dhp Documentation

Release 0.0.15

Jeff Hinrichs

June 06, 2016
1 Dirty Hungarian Phrasebook
  1.1 Phrasebook Examples ................................................................. 1
  1.2 Supports .................................................................................. 1
  1.3 Requirements ......................................................................... 2
  1.4 Installation ............................................................................. 2
  1.5 Download ............................................................................... 2
  1.6 Project Site ............................................................................ 2
  1.7 License .................................................................................. 2
  1.8 Documentation ....................................................................... 2
  1.9 Change Log ........................................................................... 2
  1.10 Contributing ......................................................................... 2
  1.11 Contributors ........................................................................ 2

2 Indices and tables

Python Module Index
Dirty Hungarian Phrasebook

dhp is a library of snippets, almost guaranteed to get you into trouble. I obtained it, from a vendor, on the corner, outside of PyCon. Actually, this is a growing repository of routines that I find helpful from time to time. I think you might too.

1.1 Phrasebook Examples

dhp.doq – Use ORM like expressions to query simple data sources.
dhp.search – Search related method and functions.
  - fuzzy_search - search like “Sublime Text”
dhp.structures – Unique structures that build on Python’s built-ins.
  - DictDot - Ever wish the dictionary supported dot access?
dhp.test – Helpful test helper routines.
  - tempfile_containing - generate a temporary file that contains indicated contents and returns the filename for use. When finished the tempfile is removed.
dhp.transforms
  - to_snake - transform a “camelCased” name into a pythonized version, “camel_cased”.
dhp.VI
  - iteritems - return the proper iteritems method for a dictionary based on the version of Python
dhp.xml
  - xml_to_dict - parse any ugly, but valid, xml to a python dictionary.
  - ppxml - format/reformat any ugly but valid xml, a pretty printer for xml

1.2 Supports

Tested on Python 2.7, 3.2, 3.3, 3.4
1.3 Requirements

None.

1.4 Installation

Make sure to get the latest version.
   
   pip install dhp

1.5 Download

   • https://pypi.python.org/pypi/dhp

1.6 Project Site

   • https://bitbucket.org/dundeemt/dhp

1.7 License

BSD

1.8 Documentation

   • http://dhp.rtfd.org/

1.9 Change Log

See Change Log

1.10 Contributing

See Contributing

1.11 Contributors

See Contributors

Contents:
1.11.1 Change Log

0.0.15 (dev)

- dhp.math.log_nfactorial - using Ramanujan’s approximation
- dhp.math.prob_unique - probability of no collision
- dhp.math.choose - nCk
- dhp.VI.to_unicode - helper for dealing with unicode in Py2
- dhp.VI.set_output_encoding - helper for dealing with unicode in Py2

0.0.14 (released 2016-02-05)

- added dhp.structures.ComparableMixin to aid in creating classes with rich comparisons.
- dropped support for Python3.2

0.0.13 (released 2015-12-20)

- integration with Appveyor CI for windows testing
- dhp.tempus package - humane time interval transforms
  - dhp.tempus.interval_from_delta - transform a datetime.timedelta to an interval.
  - dhp.tempus.delta_from_interval - transform an interval string to a datetime.timedelta object.
- dhp.cache package - a simple cache class

0.0.12 (released 2015-09-27)

- dhp.doq.DOQ - implemented range operator for lookups
- dhp.VI - StringIO - export StringIO from the proper package based on py2/py3
- dhp.search - Improved documentation.
- dhp.math - Improved documentation. Improved type tolerance. int/float/decimal
- test coverage improved on all submodule that were less than 100%

0.0.11 (released 2015-09-23)

- dhp.doq.DOQ - Duke is on the job to handle all your simple data source querying needs.

0.0.10 (released 2015-09-22)

- dhp.structures.DictDot - initial implementation

0.0.9 (released 2015-08-23)

- dhp.search.fuzzy_search - made case insensitive
0.0.8 (released 2015-08-19)

- refactor of test suite now that we are using `pip install -e`.

0.0.7 (released 2015-06-27)

- `dhp.search.fuzzy_search` and `.fuzzy_distance`

1.11.2 Contributing

Notes on how to contribute

Setting up a dev environment

These instructions assume you are developing in a virtualenv, you are, aren’t you?

1. Clone the code into your virtualenv
2. You should have the packages in `dev-requirements.txt` installed
   
   ```shell
   pip install -r requirements-dev.txt
   ```
   
3. Install dhp as editable
   
   ```shell
   pip install -e .
   ```
   
4. Tests should be passing locally
   
   ```shell
   py.test
   ```
   
5. Editing documentation - you will need to build the docs initially then use docwatch, to auto build the docs when saved as you edit.
   
   ```shell
   cd docs
   make html
   cd ..
   
   python docwatch.py
   ```

Pull Requests

- Code should be passing all tests locally, bonus points for passing drone.io
- New code should have new tests to go along with it.
- Code should be pep8 compliant
- Update documentation as necessary
- Update contributors.rst
- Make a pull request
1.11.3 Contributors

People who have contributed to the project

- Jeff Hinrichs <jefffh (at) dundeemt.com>

1.11.4 dhp.doq

**DOQ**

pronounced *Duke* allows you to query an list, iterable or generator of objects with a Django ORM like / Fluent interface. This is useful for exploratory programming and also it is just a nice, comfortable interface to query your data objects. DOQ supports lazy evaluations and nested objects.

**Example**

Say you had a csv file of employee records and you wanted to list the employees in the IT department. Well you could do the traditional thing or ...

```python
EmployeeRecord = namedtuple('EmployeeRecord', 'emp_id, name, dept, hired')

def csvtuples():
    '''csv named tuple emitter.''
    reader = csv.reader(TEST_FILE)
    for emp in map(EmployeeRecord._make, reader):
        yield emp

doq = DOQ(data_objects=csvtuples())
for emp in doq.filter(dept='IT'):
    print(emp)

# Now let's list everyone who is not in IT.
for emp in doq.exclude(dept='IT'):
    print(emp)

# ok, now let's sort the not IT employees by name
for emp in doq.exclude(dept='IT').order_by('name'):
    print(emp)
```

Yes, it is just that easy. You can chain .filter() and .exclude(). There is a .get method that raises DoesNotExist and MultipleObjectsReturned. All that ooohey gooey goodness of an full blown ORM but quick and easy and works without a lot of setup.

