wheezy.core documentation

Release latest

Andriy Kornatskyy

Jun 22, 2021

Contents

CHAPTER 1

Introduction

wheezy.core is a python package written in pure Python code. It is a lightweight core library.

It is optimized for performance, well tested and documented.

Resources:

- source code, examples and issues tracker are available on github
- documentation

CHAPTER 2

Contents

2.1 Getting Started

2.1.1 Install

wheezy.core requires python version 3.6+. It is operating system independent. You can install it from the pypi site:

```
$ pip install wheezy.core
```

2.2 Examples

Before we proceed let's setup a virtualenv environment, activate it and install:

```
$ pip install wheezy.core
```

2.3 User Guide

wheezy.core comes with extensions to the following features:

- benchmark
- collections
- config
- datetime
- db
- descriptors
- feistel

- gzip
- i18n
- introspection
- json
- luhn
- mail
- pooling
- retry
- url
- uuid

2.3.1 benchmark

The *benchmark* module contains a single class *Benchmark* that measures execution time of your code.

Here is an example:

```
class BenchmarkTestCase(PublicTestCase):

    def runTest(self):
        p = Benchmark((
            self.test_home,
            self.test_about
            ), 1000)
        p.report('public', baselines={
                'test_home': 1.0,
                'test_about': 0.926
        })
```

Sample output:

```
public: 2 x 1000
baseline throughput change target
  100.0%     839rps  +0.0% test_home
   96.2%     807rps  +3.9% test_about
```

Each of the test cases has been run 1000 times. The shows productivity relative to the first test case (which serves as a baseline for other tests), throughput in requests per second, change from `baselines` argument passed to `report` method and target being benchmarked.

Reports are being printed as soon as results are available.

2.3.2 collections

The *collections* module contains types and functions that define various collections, iterators and algorithms.

Classes:

- *ItemAdapter* - adapts `defaultdict(list).__getitem__` accessor to return item at `index` from the list. If `key` is not found return None.
- *attrdict* - a dictionary with attribute-style access. Maps attribute access to dictionary.

- *defaultattrdict* - a dictionary with attribute-style access. Maps attribute access to dictionary. Extends `defaultdict`.

Functions:

- *first_item_adapter()* - adapts `defaultdict(list).__getitem__` accessor to return the first item from the list.
- *last_item_adapter()* - adapts `defaultdict(list).__getitem__` accessor to return the last item from the list.
- *distinct()* - returns generator for unique items in `seq` with preserved order.
- *gzip_iterator()* - iterates over `items` and returns generator of gzipped items. Argument `compress_level` sets compression level.

2.3.3 config

Config - promotes `options` dict to attributes. If an option can not be found in `options`, tries to get it from `master`. `master` must have a requested option otherwise an error is raised:

```
m = {'DEBUG': False}
c = Config(options={'DEBUG': True}, master=m)
assert True == c.DEBUG
```

`master` - object with dictionary or attribute style of access.

2.3.4 datetime

Represents an instant in time, typically expressed as a date and time of day.

Classes:

- *utc* - defines UTC timezone. There are two instances of the class: GMT and UTC.

Functions:

- *format_http_datetime()* - formats datetime to a string following rfc1123 pattern:

```
>>> from wheezy.core.datetime import UTC
>>> now = datetime(2011, 9, 19, 10, 45, 30, 0, UTC)
>>> format_http_datetime(now)
'Mon, 19 Sep 2011 10:45:30 GMT'
```

- *parse_http_datetime()* - parses a string in rfc1123 format to `datetime`:

```
>>> parse_http_datetime('Mon, 19 Sep 2011 10:45:30 GMT')
datetime.datetime(2011, 9, 19, 10, 45, 30)
```

- *total_seconds()* - returns a total number of seconds for the given time delta (`datetime.timedelta` or `int`):

```
>>> total_seconds(timedelta(hours=2))
7200
```

2.3.5 db

TODO

2.3.6 descriptors

TODO

2.3.7 feistel

TODO

2.3.8 gzip

One-shot compression and decompression is provided through the *compress()* and *decompress()* functions.

