Arrow Documentation

Release 0.11.0

Chris Smith

July 16, 2017

Contents

CHAPTER 1

What?

Arrow is a Python library that offers a sensible, human-friendly approach to creating, manipulating, formatting and converting dates, times, and timestamps. It implements and updates the datetime type, plugging gaps in functionality, and provides an intelligent module API that supports many common creation scenarios. Simply put, it helps you work with dates and times with fewer imports and a lot less code.

Arrow is heavily inspired by moment.js and requests.

CHAPTER 2

Why?

Python's standard library and some other low-level modules have near-complete date, time and timezone functionality but don't work very well from a usability perspective:

- Too many modules: datetime, time, calendar, dateutil, pytz and more
- Too many types: date, time, datetime, tzinfo, timedelta, relativedelta, etc.
- Timezones and timestamp conversions are verbose and unpleasant
- Timezone naivety is the norm
- Gaps in functionality: ISO-8601 parsing, time spans, humanization

CHAPTER 3

Features

- Fully implemented, drop-in replacement for datetime
- Supports Python 2.6, 2.7, 3.3, 3.4 and 3.5
- Timezone-aware & UTC by default
- Provides super-simple creation options for many common input scenarios
- Updated .replace method with support for relative offsets, including weeks
- Formats and parses strings automatically
- Partial ISO-8601 support
- Timezone conversion
- Timestamp available as a property
- Generates time spans, ranges, floors and ceilings in time frames from year to microsecond
- Humanizes and supports a growing list of contributed locales
- Extensible for your own Arrow-derived types

CHAPTER 4

Quickstart

```
$ pip install arrow
```

```
>>> import arrow
>>> utc = arrow.utcnow()
>>> utc
<Arrow [2013-05-11T21:23:58.970460+00:00]>

>>> utc = utc.shift(hours=-1)
>>> utc
<Arrow [2013-05-11T20:23:58.970460+00:00]>

>>> local = utc.to('US/Pacific')
>>> local
<Arrow [2013-05-11T13:23:58.970460-07:00]>

>>> arrow.get('2013-05-11T21:23:58.970460+00:00')
<Arrow [2013-05-11T21:23:58.970460+00:00]>

>>> local.timestamp
1368303838

>>> local.format()
'2013-05-11 13:23:58 -07:00'

>>> local.format('YYYY-MM-DD HH:mm:ss ZZ')
'2013-05-11 13:23:58 -07:00'

>>> local.humanize()
'an hour ago'

>>> local.humanize(locale='ko_kr')
'1  '
```

User's Guide

Creation

Get 'now' easily:

```
>>> arrow.utcnow()
<Arrow [2013-05-07T04:20:39.369271+00:00]>

>>> arrow.now()
<Arrow [2013-05-06T21:20:40.841085-07:00]>

>>> arrow.now('US/Pacific')
<Arrow [2013-05-06T21:20:44.761511-07:00]>
```

Create from timestamps (ints or floats, or strings that convert to a float):

```
>>> arrow.get(1367900664)
<Arrow [2013-05-07T04:24:24+00:00]>

>>> arrow.get('1367900664')
<Arrow [2013-05-07T04:24:24+00:00]>

>>> arrow.get(1367900664.152325)
<Arrow [2013-05-07T04:24:24.152325+00:00]>

>>> arrow.get('1367900664.152325')
<Arrow [2013-05-07T04:24:24.152325+00:00]>
```

Use a naive or timezone-aware datetime, or flexibly specify a timezone:

```
>>> arrow.get(datetime.utcnow())
<Arrow [2013-05-07T04:24:24.152325+00:00]>

>>> arrow.get(datetime(2013, 5, 5), 'US/Pacific')
<Arrow [2013-05-05T00:00:00-07:00]>

>>> from dateutil import tz
>>> arrow.get(datetime(2013, 5, 5), tz.gettz('US/Pacific'))
<Arrow [2013-05-05T00:00:00-07:00]>

>>> arrow.get(datetime.now(tz.gettz('US/Pacific')))
<Arrow [2013-05-06T21:24:49.552236-07:00]>
```

Parse from a string:

```
>>> arrow.get('2013-05-05 12:30:45', 'YYYY-MM-DD HH:mm:ss')
<Arrow [2013-05-05T12:30:45+00:00]>
```

Search a date in a string:

```
>>> arrow.get('June was born in May 1980', 'MMMM YYYY')
<Arrow [1980-05-01T00:00:00+00:00]>
```

