domdf-python-tools

Release 3.8.0.post2

Helpful functions for Python

Dominic Davis-Foster

May 15, 2024

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install domdf_python_tools --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
$ conda config --add channels https://conda.anaconda.org/domdfcoding
```

Then install

```
$ conda install domdf_python_tools
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/domdf_python_tools@master --user
```

Chapter

TWO

Highlights

- *utils.head()*

  ```
  head(
    obj: Union[Tuple, List, DataFrame, Series, String, HasHead],
    n: int = 10,
    ) -> Optional[str]
  ```

  Returns the head of the given object.

- *utils.strtobool()*

  ```
  strtobool(
    val: Union[str, int],
    ) -> bool
  ```

  Convert a string representation of truth to `True` or `False`.

- *paths.PathPlus*

  ```
  PathPlus(*args, **kwargs)
  ```

  Subclass of `pathlib.Path` with additional methods and a default encoding of UTF-8.

- *stringlist.DelimitedList*

  ```
  DelimitedList(iterable = ())
  ```

  Subclass of `list` that supports custom delimiters in format strings.

Part I

API Reference

Chapter

THREE

bases

Useful base classes.

Classes:

Dictable(*args, **kwargs)	The basic structure of a class that can be converted into a dictionary.
Lineup([initlist])	List-like type with fluent methods and some star players.
NamedList([initlist])	A list with a name.
UserFloat([value])	Class which simulates a float.
UserList([initlist])	Typed version of collections.UserList.

Data:

_F	Invariant TypeVar bound to *domdf_python_tools.bases.UserFloat*.
_LU	Invariant TypeVar bound to *domdf_python_tools.bases.Lineup*.
_S	Invariant TypeVar bound to *domdf_python_tools.bases.UserList*.

Functions:

namedlist([name])	A factory function to return a custom list subclass with a name.

3.1 Type Variables

_F = TypeVar(_F, bound=UserFloat)
Type: TypeVar

Invariant TypeVar bound to *domdf_python_tools.bases.UserFloat*.

_LU = TypeVar(_LU, bound=Lineup)
Type: TypeVar

Invariant TypeVar bound to *domdf_python_tools.bases.Lineup*.

_S = TypeVar(_S, bound=UserList)
Type: TypeVar

Invariant TypeVar bound to *domdf_python_tools.bases.UserList*.

_T = TypeVar(_T)
Type: TypeVar

Invariant TypeVar.

_V = TypeVar(_V)
Type: TypeVar

Invariant TypeVar.

3.2 Dictable

class Dictable(**args*, ***kwargs*)

Bases: Iterable[Tuple[str, ~_V]]

The basic structure of a class that can be converted into a dictionary.

Attributes:

__class_getitem__

Methods:

__eq__(other)	Return `self == other`.
__iter__()	Iterate over the attributes of the class.
__repr__()	Return a string representation of the *Dictable*.
__str__()	Return `str(self)`.

__class_getitem__ = <bound method GenericAlias of <class 'domdf_python_tools.bases.Dic

Type: MethodType

__eq__(*other*)

Return `self == other`.

Return type bool

__iter__()

Iterate over the attributes of the class.

Return type Iterator[Tuple[str, ~_V]]

__repr__()

Return a string representation of the *Dictable*.

Return type str

__str__()

Return `str(self)`.

Return type str

3.3 UserList

class UserList(*initlist=None*)

Bases: MutableSequence[~_T]

Typed version of `collections.UserList`.

Class that simulates a list. The instance's contents are kept in a regular list, which is accessible via the *data* attribute of *UserList* instances. The instance's contents are initially set to a copy of list, defaulting to the empty list `[]`.

New in version 0.10.0.

Parameters **initlist** (Optional[Iterable[~_T]]) – The initial values to populate the *UserList* with. Default [].

Subclassing requirements

Subclasses of *UserList* are expected to offer a constructor which can be called with either no arguments or one argument. List operations which return a new sequence attempt to create an instance of the actual implementation class. To do so, it assumes that the constructor can be called with a single parameter, which is a sequence object used as a data source.

If a derived class does not wish to comply with this requirement, all of the special methods supported by this class will need to be overridden; please consult the sources for information about the methods which need to be provided in that case.

