collections_backport

index
/Volumes/gz01-secure/gz01-git-boffin/f2010/courseworks/cw2/src/gz01/collections_backport.py

```
# This file is a backport of the PEP 372 OrderedDict class to be added
# to Python 2.7 and 3.1.
#
```

Classes

__builtin__.dict(__builtin__.object)
        OrderedDict(__builtin__.dict, _abcoll.MutableMapping)
_abcoll.MutableMapping(_abcoll.Mapping)
        OrderedDict(__builtin__.dict, _abcoll.MutableMapping)

class **OrderedDict**(__builtin__.dict, _abcoll.MutableMapping)

Method resolution order:
        OrderedDict
        __builtin__.dict
        _abcoll.MutableMapping
        _abcoll.Mapping
        _abcoll.Sized
        _abcoll.Iterable
        _abcoll.Container
        __builtin__.object

Methods defined here:

__delitem__(self, key)

__eq__(self, other)

__init__(self, *args, **kwds)

__iter__(self)

__reduce__(self)

__reversed__(self)

__setitem__(self, key, value)

clear(self)

copy(self)

dequeueitem(self)

items(self)

keys(self)

pop(self, key, default=<object object>)

popitem(self)

setdefault(self, key, default=None)

update(self, other=(), **kwds)

values(self)

Class methods defined here:

fromkeys(cls, iterable, value=None) from abc.ABCMeta

Data descriptors defined here:

__dict__
```
    dictionary for instance variables (if defined)
```

__weakref__
```
    list of weak references to the object (if defined)
```

Data and other attributes defined here:

__abstractmethods__ = frozenset([])

Methods inherited from __builtin__.dict:

__cmp__(...)
```
    x.__cmp__(y) <==> cmp(x,y)
```

__contains__(...)
```
    D.__contains__(k) -> True if D has a key k, else False
```

__ge__(...)
```
    x.__ge__(y) <==> x>=y
```

__getattribute__(...)
```
    x.__getattribute__('name') <==> x.name
```

__getitem__(...)
```
    x.__getitem__(y) <==> x[y]
```

__gt__(...)
```
    x.__gt__(y) <==> x>y
```

__le__(...)

```
    x.__le__(y) <==> x<=y
```

__len__(...)
```
    x.__len__() <==> len(x)
```

__lt__(...)
```
    x.__lt__(y) <==> x<y
```

__ne__(...)
```
    x.__ne__(y) <==> x!=y
```

__repr__(...)
```
    x.__repr__() <==> repr(x)
```

__sizeof__(...)
```
    D.__sizeof__() -> size of D in memory, in bytes
```

get(...)
```
    D.get(k[,d]) -> D[k] if k in D, else d.  d defaults to None
```

has_key(...)
```
    D.has_key(k) -> True if D has a key k, else False
```

iteritems(...)
```
    D.iteritems() -> an iterator over the (key, value) items of
```

iterkeys(...)
```
    D.iterkeys() -> an iterator over the keys of D
```

itervalues(...)
```
    D.itervalues() -> an iterator over the values of D
```

Data and other attributes inherited from __builtin__.dict:

__hash__ = None

__new__ = <built-in method __new__ of type object>
```
    T.__new__(S, ...) -> a new object with type S, a subtype of
```

Class methods inherited from _abcoll.Sized:

__subclasshook__(cls, C) from abc.ABCMeta

Data and other attributes inherited from _abcoll.Sized:

__metaclass__ = <class 'abc.ABCMeta'>
```
    Metaclass for defining Abstract Base Classes (ABCs).

    Use this metaclass to create an ABC.  An ABC can be subclas
    d
    directly, and then acts as a mix-in class.  You can also re
```

```
ster
unrelated concrete classes (even built-in classes) and unre
ted
ABCs as 'virtual subclasses' -- these and their descendants
ill
be considered subclasses of the registering ABC by the buil
in
issubclass() function, but the registering ABC won't show u
in
their MRO (Method Resolution Order) nor will method
implementations defined by the registering ABC be callable
ot
even via super()).
```