Some ISO-8601 compliant strings are recognized and parsed without a format string:

```
>>> arrow.get('2013-09-30T15:34:00.000-07:00')
<Arrow [2013-09-30T15:34:00-07:00]>
```

Arrow objects can be instantiated directly too, with the same arguments as a datetime:

```
>>> arrow.get(2013, 5, 5)
<Arrow [2013-05-05T00:00:00+00:00]>

>>> arrow.Arrow(2013, 5, 5)
<Arrow [2013-05-05T00:00:00+00:00]>
```

Properties

Get a datetime or timestamp representation:

```
>>> a = arrow.utcnow()
>>> a.datetime
datetime.datetime(2013, 5, 7, 4, 38, 15, 447644, tzinfo=tzutc())

>>> a.timestamp
1367901495
```

Get a naive datetime, and tzinfo:

```
>>> a.naive
datetime.datetime(2013, 5, 7, 4, 38, 15, 447644)

>>> a.tzinfo
tzutc()
```

Get any datetime value:

```
>>> a.year
2013
```

Call datetime functions that return properties:

```
>>> a.date()
datetime.date(2013, 5, 7)

>>> a.time()
datetime.time(4, 38, 15, 447644)
```

Replace & shift

Get a new `Arrow` object, with altered attributes, just as you would with a datetime:

```
>>> arw = arrow.utcnow()
>>> arw
<Arrow [2013-05-12T03:29:35.334214+00:00]>

>>> arw.replace(hour=4, minute=40)
<Arrow [2013-05-12T04:40:35.334214+00:00]>
```

Or, get one with attributes shifted forward or backward:

```
>>> arw.shift(weeks=+3)
<Arrow [2013-06-02T03:29:35.334214+00:00]>
```

Even replace the timezone without altering other attributes:

```
>>> arw.replace(tzinfo='US/Pacific')
<Arrow [2013-05-12T03:29:35.334214-07:00]>
```

Format

```
>>> arrow.utcnow().format('YYYY-MM-DD HH:mm:ss ZZ')
'2013-05-07 05:23:16 -00:00'
```

Convert

Convert to timezones by name or tzinfo:

```
>>> utc = arrow.utcnow()
>>> utc
<Arrow [2013-05-07T05:24:11.823627+00:00]>

>>> utc.to('US/Pacific')
<Arrow [2013-05-06T22:24:11.823627-07:00]>

>>> utc.to(tz.gettz('US/Pacific'))
<Arrow [2013-05-06T22:24:11.823627-07:00]>
```

Or using shorthand:

```
>>> utc.to('local')
<Arrow [2013-05-06T22:24:11.823627-07:00]>

>>> utc.to('local').to('utc')
<Arrow [2013-05-07T05:24:11.823627+00:00]>
```

Humanize

Humanize relative to now:

```
>>> past = arrow.utcnow().shift(hours=-1)
>>> past.humanize()
'an hour ago'
```

Or another Arrow, or datetime:

```
>>> present = arrow.utcnow()
>>> future = present.shift(hours=2)
>>> future.humanize(present)
'in 2 hours'
```

Support for a growing number of locales (see *locales.py* for supported languages):

```
>>> future = arrow.utcnow().shift(hours=1)
>>> future.humanize(a, locale='ru')
' 2 (,)'
```

Ranges & spans

Get the time span of any unit:

```
>>> arrow.utcnow().span('hour')
(<Arrow [2013-05-07T05:00:00+00:00]>, <Arrow [2013-05-07T05:59:59.999999+00:00]>)
```

Or just get the floor and ceiling:

```
>>> arrow.utcnow().floor('hour')
<Arrow [2013-05-07T05:00:00+00:00]>

>>> arrow.utcnow().ceil('hour')
<Arrow [2013-05-07T05:59:59.999999+00:00]>
```

You can also get a range of time spans:

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.span_range('hour', start, end):
...     print r
...
(<Arrow [2013-05-05T12:00:00+00:00]>, <Arrow [2013-05-05T12:59:59.999999+00:00]>)
(<Arrow [2013-05-05T13:00:00+00:00]>, <Arrow [2013-05-05T13:59:59.999999+00:00]>)
(<Arrow [2013-05-05T14:00:00+00:00]>, <Arrow [2013-05-05T14:59:59.999999+00:00]>)
(<Arrow [2013-05-05T15:00:00+00:00]>, <Arrow [2013-05-05T15:59:59.999999+00:00]>)
(<Arrow [2013-05-05T16:00:00+00:00]>, <Arrow [2013-05-05T16:59:59.999999+00:00]>)
```

Or just iterate over a range of time:

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.range('hour', start, end):
...     print repr(r)
...
<Arrow [2013-05-05T12:30:00+00:00]>
<Arrow [2013-05-05T13:30:00+00:00]>
<Arrow [2013-05-05T14:30:00+00:00]>
<Arrow [2013-05-05T15:30:00+00:00]>
<Arrow [2013-05-05T16:30:00+00:00]>
```