2.3.9 httpclient

HTTPClient sends HTTP requests to a server in order to accomplish an application specific use cases, e.g. remote web server API, etc:

```
>>> from wheezy.core.httpclient import HTTPClient
>>> c = HTTPClient('http://buildbot.buildbot.net/json/')
>>> c.get('project')
200
>>> project = c.json
>>>> str(project.title)
Buildbot
```

Here is another example that demonstarates etag handling (the second time we request events the server responds with HTTP status code 304, not modified):

```
>>> c = HTTPClient('https://api.github.com/repos/python/cpython/')
>>> c.get('events')
200
>>> c.headers['content-encoding']
['gzip']
>>> c.get('events')
304
```

Supports: HTTP(S) GET/HEAD/POST verbs, follows redirects, handles cookies and etags between requests, gzip content encoding.

2.3.10 i18n

Internationalisation is the process of adapting an application to different languages, regional differences and technical requirements. Internationalization of software is designing an application so that it can be adapted to various languages and regions without engineering changes.

`gettext` is an internationalization and localization (i18n) system commonly used for writing multilingual programs on Unix-like operating systems.

TranslationsManager - manages several languages and translation domains. You can use method *load()* to load all available languages and domains from the given directory (typically it is `i18n` directory within our application root directory).

Translations directory structure must follow `gettext` requirements (this is how it looks up data below the `i18n` directory):

```
{localedir}/{lang}/LC_MESSAGES/{domain}.mo
```

In order to generate a .mo file from a .po file:

```
$ msgfmt domain.po
```

TranslationsManager supports the following arguments in initialization:

- `directories` - a list of directories that holds translations.
- `default_lang` - a default language in translations. Defaults to `en`.

TranslationsManager supports a fallback mechanism. You can use *add_fallback()* to add fallback languages.

```
>>> from wheezy.core.i18n import TranslationsManager
>>> tm = TranslationsManager(['i18n'], default_lang='en')
>>> tm.add_fallback(('uk', 'ru'))
>>> tm.fallbacks
{'uk': ('uk', 'ru', 'en')}
```

The default language is always appended to the fallback list.

TranslationsManager supports dictionary access that accepts a language code as a key. So the following represents all translations related to `en` language code:

```
lang = tm['en']
```

`lang` is an instance of *defaultattrdict* where attributes correspond to translation file (translation domain), if it is not available there is fallback to an instance of `gettext.NullTranslations`:

```
assert 'Hello' == lang.messages.gettext('hello')
```

Seamless integration with `gettext` module simplifies your application internationalization and localization.

2.3.11 introspection

Type introspection is a capability to determine the type of an object at runtime.

import_name() - dynamically imports an object by its full name. The following two imports are equivalent:

```
from datetime import timedelta
import_name('datetime.timedelta')
```

import_name() lets you introduce lazy imports into your application.

2.3.12 json

Extends the standard `json` module from Python2.6 and `simplejson` for Python2.5 with support for `date`, `datetime`, `time` and `Decimal` types.

- *json_encode()* encodes obj as a JSON formatted string. Correctly escapes forward slashes to be able to embed javascript code. Decimal objects are converted to string (same applies when used with simplejson).
- *json_decode()* decodes a JSON document to a Python object. Float is parsed as Decimal.

2.3.13 luhn

TODO

2.3.14 mail

TODO

2.3.15 pooling

TODO

2.3.16 retry

TODO

2.3.17 url

Every URL consists of the following: the scheme name (or protocol), followed by a colon and two slashes, then, a domain name (alternatively, IP address), a port number (optionally), the path of the resource to be fetched, a query string, and an optional fragment identifier. Here is the syntax:

```
scheme://domain:port/path?query_string#fragment_id
```

The *url* module provides integration with urlparse module.

UrlParts - concrete class for urlparse.urlsplit() results, where argument parts is a tupple of length 6. There are the following methods:

- geturl() - returns the re-combined version of the original URL as a string.
- join(other) - joins with another UrlParts instance by taking none-empty values from other. Returns new UrlParts instance.

There is a factory function *urlparts()* for *UrlParts*, that let you create an instance of *UrlParts* with partial content.

2.3.18 uuid

A universally unique identifier (UUID) is an identifier that enable distributed systems to uniquely identify information without significant central coordination. A UUID is a 16-byte (128-bit) number.

