mom Documentation

Release 0.1.0

Yesudeep Mangalapilly

July 30, 2015

Contents

Mother of all our Python projects. Batteries for Python.

CHAPTER 1

Getting the library

```
$ pip install mom
```

or

```
$ git clone git://github.com/gorakhargosh/mom.git

or

$ git clone http://code.google.com/p/python-mom/

$ cd mom
$ python setup.py install
```

CHAPTER 2

User Guides

2.1 Contributing

Welcome hackeratti! So you have got something you would like to see in mom? Whee. This document will help you get started.

2.1.1 Important URLs

mom uses git to track code history and hosts its code repository at github. The issue tracker is where you can file bug reports and request features or enhancements to mom.

2.1.2 Before you start

Ensure your system has the following programs and libraries installed before beginning to hack:

1. Python
2. git
3. ssh

2.1.3 Setting up the Work Environment

mom makes extensive use of zc.buildout to set up its work environment. You should get familiar with it.

Steps to setting up a clean environment:

1. Fork the code repository into your github account. Let us call you `hackeratti`. That *is* your name innit? Replace `hackeratti` with your own username below if it isn't.
2. Clone your fork and setup your environment:

```
$ git clone --recursive git@github.com:hackeratti/mom.git
$ cd mom
$ python bootstrap.py --distribute
$ bin/buildout
```

Important: Re-run `bin/buildout` every time you make a change to the `buildout.cfg` file.

That's it with the setup. Now you're ready to hack on mom.

2.1.4 Enabling Continuous Integration

The repository checkout contains a script called `autobuild.sh` which you should run prior to making changes. It will detect changes to Python source code or restructuredText documentation files anywhere in the directory tree and rebuild sphinx documentation, run all tests using unittest2, and generate coverage reports.

Start it by issuing this command in the `mom` directory checked out earlier:

```
$ tools/autobuild.sh
...
```

Happy hacking!

CHAPTER 3

API Documentation

3.1 *mom*

synopsis Mother of all our Python projects.

module mom

3.1.1 How many times have you noticed a `utils` subpackage or module?

Yeah. There is a lot of code duplication that occurs throughout our Python-based projects and results in code that is harder to maintain in the long term. Not to mention all the duplicate test code and more maintenance.

Therefore, we have decided to move all those `util` modules and subpackages to a central library, which we use throughout our projects. If you have a `utils` module, chances are you're duplicating and wasting effort whereas instead you could use tested code provided by this library. If there's something not included in this library and think it should, speak up.

synopsis Deals with a lot of cross-version issues.

module mom.builtins

`bytes`, `str`, `unicode`, and `basestring` mean different things to Python 2.5, 2.6, and 3.x.

These are the original meanings of the types.

Python 2.5

- `bytes` is not available.
- `str` is a byte string.
- `unicode` converts to unicode string.
- `basestring` exists.

Python 2.6

- `bytes` is available and maps to str
- `str` is a byte string.
- `unicode` converts to unicode string
- `basestring` exists.

Python 3.x

- `bytes` is available and does not map to `str`.
- `str` maps to the earlier `unicode`, but `unicode` has been removed.
- `basestring` has been removed.
- `unicode` has been removed

This module introduces the "bytes" type for Python 2.5 and adds a few utility functions that will continue to keep working as they should even when Python versions change.

Rules to follow: * Use `bytes` where you want byte strings (binary data).

The meanings of these types have been changed to suit Python 3.

Encodings

`mom.builtins.`**`bin`**(*number*, *prefix='0b'*)
Converts a long value to its binary representation.

Parameters

- **`number`** – Long value.
- **`prefix`** – The prefix to use for the bitstring. Default "0b" to mimic Python builtin `bin()`.

Returns Bit string.

`mom.builtins.`**`hex`**(*number*, *prefix='0x'*)
Converts a integer value to its hexadecimal representation.

Parameters

- **`number`** – Integer value.
- **`prefix`** – The prefix to use for the hexadecimal string. Default "0x" to mimic `hex()`.

Returns Hexadecimal string.

`mom.builtins.`**`byte`**(*number*)
Converts a number between 0 and 255 (both inclusive) to a base-256 (byte) representation.

Use it as a replacement for `chr` where you are expecting a byte because this will work on all versions of Python.

Raises :class:`struct.error` on overflow.

Parameters **`number`** – An unsigned integer between 0 and 255 (both inclusive).

Returns A single byte.

`mom.builtins.`**`byte_ord`**(*byte_*)
Returns the ordinal value of the given byte.

Parameters **`byte`** – The byte.

Returns Integer representing ordinal value of the byte.

Bits and bytes size counting

`mom.builtins.`**`bytes_leading`**(*raw_bytes*, *needle='\x00'*)
Finds the number of prefixed byte occurrences in the haystack.

Useful when you want to deal with padding.

Parameters

- **raw_bytes** – Raw bytes.
- **needle** – The byte to count. Default .

Returns The number of leading needle bytes.

mom.builtins.**bytes_trailing**(*raw_bytes*, *needle='\x00'*)
Finds the number of suffixed byte occurrences in the haystack.

Useful when you want to deal with padding.

Parameters

- **raw_bytes** – Raw bytes.
- **needle** – The byte to count. Default .

Returns The number of trailing needle bytes.

mom.builtins.**integer_bit_length**(*number*)
Number of bits needed to represent a integer excluding any prefix 0 bits.

Parameters **number** – Integer value. If num is 0, returns 0. Only the absolute value of the number is considered. Therefore, signed integers will be abs(num) before the number's bit length is determined.

Returns Returns the number of bits in the integer.

mom.builtins.**integer_bit_size**(*number*)
Number of bits needed to represent a integer excluding any prefix 0 bits.

Parameters **number** – Integer value. If num is 0, returns 1. Only the absolute value of the number is considered. Therefore, signed integers will be abs(num) before the number's bit length is determined.

Returns Returns the number of bits in the integer.

mom.builtins.**integer_byte_length**(*number*)
Number of bytes needed to represent a integer excluding any prefix 0 bytes.

Parameters **number** – Integer value. If num is 0, returns 0.

Returns The number of bytes in the integer.

mom.builtins.**integer_byte_size**(*number*)
Size in bytes of an integer.

Parameters **number** – Integer value. If num is 0, returns 1.

Returns Size in bytes of an integer.

Type detection predicates

mom.builtins.**is_bytes**(*obj*)
Determines whether the given value is a bytes instance.

Parameters **obj** – The value to test.

Returns `True` if value is a bytes instance; `False` otherwise.

mom.builtins.**is_bytes_or_unicode**(*obj*)
Determines whether the given value is an instance of a string irrespective of whether it is a byte string or a Unicode string.

Parameters **obj** – The value to test.

Returns `True` if value is any type of string; `False` otherwise.

mom.builtins.**is_integer**(*obj*)
Determines whether the object value is actually an integer and not a bool.

Parameters **obj** – The value to test.

Returns `True` if yes; `False` otherwise.

mom.builtins.**is_sequence**(*obj*)
Determines whether the given value is a sequence.

Sets, lists, tuples, bytes, dicts, and strings are treated as sequence.

Parameters **obj** – The value to test.

Returns `True` if value is a sequence; `False` otherwise.

mom.builtins.**is_unicode**(*obj*)
Determines whether the given value is a Unicode string.

Parameters **obj** – The value to test.

Returns `True` if value is a Unicode string; `False` otherwise.

Number predicates

People screw these up too. Useful in functional programming.

mom.builtins.**is_even**(*num*)
Determines whether a number is even.

Parameters **num** – Integer

Returns `True` if even; `False` otherwise.

mom.builtins.**is_negative**(*num*)
Determines whether a number is negative.

Parameters **num** – Number

Returns `True` if positive; `False` otherwise.

mom.builtins.**is_odd**(*num*)
Determines whether a number is odd.

Parameters **num** – Integer

Returns `True` if odd; `False` otherwise.

mom.builtins.**is_positive**(*num*)
Determines whether a number is positive.

Parameters **num** – Number

Returns `True` if positive; `False` otherwise.

synopsis Common collections.

module mom.

Queues

class mom.collections.**SetQueue**(*maxsize=0*)

Thread-safe implementation of an ordered set queue, which coalesces duplicate items into a single item if the older occurrence has not yet been read and maintains the order of items in the queue.

Ordered set queues are useful when implementing data structures like event buses or event queues where duplicate events need to be coalesced into a single event. An example use case is the inotify API in the Linux kernel which shares the same behavior.

Queued items must be immutable and hashable so that they can be used as dictionary keys or added to sets. Items must have only read-only properties and must implement the `__hash__()`, `__eq__()`, and `__ne__()` methods to be hashable.

Author Yesudeep Manglapilly <yesudeep@gmail.com>

Author Lukáš Lalinský <lalinsky@gmail.com>

An example item class implementation follows:

```
class QueuedItem(object):
    def __init__(self, a, b):
        self._a = a
        self._b = b

    @property
    def a(self):
        return self._a

    @property
    def b(self):
        return self._b

    def _key(self):
        return (self._a, self._b)

    def __eq__(self, item):
        return self._key() == item._key()

    def __ne__(self, item):
        return self._key() != item._key()

    def __hash__(self):
        return hash(self._key())
```

class mom.collections.**AttributeDict**(**args*, ***kw*)

A dictionary with attribute-style access. It maps attribute access to the real dictionary.

Subclass properties will override dictionary keys.

Author Alice Zoë Bevan–McGregor

License MIT License.

mom.collections.**attrdict**

alias of *AttributeDict*

synopsis Decorators used throughout the library.

module mom.decorators

mom.decorators.**deprecated**(*func*)

Marks functions as deprecated.

This is a decorator which can be used to mark functions as deprecated. It will result in a warning being emitted when the function is used.

Usage:

```
    @deprecated
    def my_func():
        pass

    @other_decorators_must_be_upper
    @deprecated
    def my_func():
        pass
```

Parameters **func** – The function to deprecate.

Returns Deprecated function object.

synopsis Functional programming primitives.

module mom.functional

Higher-order functions

These functions accept other functions as arguments and apply them over specific types of data structures. Here's an example of how to find the youngest person and the oldest person from among people. Place it into a Python module and run it:

```
import pprint

from mom import functional

PEOPLE = [
    {"name": "Harry",    "age": 100},
    {"name": "Hermione", "age": 16},
    {"name": "Rob",      "age": 200},
]

def youngest(person1, person2):
    '''Comparator that returns the youngest of two people.'''
    return person1 if person1["age"] <= person2["age"] else person2

def oldest(person1, person2):
    '''Comparator that returns the oldest of two people.'''
    return person1 if person1["age"] >= person2["age"] else person2

who_youngest = functional.reduce(youngest, PEOPLE)
who_oldest = functional.reduce(oldest, PEOPLE)

pprint.print(who_youngest)
# -> {"age" : 16, "name" : "Hermione"}
pprint.print(who_oldest)
# -> {"age" : 200, "name" : "Rob"}

# More examples.
```

```
# Now let's list all the names of the people.
names_of_people = functional.pluck(PEOPLE, "name")
pprint.print(names_of_people)
# -> ("Harry", "Hermione", "Rob")

# Let's weed out all people who don't have an "H" in their names.
pprint.print(functional.reject(lambda name: "H" not in name,
                               names_of_people))
# -> ("Harry", "Hermione")

# Or let's partition them into two groups
pprint.print(functional.partition(lambda name: "H" in name,
                                  names_of_people))
# -> (["Harry", "Hermione"], ["Rob"])

# Let's find all the members of a module that are not exported to wildcard
# imports by its ``__all__`` member.
pprint.print(functional.difference(dir(functional), functional.__all__))
# -> ["__all__",
#     "__author__",
#     "__builtins__",
#     "__doc__",
#     "__file__",
#     "__name__",
#     "__package__",
#     "_compose",
#     "_contains_fallback",
#     "_get_iter_next",
#     "_ifilter",
#     "_ifilterfalse",
#     "_leading",
#     "_some1",
#     "_some2",
#     "absolute_import",
#     "builtins",
#     "chain",
#     "collections",
#     "functools",
#     "itertools",
#     "map",
#     "starmap"]
```

Higher-order functions are extremely useful where you want to express yourself succinctly instead of writing a ton of for and while loops.