Let’s throw some remote json data at the Duke and see what happens.

```python
from dhp.structures import DictDot
from dhp.doq import DOQ
import requests

def json_ds(url):
    # fetch some json data, transform the returned dict to DictDot so
    # we can access attributes with dotted notation and then return
    # a DOQ with that data.
    data_objects = [DictDot(x) for x in requests.get(url).json()]
    return DOQ(data_objects=data_objects)
```

1.11. Contributors 5
users = json_ds('http://jsonplaceholder.typicode.com/users')
type(users)  # prints <class 'dhp.doq.DOQ'>
users.all().count  # prints 10
user = users.all()[0]
type(user)  # prints <class 'dhp.structures.DictDot'>
user.id  # prints 1
user.address.suite  # prints u'Apt. 556'
users.filter(address__suite__startswith='Apt.').count  # prints 3

One quick note before we head into the full documentation. DOQ is NOT a full blown Object Relation Manager. It does not create databases, nor know how to access them. If that is what you desire, then SQLAlchemy, Pony, PeeWeeDB or Django’s ORM is probably going to get you what you want.

If you are looking to slap some lipstick on a simple data source, well then, DOQ is your color. dhp.doq package for api specifics.

### 1.11.5 dhp.math

**fequal**

compare to floats to see if they are equal within a tolerance

```python
fequal(num1, num2, tolerance=0.000001)
```

return True if num1 is within tolerance of num2, else false

**Parameters**

- **num1** – float
- **num2** – float
- **tolerance** – float

**Return type** boolean

```python
from dhp.math import fequal

assert fequal(1.123456, 1.1234561)
```

**Use case:** comparing floats can be interesting due to internal representations

**is_even**

returns True if integer is even

```python
is_even(num)
```

**Parameters** num – int

**Return type** boolean

**is_odd**

returns True if integer is odd

```python
is_odd(num)
```

**Parameters** num – int
Return type  boolean

mean

returns the Arithmetic mean (a/k/a average) of a list of numbers

\texttt{mean}(\texttt{lst})

  Parameters  \texttt{list} – float | int | mixed
  Return type  float

\textbf{gmean}

returns the Geometric mean of a list of numbers

\texttt{gmean}(\texttt{lst})

  Parameters  \texttt{list} – float | int | mixed
  Return type  float

\textbf{hmean}

returns the Harmonic mean of a list of numbers

\texttt{hmean}(\texttt{lst})

  Parameters  \texttt{list} – float | int | mixed
  Return type  float

\textbf{1.11.6 dhp.search}

fuzzy\_search

given a list of strings(haystack) to search, return those elements, ranked, that fuzzily match the search term(needle).

\texttt{fuzzy\_search}(\texttt{needle}, \texttt{haystack})

  return a ranked list of elements from haystack that fuzzily match needle

  Parameters

  \begin{itemize}
  \item \texttt{needle} – what you are searching to find
  \item \texttt{haystack} – list of things to search
  \end{itemize}

  Return type  ranked sublist of haystack elements matching needle

\texttt{from dhp.search import fuzzy\_search}

\texttt{haystack = ['\texttt{.bob}', 'bob\texttt{.}', 'bo.b', 'fred']}

\texttt{assert fuzzy\_search(needle='bob', haystack) == ['\texttt{.bob}', '.bob', 'bo.b']}

\textbf{Use case:} create a “Sublime Text” like search experience

\textbf{1.11. Contributors}
1.11.7 dhp.structures

DictDot

DictDot subclasses Python’s built-in dict object and offers attribute access to the dictionary. A little code says alot:

```python
from dhp.structures import DictDot

my_dict = {'hovercraft': 'eels', 'speed': 42}
dicdot = DictDot(my_dict)
assert dicdot.hovercraft == 'eels'
assert dicdot.speed == 42

# ok, how about this?
dicdot = DictDot(hovercraft='eels', speed=42)
assert dicdot.hovercraft == 'eels'
assert dicdot.speed == 42

# or if your attacker has a pointed stick
dicdot = DictDot(my_dict, bunch='bananas')
assert dicdot.speed == 42
assert dicdot.bunch == 'bananas'

dicdot.new_value = 17
assert dicdot['new_value'] == 17
assert dicdot['hovercraft'] == 'eels'

# and now this ...
import json
assert json.dumps(dicdot) == '{"new_value": 17, "speed": 42, "hovercraft": "eels", "bunch": "bananas"}
```

All of the methods and functions of a normal Python dictionary are present and available for you to use.

**Use case:** Those times when you don’t want to type `['...']` but still want the goodness that is Python’s dictionary.

ComparableMixin

To implement comparisons and sorting for your classes just subclass the mixin and then implement the `_cmpkey()` method:

```python
from dhp.structures import ComparableMixin

class Comparable(ComparableMixin):
    def __init__(self, value):
        self.value = value

    def _cmpkey(self):
        return self.value
```

The magic methods `lt`, `le`, `eq`, `ge`, `gt` are all implemented and `NotImplemented` is returned when appropriate. Easier to use than `functools.total_ordering`. See https://wiki.python.org/moin/HowTo/Sorting for information on how the output of `_cmpkey` will sort.

1.11.8 dhp.tempus

This module includes tools to deal with time, dates, and intervals.
delta_from_interval

return a python datetime.timedelta that is represented by a human parseable Interval string. NwNdNhNmNs, i.e. 1w2d3h4m5s - One week, 2 days, 3 hours, 4 minutes and 5 seconds. Which can be quite useful if you want a human to schedule a delay or time based repeat interval.

`delta_from_interval(interval)`

return a python datetime.timedelta represented by interval.

Parameters

interval – str

Return type
datetime.timedelta

```python
from dhp.tempus import delta_from_interval
for k, val in iteritems(my_dict):
    do_something(k, val)
```

Use case: supporting python2 code that uses iteritems when targeting both 2 and 3.

PY_VER

is set to the major version of python currently running. Either 2 or 3 respectively.

StringIO

Imports the correct StringIO for the currently running version of Python.

```python
from dhp.VI import StringIO
```

1.11.9 dhp.test

tempfile_containing

generate a temporary file that contains indicated contents and returns the filename for use. When finished the tempfile is removed.

`tempfile_containing(contents[, suffix=''])`

Generate a temporary file with contents specified, clean up when done.

Parameters

- contents – what should be written to the temp file
- suffix – optional suffix of temp file, if required

Return type
filename as string

```python
from dhp.test import tempfile_containing
contents = 'I will not buy this record, it is scratched.'
with tempfile_containing(contents) as fname:
    do_something(fname)
```

Use case: When testing, some functions/modules expect one or more file names to process. This phrase creates a temporary file via Python’s `mkstemp`, writes the contents to it and closes the file so there is no contention with the module being tested on any platform. When the `with` statement goes out of scope, it cleans up the temporary file.
1.11.10 dhp.transforms

**to_snake**

given a “camelCase” string, transform it into a python-esque “camel_case”.

```
to_snake(name)
```

   return pythonized format of name, assumes name is some camelCase variant.

**Parameters**

- `name` -- camel cased name to transform

**Return type**

a pythonized string representation of the camel cased name.

```
import to_snake

assert to_snake('camelCase') == 'camel_case'
```

**Use case:** helpful when converting awful xml that uses camelCase to a python representation.

1.11.11 dhp.xml

**xml_to_dict**

There are a number of examples, on the intertubes, of doing this exact thing. However, many of them die on attributes. This has proven to be a robust routine and has dealt with all valid xml thrown at it.

```
xm_to_dict(xml)
```

convert valid xml to a python dictionary

**Parameters**

- `xml` -- string containing xml to be converted

**Return type**

dictionary

```
from dhp.xml import xml_to_dict

xml = '<vehicle type="Hovercraft"><filled/><cargo>eels</cargo></vehicle>'
xm_to_dict(xml)

{'vehicle': {'@type':'Hovercraft',
             'cargo':'eels',
             'filled': None}}
```

**Use case:** parse any ugly, but valid, xml to a python dictionary.

**ppxml**

Pretty print xml. reformat xml in a sane way. Often times xml from external/3rd party sources is delivered like a gigantic furball, making it hard for a human to parse/read, this utility function makes it a bit more palatable.

```
ppxml(xml)
```

format xml for easier viewing

**Parameters**

- `xml` -- string containing xml to be formatted

**Return type**

string
>>> from dhp.xml import ppxml
>>> xml = '<vehicle type="Hovercraft"><filled/><cargo>eels</cargo></vehicle>'
>>> ppxml(xml)
u'<?xml version="1.0" ?>
<vehicle type="Hovercraft">
 <filled/>
 <cargo>eels</cargo>
</vehicle>

>>> print ppxml(xml)
<?xml version="1.0" ?>
<vehicle type="Hovercraft">
 <filled/>
 <cargo>eels</cargo>
</vehicle>

1.11.12 dhp.VI

These are simple methods for dealing with Python 2/3 compatibility issues. They are focused on solving the problems of python 2/3 support in the dhp package. If you need more see six

iteritems

return the proper iteritems method for a dictionary based on the version of Python

\[\text{iteritems}(dct)\]
\[
\begin{tabular}{ll}
\textbf{Parameters} & dct – dictionary \\
\textbf{Return type} & iterable method \\
\end{tabular}
\]

from dhp.VI import iteritems

for k, val in iteritems(my_dict):
do_something(k, val)

Use case: supporting python2 code that uses iteritems when targeting both 2 and 3.

PY_VER

is set to the major version of python currently running. Either 2 or 3 respectively.

StringIO

Imports the correct StringIO for the currently running version of Python.

from dhp.VI import StringIO

1.11.13 Release Procedures

Notes on how to prepare, package and release a new version

1.11. Contributors
Pre-Release

1. You should have the packages in `requirements-dev.txt` installed and run setup.

   ```bash
   pip install -U -r requirements-dev.txt
   pip install -e .
   ```

2. Code should be checked in

   ```bash
   hg sum --remote
   ```

3. Tests should be passing locally

   ```bash
   py.test -v
   ```

4. drone and AppVeyor tests should be passing

   - https://drone.io/bitbucket.org/dundeemt/dhp/latest
   - https://ci.appveyor.com/project/dundeemt91221/dhp

5. Update the changelog


7. Run the release script in --dry-run mode and check that no errors or issues are outstanding. Specifically, check version information from bumpversion.

   ```bash
   ./release.sh --dry-run
   ```

Release

bumping the version, checking the build, committing tags

1. finalize the changelog (changelog.rst)

2. Run the release script

   ```bash
   ./release.sh
   ```

3. Push the commit

   ```bash
   hg push
   ```


6. Set the default docs to the new version – https://readthedocs.org/dashboard/dhp/versions/

7. upload to pypi

   ```bash
   twine upload dist/dhp-x.y.z.tar.gz
   ```

8. InsecurePlatformWarning - If you get this warning on python2.7+ you will need to install some additional modules

   ```bash
   pip install pyopenssl ndg-httpsclient pyasn1
   ```

9. Check PyPi for problems and make sure docs and package is correct – https://pypi.python.org/pypi/dhp
Profit

You and the rest of the world can enjoy

1.11.14 API Documentation

dhp package

Subpackages

dhp.VI package

Module contents  collection of routines to support python 2&3 code in this package

dhp.VI.iteritems\((dct)\)
    return the appropriate method

dhp.VI.py_ver()
    return the Major python version, 2 or 3

dhp.VI.set_output_encoding\((encoding=u'utf-8')\)
    If Python has punted on output encoding, give it a nudge. (def utf-8)
        Python knows the encoding needed when piping to the terminal and automatically sets it. However, when piping
        to another program (i.e. | less), it is None. Which means it defaults to ascii.

dhp.VI.to_unicode\((obj, encoding=u'utf-8')\)
    Convert to unicode if possible.

    Parameters
        •  \(obj\)\(\) – object to attempt conversion of
        •  \(encoding\)\(\) – default: utf-8

    Returns: (unicodelobj)

Exports  The following are exported by dhp.VI

StringIO  The proper version of StringIO from cStringIO or io package.

    from dhp.VI import StringIO

InstanceType  Python3 does not have old-style classes so types.InstanceType is undefined. If python3 then Instance-
    Type is set as a pointer to object – which is probably not what you want.

    from dhp.VI import InstanceType

dhp.cache package

1.11. Contributors
Module contents  Caching mechanisms that can be used as is or subclassed to enable specialization. view the source

exception dhp.cache.CacheKeyUnhashable
    Bases: exceptions.Exception
    Exception raised when the key given is not hashable.
    It is a replacement for the normal TypeError that would be raised in this situation to keep the exception logically separated.

    Parameters message (str) – Human readable string describing the exception.

message
    str – Human readable string describing the exception.

exception dhp.cache.CacheMiss
    Bases: exceptions.Exception
    Exception raised when requested key can not be found.
    Any SimpleCache operation that requires a key: .get, .put and .invalidate, and the key supplied can not be found while raise a CacheMiss. It is a replacement for the normal KeyError but is a logically separate exception.

    Parameters message (str) – Human readable string describing the exception.

message
    str – Human readable string describing the exception.

Examples

>>> from dhp.cache import SimpleCache, CacheMiss
>>> scache = SimpleCache()
>>> try:
>>>     value = scache.get(key='foo')
>>> except CacheMiss:
>>>     # do something meaningful like set value to a default
>>>     # or run your expensive operation.
>>>     pass

class dhp.cache.SimpleCache
    Bases: object
    A simplistic cache mechanism.
    SimpleCache is an in-memory dictionary based caching mechanism. It includes cache stats.

Examples

>>> from dhp.cache import SimpleCache
>>> scache = SimpleCache()
>>> scache.put(key='foo', value='bar')
>>> scache.get(key='foo')
'bar'

A key can be any Python hashable object, the restrictions on value are the same as for a Python dict – since that is the underlying mechanism.
get (key)
Return the cached entry indicated by key or raise CacheMiss

Parameters key (hashable object) – The cache key to retrieve.

Returns The object associated with key.

Raises

• CacheMiss – If the key can not be found.
• CacheKeyUnhashable – If the key is not hashable

invalidate (key)
Invalidate an element of the cache.

Parameters key (hashable object) – The key to remove from the cache.

Raises

• CacheMiss – If the key can not be found.
• CacheKeyUnhashable – If the key is not hashable.

put (key, value)
Cache the value with key.

Parameters

• key (hashable object) – The key to associate with the value. Keys are unique. The most recent .put call replaces the existing value, if any.

• value (object) – The value to cache.

Raises CacheKeyUnhashable – if the key is not hashable.

stats
(property) Returns a dictionary with cache stats.

Returns

a 4 element dictionary consisting of cache_size, cache_hits, cache_misses and cache_invalidated.

Return type (dict)

dhp.doq package

Module contents Data Object Query mapper.

pronounced Duke allows you to query an list, iterable or generator yielding objects with a Django ORM like / Fluent interface. This is useful for exploratory programming and also it is just a nice, comfortable interface to query your data objects.

Example

Say you had a csv file of employee records and you wanted to list the employees in the IT department. Well you could do the traditional thing or ...

Example:
def csvtuples():
    '''csv named tuple generator.''
    reader = csv.reader(TEST_FILE)
    for emp in map(EmployeeRecord._make, reader):
        yield emp

# Enter the Duke
doq = DOQ(data_objects=csvtuples())
for emp in doq.filter(dept='IT'):
    print(emp)

# Now let's list everyone who is not in IT.
for emp in doq.exclude(dept='IT'):
    print(emp)

# ok, now let's sort the not IT employees by name
for emp in doq.exclude(dept='IT').order_by('name'):
    print(emp)

Yes, it is just that easy. You can chain filter() and exclude(). There is a get() method that raises DoesNotExist() and MultipleObjectsReturned().

All that ooohey gooey query goodness of a traditional ORM but quick and easy and works without a lot of setup.

One quick note before we head into the full documentation. DOQ is NOT a full blown Object Relation Manager. It does not create databases, nor know how to access them. If that is what you desire, then SQLAlchemy, Pony, PeeWeeDB or Django’s ORM is probably going to get you what you want.

If you are looking to slap some lipstick on a simple data source, well then, DOQ is just your color.

class dhp.doq.DOQ(data_objects)
    Bases: object
    data object query mapper.

    all()
    Returns a cloned DOQ. Short hand for an empty filter but it reads more naturally than doq.filter().

    Parameters None –
    Returns A cloned DOQ object.

    Return type DOQ

    Example:
    for obj in doq.all():
        print(obj)

    count
    A property that returns the number of objects currently selected. Can also use len(doq).

    Returns The number of objects selected.

    Return type (int)

    Example:
```python
if doq.filter(name='Jeff').count == 1:
    do_something
result = doq.filter(emp_id=1)
assert doq.count == len(doq)
```

**exclude(** **look_ups** **)**

Returns a new DOQ containing objects that do not match the given lookup parameters.

**Parameters** *look_ups* – The lookup parameters should be in the format described in *Attribute Lookups* below. Multiple parameters are joined via AND in the underlying logic, and the whole thing is enclosed in a NOT.

**Returns** A cloned DOQ object with the specified exclude(s).

**Return type** *DOQ*

**Raises** *AttributeError* – If an attribute_name in the look_ups specified can not be found.

Example:

```python
doq.exclude(hired__gt=datetime.date(2005, 1, 3), name='Jeff')
```

**filter(** **look_ups** **)**

Returns a new DOQ containing objects that match the given lookup parameters.

**Parameters** *look_ups* – The lookup parameters should be in the format described in *Attribute Lookups* below. Multiple parameters are joined via AND in the underlying logic.

**Returns** A cloned DOQ object with the specified filter(s).

**Return type** *DOQ*

**Raises** *AttributeError* – If an attribute_name in the look_ups specified can not be found.

Example:

```python
doq.filter(name='Foo', hired__gte='2012-01-03')
```

**get(** **look_ups** **)**

Preform a get operation using 0 or more filter keyword arguments. A single object should be returned.

**Parameters** *look_ups* – The lookup parameters should be in the format described in *Attribute Lookups* below. Multiple parameters are joined via AND in the underlying logic.

**Returns** A single matching data_object from data_objects.

**Return type** *data_object*

**Raises** *AttributeError* – If an attribute_name in the look_ups specified can not be found.

Example:

```python
obj = doq.get(emp_id=1)
```

**Raises**

- *DoesNotExist* – If no matching object is found.
- *MultipleObjectsReturned* – If more than 1 object is found.

**static get_attr** (*obj, attrname*)

Retrieve a possibly nested attribute value.

**Parameters**
• **obj** (*data object*) – The data object to retrieve the value.

• **attrname** (*str*) – The attribute name/path to retrieve. A simple object access might be `name`, a nested object value might be `address__city`.

**Returns** The value of the indicated attribute.

**order_by** (*attribute_names*)

Return a new DOQ with the results ordered by the data_object’s attribute(s). The default order is ascending. Use a minus (-) sign in front of the attribute name to indicate descending order. Repeated .order_by calls are NOT additive, they replace any existing ordering.

**Parameters** **attribute_names** – 0 or more data_object attribute names. Listed from most significant order to least.

**Returns** A new DOQ object with the specified ordering.

**Return type** **DOQ**

**Example**:

```python
doq.all().order_by('emp_id')  # emp_id 1, 2, 3, ..., n
doq.all().order_by('-emp_id')  # emp_id n, n-1, n-2, ..., 1
doq.all().order_by('dept', 'emp_id')  # by dept, then by emp_id
```

to order randomly, use a `?`.

```python
doq.all().order_by('?')
```

**static order_by_key_fn** (*attrname*)

Override this method to supply a new key function for the order_by method.

The default function is:

```python
lambda obj: DOQ.get_attr(obj, attrname)
```

If you had an attribute “emp_id” that returned a number as a string ['2', '1', '3', '11']. It would be ordered by string conventions returning them in ['1', '11', '2', '3']. If you want them sorted like integers ['1', '2', '3', '11'], you would subclass DOQ and override the `order_by_key_fn` like this:

```python
class MyDOQ(DOQ):
    @staticmethod
    def order_by_key_fn(attrname):
        if attrname == 'emp_id':
            def key_fn(obj):
                # return attr as an integer
                return int(DOQ.get_attr(obj, attrname))
        else:
            def key_fn(obj):
                # return the standard function.
                return DOQ.get_attr(obj, attrname)
        return key_fn
```

```python
mydoq = MyDOQ(data_objects)
mydoq.all().order_by('emp_id')
```

**Parameters** **attrname** (*str*) – The attribute name be acted on by the order_by method.

**Returns**
A function that takes the attribute name as an argument and that also has access to the object be acted on.

**Return type** function

**Raises** AttributeError – If the attribute name specified can not be found.

**ordered**

True if an order is set, otherwise False.

**Returns** True if the order_by is set, otherwise False.

**Return type** bool

Example:

```python
results = doq.all()
assert results.ordered == False
results = results.order_by('name')
assert results.ordered == True
```

**exception** dhp.doq.DoesNotExist

Bases: exceptions.Exception

Raised when no object is found.

**exception** dhp.doq.MultipleObjectsReturned

Bases: exceptions.Exception

Raised when more than 1 object returned but should not be.

**Attribute Lookups**  Attribute lookups are similar to how you specify the meat of an SQL WHERE clause. They're specified as keyword arguments to the DOQ methods filter(), exclude() and get().

The format of look_ups is attribute_name__operation=value That is the name of the attribute to look at, a double under score(dunder) and then the lookup operator, an equals sign and then the value to compare against. The format was inspired by Django's ORM.

DOQ's inbuilt lookups are listed below.

As a convenience when no lookup type is provided (like in `doq.get(emp_id=14)`) the lookup type is assumed to be **exact**.

**exact**  Exact case-sensitive match.

```python
doq.get(emp_id__exact=4)
assert doq.get(name='Jeff') == doq.get(name__exact='Jeff')
```

**iexact**  Exact, case insensitive, match.

```python
doq.filter(name__iexact='jeff')  # would match, jEFF, Jeff, etc.
```

**lt**  Less Than.

```python
doq.filter(emp_id__lt=3)  # given [4, 3, 2, 1], would match [2, 1]
```
lte  Less Than or Equal to.

```python
doq.filter(emp_id__lte=3)  # given [4, 3, 2, 1], would match [3, 2, 1]
```

**gt**  Greater Than.

```python
doq.filter(emp_id__gt=3)  # given [4, 3, 2, 1], would match [4, ]
```

**gte**  Greater Than or Equal To.

```python
doq.filter(emp_id__gte=3)  # given [4, 3, 2, 1], would match [4, 3]
```

contains  If the value is in the attribute.

```python
doq.filter(name__contains='o')  # given ['Oscar', 'John', 'Jo'], would match ['John', 'Joe']
```

icontains  Case insensitive version of contains. See above.

```python
doq.filter(name__icontains='o')  # given ['Oscar', 'John', 'Jo'], would match ['Oscar', John', 'Joe']
```

startswith  If the attribute value startswith.

```python
doq.filter(name__startswith='O')  # given ['Oscar', 'John', 'Jo'], would match ['Oscar', ]
```

istartswith  Case insensitive version of startswith. See above.

```python
doq.filter(name__istartswith='o')  # given ['Oscar', 'John', 'Jo'], would match ['Oscar', ]
```

endswith  If the attribute value endswith.

```python
doq.filter(name__endswith='n')  # given ['Oscar', 'John', 'Jo'], would match ['John', ]
```

iendswith  Case insensitive version of endswith. See above.

```python
doq.filter(name__iendswith='N')  # given ['Oscar', 'John', 'Jo'], would match ['John', ]
```

in  If the attribute value is in the list supplied.

```python
doq.filter(emp_id__in=[1, 3])  # given [1, 2, 3, 4], would match [1, 3]
```

range  Is a short hand equivalent of a >= b and a <= c where a__range=(b, c) and b <= c

```python
doq.filter(emp_id__range=(2, 5))  # is equivalent of doq.filter(emp_id__gte=2, emp_id__lte=5)
```
Nested Objects  If you have a object that is composed of nested objects, you can access the values of the nested subobjects by using double underscores to list the path of the relationship. Say you had a list of objects with the following layout:

```python
user:
    id
    name
    address:
        street
        suite
        zipcode
    geo:
        lat
        lon
```

You would access the top-level attributes.

```python
doq.filter(id=7)
```

To access the suite information,

```python
doq.filter(address__suite='Apt. 201')
```

which would be an exact match on the attribute value. To use another operator with your lookup just specify it.

```python
doq.filter(address__suite__startswith='Apt.')
```

Ordering on a nested attribute is the same. To order by lat:

```python
doq.all().order_by('address__geo__lat')
```

Slicing DOQ (Limiting)  Slicing a DOQ is supported. Since we are not performing SQL the results of a slicing operation are immediate and return a list of data_objects.

```python
>>> type(doq.all()[2:4])
<type 'list'>
```

This also means that Negative indexing is supported.

```python
doq.all()[-1]
```

Would return the last data_object from the results.

dhp.math package

Module contents  handy math and statistics routines

Supported Number sets

- `{int}` = Set of integers
- `{float}` = Set of float
- `{decimal}` = Set of Decimal
- `{mixed-float}` = `{float} + `{int}`
- `{mixed-decimal}` = `{decimal} + `{int}`
Return Type Precedence

The type returned is based on the function, input type(s). The simplest meaningful type is returned.

- bool
- int
- float
- Decimal

**Exception** `dhp.math.MathError

Bases: exceptions.ValueError

general math error

**Exception** `dhp.math.UndefinedError

Bases: `dhp.math.MathError

When the calculation is undefined for the given input.

dhp.math.choose(n, k)

Return the number of combinations of n choose k (nCk).

\[
\binom{n}{k} = \frac{n!}{k!(n-k)!} \text{ when } k < n \text{ and } 0 \text{ for } k > n
\]

**Parameters**

- n (int) – the size of the pool of choices
- k (int) – the size of the sample

**Returns** the number of k-combinations of pool size n

**Return type** (int)

dhp.math.fequal(num1, num2, delta=1e-06)

Compare equivalency of two numbers to a given delta.

Both num1 and num2 must be from the same set of {mixed-float} OR {mixed-decimal}.

\[num1 \equiv num2 \iff |num1 - num2| < \delta\]

**Parameters**

- num1 ({mixed-float} | {mixed-decimal}) – The first number to compare.
- num2 (num1) – The second number to compare.
- delta (float) – The amount of difference allowed for equivalence. (default: 0.000001)

**Returns**

True if the absolute difference between num1 and num2 is less than delta, else False.

**Return type** (bool)

**Raises** TypeError – If testing a float and a Decimal.

dhp.math.gmean(nums)

Return the geometric mean of the list of numbers.

\[G = (x_1 \times x_2 \times \ldots \times x_N)^{1/N} = \left(\prod_{i=1}^{N} x_i\right)^{1/N}\]

**Parameters** nums (list) – list of numbers ({mixed-float} | {mixed-decimal})

**Returns** Geometric Mean of the list.
Return type (float|decimal)

Raises
- (UndefinedError) – If nums is empty. $N = 0$
- (TypeError) – If nums contains both float and Decimal numbers.

```python
dhp.math.hmean(nums)
```
Return the harmonic mean of a list of numbers.

$$H = \frac{N}{\frac{1}{x_1} + \frac{1}{x_2} + \ldots + \frac{1}{x_N}} = \frac{N}{\sum_{i=1}^{N} \frac{1}{x_i}}$$

Parameters `nums (list)` – list of numbers ([mixed-float]|{mixed-decimal})

Returns Harmonic Mean of the list.

Return type (float|decimal)

Raises (UndefinedError) – If the list is empty. $N = 0$

```python
dhp.math.is_even(num)
```
Return True if num is even, else False.

An integer is even if it is ‘evenly divisible’ by two.

Even = \{2k : k ∈ Z\}

Parameters `num (int)` – The num to check.

Returns True if num is even, else False.

Return type (bool)

Raises (MathError) – If num is not an integer.

```python
dhp.math.is_odd(num)
```
Return True if num is odd, else False.

An integer is odd if it is not even.

Odd = \{2k + 1 : k ∈ Z\}

A number expressed in the binary is odd if its last digit is 1 and even if its last digit is 0.

Parameters `num (int)` – The num to check.

Returns True if num is odd, else False.

Return type (bool)

Raises (MathError) – If num is not an integer.

```python
dhp.math.log_nfactorial(n)
```
calculate approximation of log n! using Srinivasa Ramanujan’s approximation of log n!

$$\log n! \approx n \log n - n + \frac{\log(n(1 + 4n(1 + 2n)))}{6} + \frac{\log(\pi)}{2}$$

Parameters `n (int)` – a very large integer

Returns log n!

Return type (float)

```python
dhp.math.mean(nums)
```
Return the arithmetic mean of the list of numbers
\[ X = \frac{x_1 + x_2 + ... + x_N}{N} = \frac{\sum_{i=1}^{N} x_i}{N} \]

**Parameters** `nums(list)` – list of numbers \{(mixed-float|mixed-Decimal)\}

**Returns** Arithmetic Mean of the list.

**Return type** (float|decimal)

**Raises**

- `(UndefinedError)` – If nums is empty, \(N = 0\)
- `(TypeError)` – If nums contains both float and Decimal numbers.

```python
dhp.math.median(nums)
```

Return the median value from the list.

Given: \(a < b < c < d\) The median of the list \([a, b, c]\) is \(b\), and, the median of the list \([a, b, c, d]\) is the mean of \(b\) and \(c\); i.e. \(\frac{b + c}{2}\)

**Parameters** `nums(list)` – list of numbers \{(mixed-float|mixed-decimal)\}

**Returns** The median of the list of numbers.

**Return type** (int|float|decimal)

```python
dhp.math.mode(lst)
```

Return the mode (most common element value) from the list.

**Parameters** `lst(list)` – list of hashable objects to search for the mode.

**Returns** The most common value in lst.

**Return type** (list element)

**Raises**

- `(UndefinedError)` – If lst is empty.
- `(MathError)` – If lst is multi-modal.

```python
dhp.math.prob_unique(nvals, ssize)
```

return the Probability of Uniqueness given:

where \(N\) is \(nvals\) and \(r\) is \(ssize\):

\[ p = \frac{N!}{N^r(N - r)!} \]

**Parameters**

- `nvals(int)` – number of unique points (i.e. birthdays=365)
- `ssize(int)` – size of sample (i.e. people)

**Returns** probability that all samples are unique

**Return type** (float)

```python
dhp.math.pstddev(lst)
```

return the population standard deviation of the elements in the list

```python
dhp.math.pvariance(lst)
```

return the population variance for the list of numbers

```python
dhp.math.sstddev(lst)
```

return the sample standard deviation of the elements in the list
dhp.math.\texttt{svariance}(lst)  
return the sample population variance for the list of numbers

\texttt{dhp.math.ttest_independent}(lst1, lst2)  
calc the ttest for two independent samples

\texttt{dhp.search package}

\texttt{Module contents}  
search type utilities

\texttt{dhp.search.fuzzy_distance}(needle, straw)  
calculate distance between needle and a straw from the haystack.

\texttt{Parameters}

- \texttt{needle(str)} – The thing to match
- \texttt{straw(str)} – The thing to match against

\texttt{Returns}

A distance of 0 indicates a search failure on one or more chars in needle. The lower the distance the closer the match, matching earlier and closer together results in a shorter distance.

\texttt{Return type} (int)

\texttt{dhp.search.fuzzy_search}(needle, haystack)  
Return a list of elements from haystack, ranked by distance from needle.

\texttt{Parameters}

- \texttt{needle(str)} – The thing to match.
- \texttt{haystack(list)} – A list of strings to match against.

\texttt{Returns}

Of strings, ranked by distance, that fuzzy match needle to one degree or another.

\texttt{Return type} (list)

Example:

```python
corpus = ['django_migrations.py',  
          'django_admin_log.py',  
          'main_generator.py',  
          'migrations.py',  
          'api_user.doc',  
          'user_group.doc',  
          'accounts.txt',  
          ]
assert fuzzy_search('mig', corpus) == ['migrations.py',  
                                       'django_migrations.py',  
                                       'main_generator.py',  
                                       'django_admin_log.py']
```

\texttt{dhp.structures package}

\texttt{Module contents}  
dhp data structures
class dhp.structures.ComparableMixin
    Bases: object

    Mixin to give proper comparisons.

    Example:

    ```python
    class Comparable(ComparableMixin):
        def __init__(self, value):
            self.value = value
        def _cmpkey(self):
            return self.value
    ```

    Returns NotImplemented if the object being compared doesn’t support the comparison.

    Raises Not ImplementedError if you have not overridden the _cmpkey method.

    Code is from Lennart Regebro https://regebro.wordpress.com/2010/12/13/python-implementing-rich-comparison-the-correct-way/

class dhp.structures.DictDot(*args, **kwargs)
    Bases: dict

    A subclass of Python’s dictionary that provides dot-style access.

    Nested dictionaries are recursively converted to DictDot. There are a number of similar libraries on PyPI. However, I feel this one does just enough to make things work as expected without trying to do too much.

    Example:

    ```python
dicdot = DictDot({
    'foo': {
        'bar': {
            'baz': 'hovercraft',
            'x': 'eels'
        }
    }
}
assert dicdot.foo.bar.baz == 'hovercraft'
assert dicdot['foo'].bar.x == 'eels'
assert dicdot['foo']['bar'].baz == 'hovercraft'
dicdot.bouncy = 'bouncy'
assert dicdot['bouncy'] == 'bouncy'
``` 

DictDot raises an AttributeError when you try to read a non-existing attribute while also allowing you to create new key/value pairs using dot notation.

DictDot also supports keyword arguments on instantiation and is built to be subclass’able.

dhp.tempus package

Module contents dhp.tempus - date, time and interval related routines.

exception dhp.tempus.IntervalError
    Bases: exceptions.ValueError

    time interval error has occurred

dhp.tempus.delta_from_interval(interval)
    convert an interval 'NwNdNhNmNs' to a timedelta.
Parameters **interval** (*str*) – a string in the form [Mw][Nd][Od][Ph][Qm][Rs]

**Returns** a timedelta object of the same interval length.

**Return type** `datetime.timedelta`

dhp.tempus.**interval_from_delta** (*tdelta*)
convert a timedelta to an interval.

Parameters **tdelta** (*datetime.timedelta*) – The timedelta object to convert to an interval.

**Returns** **interval** – The interval representation of the timedelta object.

**Return type** `str`

dhp.test package

**Module contents** routines and snippets generally useful for testing

dhp.test.**tempfile_containing** (*args, **kwds*)
create a temporary file, with optional suffix and return the filename, cleanup when finished

dhp.transforms package

**Module contents** dhp transforms library

dhp.transforms.**chunk_r** (*buf, chunk_size*)
starting from the right most character, split into groups of chunk_size.

```
>>> chunk_r('abcdefg', 3)
['a', 'bcd', 'efg']
```

Parameters

- **buf** (*str*) – a string or object that can be stringified
- **chunk_size** (*int*) – the maximum size of the groups

**Returns** chunk sized strings

**Return type** `list`

dhp.transforms.**filter_dict** (*dictionary, keys*)
filter a dictionary so it contains only specified keys.

```
>>> old = {'foo': 0, 'bar': 1, 'baz': 2}
>>> filter_dict(old, ['bar', 'baz', 'missing'])
{'bar': 1, 'baz': 2}
```

Parameters

- **dictionary** (*dict*) – the dictionary to filter out unwanted keys/vals
- **keys** (*list*) – the list of keys to return in the resultant dictionary

**Returns** the resultant dictionary with only the specified keys

**Return type** `dict`

1.11. Contributors 27
dhp.transforms.int2word(ivalue)
return the integer value as word(s)

```
>>> int2word(12)
Twelve
>>> int2word(237)
Two Hundred Thirty Seven
```

Parameters  
```
ivalue (int) – the integer to be converted
```

Returns  
```
The spelled out value of ivalue
```

Return type  str

---

dhp.transforms.to_snake(buf)
pythonize the camelCased name contained in buf

```
>>> to_snake('camelCase')
camel_case
```

Parameters  
```
buf (str) – the camelCased name to transform
```

Returns  
```
the pythonized version of the camelCased name
```

Return type  str

---

dhp.xml package

Module contents  routines generally helpful for dealing with icky xml

exception dhp.xml.MissingRequiredException
Bases: exceptions.Exception
A required data element is not present

exception dhp.xml.NoRootException
Bases: exceptions.Exception
raised when dictionary to build xml document from does not have a single ‘root’ node element. i.e. {’root’:{...}}

---

dhp.xml.dict_to_xml(dictionary, attrs=None)
return a string representation of an xml document of the dictionary. ( with optional attributes for the root node.)