Methods:

__add__(other)	Return `self + value`.
__contains__(item)	Return `key in self`.
__delitem__(i)	Delete `self[key]`.
__eq__(other)	Return `self == other`.
__ge__(other)	Return `self >= other`.
__getitem__(i)	Return `self[key]`.
__gt__(other)	Return `self > other`.
__le__(other)	Return `self <= other`.
__lt__(other)	Return `self < other`.
__mul__(n)	Return `self * value`.
__radd__(other)	Return `value + self`.
__repr__()	Return a string representation of the *UserList*.
__rmul__(n)	Return `self * value`.
__setitem__(i, item)	Set `self[key]` to `value`.
append(item)	Append `item` to the end of the *UserList*.
clear()	Remove all items from the *UserList*.
copy()	Returns a copy of the *UserList*.
count(item)	Returns the number of occurrences of `item` in the *UserList*.
extend(other)	Extend the *NamedList* by appending elements from `other`.
index(item, *args)	Returns the index of the fist element matching `item`.
insert(i, item)	Insert `item` at position `i` in the *UserList*.
pop([i])	Removes and returns the item at index `i`.
remove(item)	Removes the first occurrence of `item` from the list.
reverse()	Reverse the list in place.
sort(*[, key, reverse])	Sort the list in ascending order and return `None`.

Attributes:

__class_getitem__	
data	A real list object used to store the contents of the *UserList*.

__add__(*other*)

Return `self + value`.

Return type *~_S*

__class_getitem__ = <bound method GenericAlias of <class 'domdf_python_tools.bases.Use
Type: MethodType

__contains__(*item*)
Return `key in self`.

Return type bool

__delitem__(*i*)
Delete `self[key]`.

__eq__(*other*)
Return `self == other`.

Return type bool

__ge__(*other*)
Return `self >= other`.

Return type bool

__getitem__(*i*)
Return `self[key]`.

Return type Union[~_T, MutableSequence[~_T]]

Overloads

- *__getitem__*(**i**: int) -> ~_T
- *__getitem__*(**i**: slice) -> MutableSequence[~_T]

__gt__(*other*)
Return `self > other`.

Return type bool

__le__(*other*)
Return `self <= other`.

Return type bool

__lt__(*other*)
Return `self < other`.

Return type bool

__mul__(*n*)
Return `self * value`.

Return type ~_S

__radd__(*other*)
Return `value + self`.

__repr__()
Return a string representation of the *UserList*.

Return type str

__rmul__(*n*)
Return `self * value`.

Return type *~_S*

__setitem__(*i*, *item*)
Set `self[key]` to `value`.

Overloads

- *__setitem__*(**i**: int, **o**: *~_T*)
- *__setitem__*(**i**: slice, **o**: Iterable[*~_T*])

append(*item*)
Append `item` to the end of the *UserList*.

clear()
Remove all items from the *UserList*.

copy()
Returns a copy of the *UserList*.

Return type *~_S*

count(*item*)
Returns the number of occurrences of `item` in the *UserList*.

Return type int

data
Type: List[*~_T*]

A real list object used to store the contents of the *UserList*.

extend(*other*)
Extend the *NamedList* by appending elements from `other`.

Parameters **other** (Iterable[*~_T*])

index(*item*, **args*)
Returns the index of the fist element matching `item`.

Parameters

- **item** (*~_T*)
- **args**

Raises **ValueError** – if the item is not present.

Return type int

insert(*i*, *item*)
Insert `item` at position `i` in the *UserList*.

pop(*i=- 1*)
Removes and returns the item at index `i`.

Raises **IndexError** – if list is empty or index is out of range.

Return type *~_T*

remove(*item*)
Removes the first occurrence of `item` from the list.

Parameters **item** (*~_T*)

Raises **ValueError** – if the item is not present.

reverse()
Reverse the list in place.

sort(*, *key=None*, *reverse=False*)
Sort the list in ascending order and return None.

The sort is in-place (i.e. the list itself is modified) and stable (i.e. the order of two equal elements is maintained).

If a key function is given, apply it once to each list item and sort them, ascending or descending, according to their function values.

The reverse flag can be set to sort in descending order.

3.4 NamedList

Both *NamedList* and *namedlist()* can be used to create a named list.

namedlist() can be used as follows:

```
>>> ShoppingList = namedlist("ShoppingList")
>>> shopping_list = ShoppingList(["egg and bacon", "egg sausage and bacon", "egg and␣
↪spam", "egg bacon and spam"])
>>>
```

If you wish to create a subclass with additional features it is recommended to subclass from *NamedList* rather than from *namedlist()*. For example, do this:

```
>>> class ShoppingList(NamedList):
...     pass
>>>
```

and not this:

```
>>> class ShoppingList(namedlist())
...     pass
>>>
```

This avoids any potential issues with mypy.

class NamedList(*initlist=None*)
Bases: *UserList*[*~_T*]

A list with a name.

The name of the list is taken from the name of the subclass.

Changed in version 0.10.0: *NamedList* now subclasses *UserList* rather than collections.UserList.

namedlist(*name='NamedList'*)
A factory function to return a custom list subclass with a name.

Parameters **name** (str) – The name of the list. Default 'NamedList'.

Return type Type[*NamedList*]

3.5 UserFloat

class UserFloat(*value=0.0*)

Bases: Real

Class which simulates a float.

New in version 1.6.0.

Parameters **value** (Union[SupportsFloat, *SupportsIndex*, str, bytes, bytearray]) – The values to initialise the *UserFloat* with. Default 0.0.

Methods:

__abs__()	Return abs(self).
__add__(other)	Return self + value.
__bool__()	Return self != 0.
__complex__()	Return complex(self).
__divmod__(other)	Return divmod(self, value).
__eq__(other)	Return self == other.
__float__()	Return float(self).
__floordiv__(other)	Return self // value.
__ge__(other)	Return self >= other.
__gt__(other)	Return self > other.
__int__()	Return int(self).
__le__(other)	Return self <= other.
__lt__(other)	Return self < other.
__mod__(other)	Return self % value.
__mul__(other)	Return self * value.
__ne__(other)	Return self != other.
__neg__()	Return - self.
__pos__()	Return + self.
__pow__(other[, mod])	Return pow(self, value, mod).
__radd__(other)	Return value + self.
__rdivmod__(other)	Return divmod(value, self).
__repr__()	Return a string representation of the *UserFloat*.
__rfloordiv__(other)	Return value // self.
__rmod__(other)	Return value % self.
__rmul__(other)	Return value * self.
__round__([ndigits])	Round the *UserFloat* to ndigits decimal places, defaulting to 0.
__rpow__(other[, mod])	Return pow(value, self, mod).
__rsub__(other)	Return value - self.
__rtruediv__(other)	Return value / self.
__str__()	Return str(self).
__sub__(other)	Return value - self.
__truediv__(other)	Return self / value.
__trunc__()	Truncates the float to an integer.
as_integer_ratio()	Returns the float as a fraction.
fromhex(string)	Create a floating-point number from a hexadecimal string.
hex()	Returns the hexadecimal (base 16) representation of the float.
is_integer()	Returns whether the float is an integer.

__abs__()
Return `abs(self)`.

Return type *~_F*

__add__(*other*)
Return `self + value`.

Return type *~_F*

__bool__()
Return `self != 0`.

Return type `bool`

__complex__()
Return `complex(self)`.

```
complex(self) == complex(float(self), 0)
```

Return type `complex`

__divmod__(*other*)
Return `divmod(self, value)`.

Return type `Tuple`[*~_F*, *~_F*]

__eq__(*other*)
Return `self == other`.

Return type `bool`

__float__()
Return `float(self)`.

Return type `float`

__floordiv__(*other*)
Return `self // value`.

Return type *~_F*

__ge__(*other*)
Return `self >= other`.

Return type `bool`

__gt__(*other*)
Return `self > other`.

Return type `bool`

__int__()
Return `int(self)`.

Return type `int`

__le__(*other*)
Return `self <= other`.

Return type `bool`

__lt__(*other*)
Return `self < other`.

Return type `bool`

__mod__(*other*)
Return `self % value`.

Return type *~_F*

__mul__(*other*)
Return `self * value`.

Return type *~_F*

__ne__(*other*)
Return `self != other`.

Return type `bool`

__neg__()
Return `- self`.

Return type *~_F*

__pos__()
Return `+ self`.

Return type *~_F*

__pow__(*other*, *mod=None*)
Return `pow(self, value, mod)`.

Return type *~_F*

__radd__(*other*)
Return `value + self`.

Return type *~_F*

__rdivmod__(*other*)
Return `divmod(value, self)`.

Return type `Tuple`[*~_F*, *~_F*]

__repr__()
Return a string representation of the *UserFloat*.

Return type str

__rfloordiv__(*other*)
Return value // self.

Return type ~_F

__rmod__(*other*)
Return value % self.

Return type ~_F

__rmul__(*other*)
Return value * self.

Return type ~_F

__round__(*ndigits=None*)
Round the *UserFloat* to ndigits decimal places, defaulting to 0.

If ndigits is omitted or None, returns an int, otherwise returns a float. Rounds half toward even.

Parameters **ndigits** (Optional[int]) – Default None.

Return type Union[int, float]

__rpow__(*other*, *mod=None*)
Return pow(value, self, mod).

Return type ~_F

__rsub__(*other*)
Return value - self.

Return type ~_F

__rtruediv__(*other*)
Return value / self.

Return type ~_F

__str__()
Return str(self).

Return type str

__sub__(*other*)
Return value - self.

Return type ~_F

__truediv__(*other*)
Return `self / value`.

Return type *~_F*

__trunc__()
Truncates the float to an integer.

Return type `int`

as_integer_ratio()
Returns the float as a fraction.

Return type `Tuple`[`int`, `int`]

classmethod fromhex(*string*)
Create a floating-point number from a hexadecimal string.

Parameters **string** (`str`)

Return type *~_F*

hex()
Returns the hexadecimal (base 16) representation of the float.

Return type `str`

is_integer()
Returns whether the float is an integer.

Return type `bool`

3.6 Lineup

class Lineup(*initlist=None*)

Bases: *UserList*[*~_T*]

List-like type with fluent methods and some star players.

Methods:

append(item)	Append `item` to the end of the *UserList*.
clear()	Remove all items from the *UserList*.
extend(other)	Extend the *NamedList* by appending elements from `other`.
insert(i, item)	Insert `item` at position `i` in the *UserList*.
remove(item)	Removes the first occurrence of `item` from the list.
replace(what, with_)	Replace the first instance of `what` with `with_`.
reverse()	Reverse the list in place.
sort(*[, key, reverse])	Sort the list in ascending order and return the self.

append(*item*)

Append `item` to the end of the *UserList*.

Return type *~_LU*

clear()

Remove all items from the *UserList*.

Return type *~_LU*

extend(*other*)

Extend the *NamedList* by appending elements from `other`.

Parameters **other** (`Iterable`[*~_T*])

Return type *~_LU*

insert(*i*, *item*)

Insert `item` at position `i` in the *UserList*.

Return type *~_LU*

remove(*item*)

Removes the first occurrence of `item` from the list.

Parameters **item** (*~_T*)

Return type *~_LU*

Raises **ValueError** – if the item is not present.

replace(*what*, *with_*)
Replace the first instance of `what` with `with_`.

Parameters

- **what** (*~_T*) – The object to find and replace.
- **with_** (*~_T*) – The new value for the position in the list.

Return type *~_LU*

reverse()
Reverse the list in place.

Return type *~_LU*

sort(*, *key=None*, *reverse=False*)
Sort the list in ascending order and return the self.

The sort is in-place (i.e. the list itself is modified) and stable (i.e. the order of two equal elements is maintained).

If a key function is given, apply it once to each list item and sort them, ascending or descending, according to their function values.

The reverse flag can be set to sort in descending order.

Return type *~_LU*

Chapter

FOUR

compat

Cross-version compatibility helpers.

New in version 0.12.0.

Provides the following:

PYPY
Type: bool

True if running on PyPy rather than CPython.

New in version 2.3.0.

PYPY36
Type: bool

True if running on PyPy 3.6.

New in version 2.6.0.

PYPY37
Type: bool

True if running on PyPy 3.7.

New in version 2.6.0.

PYPY37_PLUS
Type: bool

True if running on PyPy 3.7 or newer.

New in version 3.3.0.

PYPY38
Type: bool

True if running on PyPy 3.8.

New in version 3.2.0.

PYPY38_PLUS
Type: bool

True if running on PyPy 3.8 or newer.

New in version 3.3.0.

PYPY39
Type: bool

True if running on PyPy 3.9.

New in version 3.3.0.

PYPY39_PLUS
Type: bool

True if running on PyPy 3.9 or newer.

New in version 3.3.0.

importlib_resources
importlib_resources on Python 3.6; importlib.resources on Python 3.7 and later.

importlib_metadata
importlib_metadata on Python 3.8 and earlier; `importlib.metadata` on Python 3.9 and later.

New in version 1.1.0.

Changed in version 2.5.0: importlib_metadata is now used on Python 3.8 in place of the stdlib version.

class nullcontext(*enter_result=None*)
Bases: `ContextManager`[`Optional`[`~_T`]]

Context manager that does no additional processing.

Used as a stand-in for a normal context manager, when a particular block of code is only sometimes used with a normal context manager:

```
cm = optional_cm if condition else nullcontext()
with cm:
    # Perform operation, using optional_cm if condition is True
```

New in version 2.1.0.

In Python 3.7 and above the version from the standard library is used instead of this one, but the implementations are identical.

Parameters **enter_result** (`Optional`[`~_T`]) – An optional value to return when entering the context. Default `None`.

Chapter

FIVE

dates

Utilities for working with dates and times.

Attention: This module has the following additional None:

```
pytz>=2019.1
```

This can be installed as follows:

```
$ python -m pip install domdf-python-tools[dates]
```

Data:

months	Mapping of 3-character shortcodes to full month names.
month_full_names	List of the full names for months in the Gregorian calendar.
month_short_names	List of the short names for months in the Gregorian calendar.

Functions:

calc_easter(year)	Returns the date of Easter in the given year.
check_date(month, day[, leap_year])	Returns `True` if the day number is valid for the given month.
current_tzinfo()	Returns a tzinfo object for the current timezone.
get_month_number(month)	Returns the number of the given month.
get_timezone(tz[, date])	Returns a localized `pytz.timezone` object for the given date.
get_utc_offset(tz[, date])	Returns the offset between UTC and the requested timezone on the given date.
is_bst(the_date)	Calculates whether the given day falls within British Summer Time.
parse_month(month)	Converts an integer or shorthand month into the full month name.
set_timezone(obj, tzinfo)	Sets the timezone / tzinfo of the given `datetime.datetime` object.
utc_timestamp_to_datetime(utc_timestamp[, ...])	Convert UTC timestamp (seconds from UNIX epoch) to a `datetime.datetime` object.

calc_easter(*year*)

Returns the date of Easter in the given year.

New in version 1.4.0.

Parameters **year** (`int`)

Return type `date`

check_date(*month*, *day*, *leap_year=True*)
Returns `True` if the day number is valid for the given month.

Note: This function will return `True` for the 29th Feb. If you don't want this behaviour set `leap_year` to `False`.

Parameters

- **month** (`Union`[`str`, `int`]) – The month to test.
- **day** (`int`) – The day number to test.
- **leap_year** (`bool`) – Whether to return `True` for 29th Feb. Default `True`.

Return type `bool`

current_tzinfo()
Returns a tzinfo object for the current timezone.

Return type `Optional`[`tzinfo`]

get_month_number(*month*)
Returns the number of the given month. If `month` is already a number between 1 and 12 it will be returned immediately.

Parameters **month** (`Union`[`str`, `int`]) – The month to convert to a number

Return type `int`

Returns The number of the month

get_timezone(*tz*, *date=None*)
Returns a localized `pytz.timezone` object for the given date.

If `date` is `None` then the current date is used.

Parameters

- **tz** (`str`) – A string representing a pytz timezone
- **date** (`Optional`[`datetime`]) – The date to obtain the timezone for. Default `None`.

Return type `Optional`[`tzinfo`]

get_utc_offset(*tz*, *date=None*)
Returns the offset between UTC and the requested timezone on the given date. If `date` is `None` then the current date is used.

Parameters

- **tz** (`Union`[`tzinfo`, `str`]) – `pytz.timezone` or a string representing the timezone
- **date** (`Optional`[`datetime`]) – The date to obtain the UTC offset for. Default `None`.

Return type `Optional`[`timedelta`]

is_bst(*the_date*)

Calculates whether the given day falls within British Summer Time.

This function should also be applicable to other timezones which change to summer time on the same date (e.g. Central European Summer Time).

Note: This function does not consider the time of day, and therefore does not handle the fact that the time changes at 1 AM GMT. It also does not account for historic deviations from the current norm.

New in version 3.5.0.

Parameters **the_date** (Union[struct_time, date]) – A time.struct_time, datetime.date or datetime.datetime representing the target date.

Return type bool

Returns True if the date falls within British Summer Time, False otherwise.

parse_month(*month*)

Converts an integer or shorthand month into the full month name.

Parameters **month** (Union[str, int]) – The month number or shorthand name

Return type str

Returns The full name of the month

set_timezone(*obj*, *tzinfo*)

Sets the timezone / tzinfo of the given datetime.datetime object. This will not convert the time (i.e. the hours will stay the same). Use datetime.datetime.astimezone() to accomplish that.

Parameters

- **obj** (datetime)
- **tzinfo** (tzinfo)

Return type datetime

utc_timestamp_to_datetime(*utc_timestamp*, *output_tz=None*)

Convert UTC timestamp (seconds from UNIX epoch) to a datetime.datetime object.

If output_tz is None the timestamp is converted to the platform's local date and time, and the local timezone is inferred and set for the object.

If output_tz is not None, it must be an instance of a datetime.tzinfo subclass, and the timestamp is converted to output_tz's time zone.

Parameters

- **utc_timestamp** (Union[float, int]) – The timestamp to convert to a datetime object
- **output_tz** (Optional[tzinfo]) – The timezone to output the datetime object for. If None it will be inferred. Default None.

Return type datetime

Returns The timestamp as a datetime object.

Raises **OverflowError** – if the timestamp is out of the range of values supported by the platform C localtime() or gmtime() functions, and OSError on localtime() or gmtime() failure. It's common for this to be restricted to years in 1970 through 2038.

months

Type: OrderedDict[str, str]

Mapping of 3-character shortcodes to full month names.

Essentially:

```
months = {
    Jan="January",
    Feb="February",
    Mar="March",
    Apr="April",
    May="May",
    ...,
    Dec="December",
}
```

month_full_names = ('January', 'February', ..., 'December')

Type: tuple

List of the full names for months in the Gregorian calendar.

New in version 2.0.0.

month_short_names = ('Jan', 'Feb', 'Mar', ..., 'Dec')

Type: tuple

List of the short names for months in the Gregorian calendar.

New in version 2.0.0.

Chapter

SIX

delegators

Decorators for functions that delegate parts of their functionality to other functions.

New in version 0.10.0.

Data:

_C	Invariant `TypeVar` bound to `typing.Callable`.

Functions:

delegate_kwargs(to, *except_)	Decorator to replace `**kwargs` in function signatures with the parameter names from the delegated function.
delegates(to)	Decorator to replace `*args, **kwargs` function signatures with the signature of the delegated function.

_C = TypeVar(_C, bound=typing.Callable)

Type: `TypeVar`

Invariant `TypeVar` bound to `typing.Callable`.

delegate_kwargs(*to*, **except_*)

Decorator to replace `**kwargs` in function signatures with the parameter names from the delegated function.

Parameters

- **to** (`Callable`) – The function **kwargs is passed on to.
- ***except_** (`str`) – Parameter names not to delegate.

Raises **ValueError** – if a non-default argument follows a default argument.

Return type `Callable`[[`~_C`], `~_C`]

delegates(*to*)

Decorator to replace `*args, **kwargs` function signatures with the signature of the delegated function.

Parameters **to** (`Callable`) – The function the arguments are passed on to.

Return type `Callable`[[`~_C`], `~_C`]

Chapter

SEVEN

doctools

Utilities for documenting functions, classes and methods.

Data:

_F	Invariant TypeVar bound to typing.Callable[..., typing.Any].
_T	Invariant TypeVar bound to typing.Type.

Functions:

append_docstring_from(original)	Decorator to appends the docstring from the original function to the target function.
append_doctring_from_another(target, original)	Sets the docstring of the target function to that of the original function.
deindent_string(string)	Removes all indentation from the given string.
document_object_from_another(target, original)	Sets the docstring of the target function to that of the original function.
is_documented_by(original)	Decorator to set the docstring of the target function to that of the original function.
make_sphinx_links(input_string[, builtins_list])	Make proper sphinx links out of double-backticked strings in docstring.
prettify_docstrings(obj)	Decorator to prettify the default object docstrings for use in Sphinx documentation.
sphinxify_docstring()	Decorator to make proper sphinx links out of double-backticked strings in the docstring.

_F = TypeVar(_F, bound=typing.Callable[..., typing.Any])

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., typing.Any].

_T = TypeVar(_T, bound=typing.Type)

Type: TypeVar

Invariant TypeVar bound to typing.Type.

append_docstring_from(*original*)

Decorator to appends the docstring from the original function to the target function.

This may be useful for subclasses or wrappers that use the same arguments.

Any indentation in either docstring is removed to ensure consistent indentation between the two docstrings. Bear this in mind if additional indentation is used in the docstring.

Parameters **original** (Callable)

Return type Callable[[~_F], ~_F]

append_doctring_from_another(*target*, *original*)
Sets the docstring of the `target` function to that of the `original` function.

This may be useful for subclasses or wrappers that use the same arguments.

Any indentation in either docstring is removed to ensure consistent indentation between the two docstrings. Bear this in mind if additional indentation is used in the docstring.

Parameters

- **target** (Union[Type, Callable]) – The object to append the docstring to
- **original** (Union[Type, Callable]) – The object to copy the docstring from

deindent_string(*string*)
Removes all indentation from the given string.

Parameters **string** (Optional[str]) – The string to deindent

Return type str

Returns The string without indentation

document_object_from_another(*target*, *original*)
Sets the docstring of the `target` function to that of the `original` function.

This may be useful for subclasses or wrappers that use the same arguments.

Parameters

- **target** (Union[Type, Callable]) – The object to set the docstring for
- **original** (Union[Type, Callable]) – The object to copy the docstring from

is_documented_by(*original*)
Decorator to set the docstring of the `target` function to that of the `original` function.

This may be useful for subclasses or wrappers that use the same arguments.

Parameters **original** (Callable)

Return type Callable[[~_F], ~_F]

make_sphinx_links(*input_string*, *builtins_list=None*)
Make proper sphinx links out of double-backticked strings in docstring.

i.e. ``str`` becomes :class:`str`

Make sure to include the following in your `conf.py` file for Sphinx:

```
intersphinx_mapping = {"python": ("https://docs.python.org/3/", None)}
```

Parameters

- **input_string** (str) – The string to process.
- **builtins_list** (Optional[Sequence[str]]) – A list of builtins to make links for. Default dir(builtins).

Return type str

Returns Processed string with links.

prettify_docstrings(*obj*)

Decorator to prettify the default `object` docstrings for use in Sphinx documentation.

New in version 0.8.0.

Parameters **obj** (*~_T*) – The object to prettify the method docstrings for.

Return type *~_T*

sphinxify_docstring()

Decorator to make proper sphinx links out of double-backticked strings in the docstring.

i.e. ``str`` becomes `:class:`str``

Make sure to include the following in your `conf.py` file for Sphinx:

```
intersphinx_mapping = {
    "python": ("https://docs.python.org/3/", None),
}
```

Return type Callable[[*~_F*], *~_F*]

Chapter

EIGHT

getters

Variants of operator.attrgetter(), operator.itemgetter() and operator.methodcaller() which operate on values within sequences.

New in version 3.2.0.

Classes:

attrgetter(idx, attr)	Returns a callable object that fetches attr from the item at index idx in its operand.
itemgetter(idx, item)	Returns a callable object that fetches item from the item at index idx in its operand, using the __getitem__() method.
methodcaller(_idx, _name, *args, **kwargs)	Returns a callable object that calls the method name on the item at index idx in its operand.

class attrgetter(*idx*, *attr*)

Returns a callable object that fetches attr from the item at index idx in its operand.

The attribute name can contain dots. For example:

- After f = attrgetter(0, 'name'), the call call f(b) returns b[0].name.
- After f = attrgetter(3, 'name.first'), the call f(b) returns b[3].name.first.

