CleressianDate Documentation

Lily Ellington

Jan 24, 2021

CONTENTS:

CHAPTER

ONE

CLERESSIANDATE

class CleressianDate([*grand_cycle=1*, *cycle=1*, *year=1*, *month=1*, *day=1*])

Create a CleressianDate object with the given specification.

Parameters

- **grand_cycle** (*int*) – The grand cycle component. Must be "interpretable" as an integer.
- **cycle** (*int*) – The cycle component. Must be "interpretable" as an integer between 1 and 23.
- **year** (*int*) – The year component. Must be "interpretable" as an integer between 1 and 13.
- **month** (*int or str*) – The month component. Must be either the full name of the month or interpretable as an integer between 1 and 10.
- **day** (*int*) – The day component. Must be interpretable as an integer between 1 and the maximum number of days in the given month, as returned by *days_in_month()*.

Raises

- **TypeError** – If any component is not interpretable as an integer or if *month* is not a string.
- **ValueError** – If any other validation fails.

MONTHS

A list of the month names. Note that the first element is "" so that the others are 1-indexed.

```
>>> CleressianDate.MONTHS
['', 'Sirelle', 'Tiri', 'Enna', 'Fis', 'Klesni', 'Pelio', 'Kria', 'Sui',
↪'Brilia', 'Neyu']
>>> CleressianDate.MONTHS[4]
'Fis'
```

property grand_cycle

The grand cycle component.

When set, the new value must be interpretable as an integer.

property cycle

The cycle component.

When set, the new value must be interpretable as an integer between 1 and 23.

property year

The year component.

When set, the new value must be interpretable as an integer between 1 and 13.

property month
The month component, as an integer.

When set, the new value must be interpretable as an integer between 1 and 10 or the full name of a month as a string.

property day
The day component.

When set, the new value must be interpretable as an integer between 1 and the maximum number of days in the date's month, as returned by *days_in_month()*.

static is_leap_year(*cycle*, *year*)
Determines if the given year points to a leap year.

Parameters

- **cycle** (*int*) – The cycle component.
- **year** (*int*) – The year component.

Returns `True` if _:cycle:year points to a leap year; `False` otherwise.

Return type bool

static days_in_month(*cycle*, *year*, *month*)
Return the number of days in the given month.

Parameters

- **cycle** (*int*) – The cycle component.
- **year** (*int*) – The year component.
- **month** (*int or str*) – The month component: either the month index or full month name.

Returns The number of days in the given month: 7 for Neyu in leap years, 6 for Neyu in non-leap years, and 34 for all other months.

Return type int

static days_in_year(*cycle*, *year*)
Return the number of days in the given year.

Parameters

- **cycle** (*int*) – The cycle component.
- **year** (*int*) – The year component.

Returns The number of days in the year _:cycle:year: 313 when this is a leap year, 312 otherwise.

Return type int

replace(***repl*)
Return a new `CleressianDate` object with the given replacements.

```
>>> a = CleressianDate(1, 2, 3, 4, 5)
>>> print(a)
1:2:3 Fis 5
>>> b = a.replace(grand_cycle=6, month=10)
>>> print(b)
6:2:3 Neyu 5
```

Parameters **repl** – A dictionary-like of replacements to make.

Returns A new date object with the given replacements.

Return type *CleressianDate*

Raises **TypeError or ValueError** – As described in the *initialization*.

copy()

Return a (deep) copy of this `CleressianDate` object. Note that `a.copy()` is equivalent to `a.replace()`.

This is also the implementation of `__copy__` and `__deepcopy__`, allowing this class to work with the standard library copy module.

Returns A copy of this object.

Return type *CleressianDate*

classmethod from_absolute_date(*year = 1*, *day = 1*)

An alternate constructor using the absolute year and day, instead of the "G:C:Y MMM D" standard. Note that 1:1:1 is absolute year 1.:

```
>>> a = CleressianDate.from_absolute_date(677, 43)
>>> type(a)
<class 'CleressianDate'>
>>> print(a)
3:7:1 Tiri 9
```

Parameters

- **year** (*int*) – The absolute year. `year = 1` corresponds to standard form year 1:1:1.
- **day** (*int*) – The index (1-based) of the day within the year.

Returns The date with the given absolute index.

Return type *CleressianDate*

Raises

- **TypeError** – If `year` or `day` are not numeric.
- **ValueError** – If `day` is too large to fit in the given year.

to_absolute_date()

Convert a date to an absolute date.:

```
>>> a = CleressianDate(3, 7, 1, 'Tiri', 9)
>>> a.to_absolute_date()
AbsoluteDate(year=677, day=43)
>>> a.to_absolute_date().year
677
```

Returns The absolute date, given in years and days. Absolute year 1 corresponds to standard year 1:1:1.

Return type *AbsoluteDate*

strftime(*template = "%x"*)

Convert the date object to a string following the given template. Information about the available format codes can be found *below*.