Warning: About consuming iterators multiple times
Now before you go all guns blazing with this set of functions, please note that Python generators/iterators are for single use only. Attempting to use the same iterator multiple times will cause unexpected behavior in your code. Be careful.

Terminology

- A **predicate** is a function that returns the truth value of its argument.
- A **complement** is a predicate function that returns the negated truth value of its argument.
- A **walker** is a function that consumes one or more items from a sequence at a time.

- A **transform** is a function that transforms its arguments to produce a result.
- **Lazy evaluation** is evaluation delayed until the last possible instant.
- **Materialized iterables** are iterables that take up memory equal to their size.
- **Dematerialized iterables** are iterables (usually iterators/generators) that are evaluated lazily.

Iteration and aggregation

mom.functional.**each**(*walker*, *iterable*)
Iterates over iterable yielding each item in turn to the walker function.

Parameters

- **walker** – The method signature is as follows:

  f(x, y)

  where `x, y` is a `key, value` pair if iterable is a dictionary, otherwise `x, y` is an `index, item` pair.
- **iterable** – Iterable sequence or dictionary.

mom.functional.**reduce**(*transform*, *iterable*, **args*)
Aggregate a sequence of items into a single item. Python equivalent of Haskell's left fold.

Please see Python documentation for reduce. There is no change in behavior. This is simply a wrapper function.

If you need reduce_right (right fold):

```
reduce_right = foldr = lambda f, i: lambda s: reduce(f, s, i)
```

Parameters

- **transform** – Function with signature:

```
f(x, y)
```

- **iterable** – Iterable sequence.
- **args** – Initial value.

Returns Aggregated item.

Logic and search

mom.functional.**every**(*predicate*, *iterable*)
Determines whether the predicate is true for all elements in the iterable.

Parameters

- **predicate** – Predicate function of the form:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.

Returns `True` if the predicate is true for all elements in the iterable.

mom.functional.**find**(*predicate*, *iterable*, *start=0*)
Determines the first index where the predicate is true for an element in the iterable.

Parameters

- **predicate** – Predicate function of the form:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.
- **start** – Start index.

Returns -1 if not found; index (>= start) if found.

mom.functional.**none**(*predicate*, *iterable*)
Determines whether the predicate is false for all elements in in iterable.

Parameters

- **predicate** – Predicate function of the form:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.

Returns `True` if the predicate is false for all elements in the iterable.

mom.functional.**some**(*predicate*, *iterable*)
Determines whether the predicate applied to any element of the iterable is true.

Parameters

- **predicate** – Predicate function of the form:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.

Returns `True` if the predicate applied to any element of the iterable is true; `False` otherwise.

Filtering

mom.functional.**ireject**(*predicate*, *iterable*)
Reject all items from the sequence for which the predicate is true.

ireject(function or None, sequence) –> iterator

Parameters

- **predicate** – Predicate function. If `None`, reject all truthy items.
- **iterable** – Iterable to filter through.

Yields A sequence of all items for which the predicate is false.

mom.functional.**iselect**(*predicate*, *iterable*)
Select all items from the sequence for which the predicate is true.

iselect(function or None, sequence) –> iterator

Parameters

- **predicate** – Predicate function. If `None`, select all truthy items.
- **iterable** – Iterable.

Yields A iterable of all items for which the predicate is true.

`mom.functional.`**partition**(*predicate*, *iterable*)
Partitions an iterable into two iterables where for the elements of one iterable the predicate is true and for those of the other it is false.

Parameters

- **predicate** – Function of the format:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.

Returns Tuple (selected, rejected)

`mom.functional.`**reject**(*predicate*, *iterable*)
Reject all items from the sequence for which the predicate is true.

reject(function or None, sequence) -> list

Parameters

- **predicate** – Predicate function. If `None`, reject all truthy items.
- **iterable** – The iterable to filter through.

Returns A list of all items for which the predicate is false.

`mom.functional.`**select**(*predicate*, *iterable*)
Select all items from the sequence for which the predicate is true.

select(function or None, sequence) -> list

Parameters

- **predicate** – Predicate function. If `None`, select all truthy items.
- **iterable** – Iterable.

Returns A list of all items for which the predicate is true.

Counting

`mom.functional.`**leading**(*predicate*, *iterable*, *start=0*)
Returns the number of leading elements in the iterable for which the predicate is true.

Parameters

- **predicate** – Predicate function of the form:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.
- **start** – Start index. (Number of items to skip before starting counting.)

mom.functional.**tally**(*predicate*, *iterable*)
Count how many times the predicate is true.

Taken from the Python documentation. Under the PSF license.

Parameters

- **predicate** – Predicate function.
- **iterable** – Iterable sequence.

Returns The number of times a predicate is true.

mom.functional.**trailing**(*predicate*, *iterable*, *start=-1*)
Returns the number of trailing elements in the iterable for which the predicate is true.

Parameters

- **predicate** – Predicate function of the form:

```
f(x) -> bool
```

- **iterable** – Iterable sequence.
- **start** – If start is negative, -1 indicates starting from the last item. Therefore, -2 would mean start counting from the second last item. If start is 0 or positive, it indicates the number of items to skip before beginning to count.

Function-generators

mom.functional.**complement**(*predicate*)
Generates a complementary predicate function for the given predicate function.

Parameters **predicate** – Predicate function.

Returns Complementary predicate function.

mom.functional.**compose**(*function*, **functions*)
Composes a sequence of functions such that:

```
compose(g, f, s) -> g(f(s()))
```

Parameters **functions** – An iterable of functions.

Returns A composition function.

Iterators

These functions take iterators as arguments.

mom.functional.**eat**(*iterator*, *amount*)
Advance an iterator n-steps ahead. If n is None, eat entirely.

Taken from the Python documentation. Under the PSF license.

Parameters

- **iterator** – An iterator.
- **amount** – The number of steps to advance.

Yields An iterator.

Iterable sequences

These functions allow you to filter, manipulate, slice, index, etc. iterable sequences.

Indexing and slicing

mom.functional.**chunks**(*iterable*, *size*, **args*, ***kwargs*)
Splits an iterable into materialized chunks each of specified size.

Parameters

- **iterable** – The iterable to split. Must be an ordered sequence to guarantee order.
- **size** – Chunk size.
- **padding** – This must be an iterable or None. So if you want a `True` filler, use [True] or (True,) depending on whether the iterable is a list or a tuple. Essentially, it must be the same type as the iterable.

  If a pad value is specified appropriate multiples of it will be concatenated at the end of the iterable if the size is not an integral multiple of the length of the iterable:

  tuple(chunks("aaabccd", 3, "-")) -> ("aaa", "bcc", "d–")

  tuple(chunks((1, 1, 1, 2, 2), 3, (None,))) -> ((1, 1, 1,), (2, 2, None))

  If no padding is specified, nothing will be appended if the chunk size is not an integral multiple of the length of the iterable. That is, the last chunk will have chunk size less than the specified chunk size. :yields: Generator of materialized chunks.

mom.functional.**head**(*iterable*)
Returns the first element out of an iterable.

Parameters **iterable** – Iterable sequence.

Returns First element of the iterable sequence.

mom.functional.**ichunks**(*iterable*, *size*, **args*, ***kwargs*)
Splits an iterable into iterators for chunks each of specified size.

Parameters

- **iterable** – The iterable to split. Must be an ordered sequence to guarantee order.
- **size** – Chunk size.
- **padding** – If a pad value is specified appropriate multiples of it will be appended to the end of the iterator if the size is not an integral multiple of the length of the iterable:

  map(tuple, ichunks("aaabccd", 3, "-")) -> [("a", "a", "a"), ("b", "c", "c"), ("d", "-", "-")]

  map(tuple, ichunks("aaabccd", 3, None)) -> [("a", "a", "a"), ("b", "c", "c"), ("d", None, None)]

  If no padding is specified, nothing will be appended if the chunk size is not an integral multiple of the length of the iterable. That is, the last chunk will have chunk size less than the specified chunk size. :yields: Generator of chunk iterators.

mom.functional.**ipeel**(*iterable*, *count=1*)
Returns an iterator for the meat of an iterable by peeling off the specified number of elements from both ends.

Parameters

- **iterable** – Iterable sequence.
- **count** – The number of elements to remove from each end.

Yields Peel iterator.

mom.functional.**itail**(*iterable*)
Returns an iterator for all elements excluding the first out of an iterable.

Parameters **iterable** – Iterable sequence.

Yields Iterator for all elements of the iterable sequence excluding the first.

mom.functional.**last**(*iterable*)
Returns the last element out of an iterable.

Parameters **iterable** – Iterable sequence.

Returns Last element of the iterable sequence.

mom.functional.**nth**(*iterable*, *index*, *default=None*)
Returns the nth element out of an iterable.

Parameters

- **iterable** – Iterable sequence.
- **index** – Index
- **default** – If not found, this or `None` will be returned.

Returns nth element of the iterable sequence.

mom.functional.**peel**(*iterable*, *count=1*)
Returns the meat of an iterable by peeling off the specified number of elements from both ends.

Parameters

- **iterable** – Iterable sequence.
- **count** – The number of elements to remove from each end.

Returns Peeled sequence.

mom.functional.**tail**(*iterable*)
Returns all elements excluding the first out of an iterable.

Parameters **iterable** – Iterable sequence.

Returns All elements of the iterable sequence excluding the first.

mom.functional.**round_robin**(**iterables*)
Returns items from the iterables in a round-robin fashion.

Taken from the Python documentation. Under the PSF license. Recipe credited to George Sakkis

Example:

```
round_robin("ABC", "D", "EF") --> A D E B F C"
```

Parameters **iterables** – Variable number of inputs for iterable sequences.

Yields Items from the iterable sequences in a round-robin fashion.

mom.functional.**take**(*iterable*, *amount*)
Return first n items of the iterable as a tuple.

Taken from the Python documentation. Under the PSF license.

Parameters

- **amount** – The number of items to obtain.
- **iterable** – Iterable sequence.

Returns First n items of the iterable as a tuple.

mom.functional.**ncycles**(*iterable*, *times*)
Yields the sequence elements n times.

Taken from the Python documentation. Under the PSF license.

Parameters

- **iterable** – Iterable sequence.
- **times** – The number of times to yield the sequence.

Yields Iterator.

mom.functional.**occurrences**(*iterable*)
Returns a dictionary of counts (multiset) of each element in the iterable.

Taken from the Python documentation under PSF license.

Parameters **iterable** – Iterable sequence with hashable elements.

Returns A dictionary of counts of each element in the iterable.

Manipulation, filtering

mom.functional.**contains**(*iterable*, *item*)
Determines whether the iterable contains the value specified.

Parameters

- **iterable** – Iterable sequence.
- **item** – The value to find.

Returns `True` if the iterable sequence contains the value; `False` otherwise.

mom.functional.**omits**(*iterable*, *item*)
Determines whether the iterable omits the value specified.

Parameters

- **iterable** – Iterable sequence.
- **item** – The value to find.

Returns `True` if the iterable sequence omits the value; `False` otherwise.

mom.functional.**falsy**(*iterable*)
Returns a iterable with only the falsy values.

Example:

```
falsy((0, 1, 2, False, None, True)) -> (0, False, None)
```

Parameters **iterable** – Iterable sequence.

Returns Iterable with falsy values.

mom.functional.**ifalsy**(*iterable*)
Returns a iterator for an iterable with only the falsy values.