The following functions available:

- *shrink_uuid()* - returns base64 representation of a uuid:

```
>>> shrink_uuid(UUID('a4af2f54-e988-4f5c-bfd6-351c79299b74'))
'pK8vVOmIT1y_1jUceSmbdA'
```

- *parse_uuid()* - decodes a base64 string to `uuid`:

```
>>> parse_uuid('pK8vVOmIT1y_1jUceSmbdA')
UUID('a4af2f54-e988-4f5c-bfd6-351c79299b74')
```

There is also a module attribute `UUID_EMPTY` defined, that is just an instance of UUID `'00000000-0000-0000-0000-000000000000'`.

2.4 Modules

2.4.1 wheezy.core

2.4.2 wheezy.core.benchmark

`bechmark` module.

class `wheezy.core.benchmark.`**Benchmark**(*targets*, *number*, *warmup_number=None*, *timer=None*)

Measure execution time of your code.

class `wheezy.core.benchmark.`**Timer**(*target*, *name*)

Intercept a call to given method in order to compute timing.

2.4.3 wheezy.core.collections

The `collections` module contains types and functions that define various collections and algorithms.

class `wheezy.core.collections.`**ItemAdapter**(*adaptee*, *index*)

Adapts `defaultdict(list).__getitem__` accessor to return item at `index` from the list. If `key` is not found return None.

get(*key*, *default=None*)

Return the value for key if key is in the adaptee, else default. If default is not given, it defaults to None, so that this method never raises a KeyError.

```
>>> d = defaultdict(list)
>>> d['a'].extend([1, 2, 3])
>>> a = ItemAdapter(d, 0)
>>> a.get('a')
1
>>> a.get('b', 100)
100
```

class `wheezy.core.collections.`**attrdict**

A dictionary with attribute-style access. Maps attribute access to dictionary.

```
>>> d = attrdict(a=1, b=2)
>>> sorted(d.items())
[('a', 1), ('b', 2)]
>>> d.a
1
```

```
>>> d.c = 3
>>> d.c
3
>>> d.d # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
AttributeError: ...
```

class wheezy.core.collections.**defaultattrdict**

A dictionary with attribute-style access. Maps attribute access to dictionary.

```
>>> d = defaultattrdict(str, a=1, b=2)
>>> d.a
1
```

```
>>> d.c = 3
>>> d.c
3
>>> d.d
''
```

wheezy.core.collections.**distinct**(*seq*)

Returns generator for unique items in `seq` with preserved order.

```
>>> list(distinct('1234512345'))
['1', '2', '3', '4', '5']
```

If the order is not important consider using `set` which is approximately eight times faster on large sequences.

```
>>> sorted(list(set('1234512345')))
['1', '2', '3', '4', '5']
```

wheezy.core.collections.**first_item_adapter**(*adaptee*)

Adapts `defaultdict(list).__getitem__` accessor to return the first item from the list.

```
>>> d = defaultdict(list)
>>> d['a'].extend([1, 2, 3])
>>> a = first_item_adapter(d)
```

Return a first item from the list.

```
>>> a['a']
1
```

wheezy.core.collections.**gzip_iterator**(*items*, *compress_level=6*)

Iterates over `items` and returns generator of gzipped items.

`items` - a list of bytes

```
>>> items = ['Hello World'.encode('latin1')]
>>> result = list(gzip_iterator(items))
>>> assert 3 == len(result)
>>> assert GZIP_HEADER == result[0]
```

wheezy.core.collections.**last_item_adapter**(*adaptee*)

Adapts `defaultdict(list).__getitem__` accessor to return the last item from the list.

```
>>> d = defaultdict(list)
>>> d['a'].extend([1, 2, 3])
>>> a = last_item_adapter(d)
```

Return a last item from the list.

```
>>> a['a']
3
```

wheezy.core.collections.**list_values**(*keys*, *dictionary*)

Returns *dictionary* values orderd by *keys*.

```
>>> d = {'1': 1, '2': 2}
>>> list_values(['1', '2', '3'], d)
[1, 2, None]
```

wheezy.core.collections.**map_keys**(*function*, *dictionary*)

Apply *function* to every key of *dictionary* and return a dictionary of the results.

```
>>> d = {'1': 1, '2': 2}
>>> sorted_items(map_keys(lambda key: 'k' + key, d))
[('k1', 1), ('k2', 2)]
```

wheezy.core.collections.**map_values**(*function*, *dictionary*)

Apply *function* to every value of *dictionary* and return a dictionary of the results.

```
>>> d = {'1': 1, '2': 2}
>>> sorted_items(map_values(lambda value: 2 * value, d))
[('1', 2), ('2', 4)]
```

wheezy.core.collections.**sorted_items**(*dictionary*)

Returns *dictionary* items sorted by key.

```
>>> d = {'1': 1, '2': 2}
>>> sorted_items(d)
[('1', 1), ('2', 2)]
```

2.4.4 wheezy.core.config

`config` module.

class wheezy.core.config.**Config**(*options=None*, *master=None*)

Promotes `options` dict to attributes. If an option can not be found in `options` tries to get it from `master`. `master` must have a requested option otherwise raises error.

`master` can be a module.

```
>>> from sys import modules
>>> m = modules[Config.__module__]
>>> c = Config(master=m)
>>> c.DEBUG
False
```

or an instance of `dict`.

```
>>> c = Config(master={'DEBUG': False})
>>> c.DEBUG
False
```

`options` override `master`.

```
>>> c = Config(options={'DEBUG': True}, master=m)
>>> c.DEBUG
True
```

If option is not defined it takes from `master`.

```
>>> c = Config(master=m)
>>> c.DEBUG
False
```

Configs can be nested

```
>>> m = Config(dict(B='b'))
>>> c = Config(dict(A='a'), master=m)
>>> c.B
'b'
```

if `options` is an instance of `Config` than use its options only so this config can have own master.

```
>>> options = Config(dict(A='a'))
>>> c = Config(options)
>>> c.A
'a'
```

2.4.5 wheezy.core.datetime

`datetime` module.

`wheezy.core.datetime.`**`format_http_datetime`**`(`*`stamp`*`)`

Formats datetime to a string following rfc1123 pattern.

```
>>> now = datetime(2011, 9, 19, 10, 45, 30, 0, UTC)
>>> format_http_datetime(now)
'Mon, 19 Sep 2011 10:45:30 GMT'
```

if timezone is not set in datetime instance the `stamp` is assumed to be in UTC (`datetime.utcnow`).

```
>>> now = datetime(2011, 9, 19, 10, 45, 30, 0)
>>> format_http_datetime(now)
'Mon, 19 Sep 2011 10:45:30 GMT'
```

```
>>> now = datetime.utcnow()
>>> assert format_http_datetime(now)
```

if `stamp` is a string just return it

```
>>> format_http_datetime('x')
'x'
```

```
>>> format_http_datetime(100) # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
TypeError: ...
```

wheezy.core.datetime.**format_iso_datetime**(*stamp*)

Return a string representing the date and time in ISO 8601 format. If the time is in UTC, adds a 'Z' directly after the time without a space.

see http://en.wikipedia.org/wiki/ISO_8601.

```
>>> class EET(tzinfo):
...     def utcoffset(self, dt):
...         return timedelta(minutes=120)
...     def dst(self, dt):
...         return timedelta()
>>> format_iso_datetime(datetime(2012, 2, 22, 12, 52, 29, 300))
'2012-02-22T12:52:29'
>>> format_iso_datetime(datetime(2012, 2, 22, 12, 52, 29, 300,
...     tzinfo=UTC))
'2012-02-22T12:52:29Z'
>>> format_iso_datetime(datetime(2012, 2, 22, 12, 52, 29, 300,
...     tzinfo=EET()))
'2012-02-22T12:52:29+02:00'
```

wheezy.core.datetime.**format_iso_time**(*stamp*)

Return a string representing the time in ISO 8601 format. If the time is in UTC, adds a 'Z' directly after the time without a space.