This is used for the class's string representation: `str(a) == a.strftime()`.

```
>>> a = CleressianDate(1, 2, 3, 'Fis', 5)
>>> print(a.strftime())
1:2:3 Fis 5
>>> print(a.strftime("%X"))  # absolute date form
0016.107
>>> print(a.strftime("Month '%B' has abbreviation '%b' and index %m."))
Month 'Fis' has abbreviation 'Fis' and index 4.
```

Parameters **template** (*str*) – The format template.

Returns The formatted string representation.

Return type str

Raises **TypeError** – If `template` is not a string.

classmethod strptime(*date_str*[, *template = "%x"*])

Create a new `CleressianDate` object from a string, using the given format template. Information about the available format codes can be found *below*.

Warning: Because of the tag parsing order (`Y > j > g > c > y > m > b > B > d`), if duplicate values are found (e.g., by passing `%Y` and `%g`), values found later in the *parsing* take priority, regardless of where they occur in the template string. That is, standard form values take priority over absolute form values, and full month name beats abbreviated name beats month index. In instances where these overwrites occur, a warning is issued by `logging.warning`; these can be suppressed by, e.g., setting the logger level above warning or directing the logger to /dev/null.

```
>>> CleressianDate.strptime('3:7:1 Tiri 9')
CleressianDate(grand_cycle=3, cycle=7, year=1, month=2, day=9)
>>>
>>> # partially-filled with defaults
>>> CleressianDate.strptime('Tiri 9', "%B %d")
CleressianDate(grand_cycle=1, cycle=1, year=1, month=2, day=9)
>>>
>>> # competing specifications (standard > absolute)
>>> a = CleressianDate.strptime('0677.043 / 7:16:9 Tiri 6', '%X / %x')
WARNING:root:strptime: grand_cycle overwritten: 0677 (3:7:1) -> 1873 (7:7:1)
WARNING:root:strptime: cycle overwritten: 1873 (7:7:1) -> 1990 (7:16:1)
WARNING:root:strptime: year overwritten: 1990 (7:16:1) -> 1998 (7:16:9)
WARNING:root:strptime: day overwritten: .043 (Tiri 09) -> .040 (Tiri 06)
>>> a
CleressianDate(grand_cycle=7, cycle=16, year=9, month=2, day=6)
>>>
>>> # competing specifications for month (%B > %b > %m)
>>> x = CleressianDate.strptime('m=2, b=Kle, B=Sirelle', 'm=%m, b=%b, B=%B')
WARNING:root:strptime: month overwritten: .035 (Tiri 01) -> .137 (Klesni 01)
WARNING:root:strptime: month overwritten: .137 (Klesni 01) -> .001 (Sirelle␣
↪01)
>>> x.month_name
'Sirelle'
>>>
>>> # ...even when they aren't in that order!
>>> CleressianDate.strptime("B=Sirelle, b=Kle, m=2", "B=%B, b=%b, m=%m")
```

(continues on next page)

(continued from previous page)

```
WARNING:root:strptime: month overwritten: .035 (Tiri 01) -> .137 (Klesni 01)
WARNING:root:strptime: month overwritten: .137 (Klesni 01) -> .001 (Sirelle␣
↪01)
>>> y.month_name
'Sirelle'
>>>
>>> # disabling the logger
>>> import logging
>>> logging.basicConfig(level=logging.ERROR)
>>> y = CleressianDate.strptime("B=Sirelle, b=Kle, m=2", "B=%B, b=%b, m=%m")
>>> y.month_name
'Sirelle'
```

Parameters

- **date_str** (*str*) – The date string to parse.
- **template** (*str*) – The template to use to parse. It may include any of the `%_` tags described below.

Returns The parsed CleressianDate object.

Return type *CleressianDate*

Raises

- **TypeError** – If either argument is not a string.
- **ValueError** – If the date string does not match the given format template.

__add__(*other*)

Implements `a + other`. Return the date a given number of years and days in the future.

Parameters **other** (*int or Iterable[int]*) – When this is an integer, it represents the number of days to add; when it is an iterable, it should be of the form (years, days).

Returns The date (years, days) in the future.

Return type *CleressianDate*

Raises **TypeError** – If `other` is neither an integer nor an iterable of two integers.

__sub__(*other*)

Implements `a - other`. Returns

1. the number of years/days between two dates, or
2. the date a given number of years/days in the past.

```
>>> # date - date
>>> a = CleressianDate.strptime('7:17:11 Enna 28')
>>> b = CleressianDate.strptime('7:16:9 Klesni 11')
>>> a - b
DateDelta(years=14, days=262)
>>> b - a
DateDelta(years=-14, days=-262)
>>>
>>> # date - int
>>> print(a - 3)
7:17:11 Enna 28
>>>
```

(continues on next page)

(continued from previous page)

```
>>> # date - iterable
>>> print(a - (14, 262))
7:16:9 Klesni 11
>>> print(a - DateDelta(14, 262))  # DateDelta is iterable
7:16:9 Klesni 11
```

Parameters **other** (*CleressianDate or int or Iterable[int]*) – If a CleressianDate, it represents the date to subtract. If an integer, the number of days to subtract. If an iterable, the (years, days) to subtract.

