Sequential Dictionary

Description:

The Python dictionary accesses its keys by hash values for performance reasons. As a result there is no specified order of its elements. Consequently there are no sort or slicing methods on ordinary dictionaries.

A little German-English dictionary:

```
dict={"Abend"     : "evening",
      "aber"      : "but",
      "Bild"      : "picture",
      "deshalb"   : "therefore",
      "Erkennung" : "recognition",
      "Flöte"     : "flute",
      "gewinnen"  : "gain" }

>>> dict
{'gewinnen': 'gain', 'deshalb': 'therefore', 'Abend': 'evening', 'aber':
'but', 'Bild': 'picture', 'Erkennung': 'recognition', 'Fl\366te': 'flute’ }
```

Although the dictionary was created with items in order there is no order and no reliable sequence in the dictionary’s representation.

The sequential dictionary incorporates a list holding the dictionary’s keys and the dictionary.
An empty sequential dictionary:

```
>>> seqdict.seqdict()
seqdict(
[],           # list holds the keys in sequence
{})           # dictionary holds the keys and values
```

A sequential dictionary of dict is created:

```
>>> s = seqdict.seqdict(dict)
>>> s
seqdict(
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te'],
{'gewinnen': 'gain', 'deshalb': 'therefore', 'Abend': 'evening', 'aber':
'but', 'Bild': 'picture', 'Erkennung': 'recognition', 'Fl\366te': 'flute'})
```

After sorting:

```
>>> s.sort(lambda x,y:cmp(string.lower(x),string.lower(y)))
>>> s
seqdict(
['Abend', 'aber', 'Bild', 'deshalb', 'Erkennung', 'Fl\366te', 'gewinnen'],
{'gewinnen': 'gain', 'deshalb': 'therefore', 'Abend': 'evening', 'aber':
'but', 'Bild': 'picture', 'Erkennung': 'recognition', 'Fl\366te': 'flute'})
```

Voila, here it is sorted again!

```
>>> for key,value in s.items():
...       print '%11s : %s'%(key,value)
...

      Abend : evening
       aber : but
       Bild : picture
    deshalb : therefore
  Erkennung : recognition
      Flöte : flute
   gewinnen : gain
```

Of course, keys and items also appear in sorted sequence:

```
>>> s.keys()
['Abend', 'aber', 'Bild', 'deshalb', 'Erkennung', 'Fl\366te', 'gewinnen']

>>> s.values()
['evening', 'but', 'picture', 'therefore', 'recognition', 'flute', 'gain']

>>> s.items()
[('Abend', 'evening'), ('aber', 'but'), ('Bild', 'picture'), ('deshalb',
'therefore'), ('Erkennung', 'recognition'), ('Fl\366te', 'flute'),
('gewinnen', 'gain')]
```

A short Tutorial:

slicing by index:

```
>>> s[2:5]
seqdict(
['Bild', 'deshalb', 'Erkennung'],
{'Bild': 'picture', 'Erkennung': 'recognition', 'deshalb': 'therefore'})
```

insert:

```
>>> d=seqdict.seqdict({"Fliege":"fly","Hand":"hand"})
>>> s.insert(3,d)
>>> s
seqdict(
['Abend', 'Bild', 'Erkennung', 'Hand', 'Fliege', 'Fl\366te', 'aber',
'deshalb', 'gewinnen'],
{'Hand': 'hand', 'Fliege': 'fly', 'gewinnen': 'gain', 'deshalb':
'therefore', 'Abend': 'evening', 'aber': 'but', 'Bild': 'picture',
'Erkennung': 'recognition', 'Fl\366te': 'flute'})
```

append:

```
>>> x=seqdict.seqdict(dict)
>>> x.keys()
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te']
>>> x.append('Zug','train')
>>> x.keys()
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te',
'Zug']
```

check:

Checks the integrity of the dictionary. This method is suitable for debugging reasons.

```
>>> x.check()
1   # ok
>>> x.list[0]='spam' # don't even think to modify internals!
>>> x.check()
0   # list and dict are of same length, but keys don't match
>>> del x.list[1]    # don't even think to modify internals!
>>> x.check()
-1  # list and dict are not of same length
```

clear:

```
>>> x.clear()
>>> x
seqdict(
[],
{})
```

copy:

```
>>> s1=s.copy()  # copy of the dictionary
>>> s2=s[:]      # same as s.copy
>>> s3=s         # copies the reference
>>> s==s1==s2==s3
1
>>> s is s1 or s is s2 or s1 is s2
0
>>> s is s3
1
```

count:

Counts the number of appearance of the value within the dictionary's values

```
>>> u=seqdict.seqdict({1:2, 2:3, 3:4, 4:5, 5:3})
>>> u.count(2)
1
>>> u.count(3)
2
```

del:

```
>>> x=seqdict.seqdict(dict)
>>> x.keys()
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te']

>>> del x[1:3]
>>> x.keys()
['gewinnen', 'aber', 'Bild', 'Erkennung', 'Fl\366te']

>>> del x['Erkennung']
>>> x.keys()
['gewinnen', 'aber', 'Bild', 'Fl\366te']

>>> del x
>>> x
Traceback (innermost last):
  File "<stdin>", line 1, in ?
NameError: x
```

extend:

update:

Unless the keys are member of s they will be appended, otherwise updated.

```
>>> s.keys()
['Abend', 'Bild', 'Erkennung', 'Hand', 'Fliege', 'Fl\366te', 'aber',
'deshalb', 'gewinnen']
>>> d["Melone"]="melon"
>>> d.keys()
['Hand','Fliege','Melone']
>>> s.extend(d) # same as s.update(d)
>>> s.keys()
['Abend', 'Bild', 'Erkennung', 'Hand', 'Fliege', 'Fl\366te', 'aber',
'deshalb', 'gewinnen','Melone']
```

filter:

```
>>> x=seqdict.seqdict(dict)
>>> x.keys()
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te']

>>> import re
>>> x.filter(re.compile('a',re.I).match) # all keys beginning with a or A
seqdict(
['Abend', 'aber'],
{'Abend': 'evening', 'aber': 'but'})
```

get:

```
>>> s.get('car')  # No return value, as 'car' is not a key of s
>>> s.get('car','key not available')
'key not available'
```

index:

```
>>> s.index("Erkennung")
2
```

items:

```
>>> s.items()
[('Abend', 'evening'), ('Bild', 'picture'), ('Erkennung', 'recognition'),
('Hand', 'hand'), ('Fliege', 'fly'), ('Fl\366te', 'flute'), ('aber', 'but'),
('deshalb', 'therefore'), ('gewinnen', 'gain')]
```

has_key:

```
>>> s.has_key('car')
0
```

keys():

```
>>> s.keys()
['Abend', 'Bild', 'Erkennung', 'Hand', 'Fliege', 'Fl\366te', 'aber',
'deshalb', 'gewinnen']
```

len:

```
>>> len(s)
9
```

map:
```
>>> swap = lambda (key,value):(value,key)
>>> x=seqdict.seqdict(dict)
>>> x.map(swap) # There is also a swap-method for seqdict
seqdict(
['gain', 'therefore', 'evening', 'but', 'picture', 'recognition', 'flute'],
{'but': 'aber', 'flute': 'Fl\366te', 'picture': 'Bild', 'gain': 'gewinnen',
'evening': 'Abend', 'therefore': 'deshalb', 'recognition': 'Erkennung'})
```

pop:

push:

You can use the dictionary like a stack
```
>>> s.pop()
{'gewinnen': 'gain'}
>>> s.push('Auto','car')
>>> s.keys()
['Abend', 'Bild', 'Erkennung', 'Hand', 'Fliege', 'Fl\366te', 'aber',
'deshalb', 'Auto']

>>> s.pop('Hand')
{'Hand': 'hand'}
>>> s.keys()
['Abend', 'Bild', 'Erkennung', 'Fliege', 'Fl\366te', 'aber', 'deshalb',
'Auto']
```

reduce:
```
>>> g=lambda x,(u,v):('%s%10s : %s\n'%(x,u,v))
>>> x=seqdict.seqdict(dict)

>>> x.reduce(g,'')
'  gewinnen : gain\012   deshalb : therefore\012     Abend : evening\012
aber : but\012      Bild : picture\012 Erkennung : recognition\012
Fl\366te : flute\012'

>>> print x.reduce(g,'')[:-1]
  gewinnen : gain
   deshalb : therefore
     Abend : evening
      aber : but
      Bild : picture
 Erkennung : recognition
     Flöte : flute
```

remove:

del:
```
>>> s.remove('Fliege')
>>> s.keys()
['Abend', 'Bild', 'Erkennung', 'Fl\366te', 'aber', 'deshalb', 'Auto']

>>> del s['Bild']
>>> s.keys()
['Abend', 'Erkennung', 'Fl\366te', 'aber', 'deshalb', 'Auto']

>>> del s[1:3]
>>> s.keys()
['Abend', 'aber', 'deshalb', 'Auto']
```

reverse:
```
>>> s.reverse()
>>> s.keys()
['Auto', 'deshalb', 'aber', 'Abend']
```

slice: # slicing by key

Slicing with the "['key1':'key2'] is not possible as python accept integer indexes only within this construct. Thus the slice method implements this as an ordinary function call. An extension is slicing by step.
```
>>> t=seqdict.seqdict(dict)
>>> t.keys()
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te']

>>> t.slice('Bild')
seqdict(
['Bild'],
{'Bild': 'picture'})

>>> t.slice('deshalb','Erkennung')
seqdict(
['deshalb', 'Abend', 'aber', 'Bild'],
{'Abend': 'evening', 'aber': 'but', 'Bild': 'picture', 'deshalb':
'therefore'})

>>> t.slice('deshalb','Erkennung',2)
seqdict(
['deshalb', 'aber'],
{'deshalb': 'therefore', 'aber': 'but'})
```

sort:
```
>>> s.sort()
>>> s.keys()
['Abend', 'Auto', 'aber', 'deshalb']

>>> s.sort(lambda x,y:cmp(string.lower(x),string.lower(y)))
>>> s.keys()
['Abend', 'aber', 'Auto', 'deshalb']
```

split:

The split-method returns a new sequential dictionary (seqdict). It expects a function which accepts a key and a value of the dictionary. If the function returns a key 'tkey' the dictionary-item is copied into the new seqdict (t) as t[tkey][key]=value. For every tkey a seqdict is hold as value.

Following example splits the seqdict u into one seqdict for every lowercase beginning letter. The lowercase beginning letter is the key and the splitted seqdict is the value for the new seqdict su.

```
>>> u=seqdict.seqdict(dict)
>>> u.sort(lambda x,y:cmp(string.lower(x),string.lower(y)))
>>> u.keys()
['Abend', 'aber', 'Bild', 'deshalb', 'Erkennung', 'Fl\366te', 'gewinnen']

>>> su = u.split(lambda x:string.lower(x)[0])
>>> su
seqdict(
['a', 'b', 'd', 'e', 'f', 'g'],
{'f': seqdict(
['Fl\366te'],
{'Fl\366te': 'flute'}), 'g': seqdict(
['gewinnen'],
{'gewinnen': 'gain'}), 'd': seqdict(
['deshalb'],
{'deshalb': 'therefore'}), 'e': seqdict(
['Erkennung'],
{'Erkennung': 'recognition'}), 'b': seqdict(
['Bild'],
{'Bild': 'picture'}), 'a': seqdict(
['Abend', 'aber'],
{'Abend': 'evening', 'aber': 'but'})})


>>> for tkey in su.keys():
...   print tkey , su[tkey].keys()
...
a ['Abend', 'aber']
b ['Bild']
d ['deshalb']
e ['Erkennung']
f ['Fl\366te']
g ['gewinnen']
```

swap:

This method can only be applied when all values of the dictionary are immutable. The Python dictionary cannot hold mutable keys! So swap doesn’t work if only one of the values has the type list or dictionary. Tuples and instances of classes are save as long as they don’t emulate lists or dictionaries.

```
>>> x=seqdict.seqdict(dict)
>>> x     # A small German - English dictionary
seqdict(
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te'],
{'gewinnen': 'gain', 'deshalb': 'therefore', 'Abend': 'evening', 'aber':
'but', 'Bild': 'picture', 'Erkennung': 'recognition', 'Fl\366te': 'flute'})

>>> x.swap() # swaps keys and items of x
>>> x     # A small English - German dictionary
seqdict(
['gain', 'therefore', 'evening', 'but', 'picture', 'recognition', 'flute'],
{'but': 'aber', 'flute': 'Fl\366te', 'picture': 'Bild', 'gain': 'gewinnen',
'evening': 'Abend', 'therefore': 'deshalb', 'recognition': 'Erkennung'})

>>> f=seqdict.seqdict({1:[1,2,3]}) # value is mutable list!
>>> f
seqdict(
[1],
{1: [1, 2, 3]})
>>> f.swap()
Traceback (innermost last):
  File "<stdin>", line 1, in ?
  File "ndict/py", line 150, in swap
  File "ndict/py", line 22, in __init__
TypeError: unhashable type
```

operator + :
```
>>> s.keys()
['Abend', 'aber', 'Auto', 'deshalb']

>>> t.keys()
['gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung', 'Fl\366te']

>>> (s+t).keys()
['Auto', 'gewinnen', 'deshalb', 'Abend', 'aber', 'Bild', 'Erkennung',
'Fl\366te']

>>> (t+s).keys()
['gewinnen', 'Bild', 'Erkennung', 'Fl\366te', 'Abend', 'aber', 'Auto',
'deshalb']
```

operator %:
```
>>> print """'Bild' is the German word for '%(Bild)s' and
... 'Flöte' is '%(Flöte)s' in English"""%t
'Bild' is the German word for 'picture' and
'Flöte' is 'flute' in English
```

Initialising the dictionary:

An empty seqdict:
```
>>> ndict.seqdict()
seqdict(
[],
{})
```

3 ways to initialize the seqdict:
```
>>> d1=seqdict.seqdict(dict.keys(),dict.values())
>>> d2=seqdict.seqdict(dict.keys(),dict)
>>> d3=seqdict.seqdict(dict)
>>> d1==d2==d3
1
```

Things which will not work:

doesn’t work:	use:
`for key in t:`	`for key in t.keys():`
`key in t:`	`key in t.keys():`