Diff It To Dig It

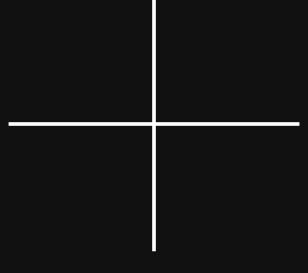
A dive into Python types

By Sep Ehr

zepworks.com

github.com/seperman/deepdiff

April 5 2016



Got Diff?

Deep Diff

pip install deepdiff

Our goal

- Diff nested objects
- Get the path and value of changes
- Ignore order on demand
- Work with Py2 and py3

Object categories in Py

1. Text Sequences
2. Numerics
3. Sets
5. Mappings
6. Other Iterables (List, Generator, Deque, Tuple, Custom Iterables)
7. User Defined Objects

Diff Text Sequences with Difflib

```
>>> import difflib
>>> t1="""
... Hello World!
... """.splitlines()
>>> t2="""
... Hello World!
... It is ice-cream time.
... """.splitlines()
>>> g = difflib.unified_diff(t1, t2, lineterm='')
>>> print('\n'.join(list(g)))
---
+++
@@ -1,2 +1,3 @@

 Hello World!
+It is ice-cream time.
```

Diff Sets, Frozensets

```
>>> t1 = {1,2,3}
>>> t2 = {3,4,5}
>>> items_added = t2 - t1
>>> items_removed = t1 - t2
>>> items_added
set([4, 5])
>>> items_removed
set([1, 2])
```

Diff Mapping

Dict, OrderedDict, Defaultdict

```
t1_keys= set(t1.keys())
t2_keys= set(t2.keys())

same_keys = t2_keys.intersection(t1_keys)

added = t2_keys - same_keys
removed = t1_keys - same_keys
```

And then recursively check same_keys values

Diff Iterables

Consider Order

```
>>> t1 = [1, 2, 3]
>>> t2 = [1, 2, 5, 6]
```

Diff Iterables

Consider Order

```
>>> t1 = [1, 2, 3]
>>> t2 = [1, 2, 5, 6]
>>>
>>> class NotFound(object):
...     "Fill value for zip_longest"
...     def __repr__(self):
...         return "NotFound"
...     def __str__(self):
...         return "NotFound Str"
...
>>> notfound = NotFound()
>>>
>>> list(zip_longest(t1, t2, fillvalue=notfound))
[(1, 1), (2, 2), (3, 5), (NotFound, 6)]
```

Diff Iterables

Consider Order

```
>>> for (x, y) in zip_longest(t1, t2, fillvalue=NotFound):
...     if x != y:
...         if y is NotFound:
...             removed.append(x)
...         elif x is NotFound:
...             added.append(y)
...         else:
...             modified.append("{} -> {}".format(x, y))
...
>>> print removed
[]
>>> print added
[6]
>>> print modified
['3 -> 5']
```

Diff Iterables

Ignore Order

Diff Iterables

Ignore Order

```
>>> t1=[1,2]
>>> t2=[1,3,4]
>>> t1set=set(t1)
>>> t2set=set(t2)
>>> t1set-t2set
{2}
>>> t2set-t1set
{3, 4}
```

Diff Iterables

Ignore Order

but ...

```
>>> t1=[1, 2, {3:3}]
>>> t2=[1]
>>> t1set = set(t1)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'dict'
```

A set object is an unordered collection of distinct hashable objects.

Mutable vs. Immutable

Mutable vs. Immutable

```
>>> a=[1,2]
>>> id(a)
400304246
>>> a.append(3)
>>> id(a)
400304246
>>> b=(1,2)
>>> id(b)
399960722
>>> b += (3,)
>>> id(b)
400670561
```

Hashable

Hashable

- __hash__ with output that does NOT change over object's lifetime.
- __eq__ for comparison

Unhashable vs. Mutable

Hashable that is Mutable

```
>>> class A:
...     aa=1
...
>>> hash(A)
2857987
>>> A.aa=2
>>> hash(A)
2857987
```

Diff Iterables

Ignore Order: approach 1: sort

```
>>> t1=[{1:1}, {3:3}, {4:4}]
>>> t2=[{3:3}, {1:1}, {4:4}]
>>> t1.sort()
>>> t1
[{1: 1}, {3: 3}, {4: 4}]
>>> t2.sort()
>>> t2
[{1: 1}, {3: 3}, {4: 4}]
>>> [(a, b) for a, b in zip(t1,t2) if a != b]
[]
```

Py2

Diff Iterables

Ignore Order: approach 1: sort

Py3

```
>>> t1=[{1:1}, {3:3}, {4:4}]
>>> t2=[{3:3}, {1:1}, {4:4}]
>>> t1.sort()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: dict() < dict()
```

Diff Iterables

Ignore Order: approach 1: sort

Sort key

Diff Iterables

Ignore Order: approach 1: sort

```
>>> students = [
        ('john', 'A', 15),
        ('jane', 'B', 12),
        ('dave', 'B', 10),
]
>>> sorted(students, key=lambda s: s[2])
[('dave', 'B', 10),
('jane', 'B', 12),
('john', 'A', 15)]
```

Diff Iterables

Ignore Order: approach 1: sort

What to use for sort key to order list of dictionaries?

Diff Iterables

Ignore Order: approach 1: sort

Sort key: hash of dictionary contents

```
>>> from json import dumps
>>> t1=[{1:1}, {3:3}, {4:4}]
>>> t2=[{3:3}, {1:1}, {4:4}]
>>> t1.sort(key=lambda x: hash(dumps(x)))
>>> t2.sort(key=lambda x: hash(dumps(x)))
>>> t1
[{1: 1}, {3: 3}, {4: 4}]
>>> t2
[{1: 1}, {3: 3}, {4: 4}]
>>> [(a, b) for a, b in zip(t1,t2) if a != b]
[]
```

Py2 & 3

Diff Iterables

Ignore Order: approach 1: sort

Iterables with different length

Diff Iterables

Ignore Order: approach 1: sort

iterables with different lengths

```
>>> import json
>>>
>>> t1=[10, {1:1}, {3:3}, {4:4}]
>>> t1.sort(key=lambda x: hash(json.dumps(x)))
>>>
>>> t2=[{3:3}, {1:1}, {4:4}]
>>> t2.sort(key=lambda x: hash(json.dumps(x)))
>>> t1
[{1: 1}, {3: 3}, {4: 4}, 10]
>>> t2
[{1: 1}, {3: 3}, {4: 4}]
```

Diff Iterables

Ignore Order: approach 1: sort

iterables with different lengths

```
>>> t1=[10, "a", {1:1}, {3:3}, {4:4}]
>>> t1.sort(key=lambda x: hash(dumps(x)))
>>> t1
['a', {1: 1}, {3: 3}, {4: 4}, 10]
>>> t2
[{1: 1}, {3: 3}, {4: 4}]
...

>>> modified
['a -> {1: 1}', '{1: 1} -> {3: 3}',
'{3: 3} -> {4: 4}']
```

Diff Iterables

Ignore Order: approach 2: hashtable

Put items in a dictionary of
{item_hash: item}

Diff Iterables

Ignore Order: approach 2: hashtable

```
>>> t1 = [10, "a", {1:1}, {3:3}, {4:4}]
>>> t2 = [{3:3}, {1:1}, {4:4}, "b"]
>>> def create_hashtable(t):
...     hashes = {}
...     for item in t:
...         try:
...             item_hash = hash(item)
...         except TypeError:
...             try:
...                 item_hash = hash(json.dumps(item))
...             except:
...                 pass # For presentation purposes
...             else:
...                 hashes[item_hash] = item
...         else:
...             hashes[item_hash] = item
...     return hashes
```

Diff Iterables

Ignore Order: approach 2: hashtable

```
>>> h1 = create_hashtable(t1)
>>> h2 = create_hashtable(t2)
>>>
>>> items_added = [h2[i] for i in h2 if i not in h1]
>>> items_removed = [h1[i] for i in h1 if i not in h2]
>>>
>>> items_added
['b']
>>> items_removed
['a', 10]
```

Diff Iterables

Ignore Order: approach 2: hashtable

What if the object is not json serializable?

What if json serializable version of 2 different objects are the same?

Diff Iterables

Ignore Order: approach 2: hashtable

Pickle

Diff Iterables

Ignore Order: approach 2: hashtable

```
>>> from pickle import dumps
>>> t = ({1: 1, 2: 4, 3: 6, 4: 8, 5: 10},
'Hello World', (1, 2, 3, 4, 5), [1, 2, 3, 4, 5])
>>> dumps(t)
"((dp0\nI1\nI1\nsI2\nI4\nsI3\nI6\nsI4\nI8\nsI5\
nI10\nsS'Hello World'\np1\n(I1\nI2\nI3\nI4\nI5\
ntp2\n(lp3\nI1\naI2\naI3\naI4\naI5\natp4\n."
>>> dumps(({1: 1, 2: 4, 3: 6, 4: 8, 5: 10},
           'Hello World', (1, 2, 3, 4, 5),
            [1, 2, 3, 4, 5]))
"((dp0\nI1\nI1\nsI2\nI4\nsI3\nI6\nsI4\nI8\nsI5\
nI10\nsS'Hello World'\np1\n(I1\nI2\nI3\nI4\nI5
\ntp2\n(lp3\nI1\naI2\naI3\naI4\naI5\natp4\n."
```

Diff Iterables

Ignore Order: approach 2: hashtable

What about cPIckle? It is faster than Pickle!

```
>>> from cPickle import dumps
>>> t = ({1: 1, 2: 4, 3: 6, 4: 8, 5: 10},
'Hello World', (1, 2, 3, 4, 5), [1, 2, 3, 4, 5])
>>> dumps(t)
"((dp1\nI1\nI1\nsI2\nI4\nsI3\nI6\nsI4\nI8\nsI5\n
I10\nsS'Hello World'\np2\n(I1\nI2\nI3\nI4\nI5\n
tp3\n(lp4\nI1\naI2\naI3\naI4\naI5\nat."
>>> dumps(({1: 1, 2: 4, 3: 6, 4: 8, 5: 10},
           'Hello World', (1, 2, 3, 4, 5),
            [1, 2, 3, 4, 5]))
"((dp1\nI1\nI1\nsI2\nI4\nsI3\nI6\nsI4\nI8\nsI5\n
I10\nsS'Hello World'\n(I1\nI2\nI3\nI4\nI5\nt(lp2
\nI1\naI2\naI3\naI4\naI5\natp3\n."
```

Diff Iterables

Ignore Order: approach 2: hashtable

cPickle includes if the object is referenced in the serialization!

Diff Iterables

Ignore Order: approach 2: hashtable

Note 2: Pickle does not include class attributes

```
class Foo:
    attr = 'not in pickle'
picklestring = pickle.dumps(Foo)
```

Diff Iterables

Ignore Order: approach 2: hashtable

Do we care?

No

Not in Deep Diff

Diff Iterables

What did we learn from diffing iterables?

- Difference of unhashable and mutable
- Sets can only contain hashable
- Create hash for dictionary
- Custom sorting with a key function
- Converting a squence into hashtable
- Pickling

Diff Custom Objects

__dict__

Diff Custom Objects

```
>>> class CL:
...     attr1 = 0
...     def __init__(self, thing):
...         self.thing = thing

>>> obj1 = CL(1)
>>> obj2 = CL(2)
>>> obj2.attr1 = 10
>>> obj1.__dict__
{'thing': 1}  # Notice that att1 is not here
>>> obj2.__dict__
{'attr1': 10, 'thing': 2}
```

Diff Custom Objects

__slots__

Diff Custom Objects

```
>>> class ClassA(object):
...     __slots__ = ['x', 'y']
...     def __init__(self, x, y):
...         self.x = x
...         self.y = y
...
>>> t1 = ClassA(1, 1)
>>> t2 = ClassA(1, 2)
>>>
>>> t1.new = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'ClassA' object has no attribute 'new'
```

Diff Custom Objects

```
>>> t1 = {i: getattr(t1, i) for i in t1.__slots__}
>>> t2 = {i: getattr(t2, i) for i in t2.__slots__}
>>> t1
{'x': 1, 'y': 1}
>>> t2
{'x': 1, 'y': 2}
```

Diff Custom Objects

Loops

```
>>> class LoopTest(object):
...     def __init__(self, a):
...         self.loop = self
...         self.a = a
...
>>> t1 = LoopTest(1)
>>> t2 = LoopTest(2)
>>> t1
<__main__.LoopTest object at 0x02B9A910>
>>> t1.__dict__
{'a': 1, 'loop': <__main__.LoopTest object at 0x02B9A910>}
```

Diff Custom Objects

Detect Loop with ID

A --> B --> C --> A

11 --> 23 --> 2 --> 11

Diff Custom Objects

Detect Loop with ID

```
def diff_common_children_of_dictionary(t1, t2,
                t_keys_intersect, parents_ids):

    for item_key in t_keys_intersect:

        t1_child = t1[item_key]
        t2_child = t2[item_key]

        item_id = id(t1_child)

        if parents_ids and item_id in parents_ids:
            print ("Warning, a loop is detected.")
            continue

        parents_added = set(parents_ids)
        parents_added.add(item_id)
        parents_added = frozenset(parents_added)

        diff(t1_child, t2_child, parents_ids=parents_added)
```

Diff Custom Objects

What did we learn about diffing custom objects

- __dict__ or __slots__
- Then diff as dictionary
- Objects can point to self or parent
- Detecting loops with IDs

Why Diff

- Debugging
- Testing, assertEqual with diff
- Emotional Stability

Deep Diff

Zepworks.com

https://github.com/seperman/deepdiff