ct

(CoolTyping)

Version : 1.0
Date : 19/08/2006
Author : ¨Patrick Germain Placidoux

SUMMARY

1 INTRODUCTION

CoolTyping is an ultra lite serialization lib for Python/Jython.
It is handy but weak because it cannot handle any expression, see restriction.

Note: should work in Jython althougth it is experimental.

2 PREREQUISITES

Oses (every where python works)

Linux, AIX ®, Windows ®

Langages

Python >= 2.3 < 3
Jython >= 2.5 (experimental)

3 INSTALL

Download ct (cooltyping) at www.sourceforge.net

Unzip the file ct_#.##.zip in the directory of your choice.

4 QUICK VIEW

Spoking fast a CoolTyped expression is a python expression (bool, int, long, float, tuple, dict) converted to str but with no "", no""" and no trailing space arround structural character (like (,),[,], {,}:).

A CoolTyped expression is very serializable to a text file, a DB, a command line or an http stream, ...

The CoolTyping library comes with too functions :

unDress: converts a python expression to a CoolTyped expression
and
dress: converts a CoolTyped expression to a python expression.

Restriction : ct lib do not deserialize structural characters like : (,),[,],{,}:) as values.
For example if yoiu need to serialize :
s='got structural chars, in: my value'
>s=ct.unDress(d) **would work.**
>ct.dress(s) **wont work.**

4.1 THE UNDRESS FUNCTION

The unDress function starts from a python expresion and converts it to a CoolTyped expresion.

>>> import ct

4.1.1 bool

>>> **ct.unDress(**True**)**
'True'

Going back:
>>> type(ct.dress(ct.unDress(True)))
<type 'bool'>

>>> **ct.unDress(**False**)**
'False'

Going back:
>>> type(ct.dress(ct.unDress(False)))

<type 'bool'>

4.1.2 int

```
>>> ct.unDress(1234)
'1234'
```

Going back:

```
>>> type(ct.dress(ct.unDress(1234)))
<type 'int'>
```

```
>>> ct.unDress(123456789)
'123456789'
```

Going back:

```
>>> type(ct.dress(ct.unDress(123456789)))
<type 'int'>
```

4.1.3 long

```
>>> ct.unDress(12345678901234567890)
'12345678901234567890'
```

Going back:

```
>>> type(ct.dress(ct.unDress(12345678901234567890)))
<type 'long'>
```

4.1.4 float

```
>>> ct.unDress(1234.567)
'1234.567'
```

Going back:

```
>>> type(ct.dress(ct.unDress(1234.567)))
<type 'float'>
```

4.1.5 tuple/list

```
>>> ct.unDress((1, 2, 3, 4))
'(1,2,3,4)'
```

Going back:

>>> type(ct.dress(ct.unDress((1, 2, 3, 4))))
<type 'tuple'>

>>> **ct.unDress(**[1, 2, 3, 4]**)**
'[1,2,3,4]'

Going back:
>>> type(ct.dress(ct.unDress([1, 2, 3, 4])))
type 'list'>

Mixed tuple/list:

>>> l=(1, 'abc', 3, True, 4, 12345678901234567890, 1234.567)

>>> **ct.unDress(**l**)**
'(1,abc,3,True,4,12345678901234567890,1234.567)'

Going back:
>>> type(ct.dress(ct.unDress(l)
<type 'tuple'>

4.1.6 dict

>>> d={'kind':'cat', 'fly':False, 'color':'pink and blue', 'number':3}

>>> **ct.unDress(d)**
'{fly:False,color:pink and blue,kind:cat,number:3}'

Going back:
>>> type(ct.dress(ct.unDress(d)
<type 'dict'>

Mixed dict:

>>> d={'kind':'cat', 'fly':False, 'color':'pink and blue', 'val1':4, 'val2':12345678901234567890,
'val3':1234.567, 'number':3,
'title':'the Good the Bad and the   Ugly'}

>>> ct.unDress(d)
'{fly:False,kind:cat,
title:the Good the Bad and the   Ugly,color:pink and blue,number:3,
val3:1234.567,val2:12345678901234567890,val1:4}'

Going back:
>>> type(ct.dress(ct.unDress(d)
<type 'dict'>

4.1.7 imbricated mixed tuple/list and dict

Imbricated mixed tuple

```
>>>  t=(1, 'abc', 3, True, 4, 12345678901234567890, 1234.567,
    {'kind':'cat', 'fly':False, 'color':'pink and blue', 'number':3, 'title':'the Good the  Bad and the   Ugly'})

>>> ct.unDress(t)
'(1,abc,3,True,4,12345678901234567890,1234.567,
{fly:False,color:pink and blue,kind:cat,number:3,title:the Good the  Bad and the   Ugly})'
```

Going back:

```
>>> type(ct.dress(ct.unDress(l))
<type 'tuple'>
```

Imbricated mixed dict

```
>>> d={'kind':'cat', 'fly':False, 'color':'pink and blue', 'val1':4,
    'val2':12345678901234567890, 'val3':1234.567, 'number':3, 'title':'the Good the  Bad and the   Ugly',
    'other dict':{'kind':'cat', 'fly':False, 'color':'pink and blue', 'number':3},
    'a tuple':(1, 'abc', 3, True, 4, 12345678901234567890, 1234.567)
}
 >>> ct.unDress(d)
'{fly:False,kind:cat,title:the Good the  Bad and the   Ugly,color:pink and blue,number:3,\
a tuple:(1,abc,3,True,4,12345678901234567890,1234.567),\
other dict:{fly:False,color:pink and blue,kind:cat,number:3},
val3:1234.567,val2:12345678901234567890,val1:4}'
```

Going back:

```
>>> type(ct.dress(ct.unDress(d))
<type 'dict'>
```

4.2 THE DRESS FUNCTION

Does the same thing as the unDress function but on the opposite way, goes back to the original python type.

So for example, for this python dict:
mydict={'kind':'cat', 'fly':False, 'color':'pink and blue', 'number':3}

With this CoolTyped expression:

```
>>> ct.undress(myDict)
'{fly:False,color:pink and blue,kind:cat,number:3}'
```

This is Tue:
mydict==ct.dress(ct.undress(myDict))

And This is always Tue:
py_expression==ct.dress(ct.undress(py_expression))

5 TRADEMARKS: