PyStrExt Documentation

Release 0.1.6

Eric Lapouyade

November 04, 2015

Contents

1 Indices and tables 11

Python Module Index 13

Quickstart

To install:

```
pip install pystrext
```

Usage:

```
>>> import pystrext as strext

>>> h,m,s = strext.extracts('elapse time : 5h 7m 36s','(\d+)h (\d+)m (\d+)s',['0']*3)
>>> h,m,s
('5', '7', '36')
...
```

Functions index

MB_GB(str[, MB, GB])	Transforms a string representing a memory MegaByte value into GigaBytes if value >
MHz_GHz(str)	Transforms a string representing a frequency from Mhz to Ghz if value > 1024
base62_decode(str)	decode a base62 encoded number
base62_encode(i)	encode an integer into base62
compress(s)	Compress a string with gzip
extract(str, pattern)	Search a pattern and return the first group of the first match.
extracts(str, pattern, default)	Search a pattern and return groups of the first match.
file_unicode(f)	Convert a filename (unicode, utf-8 or iso-8859-1) into unicode
file_unicode_list(l)	Convert filenames into unicode strings inside a list
get_col(str, **kwargs)	Extract a column from a string
if_val(val, str1, str2)	Select one string depending on a value
is_email_valid(email)	Check string is a correct email address
is_ip_valid(address)	Check string is a correct IP address (ipv4)
no_one_many(n, str0, str1, str2)	Select one string depending on a n equal to 0,1 or many
plural(str1, str2, n)	Select one string depending on a numeric value
random_password([length])	Generate an “easy to memorize” random password
remove_accents(s)	Remove accents
slugify(value)	Convert string to a slug
truncate(str, maxsize[, max_end_str, ...])	Truncate a string to a specified length and add a suffix (if truncated only)
uncompress(s)	Uncompress a gzipped string
version_lt(v1, v2)	Compare 2 strings representing dotted version number
vjust(str[, level, delim, bitsize, fillchar])	Justify a string representing dotted version number

Functions documentation

Python String Extension

With pystrext module, you will be able to manipulate strings.

Created on 2 Feb. 2010
@author: elapouya

pystrext.**MB_GB**(*str*, *MB='MB'*, *GB='GB'*)
Transforms a string representing a memory MegaByte value into GigaBytes if value > 1024

One can specify his own unit

Parameters

- **str** (*str*) – string representing a value to be devided by 1024 if > 1024
- **MB** (*str*) – string for the lowest unit
- **GB** (*str*) – string for the highest unit

Returns The converted string

Return type str

Examples

```
>>> MB_GB('512')
>>> '512 MB'
>>> MB_GB('2048')
>>> '2 GB'
>>> MB_GB('2048','MBytes','GBytes')
>>> '2 GBytes'
```

pystrext.**MHz_GHz**(*str*)

Transforms a string representing a frequency from Mhz to Ghz if value > 1024

Parameters **str** (*str*) – string representing the frequency (Mhz)

Returns The converted string

Return type str

Examples

```
>>> MHz_GHz('377')
>>> '377 MHz'
>>> MHz_GHz('2048')
>>> '2.048 GHz'
```

pystrext.**base62_decode**(*str*)

decode a base62 encoded number

Parameters **str** (*str*) – base62 string

Returns decoded integer

Return type int

Examples

```
>>> base62_decode('GW')
>>> 1024
```

pystrext.**base62_encode**(*i*)

encode an integer into base62

This can be usefull when encoding an item id to build a short url : see url shorteners like http://pack.li

Parameters **i** (*int*) – the number to convert

Returns base62 encoded string

Return type str

Examples

```
>>> base62_encode(1024)
>>> 'GW'
```

pystrext.**compress**(*s*)

Compress a string with gzip

It could be useful to compress some data without creating a file, this function do that.

Parameters **s** (*str*) – The strings to compress

Returns The compressed string

Return type str

Examples

```
>>> s = "monty python" * 80
>>> len(s)
960
>>> c = compress(s)
>>> len(c)
41
>>> u = uncompress(c)
>>> len(u)
960
>>> s == u
True
```

pystrext.**extract**(*str*, *pattern*)

Search a pattern and return the first group of the first match.

The pattern must include a group selection, ie : it must include parentheses. Only the part inside the parentheses will be returned.

Parameters

- **str** (*str*) – The string to search a pattern
- **pattern** (*RegexObject or str*) – A regular expression object or a string for the pattern to search

Returns The extracted strings that matches the pattern or **None** if no match.

Return type str

Examples

```
>>> extract('the full monty python','(\w+) python')
'monty'
>>> r=re.compile('>([^<]*)<')
>>> extract('this is text form : >the answer<',r)
'the answer'
```

pystrext.**extracts**(*str*, *pattern*, *default*)
Search a pattern and return groups of the first match.

The pattern must include a group selections, ie : it must include parentheses. Only the part inside the parentheses will be returned.

Parameters

- **str** (*str*) – The string to search a pattern
- **pattern** (*RegexObject or str*) – A regular expression object or a string for the pattern to search

Returns The extracted string for each group that matches the pattern or *default* argument if no match.

Return type list

Examples

```
>>> h,m,s = extracts('elapse time : 5h 7m 36s','(\d+)h (\d+)m (\d+)s',['0']*3)
>>> h,m,s
('5', '7', '36')
```

pystrext.**file_unicode**(*f*)
Convert a filename (unicode, utf-8 or iso-8859-1) into unicode

Parameters **f** (*str*) – a filename

Returns converted filename

Return type unicode

pystrext.**file_unicode_list**(*l*)
Convert filenames into unicode strings inside a list

Parameters **l** (*list*) – a filename list

Returns converted filename list

Return type list

pystrext.**get_col**(*str*, ***kwargs*)
Extract a column from a string

Some strings have got many columns seperated with a separator. Some strings may also have some sub-columns seperated with another separator. get_col() can extract **one** column/sub-column at any depth level. You have to specify one separator and one column number for each depth level you want to select.

Arguments name is important, it must be : col<n> and sep<n>.

Arguments are sorted so sep1/col1 is searched before sep2/col2

The separator can be a regular expression (by default will be 'W+')

Parameters

- **str** (*str*) – The listing row to parse
- **sep1** (*str*) – sperator 1
- **col1** (*str*) – column number 1

- **sepn** (*str*) – sperator n
- **coln** (*str*) – column number n

Returns The column/sub-column requested

Return type str

Examples

```
>>> get_col("  4   0  95   0   0   0|   0    72k| 352k   40k|   0     0 | 435   138 ",col1=2,sep
'40k'
>>> get_col("/a/b/c/basename.date.jpg",col1=-1,sep1='/',col2=1,sep2='\.')
'date'
```

pystrext.**if_val**(*val*, *str1*, *str2*)

Select one string depending on a value

This function return str1 if *val* True, str2 if False.

str1 or str2 can be a string that may include '%(val)s' : it will replaced by *val* value.
str1 or str2 can be a callable : it will be called with *val* as argument

Parameters

- **val** (*any type*) – value to test
- **str1** (*str or callable*) – The string to return if *val*
- **str2** (*str or callable*) – The string to return if not *val*

Returns str1 or str2 or str1(val) or str2(val)

Return type str

Examples

```
>>> print if_val(3,"Item(s) found : %(val)s","No item found")
Item(s) found : 3
>>> print if_val(0,"Item(s) found : %(val)s","No item found")
No item found
```

pystrext.**is_email_valid**(*email*)

Check string is a correct email address

It just checks string syntax. Useful for form checking.

Parameters **address** (*str*) – email address string to check

Returns True if email string syntax is correct.

Return type bool

Examples

```
>>> is_email_valid('hello@world.com')
>>> True
>>> is_email_valid('hello@world')
>>> False
>>> is_email_valid('hello world')
>>> False
```

pystrext.**is_ip_valid**(*address*)

Check string is a correct IP address (ipv4)

It just checks IP string syntax. Useful for form checking.

Parameters **address** (*str*) – IP address string to check

Returns True if IP string syntax is correct.

Return type bool

Examples

```
>>> is_ip_valid('12.23.34.45')
>>> True
>>> is_ip_valid('12.23.34.345')
>>> False
>>> is_ip_valid('12.23.34')
>>> False
>>> is_ip_valid('12.23.34a.45')
>>> False
>>> is_ip_valid('12.23.34.45.56')
>>> False
```

pystrext.**no_one_many**(*n*, *str0*, *str1*, *str2*)

Select one string depending on a n equal to 0,1 or many

This function will return :

- str0 if n <= 0
- str1 if n == 1
- str2 if n >= 2

str0, str1 or str2 can be a string that may include '%(n)s' : it will replaced by *n* value.
str0, str1 or str2 can be a callable : it will be called with *n* as argument

Parameters

- **n** (*int*) – value to test
- **str0** (*str or callable*) – The string to return if n <= 0
- **str1** (*str or callable*) – The string to return if n == 1
- **str2** (*str or callable*) – The string to return if n >= 2

Returns str0 or str1 or str2

Return type str

Examples

```
>>> print no_one_many(0,"No item","One item","%(n)s items")
No item
>>> print no_one_many(1,"No item","One item","%(n)s items")
One item
>>> print no_one_many(36,"No item","One item","%(n)s items")
36 items
```

pystrext.**plural**(*str1*, *str2*, *n*)

Select one string depending on a numeric value

Parameters

- **str1** (*str*) – The string to return if *n* <= 1
- **str2** (*str*) – The string to return if *n* > 1
- **n** (*int*) – the numeric value to test

Returns str1 if *n* is 1 or less, str2 otherwise.