Factories

Use factories to harness Arrow's module API for a custom Arrow-derived type. First, derive your type:

```
>>> class CustomArrow(arrow.Arrow):
...
...     def days_till_xmas(self):
...
...         xmas = arrow.Arrow(self.year, 12, 25)
...
...         if self > xmas:
...             xmas = xmas.shift(years=1)
...
...         return (xmas - self).days
```

Then get and use a factory for it:

```
>>> factory = arrow.ArrowFactory(CustomArrow)
>>> custom = factory.utcnow()
>>> custom
>>> <CustomArrow [2013-05-27T23:35:35.533160+00:00]>

>>> custom.days_till_xmas()
>>> 211
```

Tokens

Use the following tokens in parsing and formatting. Note that they're not the same as the tokens for strptime(3):

	Token	Output
Year	YYYY	2000, 2001, 2002 ... 2012, 2013
	YY	00, 01, 02 ... 12, 13
Month	MMMM	January, February, March ... [1]
	MMM	Jan, Feb, Mar ... [1]
	MM	01, 02, 03 ... 11, 12
	M	1, 2, 3 ... 11, 12
Day of Year	DDDD	001, 002, 003 ... 364, 365
	DDD	1, 2, 3 ... 4, 5
Day of Month	DD	01, 02, 03 ... 30, 31
	D	1, 2, 3 ... 30, 31
	Do	1st, 2nd, 3rd ... 30th, 31st
Day of Week	dddd	Monday, Tuesday, Wednesday ... [2]
	ddd	Mon, Tue, Wed ... [2]
	d	1, 2, 3 ... 6, 7
Hour	HH	00, 01, 02 ... 23, 24
	H	0, 1, 2 ... 23, 24
	hh	01, 02, 03 ... 11, 12
	h	1, 2, 3 ... 11, 12
AM / PM	A	AM, PM, am, pm [1]
	a	am, pm [1]
Minute	mm	00, 01, 02 ... 58, 59
	m	0, 1, 2 ... 58, 59
Second	ss	00, 01, 02 ... 58, 59
	s	0, 1, 2 ... 58, 59
Sub-second	S...	0, 02, 003, 000006, 123123123123... [3]
Timezone	ZZZ	Asia/Baku, Europe/Warsaw, GMT ... [4]
	ZZ	-07:00, -06:00 ... +06:00, +07:00
	Z	-0700, -0600 ... +0600, +0700
Timestamp	X	1381685817

[1] localization support for parsing and formatting
[2] localization support only for formatting
[3] the result is truncated to microseconds, with half-to-even rounding.
[4] timezone names from tz database provided via dateutil package

CHAPTER 6

API Guide

arrow.arrow

Provides the *Arrow* class, an enhanced `datetime` replacement.

class `arrow.arrow.`**`Arrow`**(*year*, *month*, *day*, *hour=0*, *minute=0*, *second=0*, *microsecond=0*, *tzinfo=None*)

An *Arrow* object.

Implements the `datetime` interface, behaving as an aware `datetime` while implementing additional functionality.

Parameters

- **`year`** – the calendar year.
- **`month`** – the calendar month.
- **`day`** – the calendar day.
- **`hour`** – (optional) the hour. Defaults to 0.
- **`minute`** – (optional) the minute, Defaults to 0.
- **`second`** – (optional) the second, Defaults to 0.
- **`microsecond`** – (optional) the microsecond. Defaults 0.
- **`tzinfo`** – (optional) A timezone expression. Defaults to UTC.

Recognized timezone expressions:

•A `tzinfo` object.

•A `str` describing a timezone, similar to 'US/Pacific', or 'Europe/Berlin'.

•A `str` in ISO-8601 style, as in '+07:00'.

•A `str`, one of the following: 'local', 'utc', 'UTC'.

Usage:

```
>>> import arrow
>>> arrow.Arrow(2013, 5, 5, 12, 30, 45)
<Arrow [2013-05-05T12:30:45+00:00]>
```

classmethod **`now`**(*tzinfo=None*)

Constructs an *Arrow* object, representing "now" in the given timezone.