Returns The amount of time between the two dates (positive if other happened in the past) or the date (years, days) in the past.

Return type *DateDelta* or *CleressianDate*

Raises **TypeError** – If other is neither an integer nor an iterable of two integers nor a CleressianDate.

static distance(*x*, *y*)

Return the number of years and days between two dates, regardless of order.

Parameters **x, y** (CleressianDate) – the two dates

Returns The number of years and days between the two dates.

Return type *DateDelta*

Raises **TypeError** – If either of x and y are not a CleressianDate.

CHAPTER

TWO

ABSOLUTEDATE

class AbsoluteDate(*year=1*, *day=1*)

A subclass of `dataclasses.dataclass`, used mostly as a helper class. It is the return type of *CleressianDate.to_absolute_date()*.

Note: This class is iterable, producing its `year` attribute, then its `day` attribute. Thus, it is possible to do:

```
>>> a = AbsoluteDate(year=677, day=43)
>>> b = CleressianDate.from_absolute_date(*a)
>>> b.strftime("%X")
'0677.43'
```

Warning: The attributes **can** be set, but they do not have any validation built-in. Thus, blindly creating a `CleressianDate` via *from_absolute_date()* may result in a `ValueError` if the day value is too high or a `TypeError` if the attributes are set to improper types.

To be sure, use *validate()*.

year: int

day: int

validate()

Determine if this absolute date is a "valid" representation. Specifically, this determines if *CleressianDate.from_absolute_date()* would result in an error.

Returns `True` if the date is valid, `False` otherwise.

Return type bool

CHAPTER

THREE

DATEDELTA

class DateDelta

A subclass of `dataclasses.dataclass`, used mostly as a helper class. It is the return type of *CleressianDate.__sub__()*.

Note: This class is iterable, producing its `years` attribute, then its `days` attribute. Thus, it may be stylistically preferable to use it as the second argument to *CleressianDate.__add__()* or *__sub__()*.

```
>>> a = CleressianDate.strptime('7:16:9 Klesni 11')
>>> b = a + DateDelta(14, 262)  # equivalent to a + (14, 262)
>>> print(b)
7:17:11 Enna 28
>>> c = b - DateDelta(14, 262)  # equivalent to b - (14, 262) or b + (-14, -262)
>>> print(c)
7:16:9 Klesni 11
```

---

years: int

days: int

CHAPTER

FOUR

STRFTIME AND STRPTIME FORMAT TAGS

The template strings used in *CleressianDate.strftime()* and *strptime()* may include any of the following format tags:

Tag	CleressianDate Specification	1989 C Specification
%g	number of grand cycle (1, 2, 3, ...)	N/A
%c	number of cycle in grand cycle (1, 2, ..., 23)	Locale's appropriate date/time representation (e.g., en_US: Tue Aug 16 21:30:00 1988)
%y	number of year in cycle (1, 2, ..., 13)	Year without century as a zero-padded decimal number (00, 01, ..., 99)
%b	abbreviated name of month (Sir, Tir, ..., Ney)	Month as locale's abbreviated name (en_US: Jan, Feb, ..., Dec)
%B	full name of month (Sirelle, Tiri, ..., Neyu)	Month as locale's full name (en_US: January, February, ..., December)
%m	number of month (1, 2, ..., 10)	Month as zero-padded decimal number (01, 02, ..., 12)
%d	number of day in month (1, 2, ..., 34)	Day of the month as a zero-padded decimal number (01, 02, ..., 31)
%j	number of day within year (1, 2, ..., 313)	Day of the yar as a zero-padded decimal number (001, 002, ..., 366)
%Y	number of absolute year (1:1:1 = year 1)	Year with century as a decimal number (0001, 0002, ..., 2013, 2014, ..., 9998, 9999)
%%	a literal `%` symbol	A literal `%` symbol

Warning: The C89 specification is provided here for comparison. While some of the tags are the same, a few are completely different—most notably `%c` and `%X`. Even those which are similar, such as `%j`, differ in that the CleressianDate specification does not provide padded values by default.

Instead, padded values can be obtained for the numeric codes by infixing the pad between the percent symbol and the tag key. For example, for a length-four pad for the absolute year, one can write `%04Y`. In the current specifications, only zero-padding is allowed. Formally, the specification requires the following regular expression to be matched: `r'%(0\d+)?K'`, where `K` is the tag key.

Note: The CleressianDate specification also includes `%x` as a shorthand for the standard form representation and `%X` as a shorthand for the absolute form representation. That is,

- `%x` → `%g:%y:%c %B %d`
- `%X` → `%04Y.%03j`

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX