do Python nos incentiva a usar o `setuptools`, mas ele nem sempre vem instalado por padrão no sistema. Caso isso aconteça, podemos instalar com o seguinte comando no terminal:

```
pip install setuptools
```

> *Dependendo do sistema, esse comando pode precisar de permissão de administrador (`sudo`) para funcionar propriamente.*

Se não tivéssemos como usar o *pip*, a solução seria usar o `setup()` do `distutils.core`, mesmo. Isso implicaria em não poder aproveitar ao máximo essa poderosa função, se limitando aos argumentos suportados por esse módulo.

Daqui para frente usaremos o `setuptools`, cuja lista de argumentos suportados pelo `setup()` é mais extensa.

Vamos começar a montar nosso código passando o parâmetro `name` para a função `setup()`, indicando o nome do nosso projeto:

```
from setuptools import setup
```

```
setup(
    name = 'aluratemp'
)
```

Tudo bem, passamos o nome de nosso projeto para o `setup()`. O problema é que apenas assim o código não funciona, porque precisamos de mais informações, como a versão (`version`), os autores (`author` e `author_email`) e os pacotes (`packages`, geralmente uma lista com o mesmo nome especificado em `name`):

```
from setuptools import setup

setup(
    name = 'aluratemp',
    version = '1.0.0',
    author = 'Yan Orestes',
    author_email = 'yan.orestes@alura.com.br',
    packages = ['aluratemp']
)
```

Agora sim! Isso é o básico que a função `setup()` deve receber. Porém, não seria legal poder passar, além desses, mais dados para aparecerem lá na página do projeto no PyPI e para facilitarem a busca? Mais uma vantagem da função `setup()` - em geral, podemos!

Por exemplo, podemos passar também uma curta descrição (`description`) para aparecer lá na página, em conjunto com uma mais longa (`long_description`).

Também podemos especificar uma URL (`url`)de página inicial para o projeto. No caso, vou colocar o repositório do GitHub onde o código se encontra. Em outro argumento (`project_urls`), podemos até passar um link de download:

```
setup(
    ## código omitido
    description = 'Um simples conversor de temperatura (Celsius - Fahrenheit)',
    long_description = 'Um simples conversor de temperatura, com funções para '
                       + 'conversão de Celsius para Fahrenheit e vice-versa, '
                       + 'usado para um post no Blog da Alura',
    url = 'https://github.com/yanorestes/aluratemp',
    project_urls = {
        'Código fonte': 'https://github.com/yanorestes/aluratemp',
        'Download': 'https://github.com/yanorestes/aluratemp/archive/1.0.0.zip'
    }
)
```

Podemos até passar a licença (`license`) que escolhemos (obviamente de maneira bem mais reduzida):

```
setup(
    # código omitido
    license = 'MIT'
)
```

Outras informações legais que podemos passar são as que influenciam diretamente na parte das buscas, como palavras-chave (**keywords**) e classificadores (**classifiers**). O próprio PyPI nos disponibiliza uma lista completa de classificadores para sabermos o que podemos usar para categorizar nosso projeto.

A lista de classificadores contém informações a respeito do status de desenvolvimento, de quem é o alvo do código, da língua nativa dele, dentre várias outras possíveis. No nosso caso ficou assim:

```
setup(
    # código omitido
    keywords = 'conversor temperatura alura',
    classifiers = [
        'Development Status :: 5 - Production/Stable',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: Portuguese (Brazilian)',
        'Operating System :: OS Independent',
        'Topic :: Software Development :: Internationalization',
        'Topic :: Scientific/Engineering :: Physics'
    ]
)
```

Ao final, nosso código ficou assim:

```
from setuptools import setup

setup(
    name = 'aluratemp',
    version = '1.0.0',
    author = 'Yan Orestes',
    author_email = 'yan.orestes@alura.com.br',
    packages = ['aluratemp'],
    description = 'Um simples conversor de temperatura (Celsius - Fahrenheit)',
    long_description = 'Um simples conversor de temperatura, com funções para '
                       + 'conversão de Celsius para Fahrenheit e vice-versa, '
                       + 'usado para um post no Blog da Alura',
    url = 'https://github.com/yanorestes/aluratemp',
    project_urls = {
        'Código fonte': 'https://github.com/yanorestes/aluratemp',
        'Download': 'https://github.com/yanorestes/aluratemp/archive/1.0.0.zip'
    },
    license = 'MIT',
    keywords = 'conversor temperatura alura',
    classifiers = [
        'Development Status :: 5 - Production/Stable',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: Portuguese (Brazilian)',
        'Operating System :: OS Independent',
        'Topic :: Software Development :: Internationalization',
        'Topic :: Scientific/Engineering :: Physics'
    ]
)
```

Agora que temos nosso projeto já configurado, podemos continuar o processo de empacotamento e upload de nosso software.

Empacotando nosso projeto

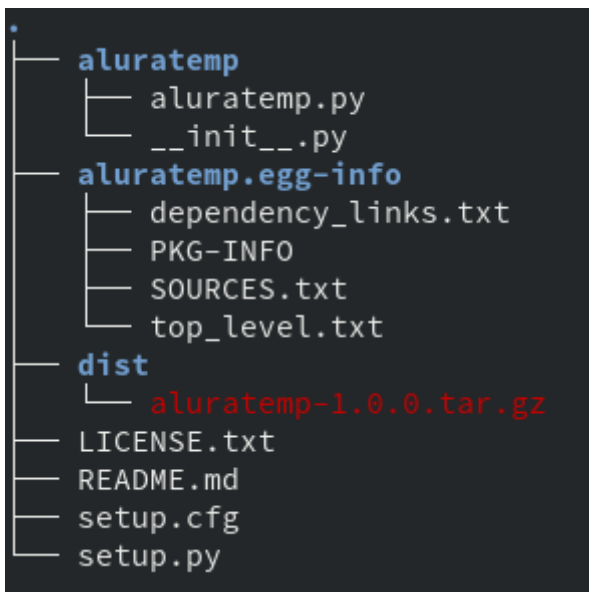
Já temos o projeto pronto, então o que fazemos agora? De alguma forma, precisamos passar para o PyPI algumas informações e metadados que permitam a instalação de nosso projeto pelo pip, ou ele não vai saber o que fazer com o nosso código!

Para isso, exploraremos o poder do arquivo `setup.py` que criamos e geraremos uma distribuição raiz, ou, tecnicamente, ***source distribution***, também chamada de ***sdist***, que é o mínimo que precisamos.

Para criarmos nossa *sdist*, basta um simples comando no terminal:

```
python setup.py sdist
```

Rodando esse comando, olha como ficou a árvore do nosso diretório:



Note que agora temos mais dois diretórios, `dist` e `aluratemp.egg-info`, que contêm um arquivo compactado do projeto e informações para o PyPI, respectivamente. Agora, podemos passar para o processo de upload.

Fazendo o upload de nosso projeto no PyPI

Agora que já temos nosso projeto pronto e empacotado, precisamos, enfim, colocá-lo mesmo lá no repositório do PyPI. Para isso, temos que **registrar nossa conta**.

Também vamos registrar uma conta no repositório de teste do PyPI, o **TestPyPI**, para testarmos todo o processo antes de colocar no repositório oficial.

Com nossas contas registradas, podemos fazer o upload. Há algumas maneiras de se fazer o upload. A maneira **recomendada pelo tutorial oficial do Python** é através do **twine**, que pode ser instalado pelo pip com o comando:

```
pip install twine
```

> *Dependendo do sistema, esse comando pode precisar de permissão de administrador (`sudo`) para funcionar propriamente.*

Com o twine instalado, usamos o comando `twine upload dist/*`. Como queremos primeiro fazer o upload no repositório teste, precisamos especificar a url dele (`https://test.pypi.org/legacy/`) com a opção `--repository-url`:

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

Teremos que digitar nossas informações de login e, enfim, **o pacote já estará disponível no repositório teste**!

Mas não é chato termos que ficar passando a URL e nossas informações de login toda vez que queremos fazer o upload? Podemos simplificar todo esse processo com um arquivo `.pypirc`, que deve ser armazenado na pasta HOME de nosso sistema:

```
[distutils]
index-servers=
    pypi
```

```
    testpypi

[pypi]
username: usuario
password: senha

[testpypi]
repository: https://test.pypi.org/legacy/
username: usuario
password: senha
```

Assim, apenas com o comando `twine upload dist/* -r testpypi` já daria certo!

> *Se não tivermos acesso ao pip, podemos usar o comando `python setup.py sdist upload -r pypitest`*

Agora, e para o repositório oficial? O comando é mais simples ainda: `twine upload dist/*`, e pronto! Nosso projeto já está disponível!

> *Se não tivermos acesso ao pip, podemos usar o comando `python setup.py sdist upload -r pypi`*

Assim, meus colegas poderão usar as funções de conversão que eu criei apenas instalando o pacote com o pip e importando-as nos seus próprios programas Python!