Parameters **`tzinfo`** – (optional) a `tzinfo` object. Defaults to local time.

classmethod `utcnow`()

Constructs an *`Arrow`* object, representing "now" in UTC time.

classmethod `fromtimestamp`(*timestamp*, *tzinfo=None*)

Constructs an *`Arrow`* object from a timestamp, converted to the given timezone.

Parameters

- **`timestamp`** – an `int` or `float` timestamp, or a `str` that converts to either.
- **`tzinfo`** – (optional) a `tzinfo` object. Defaults to local time.

Timestamps should always be UTC. If you have a non-UTC timestamp:

```
>>> arrow.Arrow.utcfromtimestamp(1367900664).replace(tzinfo='US/Pacific')
<Arrow [2013-05-07T04:24:24-07:00]>
```

classmethod `utcfromtimestamp`(*timestamp*)

Constructs an *`Arrow`* object from a timestamp, in UTC time.

Parameters **`timestamp`** – an `int` or `float` timestamp, or a `str` that converts to either.

classmethod `fromdatetime`(*dt*, *tzinfo=None*)

Constructs an *`Arrow`* object from a `datetime` and optional replacement timezone.

Parameters

- **`dt`** – the `datetime`
- **`tzinfo`** – (optional) A *timezone expression*. Defaults to `dt`'s timezone, or UTC if naive.

If you only want to replace the timezone of naive datetimes:

```
>>> dt
datetime.datetime(2013, 5, 5, 0, 0, tzinfo=tzutc())
>>> arrow.Arrow.fromdatetime(dt, dt.tzinfo or 'US/Pacific')
<Arrow [2013-05-05T00:00:00+00:00]>
```

classmethod `fromdate`(*date*, *tzinfo=None*)

Constructs an *`Arrow`* object from a `date` and optional replacement timezone. Time values are set to 0.

Parameters

- **`date`** – the `date`
- **`tzinfo`** – (optional) A *timezone expression*. Defaults to UTC.

classmethod `strptime`(*date_str*, *fmt*, *tzinfo=None*)

Constructs an *`Arrow`* object from a date string and format, in the style of `datetime.strptime`. Optionally replaces the parsed timezone.

Parameters

- **`date_str`** – the date string.
- **`fmt`** – the format string.
- **`tzinfo`** – (optional) A *timezone expression*. Defaults to the parsed timezone if `fmt` contains a timezone directive, otherwise UTC.

classmethod `range`(*frame*, *start*, *end=None*, *tz=None*, *limit=None*)

Returns a list of *`Arrow`* objects, representing an iteration of time between two inputs.

Parameters

- **frame** – the timeframe. Can be any `datetime` property (day, hour, minute...).
- **start** – A datetime expression, the start of the range.
- **end** – (optional) A datetime expression, the end of the range.
- **tz** – (optional) A *timezone expression*. Defaults to `start`'s timezone, or UTC if `start` is naive.
- **limit** – (optional) A maximum number of tuples to return.

NOTE: The `end` or `limit` must be provided. Call with `end` alone to return the entire range. Call with `limit` alone to return a maximum # of results from the start. Call with both to cap a range at a maximum # of results.

NOTE: `tz` internally **replaces** the timezones of both `start` and `end` before iterating. As such, either call with naive objects and `tz`, or aware objects from the same timezone and no `tz`.

Supported frame values: year, quarter, month, week, day, hour, minute, second.

Recognized datetime expressions:

•An `Arrow` object.

•A `datetime` object.

Usage:

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.range('hour', start, end):
...     print(repr(r))
...
<Arrow [2013-05-05T12:30:00+00:00]>
<Arrow [2013-05-05T13:30:00+00:00]>
<Arrow [2013-05-05T14:30:00+00:00]>
<Arrow [2013-05-05T15:30:00+00:00]>
<Arrow [2013-05-05T16:30:00+00:00]>
```

NOTE: Unlike Python's `range`, `end` *may* be included in the returned list:

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 13, 30)
>>> for r in arrow.Arrow.range('hour', start, end):
...     print(repr(r))
...
<Arrow [2013-05-05T12:30:00+00:00]>
<Arrow [2013-05-05T13:30:00+00:00]>
```

classmethod span_range (*frame*, *start*, *end*, *tz=None*, *limit=None*)

Returns a list of tuples, each `Arrow` objects, representing a series of timespans between two inputs.

Parameters

- **frame** – the timeframe. Can be any `datetime` property (day, hour, minute...).
- **start** – A datetime expression, the start of the range.
- **end** – (optional) A datetime expression, the end of the range.
- **tz** – (optional) A *timezone expression*. Defaults to `start`'s timezone, or UTC if `start` is naive.
- **limit** – (optional) A maximum number of tuples to return.

