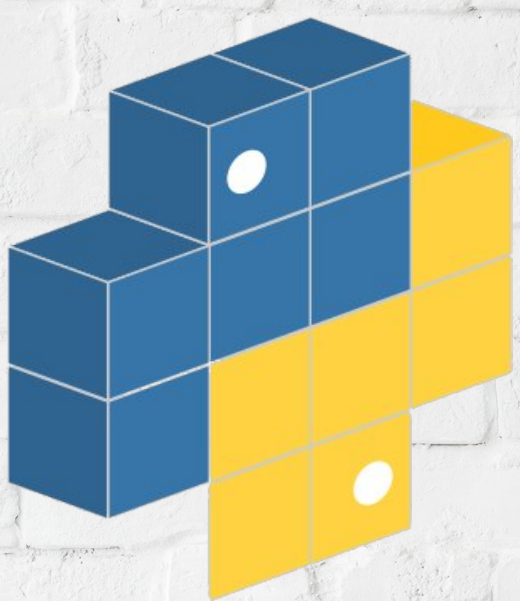
python modules and packages



André Santos, afs@inesctec.pt

Installing packages

- pip
- Python Package Index (PyPI)

```
pip install --user setuptools wheel twine
```

```
pip install --user \
  git+git://github.com/andrefs/python-aleixo50.git@master
```

Application layouts

- stand-alone script
- single script
- single package
- application with internal packages
- web application layouts

Stand-alone script

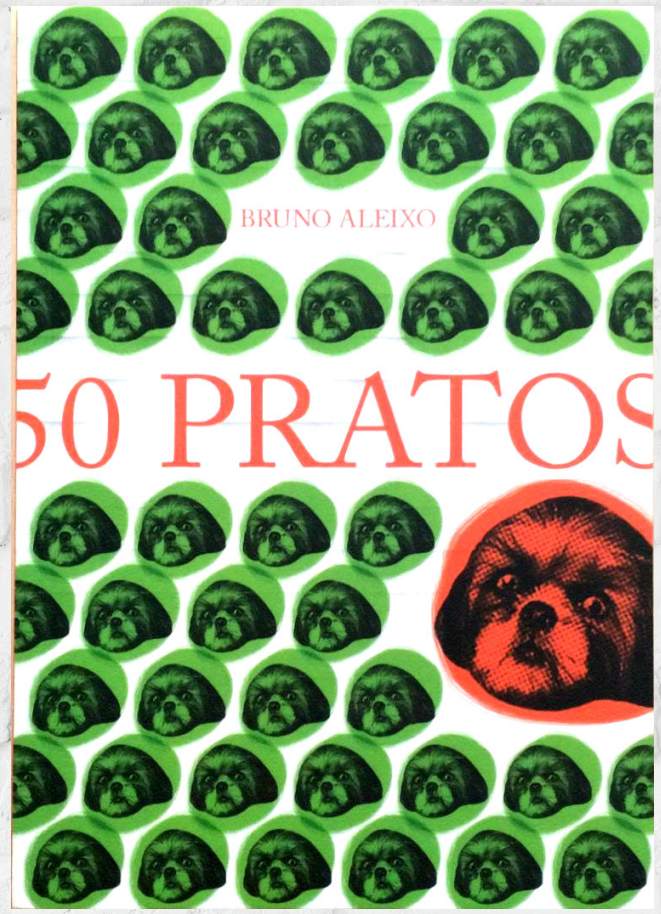
- **no dependency management**
- **not publishable on PyPI**
- **just copy it to `/usr/local/bin` or any folder in `$PATH`**

Single script

```
helloworld/
├── README.md
├── LICENSE
├── requirements.txt
├── setup.py
├── helloworld.py
└── tests.py
```

Single package: aleixo50

- Package holding information regarding Bruno Aleixo's 50 culinary dishes
- Allows to list all the dishes, select a random one, ...
- Has a command line script



Python package structure

```
aleixo50/
├── README.md
├── LICENSE
├── MANIFEST.in
├── setup.py
└── aleixo50/
```

Python package structure

```
aleixo50/aleixo50/
├── __init__.py
├── dish.py
├── recipes.json
├── dishes.py
├── rand.py
├── bin
│   └── aleixo50
├── test_dishes.py
└── test_rand.py
```

dish.py

```
class Dish(object):
    def __init__(self, name, ingrs=[], instrs=[]):
        self.name = name
        self.ingredients = ingrs
        self.instructions = instrs

    def __repr__(self):
        return ('Dish(                  + \
          '{0.name!r}, '                + \
          '{0.ingredients!r}, '         + \
          '{0.instructions!r})')          \
            .format(self)

    def __str__(self):
        return 'Dish({0.name})'.format(self)
```

recipes.json

```
[
  {"name": "Arroz de pato"},
  {"name": "Arroz de cabidela"},
  {"name": "Leitão"},
  {"name": "Dourada grelhada"},
  (...)
]
```