see http://en.wikipedia.org/wiki/ISO_8601.

```
>>> class EET(tzinfo):
...     def utcoffset(self, dt):
...         return timedelta(minutes=120)
...     def dst(self, dt):
...         return timedelta()
>>> format_iso_time(time(12, 52, 29, 300))
'12:52:29'
>>> format_iso_time(time(12, 52, 29, 300,
...     tzinfo=UTC))
'12:52:29Z'
>>> format_iso_time(time(12, 52, 29, 300,
...     tzinfo=EET()))
'12:52:29+02:00'
```

wheezy.core.datetime.**parse_http_datetime**(*stamp*)

Parses a string in rfc1123 format to `datetime`.

```
>>> parse_http_datetime('Mon, 19 Sep 2011 10:45:30 GMT')
datetime.datetime(2011, 9, 19, 10, 45, 30)
```

wheezy.core.datetime.**total_seconds**(*delta*)

Returns a total number of seconds for the given delta.

`delta` can be `datetime.timedelta`.

```
>>> total_seconds(timedelta(hours=2))
7200
```

or int:

```
>>> total_seconds(100)
100
```

otherwise raise `TypeError`.

```
>>> total_seconds('100') # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
TypeError: ...
```

class wheezy.core.datetime.**utc**(*name*)

UTC timezone.

dst(*dt*)

DST is not in effect.

```
>>> UTC.dst(None)
datetime.timedelta(0)
```

tzname(*dt*)

Name of time zone.

```
>>> GMT.tzname(None)
'GMT'
>>> UTC.tzname(None)
'UTC'
```

utcoffset(*dt*)

Offset from UTC.

```
>>> UTC.utcoffset(None)
datetime.timedelta(0)
```

2.4.6 wheezy.core.db

`session` module.

class wheezy.core.db.**NullSession**

Null session is supposed to be used in mock scenarios.

commit()

Simulates commit. Asserts the session is used in scope.

cursor(**args*, ***kwargs*)

Ensure session is entered.

class wheezy.core.db.**NullTPCSession**

Null TPC session is supposed to be used in mock scenarios.

commit()

Simulates commit. Asserts the session is used in scope.

enlist(*session*)

Ensure session is entered.

class wheezy.core.db.**Session**(*pool*)
Session works with a pool of database connections. Database connection must be implemented per Database API Specification v2.0 (see PEP0249).

commit()
Commit any pending transaction to the database.

connection
Return the session connection. Not intended to be used directly, use *cursor* method instead.

cursor(**args*, ***kwargs*)
Return a new cursor object using the session connection.

class wheezy.core.db.**TPCSession**(*format_id=7*, *global_transaction_id=None*, *branch_qualifier=''*)
Two-Phase Commit protocol session that works with a pool of database connections. Database connection must be implemented per Database API Specification v2.0 (see PEP0249).

commit()
Commit any pending transaction to the database.

enlist(*session*)
Begins a TPC transaction with the given session.

2.4.7 wheezy.core.descriptors

descriptors module.

class wheezy.core.descriptors.**attribute**(*f*)
attribute decorator is intended to promote a function call to object attribute. This means the function is called once and replaced with returned value.

```
>>> class A:
...     def __init__(self):
...         self.counter = 0
...     @attribute
...     def count(self):
...         self.counter += 1
...         return self.counter
>>> a = A()
>>> a.count
1
>>> a.count
1
```

2.4.8 wheezy.core.feistel

feistel module.

wheezy.core.feistel.**make_feistel_number**(*f*)
Generate pseudo random consistent reversal number per Feistel cypher algorithm.

see http://en.wikipedia.org/wiki/Feistel_cipher

```
>>> feistel_number = make_feistel_number(sample_f)
>>> feistel_number(1)
573852158
```

(continues on next page)

(continued from previous page)

```
>>> feistel_number(2)
1788827948
>>> feistel_number(123456789)
1466105040
```

Reversable

```
>>> feistel_number(1466105040)
123456789
>>> feistel_number(1788827948)
2
>>> feistel_number(573852158)
1
```

2.4.9 wheezy.core.gzip

One-shot compression and decompression.

`wheezy.core.gzip.`**`compress`**(*data*, *compresslevel=9*)
Compress data in one shot.

`wheezy.core.gzip.`**`decompress`**(*data*)
Decompress data in one shot.

