Live de Python #37

Parametrizando configurações

Roteiro

- Qual a importância de parametrizar programas?
- configparser - parametrizando execuções
- Turbinando a calculadora dos BBs

Qual a importância de parametrizar? [0]

Diferente do que vimos na live #36, a parametrização do fluxo de execução pode oferecer diferentes tipos de comportamento. Ou seja, podemos debugar a aplicação, podemos iniciar ela e modo de deploy ....

Qual a importância de parametrizar? [1]

Vamos usar como exemplo o arquivo de configuração do TOX (https://tox.readthedocs.io).

```
[tox]
envlist = clean,lint,py36,stats

[testenv:clean]
deps =
  coverage

commands =
  coverage erase

[testenv:lint]
deps =
  coverage
  pylama
  xenon

commands =
  coverage run --source=splitty -m unittest discover -s tests/

  xenon --max-absolute B --max-modules A --max-average A splitty
  pylama splitty/
```

Qual a importância de parametrizar? [2]

Existem quatro modos diferentes para execução dos mesmo fluxo no tox.

(Clean, Lint, py36, stats)

```
[tox]
envlist = clean,lint,py36,stats

[testenv:clean]
deps =
  coverage

commands =
  coverage erase

[testenv:lint]
deps =
  coverage
  pylama
  xenon

commands =
  coverage run --source=splitty -m unittest discover -s tests/

  xenon --max-absolute B --max-modules A --max-average A splitty
  pylama splitty/
```

Qual a importância de parametrizar? [3]

E cada uma delas tem seus comandos e fluxo diferentes.

Nesse caso, uma dependência e um único comando

```
[tox]
envlist = clean,lint,py36,stats

[testenv:clean]
deps =
  coverage

commands =
  coverage erase

[testenv:lint]
deps =
  coverage
  pylama
  xenon

commands =
  coverage run --source=splitty -m unittest discover -s tests/

  xenon --max-absolute B --max-modules A --max-average A splitty
  pylama splitty/
```

Qual a importância de parametrizar? [4]

Existem diversos sistemas baseados em arquivos ‘.ini’ e também formas diferentes de fazer a leitura do mesmo. Vamos ver hoje a bateria inclusa que toma conta desses arquivos (configParser).

Vale lembrar que existem outros formatos de arquivos de configuração. Como o .json, .yml, etc...

O python-decouple é um bom exemplo de biblioteca que trabalha com n tipos de arquivos de configuração diferente. (podemos falar sobre isso outro dia)

ConfigParser

(https://docs.python.org/3/library/configparser.html)

Entendendo o configparser

“Este módulo fornece a classe ConfigParser que implementa uma linguagem de configuração básica que fornece uma estrutura semelhante à encontrada nos arquivos INI do Microsoft Windows. Você pode usar isso para escrever programas Python que podem ser personalizados pelos usuários finais com facilidade.” (doc)

Fazer antes de explicar pode ser legal.

```
[default]
verbose = 1
debug = True
```

```python
from configparser import ConfigParser

config = ConfigParser()
config.read('simple.ini')
default_config = dict(config['default'])
```

Entendendo o que há por trás [0]

ConfigParser(**defaults**,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: É um dicionário que será inicializado junto ao config, para suprir configurações não passadas no arquivo ini, por exemplo.

Default: None

Entendendo o que há por trás [1]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Define qual o tipo de mapping que será utilizado pelo config parser, pode ser que você tenha um dicionário modificado. (abc....)

Default: collections.OrderedDict

Entendendo o que há por trás [2]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Permite que caso uma chave seja chamada sem valor, não estoure uma exceção. É uma boa pedida para sistemas complexos.

Default: False

Entendendo o que há por trás [3]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Deixa explícito quais serão os delimitadores. Podemos criar novos padrões, talvez seja necessário para formalizar algo.

Default: ('=', ':')

Entendendo o que há por trás [4]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Delimita quais serão os caracteres de comentário. Afinal, nunca se sabe onde vamos parar.

Default: ('#', ';')

Entendendo o que há por trás [5]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Oferece a opção de comentar linhas do arquivo de configuração. Pode ser uma explicação do param, etc...

Default: None

Entendendo o que há por trás [6]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Permite que os blocos se repitam no arquivo de configuração, pode ocorrer em um arquivo muito grande, ou caso você não queira mexer no que já funciona, então você sobrescreve alguma opção

Default: True

Entendendo o que há por trás [7]

ConfigParser(defaults,
dict_type,
allow_no_value,
delimiters,
comment_prefixes,
inline_comment_prefixes,
strict,
empty_lines_in_values,
default_section,
interpolation,
converters)

Defaults: Especifica uma sessão default, um dicionário contendo todas as sessões do nosso arquivo ini.

Default: configparser.DEFAULTSECT

CODE !!!