```
>>> the_dict = {’root’: {’foo’: ’1’}}
>>> dict_to_xml(the_dict)
<?xml version="1.0" ?>
<root xml:lang="en-US"><foo>1</foo></root>
```

Since the function returns a full xml document, the dictionary has to closely approximate the structure of the xml document. So the top level of the dictionary must be a string key with a dictionary for a value.

Also, ALL leaf node element values must be strings.

Parameters
```
• dictionary (dict) – an approximation of the xml document desired as a Python dictionary.
• args (dict) – a dictionary containing attributes to assign to the root level node.
```
Raises  *NoRootException*  – When there top level dictionary has more than 1 key/value or if the value of the top level key is not a dictionary.

Returns a string representing an xml document based on the inputs.

Return type  *str*

```python
dhp.xml.obj_to_xml(obj)
    serialize an object’s non-private/non-hidden data attributes

dhp.xml.ppxml(xmls, indent=\'u' \')
    pretty print xml, stripping an existing formatting
```

```python
>>> buf = \'<root><foo>1</foo></root>'
>>> ppxml(buf)
<?xml version="1.0" ?>
<root>
  <foo>1</foo>
</root>
```

**Parameters**

- `xmls (str)` – an xml string, either a fragment or document
- `indent (str)` – a string containing the white space to use for indentation.

**Returns**

A transform of that string with new lines and standardized indentation. Default is 2 spaces

Return type  *str*

```python
dhp.xml.xml_to_dict(xml_buf)
    convert xml string to a dictionary, not always pretty, but reliable
```

Module contents

dhp top level

1.11. Contributors
Indices and tables

• genindex
• modindex
• search
Python Module Index

d
- dhp, 29
- dhp.cache, 14
- dhp.doq, 15
- dhp.math, 21
- dhp.search, 25
- dhp.structures, 25
- dhp.tempus, 26
- dhp.test, 27
- dhp.transforms, 27
- dhp.VI, 13
- dhp.xml, 28
Index

A

all() (dhp.doq.DOQ method), 16

C

CacheKeyUnhashable, 14
CacheMiss, 14
choose() (in module dhp.math), 22
chunk_r() (in module dhp.transforms), 27
ComparableMixin (class in dhp.structures), 25
count (dhp.doq.DOQ attribute), 16

D
delta_from_interval() (built-in function), 9
delta_from_interval() (in module dhp.tempus), 26
dhp (module), 29
dhp.cache (module), 14
dhp.doq (module), 15
dhp.math (module), 21
dhp.search (module), 25
dhp.structures (module), 25
dhp.tempus (module), 26
dhp.test (module), 27
dhp.transforms (module), 27
dhp.VI (module), 13
dhp.xml (module), 28
dict_to_xml() (in module dhp.xml), 28
DictDot (class in dhp.structures), 26
DoesNotExist, 19
DOQ (class in dhp.doq), 16

E

exclude() (dhp.doq.DOQ method), 17

F

fequal() (built-in function), 6
fequal() (in module dhp.math), 22
filter() (dhp.doq.DOQ method), 17
filter_dict() (in module dhp.transforms), 27
fuzzy_distance() (in module dhp.search), 25
fuzzy_search() (built-in function), 7
fuzzy_search() (in module dhp.search), 25

G

g() (dhp.cache.SimpleCache method), 14
g() (dhp.doq.DOQ method), 17
g_attr() (dhp.doq.DOQ static method), 17
gmean() (built-in function), 7
gmean() (in module dhp.math), 22

H

hmean() (built-in function), 7
hmean() (in module dhp.math), 23

I

int2word() (in module dhp.transforms), 27
interval_from_delta() (in module dhp.tempus), 27
IntervalError, 26
invalidate() (dhp.cache.SimpleCache method), 15
is_even() (built-in function), 6
is_even() (in module dhp.math), 23
is_odd() (built-in function), 6
is_odd() (in module dhp.math), 23
iteritems() (built-in function), 11
iteritems() (in module dhp.VI), 13

L

log_nfactorial() (in module dhp.math), 23

M

MathError, 22
mean() (built-in function), 7
mean() (in module dhp.math), 23
median() (in module dhp.math), 24
message (dhp.cache.CacheKeyUnhashable attribute), 14
message (dhp.cache.CacheMiss attribute), 14
MissingRequiredException, 28
mode() (in module dhp.math), 24
MultipleObjectsReturned, 19

N

NoRootException, 28
O
obj_to_xml() (in module dhp.xml), 29
order_by() (dhp.doq.DOQ method), 18
order_by_key_fn() (dhp.doq.DOQ static method), 18
ordered (dhp.doq.DOQ attribute), 19

P
ppxml() (built-in function), 10
ppxml() (in module dhp.xml), 29
prob_unique() (in module dhp.math), 24
pstderrdev() (in module dhp.math), 24
put() (dhp.cache.SimpleCache method), 15
pvariance() (in module dhp.math), 24
py_ver() (in module dhp.VI), 13

S
set_output_encoding() (in module dhp.VI), 13
SimpleCache (class in dhp.cache), 14
sstdev() (in module dhp.math), 24
stats (dhp.cache.SimpleCache attribute), 15
svariance() (in module dhp.math), 24

T
tempfile_containing() (built-in function), 9
tempfile_containing() (in module dhp.test), 27
to_snake() (built-in function), 10
to_snake() (in module dhp.transforms), 28
to_unicode() (in module dhp.VI), 13
ttest_independent() (in module dhp.math), 25

U
UndefinedError, 22

X
xml_to_dict() (built-in function), 10
xml_to_dict() (in module dhp.xml), 29