Example:

```
tuple(ifalsy((0, 1, 2, False, None, True))) -> (0, False, None)
```

Parameters **iterable** – Iterable sequence.

Yields Iterator for an iterable with falsy values.

mom.functional.**itruthy**(*iterable*)
Returns an iterator to for an iterable with only the truthy values.

Example:

```
tuple(itruthy((0, 1, 2, False, None, True))) -> (1, 2, True)
```

Parameters **iterable** – Iterable sequence.

Yields Iterator for an iterable with truthy values.

mom.functional.**truthy**(*iterable*)
Returns a iterable with only the truthy values.

Example:

```
truthy((0, 1, 2, False, None, True)) -> (1, 2, True)
```

Parameters **iterable** – Iterable sequence.

Returns Iterable with truthy values.

mom.functional.**without**(*iterable*, **values*)
Returns the iterable without the values specified.

Parameters

- **iterable** – Iterable sequence.
- **values** – Variable number of input values.

Returns Iterable sequence without the values specified.

Flattening, grouping, unions, differences, and intersections

mom.functional.**flatten**(*iterable*)
Flattens nested iterables into a single iterable.

Example:

```
flatten((1, (0, 5, ("a", "b")), (3, 4))) -> [1, 0, 5, "a", "b", 3, 4]
```

Parameters **iterable** – Iterable sequence of iterables.

Returns Iterable sequence of items.

mom.functional.**flatten1**(*iterable*)
Flattens nested iterables into a single iterable only one level deep.

Example:

```
flatten1((1, (0, 5, ("a", "b")), (3, 4))) -> [1, 0, 5, ("a", "b"), 3, 4]
```

Parameters **iterable** – Iterable sequence of iterables.

Returns Iterable sequence of items.

mom.functional.**group_consecutive**(*predicate*, *iterable*)
Groups consecutive elements into subsequences:

```
things = [("phone", "android"),
          ("phone", "iphone"),
          ("tablet", "ipad"),
          ("laptop", "dell studio"),
          ("phone", "nokia"),
          ("laptop", "macbook pro")]

list(group_consecutive(lambda w: w[0], things))
-> [(("phone", "android"), ("phone", "iphone")),
    (("tablet", "ipad"),),
    (("laptop", "dell studio"),),
    (("phone", "nokia"),),
    (("laptop", "macbook pro"),)]

list(group_consecutive(lambda w: w[0], "mississippi"))
-> [("m",), ("i",),
    ("s", "s"), ("i",),
    ("s", "s"), ("i",),
    ("p", "p"), ("i",)]
```

Parameters

- **predicate** – Predicate function that returns `True` or `False` for each element of the iterable.
- **iterable** – An iterable sequence of elements.

Returns An iterator of lists.

mom.functional.**flock**(*predicate*, *iterable*)
Groups elements into subsequences after sorting:

```
things = [("phone", "android"),
          ("phone", "iphone"),
          ("tablet", "ipad"),
          ("laptop", "dell studio"),
          ("phone", "nokia"),
          ("laptop", "macbook pro")]

list(flock(lambda w: w[0], things))
-> [(("laptop", "dell studio"), ("laptop", "macbook pro")),
    (("phone", "android"), ("phone", "iphone"), ("phone", "nokia")),
    (("tablet", "ipad"),)]

list(flock(lambda w: w[0], "mississippi"))
-> [("i", "i", "i", "i"), ("m",), ("p", "p"), ("s", "s", "s", "s")]
```

Parameters

- **predicate** – Predicate function that returns `True` or `False` for each element of the iterable.
- **iterable** – An iterable sequence of elements.

Returns An iterator of lists.

mom.functional.**intersection**(*iterable*, **iterables*)
Returns the intersection of given iterable sequences.

Parameters **iterables** – Variable number of input iterable sequences.

Returns Intersection of the iterable sequences in the order of appearance in the first sequence.

mom.functional.**idifference**(*iterable1*, *iterable2*)
Difference between one iterable and another. Items from the first iterable are included in the difference.

iterable1 - iterable2 = difference

Parameters

- **iterable1** – Iterable sequence.
- **iterable2** – Iterable sequence.

Yields Generator for the difference between the two given iterables.

mom.functional.**difference**(*iterable1*, *iterable2*)
Difference between one iterable and another. Items from the first iterable are included in the difference.

iterable1 - iterable2 = difference

For example, here is how to find out what your Python module exports to other modules using wildcard imports:

```
>> difference(dir(mom.functional), mom.functional.__all__)
["__all__",
 # Elided...
 "range",
 "takewhile"]
```

Parameters

- **iterable1** – Iterable sequence.
- **iterable2** – Iterable sequence.

Returns Iterable sequence containing the difference between the two given iterables.

mom.functional.**union**(*iterable*, **iterables*)
Returns the union of given iterable sequences.

Parameters **iterables** – Variable number of input iterable sequences.

Returns Union of the iterable sequences.

mom.functional.**unique**(*iterable*, *is_sorted=False*)
Returns an iterable sequence of unique values from the given iterable.

Parameters

- **iterable** – Iterable sequence.
- **is_sorted** – Whether the iterable has already been sorted. Works faster if it is.

Returns Iterable sequence of unique values.

Dictionaries and dictionary sequences

mom.functional.**invert_dict**(*dictionary*)
Inverts a dictionary.

Parameters **dictionary** – Dictionary to invert.

Returns New dictionary with the keys and values switched.

mom.functional.**ipluck**(*dicts*, *key*, **args*, ***kwargs*)
Plucks values for a given key from a series of dictionaries as an iterator.

Parameters

- **dicts** – Iterable sequence of dictionaries.
- **key** – The key to fetch.
- **default** – The default value to use when a key is not found. If this value is not specified, a KeyError will be raised when a key is not found.

Yields Iterator of values for the key.

mom.functional.**map_dict**(*transform*, *dictionary*)
Maps over a dictionary of key, value pairs.

Parameters **transform** – Function that accepts two arguments `key, value` and returns a `(new key, new value)` pair.

Returns New dictionary of `new key=new value` pairs.

mom.functional.**pluck**(*dicts*, *key*, **args*, ***kwargs*)
Plucks values for a given key from a series of dictionaries.

Parameters

- **dicts** – Iterable sequence of dictionaries.
- **key** – The key to fetch.
- **default** – The default value to use when a key is not found. If this value is not specified, a KeyError will be raised when a key is not found.

Returns Tuple of values for the key.

mom.functional.**reject_dict**(*predicate*, *dictionary*)
Reject from a dictionary.

Parameters **predicate** – Predicate function that accepts two arguments `key, value` and returns `True` for rejected elements.

Returns New dictionary of selected `key=value` pairs.

mom.functional.**select_dict**(*predicate*, *dictionary*)
Select from a dictionary.

Parameters **predicate** – Predicate function that accepts two arguments `key, value` and returns `True` for selectable elements.

Returns New dictionary of selected `key=value` pairs.

mom.functional.**partition_dict**(*predicate*, *dictionary*)
Partitions a dictionary into two dictionaries where for the elements of one dictionary the predicate is true and for those of the other it is false.

Parameters

- **predicate** – Function of the format:

```
f(key, value) -> bool
```

- **dictionary** – Dictionary.

Returns Tuple (selected_dict, rejected_dict)

Predicates, transforms, and walkers

mom.functional.**identity**(*arg*)
Identity function. Produces what it consumes.

Parameters **arg** – Argument

Returns Argument.

mom.functional.**loob**(*arg*)
Complement of bool.

Parameters **arg** – Python value.

Returns Complementary boolean value.

mom.functional.**always**(*_*)
Predicate function that returns `True` always.

Parameters **_** – Argument

Returns `True`.

mom.functional.**never**(*_*)
Predicate function that returns `False` always.

Parameters **_** – Argument

Returns `False`.

synopsis Implements `itertools` for older versions of Python.

module mom.itertools

copyright 2010-2011 by Daniel Neuhäuser

license BSD, PSF

Borrowed from brownie.itools.

synopsis Math routines.

module mom.math

Math

mom.math.**gcd**(*num_a*, *num_b*)
Calculates the greatest common divisor.

Non-recursive fast implementation.

Parameters

- **num_a** – Long value.
- **num_b** – Long value.

Returns Greatest common divisor.

mom.math.**inverse_mod**(*num_a*, *num_b*)
Returns inverse of a mod b, zero if none

Uses Extended Euclidean Algorithm

Parameters

- **num_a** – Long value
- **num_b** – Long value

Returns Inverse of a mod b, zero if none.

mom.math.**lcm**(*num_a*, *num_b*)
Least common multiple.

Parameters

- **num_a** – Integer value.
- **num_b** – Integer value.

Returns Least common multiple.

mom.math.**pow_mod**(*base*, *power*, *modulus*)

Calculates: base**pow mod modulus

Uses multi bit scanning with nBitScan bits at a time. From Bryan G. Olson's post to comp.lang.python

Does left-to-right instead of pow()'s right-to-left, thus about 30% faster than the python built-in with small bases

Parameters

- **base** – Base
- **power** – Power
- **modulus** – Modulus

Returns base**pow mod modulus

Primes

mom.math.**generate_random_prime**(*bits*)
Generates a random prime number.

Parameters **bits** – Number of bits.

Returns Prime number long value.

mom.math.**generate_random_safe_prime**(*bits*)
Unused at the moment.

Generates a random prime number.

Parameters **bits** – Number of bits.

Returns Prime number long value.

mom.math.**is_prime**(*num*, *iterations=5*, *sieve=sieve*)
Determines whether a number is prime.

Parameters

- **num** – Number
- **iterations** – Number of iterations.

Returns `True` if prime; `False` otherwise.

synopsis MIME-Type Parser.

module mom.mimeparse

This module provides basic functions for handling mime-types. It can handle matching mime-types against a list of media-ranges. See section 14.1 of the HTTP specification [RFC 2616] for a complete explanation.

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.1

Contents

mom.mimeparse.**parse_mime_type**(*mime_type*)
Parses a mime-type into its component parts.

Parameters **mime_type** – Mime type as a byte string.

Returns

A tuple of the (type, subtype, params) where 'params' is a dictionary of all the parameters for the media range. For example, the media range b'application/xhtml;q=0.5' would get parsed into:

(b'application', b'xhtml', {'q': b'0.5'})

mom.mimeparse.**parse_media_range**(*media_range*)
Parse a media-range into its component parts.

Parameters **media_range** – Media range as a byte string.

Returns

A tuple of the (type, subtype, params) where 'params' is a dictionary of all the parameters for the media range. For example, the media range b'application/xhtml;q=0.5' would get parsed into:

(b'application', b'xhtml', {'q': b'0.5'})

In addition this function also guarantees that there is a value for 'q' in the params dictionary, filling it in with a proper default if necessary.

mom.mimeparse.**quality**(*mime_type*, *ranges*)
Return the quality ('q') of a mime-type against a list of media-ranges.

For example:

```
>>> Quality(b'text/html',b'text/*;q=0.3, text/html;q=0.7,
              text/html;level=1, text/html;level=2;q=0.4, */*;q=0.5')
0.7
```

Parameters

- **mime_type** – The mime-type to compare.
- **ranges** – A list of media ranges.

Returns Returns the quality 'q' of a mime-type when compared against the media-ranges in ranges.

mom.mimeparse.**quality_parsed**(*mime_type*, *parsed_ranges*)
Find the best match for a mime-type amongst parsed media-ranges.

Parameters

- **mime_type** – The mime-type to compare.
- **parsed_ranges** – A list of media ranges that have already been parsed by parsed_media_range().

Returns The 'q' quality parameter of the best match, 0 if no match was found.

mom.mimeparse.**best_match**(*supported*, *header*)
Return mime-type with the highest quality ('q') from list of candidates.