NOTE: The `end` or `limit` must be provided. Call with `end` alone to return the entire range. Call with `limit` alone to return a maximum # of results from the start. Call with both to cap a range at a maximum # of results.

NOTE: `tz` internally **replaces** the timezones of both `start` and `end` before iterating. As such, either call with naive objects and `tz`, or aware objects from the same timezone and no `tz`.

Supported frame values: year, quarter, month, week, day, hour, minute, second.

Recognized datetime expressions:

•An `Arrow` object.

•A `datetime` object.

NOTE: Unlike Python's `range`, `end` will *always* be included in the returned list of timespans.

Usage:

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.span_range('hour', start, end):
...     print(r)
...
(<Arrow [2013-05-05T12:00:00+00:00]>, <Arrow [2013-05-05T12:59:59.999999+00:00]>)
(<Arrow [2013-05-05T13:00:00+00:00]>, <Arrow [2013-05-05T13:59:59.999999+00:00]>)
(<Arrow [2013-05-05T14:00:00+00:00]>, <Arrow [2013-05-05T14:59:59.999999+00:00]>)
(<Arrow [2013-05-05T15:00:00+00:00]>, <Arrow [2013-05-05T15:59:59.999999+00:00]>)
(<Arrow [2013-05-05T16:00:00+00:00]>, <Arrow [2013-05-05T16:59:59.999999+00:00]>)
(<Arrow [2013-05-05T17:00:00+00:00]>, <Arrow [2013-05-05T17:59:59.999999+00:00]>)
```

classmethod `interval` (*frame*, *start*, *end*, *interval=1*, *tz=None*)

Returns an array of tuples, each `Arrow` objects, representing a series of intervals between two inputs.

Parameters

- **`frame`** – the timeframe. Can be any `datetime` property (day, hour, minute...).
- **`start`** – A datetime expression, the start of the range.
- **`end`** – (optional) A datetime expression, the end of the range.
- **`interval`** – (optional) Time interval for the given time frame.
- **`tz`** – (optional) A timezone expression. Defaults to UTC.

Supported frame values: year, quarter, month, week, day, hour, minute, second

Recognized datetime expressions:

•An `Arrow` object.

•A `datetime` object.

Recognized timezone expressions:

•A `tzinfo` object.

•A `str` describing a timezone, similar to 'US/Pacific', or 'Europe/Berlin'.

•A `str` in ISO-8601 style, as in '+07:00'.

•A `str`, one of the following: 'local', 'utc', 'UTC'.

Usage:

```
>>> start = datetime(2013, 5, 5, 12, 30)
>>> end = datetime(2013, 5, 5, 17, 15)
>>> for r in arrow.Arrow.interval('hour', start, end, 2):
...     print r
...
(<Arrow [2013-05-05T12:00:00+00:00]>, <Arrow [2013-05-05T13:59:59.999999+00:00]>)
(<Arrow [2013-05-05T14:00:00+00:00]>, <Arrow [2013-05-05T15:59:59.999999+00:00]>)
(<Arrow [2013-05-05T16:00:00+00:00]>, <Arrow [2013-05-05T17:59:59.999999+00:0]>)
```

tzinfo
Gets the `tzinfo` of the *Arrow* object.

datetime
Returns a datetime representation of the *Arrow* object.

naive
Returns a naive datetime representation of the *Arrow* object.

timestamp
Returns a timestamp representation of the *Arrow* object, in UTC time.

float_timestamp
Returns a floating-point representation of the *Arrow* object, in UTC time.

clone()
Returns a new *Arrow* object, cloned from the current one.

Usage:

```
>>> arw = arrow.utcnow()
>>> cloned = arw.clone()
```

replace(***kwargs*)
Returns a new *Arrow* object with attributes updated according to inputs.

Use property names to set their value absolutely:

```
>>> import arrow
>>> arw = arrow.utcnow()
>>> arw
<Arrow [2013-05-11T22:27:34.787885+00:00]>
>>> arw.replace(year=2014, month=6)
<Arrow [2014-06-11T22:27:34.787885+00:00]>
```

You can also replace the timezone without conversion, using a *timezone expression*:

```
>>> arw.replace(tzinfo=tz.tzlocal())
<Arrow [2013-05-11T22:27:34.787885-07:00]>
```

shift(***kwargs*)
Returns a new *Arrow* object with attributes updated according to inputs.

Use pluralized property names to shift their current value relatively:

```
>>> import arrow
>>> arw = arrow.utcnow()
>>> arw
<Arrow [2013-05-11T22:27:34.787885+00:00]>
>>> arw.shift(years=1, months=-1)
<Arrow [2014-04-11T22:27:34.787885+00:00]>
```

Day-of-the-week relative shifting can use either Python's weekday numbers (Monday = 0, Tuesday = 1 .. Sunday = 6) or using dateutil.relativedelta's day instances (MO, TU .. SU). When using weekday numbers, the returned date will always be greater than or equal to the starting date.

Using the above code (which is a Saturday) and asking it to shift to Saturday:

```
>>> arw.shift(weekday=5)
<Arrow [2013-05-11T22:27:34.787885+00:00]>
```

While asking for a Monday:

```
>>> arw.shift(weekday=0)
<Arrow [2013-05-13T22:27:34.787885+00:00]>
```

to(*tz*)

Returns a new *Arrow* object, converted to the target timezone.

Parameters **tz** – A *timezone expression*.

Usage:

```
>>> utc = arrow.utcnow()
>>> utc
<Arrow [2013-05-09T03:49:12.311072+00:00]>

>>> utc.to('US/Pacific')
<Arrow [2013-05-08T20:49:12.311072-07:00]>

>>> utc.to(tz.tzlocal())
<Arrow [2013-05-08T20:49:12.311072-07:00]>

>>> utc.to('-07:00')
<Arrow [2013-05-08T20:49:12.311072-07:00]>

>>> utc.to('local')
<Arrow [2013-05-08T20:49:12.311072-07:00]>

>>> utc.to('local').to('utc')
<Arrow [2013-05-09T03:49:12.311072+00:00]>
```

span(*frame*, *count=1*)

Returns two new *Arrow* objects, representing the timespan of the *Arrow* object in a given timeframe.

Parameters

- **frame** – the timeframe. Can be any `datetime` property (day, hour, minute...).
- **count** – (optional) the number of frames to span.

Supported frame values: year, quarter, month, week, day, hour, minute, second.

Usage:

```
>>> arrow.utcnow()
<Arrow [2013-05-09T03:32:36.186203+00:00]>

>>> arrow.utcnow().span('hour')
(<Arrow [2013-05-09T03:00:00+00:00]>, <Arrow [2013-05-09T03:59:59.999999+00:00]>)

>>> arrow.utcnow().span('day')
(<Arrow [2013-05-09T00:00:00+00:00]>, <Arrow [2013-05-09T23:59:59.999999+00:00]>)
```

```
>>> arrow.utcnow().span('day', count=2)
(<Arrow [2013-05-09T00:00:00+00:00]>, <Arrow [2013-05-10T23:59:59.999999+00:00]>)
```

floor (*frame*)

Returns a new *Arrow* object, representing the "floor" of the timespan of the *Arrow* object in a given timeframe. Equivalent to the first element in the 2-tuple returned by *span*.

Parameters **frame** – the timeframe. Can be any `datetime` property (day, hour, minute...).

Usage:

```
>>> arrow.utcnow().floor('hour')
<Arrow [2013-05-09T03:00:00+00:00]>
```

ceil (*frame*)

Returns a new *Arrow* object, representing the "ceiling" of the timespan of the *Arrow* object in a given timeframe. Equivalent to the second element in the 2-tuple returned by *span*.

Parameters **frame** – the timeframe. Can be any `datetime` property (day, hour, minute...).

Usage:

```
>>> arrow.utcnow().ceil('hour')
<Arrow [2013-05-09T03:59:59.999999+00:00]>
```

format (*fmt='YYYY-MM-DD HH:mm:ssZZ'*, *locale='en_us'*)

Returns a string representation of the *Arrow* object, formatted according to a format string.

Parameters **fmt** – the format string.

Usage:

```
>>> arrow.utcnow().format('YYYY-MM-DD HH:mm:ss ZZ')
'2013-05-09 03:56:47 -00:00'

>>> arrow.utcnow().format('X')
'1368071882'

>>> arrow.utcnow().format('MMMM DD, YYYY')
'May 09, 2013'

>>> arrow.utcnow().format()
'2013-05-09 03:56:47 -00:00'
```

humanize (*other=None*, *locale='en_us'*, *only_distance=False*, *granularity='auto'*)

Returns a localized, humanized representation of a relative difference in time.

Parameters

- **other** – (optional) an *Arrow* or `datetime` object. Defaults to now in the current *Arrow* object's timezone.
- **locale** – (optional) a `str` specifying a locale. Defaults to 'en_us'.
- **only_distance** – (optional) returns only time difference eg: "11 seconds" without "in" or "ago" part.
- **granularity** – (optional) defines the precision of the output. Set it to strings 'second', 'minute', 'hour', 'day', 'month' or 'year'.

Usage:

```
>>> earlier = arrow.utcnow().shift(hours=-2)
>>> earlier.humanize()
'2 hours ago'

>>> later = later = earlier.shift(hours=4)
>>> later.humanize(earlier)
'in 4 hours'
```

date()
Returns a `date` object with the same year, month and day.

time()
Returns a `time` object with the same hour, minute, second, microsecond.

timetz()
Returns a `time` object with the same hour, minute, second, microsecond and tzinfo.

astimezone(*tz*)
Returns a `datetime` object, converted to the specified timezone.

Parameters **tz** – a `tzinfo` object.

utcoffset()
Returns a `timedelta` object representing the whole number of minutes difference from UTC time.

dst()
Returns the daylight savings time adjustment.

timetuple()
Returns a `time.struct_time`, in the current timezone.

utctimetuple()
Returns a `time.struct_time`, in UTC time.

toordinal()
Returns the proleptic Gregorian ordinal of the date.

weekday()
Returns the day of the week as an integer (0-6).

isoweekday()
Returns the ISO day of the week as an integer (1-7).

isocalendar()
Returns a 3-tuple, (ISO year, ISO week number, ISO weekday).

isoformat(*sep='T'*)
Returns an ISO 8601 formatted representation of the date and time.

ctime()
Returns a ctime formatted representation of the date and time.

strftime(*format*)
Formats in the style of `datetime.strptime`.

Parameters **format** – the format string.

for_json()
Serializes for the `for_json` protocol of simplejson.

arrow.factory

Implements the `ArrowFactory` class, providing factory methods for common `Arrow` construction scenarios.

class `arrow.factory.`**`ArrowFactory`**(*type=<class 'arrow.arrow.Arrow'>*)

A factory for generating `Arrow` objects.

Parameters **`type`** – (optional) the `Arrow`-based class to construct from. Defaults to `Arrow`.

`get`(**args*, ***kwargs*)

Returns an `Arrow` object based on flexible inputs.

Parameters

- **`locale`** – (optional) a `str` specifying a locale for the parser. Defaults to 'en_us'.
- **`tzinfo`** – (optional) a *timezone expression* or tzinfo object. Replaces the timezone unless using an input form that is explicitly UTC or specifies the timezone in a positional argument. Defaults to UTC.

Usage:

```
>>> import arrow
```

No inputs to get current UTC time:

```
>>> arrow.get()
<Arrow [2013-05-08T05:51:43.316458+00:00]>
```

None to also get current UTC time:

```
>>> arrow.get(None)
<Arrow [2013-05-08T05:51:49.016458+00:00]>
```

One `Arrow` object, to get a copy.

```
>>> arw = arrow.utcnow()
>>> arrow.get(arw)
<Arrow [2013-10-23T15:21:54.354846+00:00]>
```

One `str`, `float`, or `int`, convertible to a floating-point timestamp, to get that timestamp in UTC:

```
>>> arrow.get(1367992474.293378)
<Arrow [2013-05-08T05:54:34.293378+00:00]>

>>> arrow.get(1367992474)
<Arrow [2013-05-08T05:54:34+00:00]>

>>> arrow.get('1367992474.293378')
<Arrow [2013-05-08T05:54:34.293378+00:00]>

>>> arrow.get('1367992474')
<Arrow [2013-05-08T05:54:34+00:00]>
```

One ISO-8601-formatted `str`, to parse it:

```
>>> arrow.get('2013-09-29T01:26:43.830580')
<Arrow [2013-09-29T01:26:43.830580+00:00]>
```

One `tzinfo`, to get the current time **converted** to that timezone:

```
>>> arrow.get(tz.tzlocal())
<Arrow [2013-05-07T22:57:28.484717-07:00]>
```

One naive `datetime`, to get that datetime in UTC:

```
>>> arrow.get(datetime(2013, 5, 5))
<Arrow [2013-05-05T00:00:00+00:00]>
```

One aware `datetime`, to get that datetime:

```
>>> arrow.get(datetime(2013, 5, 5, tzinfo=tz.tzlocal()))
<Arrow [2013-05-05T00:00:00-07:00]>
```

One naive `date`, to get that date in UTC:

```
>>> arrow.get(date(2013, 5, 5))
<Arrow [2013-05-05T00:00:00+00:00]>
```

Two arguments, a naive or aware `datetime`, and a replacement *timezone expression*:

```
>>> arrow.get(datetime(2013, 5, 5), 'US/Pacific')
<Arrow [2013-05-05T00:00:00-07:00]>
```

Two arguments, a naive `date`, and a replacement *timezone expression*:

```
>>> arrow.get(date(2013, 5, 5), 'US/Pacific')
<Arrow [2013-05-05T00:00:00-07:00]>
```

Two arguments, both `str`, to parse the first according to the format of the second:

```
>>> arrow.get('2013-05-05 12:30:45', 'YYYY-MM-DD HH:mm:ss')
<Arrow [2013-05-05T12:30:45+00:00]>
```

Two arguments, first a `str` to parse and second a `list` of formats to try:

```
>>> arrow.get('2013-05-05 12:30:45', ['MM/DD/YYYY', 'YYYY-MM-DD HH:mm:ss'])
<Arrow [2013-05-05T12:30:45+00:00]>
```

Three or more arguments, as for the constructor of a `datetime`:

```
>>> arrow.get(2013, 5, 5, 12, 30, 45)
<Arrow [2013-05-05T12:30:45+00:00]>
```

One time.struct time:

```
>>> arrow.get(gmtime(0))
<Arrow [1970-01-01T00:00:00+00:00]>
```

`utcnow``()`

Returns an *`Arrow`* object, representing "now" in UTC time.

Usage:

```
>>> import arrow
>>> arrow.utcnow()
<Arrow [2013-05-08T05:19:07.018993+00:00]>
```

`now``(`*tz=None*`)`

Returns an *`Arrow`* object, representing "now" in the given timezone.

Parameters **`tz`** – (optional) A *timezone expression*. Defaults to local time.

Usage:

```
>>> import arrow
>>> arrow.now()
<Arrow [2013-05-07T22:19:11.363410-07:00]>

>>> arrow.now('US/Pacific')
<Arrow [2013-05-07T22:19:15.251821-07:00]>

>>> arrow.now('+02:00')
<Arrow [2013-05-08T07:19:25.618646+02:00]>

>>> arrow.now('local')
<Arrow [2013-05-07T22:19:39.130059-07:00]>
```

arrow.api

Provides the default implementation of `ArrowFactory` methods for use as a module API.

`arrow.api.`**`get`**`(`**args*, ***kwargs*`)`
Calls the default `ArrowFactory` `get` method.

`arrow.api.`**`utcnow`**`()`
Calls the default `ArrowFactory` `utcnow` method.

`arrow.api.`**`now`**`(`*tz=None*`)`
Calls the default `ArrowFactory` `now` method.

`arrow.api.`**`factory`**`(`*type*`)`
Returns an `ArrowFactory` for the specified `Arrow` or derived type.

Parameters **`type`** – the type, `Arrow` or derived.

arrow.locale

`arrow.locales.`**`get_locale`**`(`*name*`)`
Returns an appropriate `Locale` corresponding to an inpute locale name.

Parameters **`name`** – the name of the locale.

class `arrow.locales.`**`Locale`**
Represents locale-specific data and functionality.

`describe``(`*timeframe*, *delta=0*, *only_distance=False*`)`
Describes a delta within a timeframe in plain language.

Parameters

- **`timeframe`** – a string representing a timeframe.
- **`delta`** – a quantity representing a delta in a timeframe.
- **`only_distance`** – return only distance eg: “11 seconds” without “in” or “ago” keywords

`day_name``(`*day*`)`
Returns the day name for a specified day of the week.

Parameters **`day`** – the `int` day of the week (1-7).

`day_abbreviation`(*day*)
Returns the day abbreviation for a specified day of the week.

Parameters **`day`** – the `int` day of the week (1-7).

`month_name`(*month*)
Returns the month name for a specified month of the year.

Parameters **`month`** – the `int` month of the year (1-12).

`month_abbreviation`(*month*)
Returns the month abbreviation for a specified month of the year.

Parameters **`month`** – the `int` month of the year (1-12).

`month_number`(*name*)
Returns the month number for a month specified by name or abbreviation.

Parameters **`name`** – the month name or abbreviation.

`year_full`(*year*)
Returns the year for specific locale if available

Parameters **`name`** – the `int` year (4-digit)

`year_abbreviation`(*year*)
Returns the year for specific locale if available

Parameters **`name`** – the `int` year (4-digit)

`meridian`(*hour*, *token*)
Returns the meridian indicator for a specified hour and format token.

Parameters

- **`hour`** – the `int` hour of the day.
- **`token`** – the format token.

`ordinal_number`(*n*)
Returns the ordinal format of a given integer

Parameters **`n`** – an integer

Index