strup

Jens B. Helmers

Oct 24, 2020

CONTENTS

strup is a package for unpacking basic data objects from a text string.

strup is written by Jens B. Helmers (c) 2020

SPDX-License-Identifier: MIT

CHAPTER

ONE

THE FUNCTION *UNPACK*

strup.**unpack**(*fmt*, *text*, **args*, ***kwargs*)

Extract basic data types from text based on fmt.

Parameters

- **fmt** (*str*) – fmt[i] defines the type of item i in the output tuple. fmt[i] must be 'i', 'f', 's', '?' or '.'. Item i will be ignored if fmt[i]=='.'.
- **text** (*str*) – The text string to extract the objects from

Other Parameters

- ***args** (*list, optional*) – Additional variable length argument list to be submitted to the constructor of Unpack
- ****kwargs** (*dict, optional*) – Additional keyword arguments to be submitted to the constructor of Unpack

Returns The tuple of objects parsed from text.

Return type tuple

Raises **ValueError** – If any parsing error occur.

Examples

```
>>> unpack("ifs?", "5 2.3   ole  True")
(5, 2.3, 'ole', True)
>>> unpack("isf", "100 'Donald Duck' 125.6", quote="'")
(100, 'Donald Duck', 125.6)
```

CHAPTER

TWO

THE CLASS *UNPACK*

class strup.**Unpack**(*fmt*, *sep=None*, *none=False*, *quote=None*, *quote_escape=None*)

Unpack is a Python package for unpacking basic data types from a text string.

Each item is of type 'int', 'float', 'string' or 'bool' depending on a format code in the constructor.

__init__(*fmt*, *sep=None*, *none=False*, *quote=None*, *quote_escape=None*)

Constructor for Unpack.

Parameters

- **fmt** (*str*) – fmt[i] defines the type of item i in the output tuple. fmt[i] must be 'i', 'f', 's', '?' or '.'. Item i will be ignored if fmt[i]=='.'.
- **sep** (*str or None, optional*) – String to separate items. See string.split() method.
- **none** (*bool, optional*) – If True: Zero-sized items are interpreted as None.
- **quote** (*str or None, optional*) – String items are sometimes enclosed by quote characters. Quotes are mandatory if string items includes the sep or quote characters. A quote character inside an item must be escaped by 'quote_escape'. (See example below). It is not possible to apply quotes if quote==''.
- **quote_escape** (*str or None, optional*) – Typical values are '""', "'", r'"' or '""'. quote_escape = None is interpreted as quote_escape = quote*2

Raises **ValueError** – If any parsing error occur.

Examples

See the __call__() examples for application of these decoders:

```
>>> decode1 = Unpack('ifssis')
>>> decode2 = Unpack('.fs', sep=',')
>>> decode3 = Unpack('isfs', sep=' ', quote='"', quote_escape='""')
```

__call__(*text*)

Extract the tuple of objects by parsing text based on self._fmt

Parameters **text** (*str*) – The text string to extract the objects from

Returns The tuple of objects parsed from text.

Return type tuple

Raises **ValueError** – If any parsing error occur.

Examples

decode1, decode2 and decode3 as defined in the __init__() examples:

```
>>> decode1("3 4.5  ole dole 5 doffen")
(3, 4.5, 'ole', 'dole', 5, 'doffen')
>>> decode2("3,4.5,  ole,dole,5,doffen")
(4.5, '  ole')
>>> decode3('3 "A ""quote"" test" 93.4 knut ignored')
(3, 'A "quote" test', 93.4, 'knut')
```

CHAPTER

THREE

`STRUP` — STRING UNPACK

This Python package is for unpacking basic objects from a text string. The standard data types `string`, `int`, `float` and `bool` are supported.

CHAPTER

FOUR

THE FUNCTION `unpack()`

We may extract the objects from a text string `text` using the utility function `unpack(fmt, text)`. Each format character in the string `fmt` indicates the data type for the corresponding object.

```
>>> from strup import unpack
>>> i, x, s, ok = unpack("ifs?", "5 2.3   ole  True")
>>> i, x, s, ok
(5, 2.3, 'ole', True)
```

The format characters for the data types are consistent with the syntax applied in the standard library module struct for handling of binary data. Characters in `fmt` are case sensitive.

Character	Data Object
i	int
f	float
s	string
?	bool
.	ignore this item

Each eventual dot inside `fmt` indicates that the corresponding item should not be part of the result.

```
>>> unpack("f..s", "2.3 ole 55   dole")
(2.3, 'dole')
```

In case of bool objects, the actual item of `text` must follow the convention applied in distutils.util.strtobool. Consequently, *y*, *yes*, *t*, *true*, *on* and *1* are interpreted as *True* and *n*, *no*, *f*, *false*, *off* and *0* as *False*. For all other values a `ValueError` exception is raised.

```
>>> strup.unpack("?????s????", "NO 0 F off False ---  yes 1 ON TruE")
(False, False, False, False, False, '---', True, True, True, True)
```

The set of items to consider from the string `text`, is by default the items returned from the standard library `text.split()` method.

Only the `len(fmt)` first items of `text.split()` are considered. Trailing dots are not needed in `fmt` and should not be specified.

CHAPTER

FIVE

OPTIONAL PARAMETERS

The optional argument `sep` as defined in the standard Python `string.split()` is also applicable in this context.

```
>>> unpack("f..s", " 2.3 ,ole,55,   dole", sep=',')
(2.3, '   dole')
```

By specifying the optional parameter `none=True`, zero-sized string items in `text` are interpreted as `None` independent of the format character. By default `none=False`.

```
>>> unpack("fissi", "2.3,,, ,12", sep=',', none=True)
(2.3, None, None, ' ', 12)
```

String objects are often defined using quotes. The optional argument `quote` has default value `None` but may be `"` or `'`.

```
>>> unpack("isf", "100 'Donald Duck' 125.6", quote="'")
(100, 'Donald Duck', 125.6)
```

Eventual quotes inside quoted strings are controlled using the optional argument `quote_escape`. By default `quote_escape=None` means that internal quotes are identified in `text` using double quotes

```
>>> unpack("isf", "100 'She''s the best' 125.6", quote="'")
(100, "She's the best", 125.6)
>>> unpack("isf", '3 "A ""quote"" test"  93.4 ignored', quote='"')
(3, 'A "quote" test', 93.4)
```

However, other escape sequences are supported like `quote_escape=r"\'"` or `quote_escape=r'\"'`

```
>>> unpack("isf", r"100 'She\'s the best' 125.6", quote="'", quote_escape=r"\'")
(100, "She's the best", 125.6)
```

CHAPTER

SIX

THE CLASS UNPACK

All processing within the function `unpack()`, as described above, is handled by the class `Unpack`.

```
>>> from strup import Unpack
```

All arguments for the function `unpack()`, except `text`, are handled by the constructor of `Unpack`. This constructor also performs preprocessing. Finally, `Unpack.__call__()` process the actual `text`.

Consequently, when the same unpack pattern is applied in loops, we may benefit from utilizing `Unpack` directly.

```
>>> mydecode = Unpack('.s..f', quote='"')      # Preprocess the pattern
>>> for line in ['5.3 "Donald Duck" 2 yes 5.4',
                 '-2.2 "Uncle Sam" 4  no 1.5',
                 '3.3  "Clint Eastwood" 7 yes 6.5']:
...       mydecode(line)
("Donald Duck", 5.4)
("Uncle Sam", 1.5)
("Clint Eastwood", 6.5)
```

CHAPTER

SEVEN

EXCEPTION HANDLING

Exceptions	Description
`ValueError`	Input error with relevant error message

```
>>> w1, w2, ival, w3 = unpack("ssis", "you,need,some,help", sep=",")
Traceback (most recent call last):
   File "e:\repositories\github\jeblohe\strup\strup\unpack.py", line 85, in unpack
   raise ValueError(msg)
ValueError: strup.unpack()
fmt='ssis'
text='you,need,some,help'
argv=(), kwargs={'sep': ','}
Error decoding element 2:'some' of items=['you', 'need', 'some', 'help']
```

CHAPTER

EIGHT

API

Docstrings from the source code are provided *here*.

CHAPTER

NINE

CONSIDERATIONS

A major goal with *strup* is to provide a clean and intuitive interface. If standard string methods are too low level and the re-module adds too much complexity, then *strup* might be your compromise.

Backward compatibility of the API is strongly emphasized.

strup will not grow into a general purpose parser. Text processing is in general a comprehensive topic. For high volume text processing it is recommended to apply optimized packages like numpy and pandas.

CHAPTER

TEN

INSTALLATION

This package is platform independent and available from PyPI and Anaconda.

To install *strup* from PyPI:

```
pip install strup           # For end users
pip install -e .[dev]       # For package development (from the root of your strup␣
↪repo)
```

or from Anaconda:

```
conda install -c jeblohe strup
```

The source code is hosted on GitHub. Continuous integration at CircleCI. The code is extensively tested on Python 2.7, 3,4, 3.5, 3.6, 3.7, 3.8 and 3.9. The test coverage is reported by Coveralls.

CHAPTER

ELEVEN

LICENSE

This software is licensed under the MIT-license.

CHAPTER

TWELVE

VERSION

12.1 1.0.0 - 2020.10.24

First official release

PYTHON MODULE INDEX

INDEX

Symbols

M

S

U