Takes a list of supported mime-types and finds the best match for all the media-ranges listed in header.

```
>>> BestMatch([b'application/xbel+xml', b'text/xml'],
               b'text/*;q=0.5,*/*; q=0.1')
b'text/xml'
```

Parameters

- **supported** – A list of supported mime-types. The list of supported mime-types should be sorted in order of increasing desirability, in case of a situation where there is a tie.
- **header** – Atring that conforms to the format of the HTTP Accept: header.

Returns Mime-type with the highest quality ('q') from the list of candidates.

synopsis string module compatibility.

module mom.string

3.2 Codecs

3.2.1 *mom.codec*

synopsis Many different types of common encode/decode function.

module mom.codec

This module contains codecs for converting between hex, base64, base85, base58, base62, base36, decimal, and binary representations of bytes.

Understand that bytes are simply base-256 representation. A PNG file:

```
\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x00
\x05\x00\x00\x00\x05\x08\x06\x00\x00\x00\x8do&
\xe5\x00\x00\x00\x1cIDAT\x08\xd7c\xf8\xff\xff?
\xc3\x7f\x06 \x05\xc3 \x12\x84\xd01\xf1\x82X\xcd
\x04\x00\x0e\xf55\xcb\xd1\x8e\x0e\x1f\x00\x00\x00
\x00IEND\xaeB`\x82
```

That is what an example PNG file looks like as a stream of bytes (base-256) in Python (with line-breaks added for visual-clarity).

If we wanted to send this PNG within an email message, which is restricted to ASCII characters, we cannot simply add these bytes in and hope they go through unchanged. The receiver at the other end expects to get a copy of exactly the same bytes that you send. Because we are limited to using ASCII characters, we need to "encode" this binary data into a subset of ASCII characters before transmitting, and the receiver needs to "decode" those ASCII characters back into binary data before attempting to display it.

> **Base-encoding raw bytes into ASCII characters is used to safely transmit binary data through any medium that does not inherently support non-ASCII data.**

Therefore, we need to convert the above PNG binary data into something that looks like (again, line-breaks have been added for visual clarity only):

```
iVBORw0KGgoAAAANSUhEUgAAAAUAAAAFCAYAAACNbyblAAAAHElEQVQI
12P4//8/w38GIAXDIBKE0DHxgljNBAAO9TXL0Y4OHwAAAABJRU5ErkJg
gg==
```

The base-encoding method that we can use is limited by these criteria:

1. The number of ASCII characters, a subset of ASCII, that we can use to represent binary data (case-sensitivity, ambiguity, base, deviation from standard characters, etc.)
2. Whether human beings are involved in the transmission of data. Ergo, visual clarity, legibility, readability, human-inputability, and even *double-click-to-select-ability*! *(Hint: try double-clicking the encoded data above to see whether it selects all of it–it won't).* This is a corollary for point 1.
3. Whether we want the process to be more time-efficient or space-efficient. That is, whether we can process binary data in chunks or whether we need to convert it into an arbitrarily large integer before encoding, respectively.

Terminology

Answer this question:

> How many **times** should I multiply 2 by itself to obtain 8?

You say:

> That's a dumb question. 3 times!

Well, congratulations! You have just re-discovered **logarithms**. In a system of equations, we may have unknowns. Given an equation with 3 parts, 2 of which are known, we often need to find the 3rd. Logarithms are used when you know the base (radix) and the number, but not the exponent.

> **Logarithms help you find exponents.**

Take for example:

```
2**0 = 1
2**1 = 2
2**2 = 4
2**3 = 8
2**4 = 16
2**5 = 32
2**6 = 64
```

Alternatively, logarithms can be thought of as answering the question:

> **Raising 2 to which exponent gets me 64?**

This is the same as doing:

```
import math
math.log(64, 2)     # 6.0; number 64, base 2.
6.0
```

read as "logarithm to the base 2 of 64" which gives 6. That is, if we raise 2 to the power 6, we get 64.

The concept of **roots** or radicals is also related. Roots help you find the base (radix) given the exponent and the number. So:

```
root(8, 3)    # 2.0; cube root. exponent 3, number 8.
```

> **Roots help you find the base.**

Hopefully, that brings you up to speed and enables you to clearly **see the relationship between powers, logarithms, and roots**.

We will often refer to the term **byte** and mean it to be an octet (8) of bits. **The number of bits in a byte is dependent on the processor architecture.** Therefore, we *can* have a 9-bit byte or even a 36-bit byte.

> **For our purposes, however, a byte means a chunk of 8 bits–that is, an octet.**

By the term "**encoding**," throughout this discussion, we mean **a way of representing a sequence of bytes in terms of a subset of US-ASCII characters, each of which uses 7 bits**. This ensures that in communication and messaging that involves the transmission of binary data, at a small increase in encoded size, we can *safely* transmit this data encoded as ASCII text. We could be pedantic and use the phrase "ASCII-subset-based encoding" everywhere, but we'll simply refer to it as "encoding" instead.

How it applies to encodings

Byte, or base-256, representation allows each byte to be represented using one of 256 values (0-255 inclusive). Modern processors can process data in chunks of 32 bits (4 bytes), 64 bits (8 bytes), and so on. Notice that these are powers of 2 given that our processors are binary machines.

We could feed a 64-bit processor with 8 bits of data at a time, but that would guarantee that the codec will be only 1/8th as time-efficient as it can be. That is, if you feed the same 64-bit processor with 64 bits of data at a time instead, the encoding process will be 8 times as fast. Whoa!

Therefore, in order to ensure that our codecs are fast, we need to feed our processors data in chunks to be more time-efficient. The two types of encoding we discuss here are:

1. big-integer-based polynomial-time base-conversions
2. chunked linear-time base-conversions.

> **These two types of encoding are not always compatible with each other.**

Big-integer based encoding

This method of encoding is generally costlier because the raw bytes (base-256 representation) are first converted into a big integer, which is then subsequently repeatedly divided to obtain an encoded sequence of bytes. Bases 58, 60, and 62 are not powers of 2, and therefore cannot be reliably or efficiently encoded in chunks of powers of 2 (used by microprocessors) so as to produce the same encoded representations as their big integer encoded representations. Therefore, using these encodings for a large amount of binary data is not advised. The base-58 and base-62 modules in this library are meant to be used with small amounts of binary data.

Chunked encoding

Base encoding a chunk of 4 bytes at a time (32 bits at a time) means we would need a way to represent each of the 256**4 (4294967296) values with our encoding:

```
256**4 # 4294967296
2**32  # 4294967296
```

Given an encoding alphabet of 85 ASCII characters, for example, we need to find an exponent (logarithm) that allows us to represent each one of these 4294967296 values:

```
85**4 # 52200625
85**5 # 4437053125

>>> 85**5 >= 2**32
True
```

Done using logarithms:

```
import math
math.log(2**32, 85)    # 4.9926740807111996
```

Therefore, we would need 5 characters from this encoding alphabet to represent 4 bytes. Since 85 is not a power of 2, there is going to be a little wastage of space and the codec will need to deal with padding and de-padding bytes to ensure the resulting size to be a multiple of the chunk size, but the byte sequence will be more compact than its base-16 (hexadecimal) representation, for example:

```
import math
math.log(2**32, 16)    # 8.0
```

As you can see, if we used hexadecimal representation instead, each 4-byte chunk would be represented using 8 characters from the encoding alphabet. This is clearly less space-efficient than using 5 characters per 4 bytes of binary data.

Base-64 as another example

Base-64 allows us to represent 256**4 (4294967296) values using 64 ASCII characters.

Bytes base-encoding

These codecs preserve bytes "as is" when decoding back to bytes. In a more mathematical sense,

> `g(f(x))` is an **identity function**

where `g` is the decoder and `f` is the encoder.

Why have we reproduced base64 encoding/decoding functions here when the standard library has them? Well, those functions behave differently in Python 2.x and Python 3.x. The Python 3.x equivalents do not accept Unicode strings as their arguments, whereas the Python 2.x versions would happily encode your Unicode strings without warning you- you know that you are supposed to encode them to UTF-8 or another byte encoding before you base64-encode them right? These wrappers are re-implemented so that you do not make these mistakes. Use them. They will help prevent unexpected bugs.

`mom.codec.`**`base85_encode`**(*raw_bytes*, *charset='ASCII85'*)

Encodes raw bytes into ASCII85 representation.

Encode your Unicode strings to a byte encoding before base85-encoding them.

Parameters

- **raw_bytes** – Bytes to encode.
- **charset** – "ASCII85" (default) or "RFC1924".

Returns ASCII85 encoded string.

mom.codec.**base85_decode**(*encoded*, *charset='ASCII85'*)
Decodes ASCII85-encoded bytes into raw bytes.

Parameters

- **encoded** – ASCII85 encoded representation.
- **charset** – "ASCII85" (default) or "RFC1924".

Returns Raw bytes.

mom.codec.**base64_encode**(*raw_bytes*)
Encodes raw bytes into base64 representation without appending a trailing newline character. Not URL-safe.

Encode your Unicode strings to a byte encoding before base64-encoding them.

Parameters **raw_bytes** – Bytes to encode.

Returns Base64 encoded bytes without newline characters.

mom.codec.**base64_decode**(*encoded*)
Decodes base64-encoded bytes into raw bytes. Not URL-safe.

Parameters **encoded** – Base-64 encoded representation.

Returns Raw bytes.

mom.codec.**base64_urlsafe_encode**(*raw_bytes*)
Encodes raw bytes into URL-safe base64 bytes.

Encode your Unicode strings to a byte encoding before base64-encoding them.

Parameters **raw_bytes** – Bytes to encode.

Returns Base64 encoded string without newline characters.

mom.codec.**base64_urlsafe_decode**(*encoded*)
Decodes URL-safe base64-encoded bytes into raw bytes.

Parameters **encoded** – Base-64 encoded representation.

Returns Raw bytes.

mom.codec.**base62_encode**(*raw_bytes*)
Encodes raw bytes into base-62 representation. URL-safe and human safe.

Encode your Unicode strings to a byte encoding before base-62-encoding them.

Convenience wrapper for consistency.

Parameters **raw_bytes** – Bytes to encode.

Returns Base-62 encoded bytes.

mom.codec.**base62_decode**(*encoded*)
Decodes base-62-encoded bytes into raw bytes.

Convenience wrapper for consistency.

Parameters **encoded** – Base-62 encoded bytes.

Returns Raw bytes.

mom.codec.**base58_encode**(*raw_bytes*)
Encodes raw bytes into base-58 representation. URL-safe and human safe.

Encode your Unicode strings to a byte encoding before base-58-encoding them.

Convenience wrapper for consistency.

Parameters **raw_bytes** – Bytes to encode.

Returns Base-58 encoded bytes.

mom.codec.**base58_decode**(*encoded*)
Decodes base-58-encoded bytes into raw bytes.

Convenience wrapper for consistency.

Parameters **encoded** – Base-58 encoded bytes.

Returns Raw bytes.

mom.codec.**base36_encode**(*raw_bytes*)
Encodes raw bytes into base-36 representation.

Encode your Unicode strings to a byte encoding before base-58-encoding them.

Convenience wrapper for consistency.

Parameters **raw_bytes** – Bytes to encode.

Returns Base-36 encoded bytes.

mom.codec.**base36_decode**(*encoded*)
Decodes base-36-encoded bytes into raw bytes.

Convenience wrapper for consistency.

Parameters **encoded** – Base-36 encoded bytes.

Returns Raw bytes.

mom.codec.**hex_encode**(*raw_bytes*)
Encodes raw bytes into hexadecimal representation.

Encode your Unicode strings to a byte encoding before hex-encoding them.

Parameters **raw_bytes** – Bytes.

Returns Hex-encoded representation.

mom.codec.**hex_decode**(*encoded*)
Decodes hexadecimal-encoded bytes into raw bytes.

Parameters **encoded** – Hex representation.

Returns Raw bytes.

mom.codec.**decimal_encode**(*raw_bytes*)
Encodes raw bytes into decimal representation. Leading zero bytes are preserved.

Encode your Unicode strings to a byte encoding before decimal-encoding them.

Parameters **raw_bytes** – Bytes.

Returns Decimal-encoded representation.

mom.codec.**decimal_decode**(*encoded*)
Decodes decimal-encoded bytes to raw bytes. Leading zeros are converted to leading zero bytes.

Parameters **encoded** – Decimal-encoded representation.

Returns Raw bytes.

mom.codec.**bin_encode**(*raw_bytes*)
Encodes raw bytes into binary representation.

Encode your Unicode strings to a byte encoding before binary-encoding them.

Parameters **raw_bytes** – Raw bytes.

Returns Binary representation.

mom.codec.**bin_decode**(*encoded*)
Decodes binary-encoded bytes into raw bytes.

Parameters **encoded** – Binary representation.

Returns Raw bytes.

synopsis ASCII-85 and RFC1924 Base85 encoding and decoding functions.

module mom.codec.base85

see http://en.wikipedia.org/wiki/Ascii85

see http://tools.ietf.org/html/rfc1924

see http://www.piclist.com/techref/method/encode.htm

Where should you use base85?

Base85-encoding is used to compactly represent binary data in 7-bit ASCII. It is, therefore, 7-bit MIME-safe but not safe to use in URLs, SGML, HTTP cookies, and other similar places. Example scenarios where Base85 encoding can be put to use are Adobe PDF documents, Adobe PostScript format, binary diffs (patches), efficiently storing RSA keys, etc.

The ASCII85 character set-based encoding is mostly used by Adobe PDF and PostScript formats. It may also be used to store RSA keys or binary data with a lot of zero byte sequences. The RFC1924 character set-based encoding, however, may be used to compactly represent 128-bit unsigned integers (like IPv6 addresses) or binary diffs. Encoding based on RFC1924 does not compact zero byte sequences, so this form of encoding is less space-efficient than the ASCII85 version which compacts redundant zero byte sequences.

About base85 and this implementation

Base-85 represents 4 bytes as 5 ASCII characters. This is a 7% improvement over base-64, which translates to a size increase of ~25% over plain binary data for base-85 versus that of ~37% for base-64.

However, because the base64 encoding routines in Python are implemented in C, base-64 may be less expensive to compute. This implementation of base-85 uses a lot of tricks to reduce computation time and is hence generally faster than many other implementations. If computation speed is a concern for you, please contribute a C implementation or wait for one.

Functions

mom.codec.base85.**b85encode**(*raw_bytes, prefix=None, suffix=None, _base85_bytes=array('B', [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117]), _padding=False, _compact_zero=True, _compact_char='z'*)

ASCII-85 encodes a sequence of raw bytes.

The character set in use is:

```
ASCII 33 ("!") to ASCII 117 ("u")
```

If the number of raw bytes is not divisible by 4, the byte sequence is padded with up to 3 null bytes before encoding. After encoding, as many bytes as were added as padding are removed from the end of the encoded sequence if `padding` is `False` (default).

Encodes a zero-group () as "z" instead of "!!!!!".

The resulting encoded ASCII string is *not URL-safe* nor is it safe to include within SGML/XML/HTML documents. You will need to escape special characters if you decide to include such an encoded string within these documents.

Parameters

- **raw_bytes** – Raw bytes.
- **prefix** – The prefix used by the encoded text. None by default.
- **suffix** – The suffix used by the encoded text. None by default.
- **_base85_bytes** – (Internal) Character set to use.
- **_compact_zero** – (Internal) Encodes a zero-group () as "z" instead of "!!!!!" if this is `True` (default).
- **_compact_char** – (Internal) Character used to represent compact groups ("z" default)

Returns ASCII-85 encoded bytes.

mom.codec.base85.**b85decode**(*encoded, prefix=None, suffix=None, _base85_bytes=array('B', [33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117]), _base85_ords={'!': 0, '#': 2, '"': 1, '%': 4, '$': 3, '"': 6, '&': 5, ')': 8, '(': 7, '+': 10, '*': 9, '-': 12, ',': 11, '/': 14, '.': 13, '1': 16, '0': 15, '3': 18, '2': 17, '5': 20, '4': 19, '7': 22, '6': 21, '9': 24, '8': 23, ';': 26, ':': 25, '=': 28, '<': 27, '?': 30, '>': 29, 'A': 32, '@': 31, 'C': 34, 'B': 33, 'E': 36, 'D': 35, 'G': 38, 'F': 37, 'I': 40, 'H': 39, 'K': 42, 'J': 41, 'M': 44, 'L': 43, 'O': 46, 'N': 45, 'Q': 48, 'P': 47, 'S': 50, 'R': 49, 'U': 52, 'T': 51, 'W': 54, 'V': 53, 'Y': 56, 'X': 55, '[': 58, 'Z': 57, ']': 60, '\\': 59, '_': 62, '^': 61, 'a': 64, '`': 63, 'c': 66, 'b': 65, 'e': 68, 'd': 67, 'g': 70, 'f': 69, 'i': 72, 'h': 71, 'k': 74, 'j': 73, 'm': 76, 'l': 75, 'o': 78, 'n': 77, 'q': 80, 'p': 79, 's': 82, 'r': 81, 'u': 84, 't': 83}, _uncompact_zero=True, _compact_char='z'*)

Decodes an ASCII85-encoded string into raw bytes.

Parameters

- **encoded** – Encoded ASCII string.
- **prefix** – The prefix used by the encoded text. None by default.
- **suffix** – The suffix used by the encoded text. None by default.
- **_base85_bytes** – (Internal) Character set to use.
- **_base85_ords** – (Internal) A function to convert a base85 character to its ordinal value. You should not need to use this.
- **_uncompact_zero** – (Internal) Treats "z" (a zero-group ()) as a "!!!!!" if `True` (default).
- **_compact_char** – (Internal) Character used to represent compact groups ("z" default)

Returns ASCII85-decoded raw bytes.

mom.codec.base85.**rfc1924_b85encode**(*raw_bytes*, *_padding=False*)

Base85 encodes using the RFC1924 character set.

The character set is:

```
0-9, A-Z, a-z, and then !#$%&()*+-;<=>?@^_`{|}~
```

These characters are specifically not included:

```
"',./:[]\
```

This is the encoding method used by Mercurial (and git?) to generate binary diffs, for example. They chose the IPv6 character set and encode using the ASCII85 encoding method while not compacting zero-byte sequences.

See http://tools.ietf.org/html/rfc1924

Parameters

- **raw_bytes** – Raw bytes.
- **_padding** – (Internal) Whether padding should be included in the encoded output. (Default `False`, which is usually what you want.)

Returns RFC1924 base85 encoded string.

mom.codec.base85.**rfc1924_b85decode**(*encoded*)

Base85 decodes using the RFC1924 character set.

This is the encoding method used by Mercurial (and git) to generate binary diffs, for example. They chose the IPv6 character set and encode using the ASCII85 encoding method while not compacting zero-byte sequences.

See http://tools.ietf.org/html/rfc1924

Parameters **encoded** – RFC1924 Base85 encoded string.

Returns Decoded bytes.

mom.codec.base85.**ipv6_b85encode**(*uint128*, *_base85_bytes=array('B', [48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 33, 35, 36, 37, 38, 40, 41, 42, 43, 45, 59, 60, 61, 62, 63, 64, 94, 95, 96, 123, 124, 125, 126])*)

Encodes a 128-bit unsigned integer using the RFC 1924 base-85 encoding. Used to encode IPv6 addresses or 128-bit chunks.

Parameters

- **uint128** – A 128-bit unsigned integer to be encoded.
- **_base85_bytes** – (Internal) Base85 encoding charset lookup table.

Returns RFC1924 Base85-encoded string.

mom.codec.base85.**ipv6_b85decode**(*encoded*, *_base85_ords={'!': 62, '#': 63, '%': 65, '$': 64, '&': 66, ')': 68, '(': 67, '+': 70, '*': 69, '-': 71, '1': 1, '0': 0, '3': 3, '2': 2, '5': 5, '4': 4, '7': 7, '6': 6, '9': 9, '8': 8, ';': 72, '=': 74, '<': 73, '?': 76, '>': 75, 'A': 10, '@': 77, 'C': 12, 'B': 11, 'E': 14, 'D': 13, 'G': 16, 'F': 15, 'I': 18, 'H': 17, 'K': 20, 'J': 19, 'M': 22, 'L': 21, 'O': 24, 'N': 23, 'Q': 26, 'P': 25, 'S': 28, 'R': 27, 'U': 30, 'T': 29, 'W': 32, 'V': 31, 'Y': 34, 'X': 33, 'Z': 35, '_': 79, '^': 78, 'a': 36, '`': 80, 'c': 38, 'b': 37, 'e': 40, 'd': 39, 'g': 42, 'f': 41, 'i': 44, 'h': 43, 'k': 46, 'j': 45, 'm': 48, 'l': 47, 'o': 50, 'n': 49, 'q': 52, 'p': 51, 's': 54, 'r': 53, 'u': 56, 't': 55, 'w': 58, 'v': 57, 'y': 60, 'x': 59, '{': 81, 'z': 61, '}': 83, '|': 82, '~': 84}*)

Decodes an RFC1924 Base-85 encoded string to its 128-bit unsigned integral representation. Used to base85-decode IPv6 addresses or 128-bit chunks.

Whitespace is ignored. Raises an `OverflowError` if stray characters are found.

Parameters

- **encoded** – RFC1924 Base85-encoded string.
- **_base85_ords** – (Internal) Look up table.

Returns A 128-bit unsigned integer.

synopsis Base-62 7-bit ASCII-safe representation for compact human-input.

module mom.codec.base62

Where should you use base-62?

Base-62 representation is 7 bit-ASCII safe, MIME-safe, URL-safe, HTTP cookie-safe, and almost **human being-safe**. Base-62 representation can:

- be readable and editable by a human being;
- safely and compactly represent numbers;
- contain only alphanumeric characters;
- not contain punctuation characters.

For examples of places where you can use base-62, see the documentation for *mom.codec.base58*.

In general, use base-62 in any 7-bit ASCII-safe compact communication where human beings and communication devices may be significantly involved.

When to prefer base-62 over base-58?

When you don't care about the visual ambiguity between these characters:

- 0 (ASCII NUMERAL ZERO)
- O (ASCII UPPERCASE ALPHABET O)

- I (ASCII UPPERCASE ALPHABET I)
- l (ASCII LOWERCASE ALPHABET L)

A practical example (versioned static asset URLs):

In order to reduce the number of HTTP requests for static assets sent to a Web server, developers often include a hash of the asset being served into the URL and set the expiration time of the asset to a very long period (say, 365 days).

This enables an almost perfect form of client-side asset caching while still serving fresh content when it changes. To minimize the size overhead introduced into the URL by such hashed-identifiers, the identifiers themselves can be shortened using base-58 or base-62 encoding. For example:

```
$ sha1sum file.js
a497f210fc9c5d02fc7dc7bd211cb0c74da0ae16
```

The asset URL for this file can be:

```
http://s.example.com/js/a497f210fc9c5d02fc7dc7bd211cb0c74da0ae16/file.js
```

where `example.com` is a canonical domain used only for informational purposes. However, the hashed-identifier in the URL is long but can be reduced using base-62 to:

```
# Base-58
http://s.example.com/js/3HzsRcRETLZ3qFgDzG1QE7CJJNeh/file.js

# Base-62
http://s.example.com/js/NU3qW1G4teZJynubDFZnbzeOUFS/file.js
```

The first 12 characters of a SHA-1 hash are sufficiently strong for serving static assets while minimizing collision overhead in the context of a small-to-medium-sized Website and considering these are URLs for static served assets that can change over periods of time. You may want to consider using the full hash for large-scale Websites. Therefore, we can shorten the original asset URL to:

```
http://s.example.com/js/a497f210fc9c/file.js
```

which can then be reduced utilizing base-58 or base-62 encoding to:

```
# Base-58
http://s.example.com/js/2QxqmqiFm/file.js

# Base-62
http://s.example.com/js/pO7arZWO/file.js
```

These are a much shorter URLs than the original. Notice that we have not renamed the file `file.js` to `2QxqmqiFm.js` or `pO7arZWO.js` because that would cause an unnecessary explosion of files on the server as new files would be generated every time the source files change. Instead, we have chosen to make use of Web server URL-rewriting rules to strip the hashed identifier and serve the file fresh as it is on the server file system. These are therefore **non-versioned assets**–only the URLs that point at them are versioned. That is if you took a diff between the files that these URLs point at:

```
http://s.example.com/js/pO7arZWO/file.js
http://s.example.com/js/2qiFqxEm/file.js
```

you would *not* see a difference. Only the URLs differ to trick the browser into caching as well as it can.

The hashed-identifier is not part of the query string for this asset URL because certain proxies do not cache files served from URLs that include query strings. That is, we are **not** doing this:

```
# Base-58 -- Don't do this. Not all proxies will cache it.
http://s.example.com/js/file.js?v=2QxqmqiFm

# Base-62 -- Don't do this. Not all proxies will cache it.
http://s.example.com/js/file.js?v=pO7arZWO
```

If you wish to support **versioned assets**, however, you may need to rename files to include their hashed identifiers and avoid URL-rewriting instead. For example:

```
# Base-58
http://s.example.com/js/file-2QxqmqiFm.js

# Base-62
http://s.example.com/js/file-pO7arZWO.js
```

Note: Do note that the base-58 encoded version of the SHA-1 hash (40 characters in hexadecimal representation) may have a length of either 27 or 28. Similarly, for the SHA-1 hash (40 characters in hex), the base62-encoded version may have a length of either 26 or 27.

Therefore, please ensure that your rewriting rules take variable length into account.

The following benefits are therefore achieved:

- Client-side caching is fully utilized
- The number of HTTP requests sent to Web servers by clients is reduced.
- When assets change, so do their SHA-1 hashed identifiers, and hence their asset URLs.
- Shorter URLs also implies that fewer bytes are transferred in HTTP responses.
- Bandwidth consumption is reduced by a noticeably large factor.
- Multiple versions of assets (if required).

Essentially, URLs shortened using base-58 or base-62 encoding can result in a faster Web-browsing experience for end-users.

Functions

mom.codec.base62.**b62encode**(*raw_bytes*, *base_bytes='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmno*
_padding=True)

Base62 encodes a sequence of raw bytes. Zero-byte sequences are preserved by default.

Parameters

- **raw_bytes** – Raw bytes to encode.
- **base_bytes** – (Internal) The character set to use. Defaults to `ASCII62_BYTES` that uses natural ASCII order.
- **_padding** – (Internal) `True` (default) to include prefixed zero-byte sequence padding converted to appropriate representation.

Returns Base-62 encoded bytes.

mom.codec.base62.**b62decode**(*encoded*, *base_bytes='0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopq*
base_ords={'1': 1, '0': 0, '3': 3, '2': 2, '5': 5, '4': 4, '7': 7, '6': 6, '9': 9, '8': 8, 'A': 10, 'C': 12, 'B': 11, 'E': 14, 'D': 13, 'G': 16, 'F': 15, 'I': 18, 'H': 17, 'K': 20, 'J': 19, 'M': 22, 'L': 21, 'O': 24, 'N': 23, 'Q': 26, 'P': 25, 'S': 28, 'R': 27, 'U': 30, 'T': 29, 'W': 32, 'V': 31, 'Y': 34, 'X': 33, 'Z': 35, 'a': 36, 'c': 38, 'b': 37, 'e': 40, 'd': 39, 'g': 42, 'f': 41, 'i': 44, 'h': 43, 'k': 46, 'j': 45, 'm': 48, 'l': 47, 'o': 50, 'n': 49, 'q': 52, 'p': 51, 's': 54, 'r': 53, 'u': 56, 't': 55, 'w': 58, 'v': 57, 'y': 60, 'x': 59, 'z': 61})

Base-62 decodes a sequence of bytes into raw bytes. Whitespace is ignored.

Parameters

- **encoded** – Base-62 encoded bytes.
- **base_bytes** – (Internal) The character set to use. Defaults to ASCII62_BYTES that uses natural ASCII order.
- **base_ords** – (Internal) Ordinal-to-character lookup table for the specified character set.

Returns Raw bytes.

synopsis Base-58 repr for unambiguous display & compact human-input.

module mom.codec.base58

Where should you use base-58?

Base-58 representation is 7 bit-ASCII safe, MIME-safe, URL-safe, HTTP cookie-safe, and **human being-safe**. Base-58 representation can:

- be readable and editable by a human being;
- safely and compactly represent numbers;
- contain only alphanumeric characters (omitting a few with visually- ambiguously glyphs–namely, "0OIl");
- not contain punctuation characters.

Example scenarios where base-58 encoding may be used:

- Visually-legible account numbers
- Shortened URL paths
- OAuth verification codes
- Unambiguously printable and displayable key codes (for example, net-banking PINs, verification codes sent via SMS, etc.)
- Bitcoin decentralized crypto-currency addresses
- CAPTCHAs
- Revision control changeset identifiers
- Encoding email addresses compactly into JavaScript that decodes by itself to display on Web pages in order to reduce spam by stopping email harvesters from scraping email addresses from Web pages.

In general, use base-58 in any 7-bit ASCII-safe compact communication where human beings, paper, and communication devices may be significantly involved.

The default base-58 character set is [0-9A-Za-z] (base-62) with some characters omitted to make them visually-legible and unambiguously printable. The characters omitted are:

- 0 (ASCII NUMERAL ZERO)
- O (ASCII UPPERCASE ALPHABET O)
- I (ASCII UPPERCASE ALPHABET I)
- l (ASCII LOWERCASE ALPHABET L)

For a practical example, see the documentation for *mom.codec.base62*.

Functions

mom.codec.base58.**b58encode**(*raw_bytes*, *base_bytes='123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrs*
_padding=True)

Base58 encodes a sequence of raw bytes. Zero-byte sequences are preserved by default.

Parameters

- **raw_bytes** – Raw bytes to encode.
- **base_bytes** – The character set to use. Defaults to ASCII58_BYTES that uses natural ASCII order.
- **_padding** – (Internal) True (default) to include prefixed zero-byte sequence padding converted to appropriate representation.

Returns Base-58 encoded bytes.

mom.codec.base58.**b58decode**(*encoded*, *base_bytes='123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstu*
base_ords={'1': 0, '3': 2, '2': 1, '5': 4, '4': 3, '7': 6, '6': 5, '9': 8, '8': 7, 'A': 9, 'C': 11, 'B': 10, 'E': 13, 'D': 12, 'G': 15, 'F': 14, 'H': 16, 'K': 18, 'J': 17, 'M': 20, 'L': 19, 'N': 21, 'Q': 23, 'P': 22, 'S': 25, 'R': 24, 'U': 27, 'T': 26, 'W': 29, 'V': 28, 'Y': 31, 'X': 30, 'Z': 32, 'a': 33, 'c': 35, 'b': 34, 'e': 37, 'd': 36, 'g': 39, 'f': 38, 'i': 41, 'h': 40, 'k': 43, 'j': 42, 'm': 44, 'o': 46, 'n': 45, 'q': 48, 'p': 47, 's': 50, 'r': 49, 'u': 52, 't': 51, 'w': 54, 'v': 53, 'y': 56, 'x': 55, 'z': 57})

Base-58 decodes a sequence of bytes into raw bytes. Whitespace is ignored.

Parameters

- **encoded** – Base-58 encoded bytes.
- **base_bytes** – (Internal) The character set to use. Defaults to ASCII58_BYTES that uses natural ASCII order.
- **base_ords** – (Internal) Ordinal-to-character lookup table for the specified character set.

Returns Raw bytes.

synopsis Routines for converting between integers and bytes.

module mom.codec.integer

Number-bytes conversion

These codecs are "lossy" as they don't preserve prefixed padding zero bytes. In a more mathematical sense,

`g(f(x))` is **almost** an identity function, but not exactly.

where `g` is the decoder and `f` is a encoder.

mom.codec.integer.**bytes_to_uint**(*raw_bytes*)
Converts a series of bytes into an unsigned integer.

Parameters **raw_bytes** – Raw bytes (base-256 representation).

Returns Unsigned integer.

mom.codec.integer.**uint_to_bytes**(*number*, *fill_size=0*, *chunk_size=0*, *overflow=False*)
Convert an unsigned integer to bytes (base-256 representation).

Leading zeros are not preserved for positive integers unless a chunk size or a fill size is specified. A single zero byte is returned if the number is 0 and no padding is specified.

When a chunk size or a fill size is specified, the resulting bytes are prefix-padded with zero bytes to satisfy the size. The total size of the number in bytes is either the fill size or an integral multiple of the chunk size.

Parameters

- **number** – Integer value
- **fill_size** – The maxmimum number of bytes with which to represent the integer. Prefix zero padding is added as necessary to satisfy the size. If the number of bytes needed to represent the integer is greater than the fill size, an `OverflowError` is raised. To suppress this error and allow overflow, you may set the `overfloww` argument to this function to `True`.
- **chunk_size** – If optional chunk size is given and greater than zero, the resulting sequence of bytes is prefix-padded with zero bytes so that the total number of bytes is a multiple of `chunk_size`.
- **overflow** – `False` (default). If this is `True`, no `OverflowError` will be raised when the fill_size is shorter than the length of the generated byte sequence. Instead the byte sequence will be returned as is.

Returns Raw bytes (base-256 representation).

Raises `OverflowError` when a fill size is given and the number takes up more bytes than fit into the block. This requires the `overflow` argument to this function to be set to `False` otherwise, no error will be raised.

synopsis More portable JSON encoding and decoding routines.

module mom.codec.json

mom.codec.json.**json_encode**(*obj*)
Encodes a Python value into its equivalent JSON string.

JSON permits but does not require forward slashes to be escaped. This is useful when json data is emitted in a <script> tag in HTML, as it prevents </script> tags from prematurely terminating the javscript. Some json libraries do this escaping by default, although python's standard library does not, so we do it here.

See http://stackoverflow.com/questions/1580647/json-why-are-forward-slashes-escaped

Parameters **obj** – Python value.

Returns JSON string.

mom.codec.json.**json_decode**(*encoded*)
Decodes a JSON string into its equivalent Python value.

Parameters **encoded** – JSON string.

Returns Decoded Python value.

synopsis Common functions for text encodings.

module mom.codec.text

Text encoding

```
"There is no such thing as plain text."
                          - Plain Text.
```

UTF-8 is one of the many ways in which Unicode strings can be represented as a *sequence of bytes*, and because UTF-8 is more portable between diverse systems, you must ensure to convert your Unicode strings to UTF-8 encoded bytes before they leave your system and ensure to decode UTF-8 encoded bytes back into Unicode strings before you start working with them in your code–that is, if you know those bytes are UTF-8 encoded.

Terminology

- The process of **encoding** is that of converting a Unicode string into a sequence of bytes. The **method** using which this conversion is done is *also* called an **encoding**:

```
Unicode string   -> Encoded bytes
---------------------------------
" Python"     -> b"\xe6\xb7\xb1\xe5\x85\xa5 Python"
```

  The **encoding** (method) used to *encode* in this example is UTF-8.