dishes.py

```
import os
import json
from .dish import Dish

_dirname = os.path.dirname(__file__)
_path = os.path.join(_dirname,”recipes.json”)

with open(_path, ’r’) as json_file:
    _recipes = json.loads(json_file.read())

dishes = [Dish(r[’name’]) for r in _recipes]
```

rand.py

```
import random
from .dishes import dishes

def rand():
    return random.choice(dishes)
```

aleixo50/__init__.py

```
from .dishes import dishes
from .rand import rand
```

bin/aleixo50

```
#!/usr/bin/python3

import getopt
import sys
import aleixo50

options, remainder = \
  getopt.getopt(sys.argv[1:], 'lr', ['list', 'rand'])
dict_opts = dict(options)

if '-r' in dict_opts or '--rand' in dict_opts:
    print(aleixo50.rand().name)
else:
    for i, r in enumerate(aleixo50.dishes):
        print(str.rjust(str(i+1), 3) + '. ' + r.name)
```

Python package structure

```
aleixo50/
├── README.md
├── LICENSE
├── MANIFEST.in
├── setup.py
└── aleixo50/
```

MANIFEST.in

```
include aleixo50/recipes.json
```

setup.py

```
import setuptools

with open("README.md", "r") as fh:
    long_description = fh.read()

setuptools.setup(
    name="aleixo50", version="0.0.1", author="André Santos",
    author_email="andrefs@andrefs.com",
    description="(...)", long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://github.com/andrefs/python-aleixo50",
    packages=setuptools.find_packages(),
    scripts=['bin/aleixo50'],
    include_package_data=True,
    classifiers=[
        "Programming Language :: Python :: 3",
        "Operating System :: OS Independent",
        "Topic :: Utilities"
    ],
)
```

Linting

```
pylint aleixo50
```

- **By default, it follows PEP8 conventions**
- **Behavior can be modified with a configuration file**

test_rand.py

```
import unittest
import aleixo50
print(aleixo50)

from aleixo50 import rand
from aleixo50.dish import Dish

class RandTest(unittest.TestCase):
    def test_rand_type(self):
        self.assertIsInstance(rand(), Dish)

if __name__ == '__main__':
    unittest.main()
```

test_dishes.py

```
import unittest
import aleixo50
print(aleixo50)

from aleixo50 import dishes

class RandTest(unittest.TestCase):
    def test_rand_type(self):
        self.assertEqual(len(dishes), 50)

if __name__ == '__main__':
    unittest.main()
```

Testing

```
python -m unittest discover
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Building the distribution

```
python3 setup.py sdist bdist_wheel
```

Publishing

Uploading to TestPyPI

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

Installing from TestPyPI

```
pip install \
  --index-url https://test.pypi.org/simple/ \
  aleixo50
```

Publishing

Uploading

```
twine upload dist/*
```

Installing

```
pip install --user aleixo50
```

Making a new release

1. **increment the version number in your `setup.py` file,**
2. **update your `CHANGES.txt` file,**
3. **if necessary, update the "Contributors" and "Thanks also to" sections of your `README.txt` file.**
4. **run `twine upload` again.**

Practical assignment #3

- **Due date: 1 Jul 2019**
- **By default, same groups**
- **Report + code + demo**

Pick one of the options described on the next slides.

Option 1

Given a large annotated corpus, create a tool capable of calculating lemmas and POS tags for an isolated word or for word(s) in a sentence.

Option 2

Create a tool capable of correcting “tracinho se” errors in written Portuguese such as “estives-te” or janta-se / jantasse.

Option 3

Create a spell checker for Mbundu (Umbundu/Kimbundu), a group of languages spoken in Angola.

Study the morfological rules of these languages, normalize a corpus and produce a list of words to be used to feed aspell, hunspell and/or jspell.

Option 4

Fetch text documents from a website (hint: pick one with an RSS feed), pre-process them, index them and implement a search functionality using the TF-IDF algorithm.

Option 5

Implement a song lyrics editor using Pat Pattison's methodology or similar: build a list of 10 words related to the song's theme, find related words and rhyming words.

Use WordNet or similar knowledge bases and rhyming dictionaries.

Option 6

Create TMX files using `Beautiful Soup` to process websites like Linguee.

Option 7

Python interface for a Prolog constraint solver for genealogy.

Option 8

Implement a program which, given a set of (rhyming) poems, calculate a rhyming dictionary.

Option 9

...

Proceed with caution

- Tell us which option your group will be doing (email)
- **Come and talk to us before starting!**
- Assignment descriptions are vague
- Most of the options need a brainstorm before begining
- We can help narrowing the scope of the assignment to make it feasible
    - ...or the inverse :)

And also

- **Bonus points for**
  - **dealing with large ammounts of text**
  - **calculating performance metrics (precision, recall, ...)**