2.4.10 wheezy.core.httpclient

`class` `wheezy.core.httpclient.`**`HTTPClient`**(*url*, *headers=None*)
HTTP client sends HTTP requests to server in order to accomplish an application specific use cases, e.g. remote web server API, etc.

`ajax_get`(*path*, ***kwargs*)
Sends GET HTTP AJAX request.

`ajax_go`(*path=None*, *method='GET'*, *params=None*, *headers=None*, *content_type=''*, *body=''*)
Sends HTTP AJAX request to web server.

`ajax_post`(*path*, ***kwargs*)
Sends POST HTTP AJAX request.

`content`
Returns a content of the response.

`follow`()
Follows HTTP redirect (e.g. status code 302).

`get`(*path*, ***kwargs*)
Sends GET HTTP request.

`go`(*path=None*, *method='GET'*, *params=None*, *headers=None*, *content_type=''*, *body=''*)
Sends HTTP request to web server.

The `content_type` takes priority over `params` to use `body`. The `body` can be a string or file like object.

`head`(*path*, ***kwargs*)
Sends HEAD HTTP request.

json
Returns a json response.

post (*path*, ***kwargs*)
Sends POST HTTP request.

2.4.11 wheezy.core.i18n

`i18n` module.

class `wheezy.core.i18n.`**TranslationsManager** (*directories=None*, *default_lang='en'*)
Manages several languages and translation domains.

add_fallback (*languages*)
Adds fallback languages.

```
>>> tm = TranslationsManager()
>>> tm.add_fallback(('uk', 'ru'))
>>> tm.fallbacks
{'uk': ('uk', 'ru', 'en')}
```

load (*localedir*)
Load all available languages and domains from the given directory.

{localedir}/{lang}/LC_MESSAGES/{domain}.mo

In order to generate .mo file from .po file: $ msgfmt domain.po

```
>>> curdir = os.path.dirname(__file__)
>>> localedir = os.path.join(curdir, 'tests', 'i18n')
>>> tm = TranslationsManager()
>>> tm.load(localedir)
>>> sorted(tm.translations.keys()) # doctest: +ELLIPSIS
[...]
```

Assess by language:

```
>>> lang = tm['en']
>>> m = lang['messages']
>>> m.gettext('hello') # doctest: +ELLIPSIS
'...ello'
>>> lang = tm['de']
>>> m = lang['messages']
>>> m.gettext('hello') # doctest: +ELLIPSIS
'...'
```

Assess by translation domain:

```
>>> messages = tm.domains['messages']
>>> m = messages['en']
>>> m.gettext('hello') # doctest: +ELLIPSIS
'...ello'
```

Fallback to English:

```
>>> m.gettext('world') # doctest: +ELLIPSIS
'...orld'
```

If translation is unknown `key` returned

```
>>> m.gettext('test')
'test'
```

2.4.12 wheezy.core.introspection

wheezy.core.introspection.**import_name**(*fullname*)

Dynamically imports object by its full name.

```
>>> from datetime import timedelta
>>> import_name('datetime.timedelta') is timedelta
True
```

class wheezy.core.introspection.**looks**(*cls*)

Performs duck typing checks for two classes.

Typical use:

```
assert looks(IFoo, ignore_argspec=['pex']).like(Foo)
```

like(*cls*, *notice=None*, *ignore_funcs=None*, *ignore_argspec=None*)

Check if *self.cls* can be used as duck typing for *cls*.

cls - class to be checked for duck typing. *ignore_funcs* - a list of functions to ignore *ignore_argspec* - a list of functions to ignore arguments spec.

2.4.13 wheezy.core.json

`json` module.

class wheezy.core.json.**JSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

wheezy.core.json.**json_decode**(*s*)

Decode `s` (containing a JSON unicode string) to a Python object.

wheezy.core.json.**json_encode**(*obj*)

Encode `obj` as a JSON formatted unicode string.

Correctly escapes forward slash to be able embed javascript code.

2.4.14 wheezy.core.luhn

`luhn` module.

