Live de Python #36

Parametrizando programas com python

Roteiro

- Qual a importância de parametrizar programas?
- Argparse, parametrizando as chamadas no terminal
- Parametrizando um scraper (download de letras de música)

Qual a importância de parametrizar? [0]

Imagine que tudo que é bom pode ficar melhor se oferecer opções. Não é verdade?

Um sanduíche é muito legal e com mais queijo por 1 real, pode ser uma opção muito atraente, não é mesmo?

Agora imagine poder ter opções de graça, É MUITO MAIS LEGAL. Claro, isso se você for um usuário, se você for um desenvolvedor terá que pagar aquele 1 real.

Qual a importância de parametrizar? [1]

Agora, antes de dizer mais nada, vamos olhar para o ls dos sistemas UNIX.

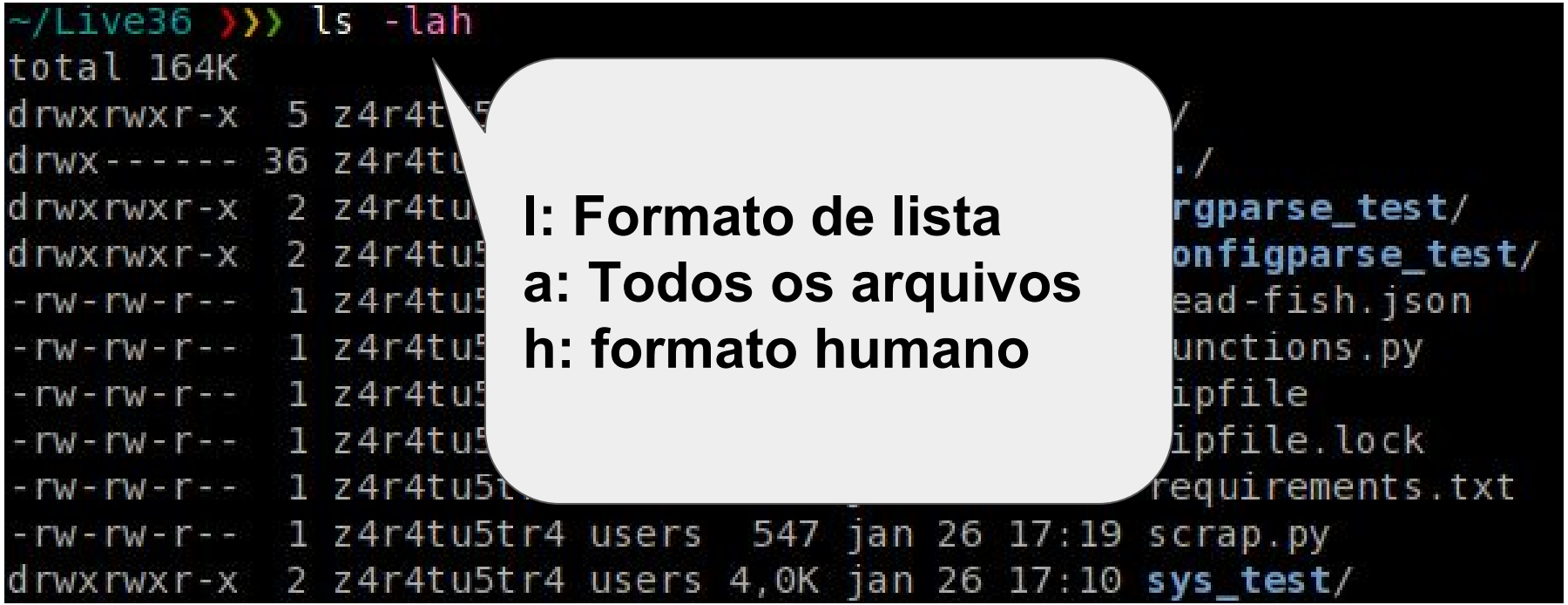
Quando uso **ls** ele me mostra todos os arquivos do diretório atual, claro, os que não estejam ocultos. Porém, sou uma pessoa curiosa. Se eu quiser ver os ocultos deve ter uma maneira de fazer isso, não? Ou vou ter que fazer um novo programa só pra isso? ls e ls**o** só para listar ocultos

Agora vamos pensar, se adicionar um ls **-a** poderia listar todos os arquivos, isso não seria legal e útil? (a: all)

Qual a importância de parametrizar? [2]

```
~/Live36 ❯❯❯ ls -lah
total 164K
drwxrwxr-x  5 z4r4tu5tr4 users 4,0K jan 26 17:17 ./
drwx------ 36 z4r4tu5tr4 users 4,0K jan 27 19:35 ../
drwxrwxr-x  2 z4r4tu5tr4 users 4,0K jan 26 17:09 argparse_test/
drwxrwxr-x  2 z4r4tu5tr4 users 4,0K jan 26 17:08 configparse_test/
-rw-rw-r--  1 z4r4tu5tr4 users 117K jan 26 17:15 dead-fish.json
-rw-rw-r--  1 z4r4tu5tr4 users  567 jan 26 17:18 functions.py
-rw-rw-r--  1 z4r4tu5tr4 users  153 jan 26 17:14 Pipfile
-rw-rw-r--  1 z4r4tu5tr4 users 5,5K jan 26 17:14 Pipfile.lock
-rw-rw-r--  1 z4r4tu5tr4 users  120 jan 26 17:16 requirements.txt
-rw-rw-r--  1 z4r4tu5tr4 users  547 jan 26 17:19 scrap.py
drwxrwxr-x  2 z4r4tu5tr4 users 4,0K jan 26 17:10 sys_test/
```

Qual a importância de parametrizar? [3]



Qual a importância de parametrizar? [4]

```
~/Live36 ››› ls --help
Uso: ls [OPÇÃO]... [ARQUIVO]...
Lista informações sobre os ARQUIVOs (no diretório atual por padrão).
Lista as entradas em ordem alfabética se não for usada nenhuma opção -cftuvSUX
nem --sort.

Argumentos obrigatórios para opções longas também o são para opções curtas.
  -a, --all                  não ignora entradas começando com .
  -A, --almost-all           não lista as entradas implícitas . e ..
      --author               com -l, emite o autor de cada arquivo
  -b, --escape               emite escapes no estilo C para caracteres não-
                               gráficos
      --block-size=TAM       o tamanho considera blocos de TAM bytes; exemplo:
                               "--block-size=M" emite tamanhos em unidades de
                               1.048.576 bytes; veja o formato de TAM abaixo
  -B, --ignore-backups       não lista as entradas implícitas terminadas com ~
  -c                         com -lt: ordena por, e mostra, ctime (hora da
                               última modificação da informação de estado do
                               arquivo);
```

Argparse

O objeto ArgumentParser

```
from argparse import ArgumentParser

parser = ArgumentParser(
    prog='O Nome do programa (default: sys.argv[0])',
    usage='String descrevendo como usar o programa',
    description='Mensagem que será exibida no help (default: none)',
    epilog='Texto que será exibido após os parametros (default: none)',
    fromfile_prefix_chars='@')  # 'Arquivos com os parâmetros pré-fixados'
```

O objeto ArgumentParser

```
~/L/argparse_test ❯❯❯ python test_1.py
usage: String descrevendo como usar o programa

Mensagem que será exibida no help (default: none)

optional arguments:
  -h, --help  show this help message and exit

Texto que será exibido após os parametros (default: none)
```

Adicionando parâmetros (add_argument)

parser.add_argument(
name or flags,
action,
nargs,
const,
default,
type,
choices,
required,
help,
metavar,
dest)

É como a opção será invocada via linha de comando.

Por exemplo:

```
parser.add_argument('-v',
                    '--verbose',
                    '--cabelo')
```

Vale lembrar que aqui acontece o desempacotamento (*args) então, tudo que não for nomeado vai servir de atalho para o mesmo parâmetro.

Adicionando parâmetros (add_argument)

parser.add_argument(

name or flags,
action,
nargs,
const,
default,
type,
choices,
required,
help,
metavar,
dest)

Outro ponto bacana de ser comentado, é que caso o argumento seja declarado sem o ‘-’ ele passará a ser obrigatório para execução:

```
parser.add_argument('x')
```

E também, a ordem de chamada será importante para o parse

Adicionando parâmetros (add_argument)

parser.add_argument(
name or flags,
action,
nargs,
const,
default,
type,
choices,
required,
help,
metavar,
dest)

A ação default é armazenar o argumento como um atributo de **parser**, porém, existem outros tipos de ação.

```
parser.add_argument('-f',
                    '--foo',
                    action='store')
```

por default, sempre será ‘store’

Adicionando parâmetros (add_argument)

Argumento	O que faz?
store	Armazena o valor do argumento
store_const	Armazena um valor (uma constante). Para valores opcionais
store_true	Armazena True
store_false	Armazena False
append	Monta uma lista com o mesmo argumento (-f=1,-f=2) [1,2]
append_const	Armazena uma constante em formato de append
count	Conta quantas vezes o mesmo parâmetro foi invocado
...	

Adicionando parâmetros (add_argument)

```
parser.add_argument(
        name or flags,
        action,
        nargs,
        const,
        default,
        type,
        choices,
        required,
        help,
        metavar,
        dest)
```

Caso você use alguma ação de append, dest vai fixar todos os parâmetros passado em uma lista, por exemplo.

```
add_argument('--str', dest='types',
             action='append_const')
```

por default, sempre será None

Adicionando parâmetros (add_argument)

parser.add_argument(
name or flags,
action,
nargs,
const,
default,
type,
choices,
required,
help,
metavar,
dest)

Default é sobre qual valor o argumento terá, caso ele não seja invocado na linha de comando

```
parser.add_argument('-f',
                    default=42)
```

Adicionando parâmetros (add_argument)

```
parser.add_argument(
        name or flags,
        action,
        nargs,
        const,
        default,
        type,
        choices,
        required,
        help,
        metavar,
        dest)
```

Type é referente ao tipo em que o dado vai estar quando o parse for feito, por default, tudo é string

Adicionando parâmetros (add_argument)

parser.add_argument(
name or flags,
action,
nargs,
const,
default,
type,
choices,
required,
help,
metavar,
dest)

Choices é uma maneira de restringir as opções do usuário, por exemplo, ele pode usar {1,2,3}. Então o parse será limitado a esse valores

Adicionando parâmetros (add_argument)

```
parser.add_argument(
        name or flags,
        action,
        nargs,
        const,
        default,
        type,
        choices,
        required,
        help,
        metavar,
        dest)
```

Required é referente a se o argumento será obrigatório ou não (bool)

Adicionando parâmetros (add_argument)

```
parser.add_argument(
        name or flags,
        action,
        nargs,
        const,
        default,
        type,
        choices,
        required,
        help,
        metavar,
        dest)
```

Quando a ajuda for chamada, help será mostrado na tela com a explicação daquele parâmetro

Adicionando parâmetros (add_argument)

```
parser.add_argument(
        name or flags,
        action,
        nargs,
        const,
        default,
        type,
        choices,
        required,
        help,
        metavar,
        dest)
```

Metavar é uma opção muito inteligente para quando seu objeto tem mais de uma chamada. Assim, como será referenciado no help?