- The process of **decoding** is that of converting a sequence of bytes into a Unicode string:

```
Encoded bytes                       -> Unicode string
-------------------------------------------------
b"\xe6\xb7\xb1\xe5\x85\xa5 Python" -> " Python"
```

  The **encoding** (method) used to *decode* in this example is UTF-8.

A very crude explanation of when to use what Essentially, inside your own system, work with:

```
" Python"
```

and not:

```
b"\xe6\xb7\xb1\xe5\x85\xa5 Python"
```

but when sending things out to other systems that may not see *" Python"* the way Python does, you encode it into UTF-8 bytes:

```
b"\xe6\xb7\xb1\xe5\x85\xa5 Python"
```

and tell those systems that you're using UTF-8 to encode your Unicode strings so that those systems can decode the bytes you sent appropriately.

When receiving text from other systems, ask for their encodings. Decode the text using the appropriate encoding method as soon as you receive it and then operate on the resulting Unicode text.

Read these before you begin to use these functions

1. http://www.joelonsoftware.com/articles/Unicode.html
2. http://diveintopython3.org/strings.html
3. http://docs.python.org/howto/unicode.html

4. http://docs.python.org/library/codecs.html

mom.codec.text.**utf8_encode**(*unicode_text*)
UTF-8 encodes a Unicode string into bytes; bytes and None are left alone.

Work with Unicode strings in your code and encode your Unicode strings into UTF-8 before they leave your system.

Parameters **unicode_text** – If already a byte string or None, it is returned unchanged. Otherwise it must be a Unicode string and is encoded as UTF-8 bytes.

Returns UTF-8 encoded bytes.

mom.codec.text.**utf8_decode**(*utf8_encoded_bytes*)
Decodes bytes into a Unicode string using the UTF-8 encoding.

Decode your UTF-8 encoded bytes into Unicode strings as soon as they arrive into your system. Work with Unicode strings in your code.

Parameters **utf8_encoded_bytes** – UTF-8 encoded bytes.

Returns Unicode string.

mom.codec.text.**utf8_encode_if_unicode**(*obj*)
UTF-8 encodes the object only if it is a Unicode string.

Parameters **obj** – The value that will be UTF-8 encoded if it is a Unicode string.

Returns UTF-8 encoded bytes if the argument is a Unicode string; otherwise the value is returned unchanged.

mom.codec.text.**utf8_decode_if_bytes**(*obj*)
Decodes UTF-8 encoded bytes into a Unicode string.

Parameters **obj** – Python object. If this is a bytes instance, it will be decoded into a Unicode string; otherwise, it will be left alone.

Returns Unicode string if the argument is a bytes instance; the unchanged object otherwise.

mom.codec.text.**utf8_encode_recursive**(*obj*)
Walks a simple data structure, converting Unicode strings to UTF-8 encoded byte strings.

Supports lists, tuples, and dictionaries.

Parameters **obj** – The Python data structure to walk recursively looking for Unicode strings.

Returns obj with all the Unicode strings converted to byte strings.

mom.codec.text.**utf8_decode_recursive**(*obj*)
Walks a simple data structure, converting bytes to Unicode strings.

Supports lists, tuples, and dictionaries.

Parameters **obj** – The Python data structure to walk recursively looking for byte strings.

Returns obj with all the byte strings converted to Unicode strings.

mom.codec.text.**bytes_to_unicode**(*raw_bytes*, *encoding='utf-8'*)
Converts bytes to a Unicode string decoding it according to the encoding specified.

Parameters

- **raw_bytes** – If already a Unicode string or None, it is returned unchanged. Otherwise it must be a byte string.
- **encoding** – The encoding used to decode bytes. Defaults to UTF-8

mom.codec.text.**bytes_to_unicode_recursive**(*obj*, *encoding='utf-8'*)
Walks a simple data structure, converting byte strings to unicode.

Supports lists, tuples, and dictionaries.

Parameters

- **obj** – The Python data structure to walk recursively looking for byte strings.
- **encoding** – The encoding to use when decoding the byte string into Unicode. Default UTF-8.

Returns obj with all the byte strings converted to Unicode strings.

mom.codec.text.**to_unicode_if_bytes**(*obj*, *encoding='utf-8'*)
Decodes encoded bytes into a Unicode string.

Parameters

- **obj** – The value that will be converted to a Unicode string.
- **encoding** – The encoding used to decode bytes. Defaults to UTF-8.

Returns Unicode string if the argument is a byte string. Otherwise the value is returned unchanged.

mom.codec.text.**ascii_encode**(*obj*)
Encodes a string using ASCII encoding.

Parameters **obj** – String to encode.

Returns ASCII-encoded bytes.

mom.codec.text.**latin1_encode**(*obj*)
Encodes a string using LATIN-1 encoding.

Parameters **obj** – String to encode.

Returns LATIN-1 encoded bytes.

synopsis Routines used by ASCII-based base converters.

module mom.codec._base

mom.codec._base.**base_encode**(*raw_bytes*, *base*, *base_bytes*, *base_zero*, *padding=True*)
Encodes raw bytes given a base.

Parameters

- **raw_bytes** – Raw bytes to encode.
- **base** – Unsigned integer base.
- **base_bytes** – The ASCII bytes used in the encoded string. "Character set" or "alphabet".
- **base_zero** –

mom.codec._base.**base_decode**(*encoded*, *base*, *base_ords*, *base_zero*, *powers*)
Decode from base to base 256.

mom.codec._base.**base_to_uint**(*encoded*, *base*, *ord_lookup_table*, *powers*)
Decodes bytes from the given base into a big integer.

Parameters

- **encoded** – Encoded bytes.
- **base** – The base to use.
- **ord_lookup_table** – The ordinal lookup table to use.

- **powers** – Pre-computed tuple of powers of length `powers_length`.

`mom.codec._base.`**`uint_to_base256`**(*number*, *encoded*, *base_zero*)
Convert uint to base 256.

3.3 Cryptography primitives

3.3.1 *mom.security*

module mom.security

synopsis Cryptography primitives.

module mom.security.hash

synopsis Convenient hashing functions.

SHA-1 digests

`mom.security.hash.`**`sha1_base64_digest`**(**inputs*)
Calculates Base-64-encoded SHA-1 digest of a variable number of inputs.

Parameters **`inputs`** – A variable number of inputs for which the digest will be calculated.

Returns Base-64-encoded SHA-1 digest.

`mom.security.hash.`**`sha1_digest`**(**inputs*)
Calculates a SHA-1 digest of a variable number of inputs.

Parameters **`inputs`** – A variable number of inputs for which the digest will be calculated.

Returns A byte string containing the SHA-1 message digest.

`mom.security.hash.`**`sha1_hex_digest`**(**inputs*)
Calculates hexadecimal representation of the SHA-1 digest of a variable number of inputs.

Parameters **`inputs`** – A variable number of inputs for which the digest will be calculated.

Returns Hexadecimal representation of the SHA-1 digest.

MD5 digests

`mom.security.hash.`**`md5_base64_digest`**(**inputs*)
Calculates Base-64-encoded MD5 digest of a variable number of inputs.

Parameters **`inputs`** – A variable number of inputs for which the digest will be calculated.

Returns Base-64-encoded MD5 digest.

`mom.security.hash.`**`md5_digest`**(**inputs*)
Calculates a MD5 digest of a variable number of inputs.

Parameters **`inputs`** – A variable number of inputs for which the digest will be calculated.

Returns A byte string containing the MD5 message digest.

`mom.security.hash.`**`md5_hex_digest`**(**inputs*)
Calculates hexadecimal representation of the MD5 digest of a variable number of inputs.

Parameters **`inputs`** – A variable number of inputs for which the digest will be calculated.

Returns Hexadecimal representation of the MD5 digest.

HMAC-SHA-1 digests

mom.security.hash.**hmac_sha1_base64_digest**(*key*, *data*)
Calculates a base64-encoded HMAC SHA-1 signature.

Parameters

- **key** – The key for the signature.
- **data** – The data to be signed.

Returns Base64-encoded HMAC SHA-1 signature.

mom.security.hash.**hmac_sha1_digest**(*key*, *data*)
Calculates a HMAC SHA-1 digest.

Parameters

- **key** – The key for the digest.
- **data** – The raw bytes data for which the digest will be calculated.

Returns HMAC SHA-1 Digest.

module mom.security.random

synopsis Random number, bits, bytes, string, sequence, & password generation.

Bits and bytes

mom.security.random.**generate_random_bits**(*n_bits*, *rand_func=<function generate_random_bytes>*)
Generates the specified number of random bits as a byte string. For example:

```
    f(x) -> y such that
    f(16) ->           1111 1111 1111 1111; bytes_to_integer(y) => 65535L
    f(17) -> 0000 0001 1111 1111 1111 1111; bytes_to_integer(y) => 131071L
```

Parameters

- **n_bits** – Number of random bits.

  if n is divisible by 8, (n / 8) bytes will be returned. if n is not divisible by 8, ((n / 8) + 1) bytes will be returned and the prefixed offset-byte will have *(n % 8)* number of random bits, (that is, *8 - (n % 8)* high bits will be cleared).

  The range of the numbers is 0 to (2**n)-1 inclusive.

- **rand_func** – Random bytes generator function.

Returns Bytes.

mom.security.random.**generate_random_bytes**(*count*)
Generates a random byte string with `count` bytes.

Parameters **count** – Number of bytes.

Returns Random byte string.

Numbers

mom.security.random.**generate_random_uint_atmost**(*n_bits*, *rand_func=<function generate_random_bytes>*)

Generates a random unsigned integer with *n_bits* random bits.

Parameters

- **n_bits** – Number of random bits to be generated at most.
- **rand_func** – Random bytes generator function.

Returns Returns an unsigned long integer with at most *n_bits* random bits. The generated unsigned long integer will be between 0 and (2**n_bits)-1 both inclusive.

mom.security.random.**generate_random_uint_exactly**(*n_bits*, *rand_func=<function generate_random_bytes>*)

Generates a random unsigned long with `n_bits` random bits.

Parameters

- **n_bits** – Number of random bits.
- **rand_func** – Random bytes generator function.

Returns Returns an unsigned long integer with `n_bits` random bits. The generated unsigned long integer will be between 2**(n_bits-1) and (2**n_bits)-1 both inclusive.

mom.security.random.**generate_random_uint_between**(*low*, *high*, *rand_func=<function generate_random_bytes>*)

Generates a random long integer between low and high, not including high.

Parameters

- **low** – Low
- **high** – High
- **rand_func** – Random bytes generator function.

Returns Random unsigned long integer value.

Sequences and choices

mom.security.random.**random_choice**(*sequence*, *rand_func=<function generate_random_bytes>*)

Randomly chooses an element from the given non-empty sequence.

Parameters **sequence** – Non-empty sequence to randomly choose an element from.

Returns Randomly chosen element.

mom.security.random.**random_shuffle**(*sequence*, *rand_func=<function generate_random_bytes>*)

Randomly shuffles the sequence in-place.

Parameters **sequence** – Sequence to shuffle in-place.

Returns The shuffled sequence itself (for convenience).

mom.security.random.**generate_random_sequence**(*length*, *pool*, *rand_func=<function generate_random_bytes>*)

Generates a random sequence of given length using the sequence pool specified.

Parameters

- **length** – The length of the random sequence.

- **pool** – A sequence of elements to be used as the pool from which random elements will be chosen.

Returns A list of elements randomly chosen from the pool.

mom.security.random.**generate_random_sequence_strong**(*entropy*, *pool*, *rand_func=<function generate_random_bytes>*)

Generates a random sequence based on entropy.

If you're using this to generate passwords based on entropy: http://en.wikipedia.org/wiki/Password_strength

Parameters

- **entropy** – Desired entropy in bits.
- **pool** – The pool of unique elements from which to randomly choose.

Returns Randomly generated sequence with specified entropy.

Strings

mom.security.random.**generate_random_string**(*length*, *pool='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQ*
rand_func=<function generate_random_bytes>)

Generates a random string of given length using the sequence pool specified.

Don't use this to generate passwords. Use generate_random_password instead.

Entropy:

H = log2(N**L)

where:

•H is the entropy in bits.

•N is the possible symbol count

•L is length of string of symbols

Entropy chart:

```
-----------------------------------------------------------------
Symbol set              Symbol Count (N)  Entropy per symbol (H)
-----------------------------------------------------------------
HEXADECIMAL_DIGITS      16                4.0000 bits
DIGITS                  10                3.3219 bits
LOWERCASE_ALPHA         26                4.7004 bits
UPPERCASE_ALPHA         26                4.7004 bits
PUNCTUATION             32                5.0000 bits
LOWERCASE_ALPHANUMERIC  36                5.1699 bits
UPPERCASE_ALPHANUMERIC  36                5.1699 bits
ALPHA                   52                5.7004 bits
ALPHANUMERIC            62                5.9542 bits
ASCII_PRINTABLE         94                6.5546 bits
ALL_PRINTABLE           100               6.6438 bits
```

Parameters

- **length** – The length of the random sequence.
- **pool** – A sequence of characters to be used as the pool from which random characters will be chosen. Default case-sensitive alpha-numeric characters.

Returns A string of elements randomly chosen from the pool.

mom.security.random.**generate_random_password**(*entropy*, *pool='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNO -./:;<=>?@[\\]^_`{|}~'*, *rand_func=<function generate_random_bytes>*)

Generates a password based on entropy.

If you're using this to generate passwords based on entropy: http://en.wikipedia.org/wiki/Password_strength

Parameters

- **entropy** – Desired entropy in bits. Choose at least 64 to have a decent password.
- **pool** – The pool of unique characters from which to randomly choose.

Returns Randomly generated password with specified entropy.

mom.security.random.**generate_random_hex_string**(*length=8*, *rand_func=<function generate_random_bytes>*)

Generates a random ASCII-encoded hexadecimal string of an even length.

Parameters

- **length** – Length of the string to be returned. Default 32. The length MUST be a positive even number.
- **rand_func** – Random bytes generator function.

Returns A string representation of a randomly-generated hexadecimal string.

Utility

mom.security.random.**calculate_entropy**(*length*, *pool='abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUV*

Determines the entropy of the given sequence length and the pool.

Parameters

- **length** – The length of the generated random sequence.
- **pool** – The pool of unique elements used to generate the sequence.

Returns The entropy (in bits) of the random sequence.

module mom.security.codec

synopsis Codecs to encode and decode keys and certificates in various formats.

PEM key decoders

mom.security.codec.**public_key_pem_decode**(*pem_key*)

Decodes a PEM-encoded public key/X.509 certificate string into internal representation.

Parameters **pem_key** – The PEM-encoded key. Must be one of: 1. RSA public key. 2. X.509 certificate.

Returns A dictionary of key information.

mom.security.codec.**private_key_pem_decode**(*pem_key*)

Decodes a PEM-encoded private key string into internal representation.

Parameters **pem_key** – The PEM-encoded RSA private key.

Returns A dictionary of key information.

3.4 Networking

3.4.1 *mom.net*

module mom.net

synopsis Makes working with Data URI-schemes easier.

module mom.net.data

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

see https://tools.ietf.org/html/rfc2397

mom.net.data_uri.**data_uri_encode**(*raw_bytes*, *mime_type='text/plain'*, *charset='US-ASCII'*, *encoder='base64'*)

Encodes raw bytes into a data URI scheme string.

Parameters

- **raw_bytes** – Raw bytes
- **mime_type** – The mime type, e.g. b"text/css" or b"image/png". Default b"text/plain".
- **charset** – b"utf-8" if you want the data URI to contain a b"charset=utf-8" component. Default b"US-ASCII". This does not mean however, that your raw_bytes will be encoded by this function. You must ensure that if you specify, b"utf-8" (or anything else) as the encoding, you have encoded your raw data appropriately.
- **encoder** – "base64" or None.

Returns Data URI.

mom.net.data_uri.**data_uri_parse**(*data_uri*)

Parses a data URI into raw bytes and metadata.

Parameters **data_uri** – The data url string. If a mime-type definition is missing in the metadata, "text/plain;charset=US-ASCII" will be used as default mime-type.

Returns

A 2-tuple:: (bytes, mime_type)

See `mom.http.mimeparse.mimeparse.parse_mime_type()` for what `mime_type` looks like.

3.5 Operating System helpers

3.5.1 *mom.os*

module mom.os

synopsis Operating system specific functions.

module mom.os.path

synopsis Directory walking, listing, and path sanitizing functions.

Functions

mom.os.path.**get_dir_walker**(*recursive*, *topdown=True*, *followlinks=False*)
Returns a recursive or a non-recursive directory walker.

Parameters **recursive** – `True` produces a recursive walker; `False` produces a non-recursive walker.

Returns A walker function.

mom.os.path.**walk**(*dir_pathname*, *recursive=True*, *topdown=True*, *followlinks=False*)
Walks a directory tree optionally recursively. Works exactly like `os.walk()` only adding the *recursive* argument.

Parameters

- **dir_pathname** – The directory to traverse.
- **recursive** – `True` for walking recursively through the directory tree; `False` otherwise.
- **topdown** – Please see the documentation for `os.walk()`
- **followlinks** – Please see the documentation for `os.walk()`

mom.os.path.**listdir**(*dir_pathname*, *recursive=True*, *topdown=True*, *followlinks=False*)
Enlists all items using their absolute paths in a directory, optionally non-recursively.

Parameters

- **dir_pathname** – The directory to traverse.
- **recursive** – `True` (default) for walking recursively through the directory tree; `False` otherwise.
- **topdown** – Please see the documentation for `os.walk()`
- **followlinks** – Please see the documentation for `os.walk()`

mom.os.path.**list_directories**(*dir_pathname*, *recursive=True*, *topdown=True*, *followlinks=False*)
Enlists all the directories using their absolute paths within the specified directory, optionally non-recursively.

Parameters

- **dir_pathname** – The directory to traverse.
- **recursive** – `True` (default) for walking recursively through the directory tree; `False` otherwise.
- **topdown** – Please see the documentation for `os.walk()`
- **followlinks** – Please see the documentation for `os.walk()`

mom.os.path.**list_files**(*dir_pathname*, *recursive=True*, *topdown=True*, *followlinks=False*)
Enlists all the files using their absolute paths within the specified directory, optionally recursively.

Parameters

- **dir_pathname** – The directory to traverse.
- **recursive** – `True` for walking recursively through the directory tree; `False` otherwise.
- **topdown** – Please see the documentation for `os.walk()`
- **followlinks** – Please see the documentation for `os.walk()`

mom.os.path.**absolute_path**(*path*)
Returns the absolute path for the given path and normalizes the path.

Parameters **path** – Path for which the absolute normalized path will be found.

Returns Absolute normalized path.

mom.os.path.**real_absolute_path**(*path*)
Returns the real absolute normalized path for the given path.

Parameters **path** – Path for which the real absolute normalized path will be found.

Returns Real absolute normalized path.

mom.os.path.**parent_dir_path**(*path*)
Returns the parent directory path.

Parameters **path** – Path for which the parent directory will be obtained.

Returns Parent directory path.

module mom.os.patterns

synopsis Wildcard pattern matching and filtering functions for paths.

Functions

mom.os.patterns.**match_path**(*pathname*, *included_patterns=None*, *excluded_patterns=None*, *case_sensitive=True*)
Matches a pathname against a set of acceptable and ignored patterns.

Parameters

- **pathname** – A pathname which will be matched against a pattern.
- **included_patterns** – Allow filenames matching wildcard patterns specified in this list. If no pattern is specified, the function treats the pathname as a match_path.
- **excluded_patterns** – Ignores filenames matching wildcard patterns specified in this list. If no pattern is specified, the function treats the pathname as a match_path.
- **case_sensitive** – `True` if matching should be case-sensitive; `False` otherwise.

Returns `True` if the pathname matches; `False` otherwise.

Raises ValueError if included patterns and excluded patterns contain the same pattern.

mom.os.patterns.**match_path_against**(*pathname*, *patterns*, *case_sensitive=True*)
Determines whether the pathname matches any of the given wildcard patterns, optionally ignoring the case of the pathname and patterns.

Parameters

- **pathname** – A path name that will be matched against a wildcard pattern.
- **patterns** – A list of wildcard patterns to match_path the filename against.
- **case_sensitive** – `True` if the matching should be case-sensitive; `False` otherwise.

Returns `True` if the pattern matches; `False` otherwise.

mom.os.patterns.**match_any_paths**(*pathnames*, *included_patterns=None*, *excluded_patterns=None*, *case_sensitive=True*)
Matches from a set of paths based on acceptable patterns and ignorable patterns.

Parameters

- **pathnames** – A list of path names that will be filtered based on matching and ignored patterns.

- **included_patterns** – Allow filenames matching wildcard patterns specified in this list. If no pattern list is specified, ["*"] is used as the default pattern, which matches all files.
- **excluded_patterns** – Ignores filenames matching wildcard patterns specified in this list. If no pattern list is specified, no files are ignored.
- **case_sensitive** – `True` if matching should be case-sensitive; `False` otherwise.

Returns `True` if any of the paths matches; `False` otherwise.

mom.os.patterns.**filter_paths**(*pathnames*, *included_patterns=None*, *excluded_patterns=None*, *case_sensitive=True*)

Filters from a set of paths based on acceptable patterns and ignorable patterns.

Parameters

- **pathnames** – A list of path names that will be filtered based on matching and ignored patterns.
- **included_patterns** – Allow filenames matching wildcard patterns specified in this list. If no pattern list is specified, ["*"] is used as the default pattern, which matches all files.
- **excluded_patterns** – Ignores filenames matching wildcard patterns specified in this list. If no pattern list is specified, no files are ignored.
- **case_sensitive** – `True` if matching should be case-sensitive; `False` otherwise.

Returns A list of pathnames that matched the allowable patterns and passed through the ignored patterns.

CHAPTER 4

Contribute

Found a bug in or want a feature added to mom? You can fork the official code repository or file an issue ticket at the issue tracker. You may also want to refer to *Contributing* for information about contributing code or documentation to mom.

4.1 Contributing

Welcome hackeratti! So you have got something you would like to see in mom? Whee. This document will help you get started.

4.1.1 Important URLs

mom uses git to track code history and hosts its code repository at github. The issue tracker is where you can file bug reports and request features or enhancements to mom.

4.1.2 Before you start

Ensure your system has the following programs and libraries installed before beginning to hack:

1. Python
2. git
3. ssh

4.1.3 Setting up the Work Environment

mom makes extensive use of zc.buildout to set up its work environment. You should get familiar with it.

Steps to setting up a clean environment:

1. Fork the code repository into your github account. Let us call you `hackeratti`. That *is* your name innit? Replace `hackeratti` with your own username below if it isn't.
2. Clone your fork and setup your environment:

```
$ git clone --recursive git@github.com:hackeratti/mom.git
$ cd mom
$ python bootstrap.py --distribute
$ bin/buildout
```

Important: Re-run `bin/buildout` every time you make a change to the `buildout.cfg` file.

That's it with the setup. Now you're ready to hack on `mom`.

4.1.4 Enabling Continuous Integration

The repository checkout contains a script called `autobuild.sh` which you should run prior to making changes. It will detect changes to Python source code or restructuredText documentation files anywhere in the directory tree and rebuild sphinx documentation, run all tests using unittest2, and generate coverage reports.

Start it by issuing this command in the `mom` directory checked out earlier:

```
$ tools/autobuild.sh
...
```

Happy hacking!

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

m

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

W