`wheezy.core.luhn.`**`digits_of`**`(n)`
Returns a list of all digits from given number.

```
>>> digits_of(123456789)
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

`wheezy.core.luhn.`**`is_luhn_valid`**`(n)`

```
>>> is_luhn_valid(1234567897)
True
>>> is_luhn_valid(473802106)
True
>>> is_luhn_valid(34518893)
False
```

`wheezy.core.luhn.`**`luhn_checksum`**`(n)`
Calculates checksum based on Luhn algorithm, also known as the "modulus 10" algorithm.

see http://en.wikipedia.org/wiki/Luhn_algorithm

```
>>> luhn_checksum(1788827948)
0
>>> luhn_checksum(573852158)
1
>>> luhn_checksum(123456789)
7
```

`wheezy.core.luhn.`**`luhn_sign`**`(n)`
Signs given number by Luhn checksum.

```
>>> luhn_sign(78482748)
784827487
>>> luhn_sign(47380210)
473802106
>>> luhn_sign(123456789)
1234567897
```

`wheezy.core.luhn.`**`sum2digits`**`(d)`
Sum digits of a number that is less or equal 18.

```
>>> sum2digits(2)
2
>>> sum2digits(17)
8
```

2.4.15 wheezy.core.mail

`mail` module.

`class` `wheezy.core.mail.`**`Alternative`**`(content, content_type='text/html', charset=None)`
Represents alternative mail message.

class wheezy.core.mail.**Attachment**(*name*, *content*, *content_type=None*, *disposition=None*, *name_charset=None*, *content_charset=None*)

An attachment to mail message.

classmethod from_file(*path*)

Creates an attachment from file.

class wheezy.core.mail.**MailMessage**(*subject=''*, *content=''*, *from_addr=None*, *to_addrs=None*, *cc_addrs=None*, *bcc_addrs=None*, *reply_to_addrs=None*, *content_type='text/plain'*, *charset='us-ascii'*)

Mail message.

class wheezy.core.mail.**Related**(*content_id*, *content*, *content_type*)

A resource related to alternative mail message.

classmethod from_file(*path*)

Creates a related mail resource from file.

class wheezy.core.mail.**SMTPClient**(*host='127.0.0.1'*, *port=25*, *use_tls=False*, *username=None*, *password=None*)

SMTP client that can be used to send mail.

send(*message*)

Sends a single mail message.

send_multi(*messages*)

Sends multiple mail messages.

wheezy.core.mail.**mail_address**(*addr*, *name=None*, *charset='utf8'*)

Returns mail address formatted string.

2.4.16 wheezy.core.pooling

`pooling` module.

class wheezy.core.pooling.**EagerPool**(*create_factory*, *size*)

Eager pool implementation.

Allocates all pool items during initialization.

count

Returns a number of available items in the pool.

class wheezy.core.pooling.**LazyPool**(*create_factory*, *size*)

Lazy pool implementation.

Allocates pool items as necessary.

acquire()

Return an item from pool. Blocks until an item is available.

count

Returns a number of available items in the pool.

class wheezy.core.pooling.**Pooled**(*pool*)

`Pooled` serves context manager purpose, effectively acquiring and returning item to the pool.

Here is an example:

```
with Pooled(pool) as item:
    # do something with item
```

2.4.17 wheezy.core.retry

`retry` module.

`wheezy.core.retry.`**`make_retry`**(*timeout*, *start*, *end=None*, *slope=1.0*, *step=0.0*)

Return a function that accepts a single argument `acquire` which should be a callable (without any arguments) that returns a boolean value when attempt to acquire some resource or perform operation succeeded or not.

If a first attempt fails the retry function sleeps for `start` and later delay time is increased linearly per `slope` and `step` until `end`. A last delay is a time remaining. A total retry time is limited by `timeout`.

`timeout` - a number of seconds for the entire retry operation from the first failed attempt to the last (excluding time for both acquire operations).

`start` - a time for initial delay.

`end` - a time for a longest delay between retry attempts.

`slope` and `step` - coefficients for linear calculation of the next delay.

Example 1:

```
# delays: 0.5, 0.5, 0.5, 0.5
retry = make_retry(timeout=10.0, start=0.5)
```

Example 2:

```
# delays: 0.5, 0.1, 1.5, 2.0
retry = make_retry(timeout=10.0, start=0.5, end=2.0, step=0.5)
```

Example 3:

```
# delays: 0.05, 0.1, 0.2, 0.4, 0.8, 1.6, 2.0
retry = make_retry(timeout=10.0, start=0.05, end=2.0, slope=2.0)
if retry(lambda: acquire('something')):
    # good to go
else:
    # timed out
```

2.4.18 wheezy.core.url

`url` module

`class` `wheezy.core.url.`**`UrlParts`**(*parts*)

Concrete class for `urlparse.urlsplit()` results.

`geturl`()

Return the re-combined version of the original URL as a string.