Return type str

Examples

```
>>> n=1
>>> print "found %d %s" % (n,plural("item","items",n))
found 1 item
>>> n=4
>>> print "found %d %s" % (n,plural("item","items",n))
found 4 items
```

pystrext.**random_password**(*length=8*)

Generate an "easy to memorize" random password

Parameters **length** (*int*) – Password length (default : 8)

Returns The random password

Return type str

Examples

```
>>> random_password()
>>> 'rixerutu'
```

pystrext.**remove_accents**(*s*)

Remove accents

Parameters **s** (*str*) – string to convert

Returns same string without any accent

Return type str

Examples

```
>>> remove_accents('Et voilà !')
>>> 'Et voila !'
```

pystrext.**slugify**(*value*)
Convert string to a slug

Parameters **value** (*str*) – the string to convert

Returns the slug

Return type str

Examples

```
>>> slugify("he'l'lO  Wörld !")
>>> 'hello-world'
```

pystrext.**truncate**(*str*, *maxsize*, *max_end_str='...'*, *end_str_inside=True*)
Truncate a string to a specified length and add a suffix (if truncated only)

Parameters

- **str** (*str*) – The string to truncate
- **maxsize** (*int*) – string maximum size before truncating and adding a suffix
- **max_end_str** (*str*) – suffix to add when string has been truncated ('...' by default)
- **end_str_inside** (*bool*) – Tells whether the suffix is inside the truncated string so the final string length is no more than maxsize or is outside the truncated string so the final string length is no more than maxsize + suffix's length

Returns The truncated string

Return type str

Examples

```
>>> truncate('hello world !',80)
>>> 'hello world !'
>>> truncate('hello world !',5)
>>> 'he...'
>>> truncate('hello world !',5,end_str_inside=False)
>>> 'hello...'
>>> truncate('hello world !',5,' <a href="?more">more...</a>',False)
>>> 'hello <a href="?more">more...</a>'
>>> truncate('hello world !',80,' <a href="?more">more...</a>',False)
>>> 'hello world !'
```

pystrext.**uncompress**(*s*)
Uncompress a gzipped string

You can uncompress both strings compressed with pystrext.compress() but also a .gz file that has been read with open() and read().

Parameters **s** (*str*) – The gzipped string to uncompress

Returns The uncompressed string

Return type str

Examples

see pystrext.compress()

pystrext.**version_lt**(*v1*, *v2*)

Compare 2 strings representing dotted version number

The goal is to test whether a version string is older than another one. A version string looks like this : "1.45.2.1" It can have as many dot you want, but the strings between dots cannot be more than 10 chars long. Strings comparaison are done "dot by dot" from left to right and stops as soon as the comparaison is not equal.

Parameters

- **v1** (*str*) – version1 string
- **v2** (*str*) – version2 string

Returns True is v1 older than v2

Return type bool

Examples

```
>>> version_lt('1.2.0','1.12.0')
>>> True
>>> version_lt('1.2.0','1.1.9')
>>> False
>>> version_lt('1.43c','1.43f')
>>> True
>>> version_lt('1.2.5.6.7.8','1.2.5.6.7.9')
>>> True
```

pystrext.**vjust**(*str*, *level=5*, *delim='.'*, *bitsize=6*, *fillchar=' '*)

Justify a string representing dotted version number

The goal is to justify/format a version string in a way it can be filtered by a SQL engine : Each substrings will get a fixed length so SQL string comparaison can be used.

Parameters

- **str** (*str*) – dotted version number
- **level** (*int*) – number max of version substrings
- **delim** (*str*) – separator (a dot by default)
- **bitsize** (*int*) – substrings max length
- **fillchar** (*str*) – the char used to fill the blanks

Returns the justified string

Return type str

Examples

```
>>> vjust('1.2')
>>> '     1.     2.      .      .      .      '
>>> vjust('1.12')
>>> '     1.    12.      .      .      .      '
>>> vjust('1.12',fillchar='0')
>>> '000001.000012.000000.000000.000000.000000'
```

CHAPTER 1

Indices and tables

- *genindex*
- *modindex*
- *search*

p

pystrext, 1

B

base62_decode() (in module pystrext), 2
base62_encode() (in module pystrext), 2

C

compress() (in module pystrext), 3

E

extract() (in module pystrext), 3
extracts() (in module pystrext), 3

F

file_unicode() (in module pystrext), 4
file_unicode_list() (in module pystrext), 4

G

get_col() (in module pystrext), 4

I

if_val() (in module pystrext), 5
is_email_valid() (in module pystrext), 5
is_ip_valid() (in module pystrext), 6

M

MB_GB() (in module pystrext), 1
MHz_GHz() (in module pystrext), 2

N

no_one_many() (in module pystrext), 6

P

plural() (in module pystrext), 7
pystrext (module), 1

R

random_password() (in module pystrext), 7
remove_accents() (in module pystrext), 7

S

slugify() (in module pystrext), 8

T

truncate() (in module pystrext), 8

U

uncompress() (in module pystrext), 8

V

version_lt() (in module pystrext), 9
vjust() (in module pystrext), 9