```
>>> from urllib.parse import urlsplit
>>> parts = urlsplit('http://www.python.org/dev/peps/pep-3333')
>>> parts = urlparts(parts)
>>> parts.geturl()
'http://www.python.org/dev/peps/pep-3333'
```

`join`(*other*)

Joins with another `UrlParts` instance by taking none-empty values from `other`. Returns new `UrlParts` instance.

Query and Fragment parts are taken unconditionally from `other`.

```
>>> from urllib.parse import urlsplit
>>> parts = urlsplit('http://www.python.org/dev/peps/pep-3333')
>>> parts = urlparts(parts)
>>> parts = parts.join(urlparts(scheme='https', path='/test'))
>>> parts.geturl()
'https://www.python.org/test'
```

wheezy.core.url.**urlparts**(*parts=None*, *scheme=None*, *netloc=None*, *path=None*, *query=None*, *fragment=None*)

Factory function for *UrlParts* that create an instance *UrlParts* with partial content.

`parts` must be a 5-tuple: (scheme, netloc, path, query, fragment)

```
>>> from urllib.parse import urlsplit
>>> parts = urlsplit('http://www.python.org/dev/peps/pep-3333')
>>> urlparts(parts)
urlparts('http', 'www.python.org', '/dev/peps/pep-3333', '', '')
>>> urlparts(scheme='https', path='/test')
urlparts('https', None, '/test', None, None)
```

Otherwise raise assertion error

```
>>> urlparts(('https', )) # doctest: +ELLIPSIS
Traceback (most recent call last):
    ...
AssertionError: ...
```

2.4.19 wheezy.core.uuid

`uuid` module.

wheezy.core.uuid.**parse_uuid**(*s*)

```
>>> parse_uuid('pK8vVOmIT1y_1jUceSmbdA')
UUID('a4af2f54-e988-4f5c-bfd6-351c79299b74')
>>> parse_uuid('0Xq6iBnDQA6t7j7Pk12ycg')
UUID('d17aba88-19c3-400e-adee-3ecf935db272')
>>> parse_uuid('Oa4T7iAqQtGRF2-2_dFppA')
UUID('39ae13ee-202a-42d1-9117-6fb6fdd169a4')
>>> parse_uuid('AAAAAAAAAAAAAAAAAAAAAA')
UUID('00000000-0000-0000-0000-000000000000')
```

Return an empty uuid in case the string is empty or length not equal to 22.

```
>>> parse_uuid('')
UUID('00000000-0000-0000-0000-000000000000')
>>> parse_uuid('x')
UUID('00000000-0000-0000-0000-000000000000')
```

Incorrect base64 padding.

```
>>> parse_uuid('AAAAAAAAAA*AAAAAAAAAA')
UUID('00000000-0000-0000-0000-000000000000')
```

wheezy.core.uuid.**shrink_uuid**(*uuid*)

Returns base64 representation of `uuid`.

```
>>> shrink_uuid(UUID('a4af2f54-e988-4f5c-bfd6-351c79299b74'))
'pK8vVOmIT1y_1jUceSmbdA'
>>> shrink_uuid(UUID('d17aba88-19c3-400e-adee-3ecf935db272'))
'0Xq6iBnDQA6t7j7Pk12ycg'
>>> shrink_uuid(UUID('39ae13ee-202a-42d1-9117-6fb6fdd169a4'))
'Oa4T7iAqQtGRF2-2_dFppA'
>>> shrink_uuid(UUID_EMPTY)
'AAAAAAAAAAAAAAAAAAAAAA'
```

Python Module Index

W

Index