```
>>> from pathlib import Path
>>> attrgetter(0, 'name')([Path("dir/code.py")])
'code.py'
>>> attrgetter(2, 'parent.name')([Path("dir/coincidence.py"), Path("dir/wheel.py
↪"), Path("dir/operator.py")])
'dir'
```

See also: operator.attrgetter() and operator.itemgetter()

Parameters

- **idx** (int) – The index of the item to obtain the attribute from.
- **attr** (str) – The name of the attribute.

class itemgetter(*idx*, *item*)

Returns a callable object that fetches item from the item at index idx in its operand, using the __getitem__() method.

For example:

- After f = itemgetter(0, 2), the call call f(r) returns r[0][2].
- After g = itemgetter(3, 5), the call g(r) returns r[3][5].

The items can be any type accepted by the item's `__getitem__()` method. Dictionaries accept any hashable value. Lists, tuples, and strings accept an index or a slice:

```
>>> itemgetter(0, 1)(['ABCDEFG'])
'B'
>>> itemgetter(1, 2)(['ABC', 'DEF'])
'F'
>>> itemgetter(0, slice(2, None))(['ABCDEFG'])
'CDEFG'
>>> army = [dict(rank='captain', name='Blackadder'), dict(rank='Private', name=
↪'Baldrick')]
>>> itemgetter(0, 'rank')(army)
'captain'
```

See also: `operator.itemgetter()`

Parameters

- **idx** (`int`) – The index of the item to call `__getitem__()` on.
- **item** (`Any`) – The value to pass to `__getitem__()`.

class methodcaller(*_idx*, *_name*, **args*, ***kwargs*)

Returns a callable object that calls the method name on the item at index `idx` in its operand.

If additional arguments and/or keyword arguments are given, they will be passed to the method as well. For example:

- After `f = methodcaller(0, 'name')`, the call `f(b)` returns `b[0].name()`.
- After `f = methodcaller(1, 'name', 'foo', bar=1)`, the call `f(b)` returns `b[1].name('foo', bar=1)`.

```
>>> from datetime import date
>>> methodcaller(0, 'upper')(["hello", "world"])
'HELLO'
>>> methodcaller(1, 'center', 9, "=")(["hello", "world"])
'==world=='
>>> methodcaller(0, 'replace', year=2019)([date(2021, 7, 6)])
datetime.date(2019, 7, 6)
```

See also: `operator.methodcaller()` and `operator.itemgetter()`

Parameters

- **_idx** – The index of the item to call the method on.
- **_attr** – The name of the method to call.
- ***args** – Positional arguments to pass to the method.
- ****kwargs** – Keyword arguments to pass to the method.

Chapter

NINE

import_tools

Functions for importing classes.

New in version 0.5.0.

Functions:

discover(package[, match_func, ...])	Returns a list of objects in the given package, optionally filtered by `match_func`.
discover_entry_points(group_name[, match_func])	Returns a list of entry points in the given category, optionally filtered by `match_func`.
discover_entry_points_by_name(group_name[, ...])	Returns a mapping of entry point names to the entry points in the given category, optionally filtered by `name_match_func` and `object_match_func`.
discover_in_module(module[, match_func, ...])	Returns a list of objects in the given module, optionally filtered by `match_func`.
iter_submodules(module)	Returns an iterator over the names of the submodules and subpackages of the given module.

discover(*package*, *match_func=None*, *exclude_side_effects=True*)

Returns a list of objects in the given package, optionally filtered by `match_func`.

Parameters

- **package** (`ModuleType`) – A Python package
- **match_func** (`Optional[Callable[[Any], bool]]`) – Function taking an object and returning `True` if the object is to be included in the output. Default `None`, which includes all objects.
- **exclude_side_effects** (`bool`) – Don't include objects that are only there because of an import side effect. Default `True`.

Return type `List[Any]`

Overloads

- *discover*(**package**, **match_func**: `Optional[Callable[[Any], bool]]` = ..., **exclude_side_effects**: `Literal[True]` = ...) -> `List[Type[Any]]`
- *discover*(**package**, **match_func**: `Optional[Callable[[Any], bool]]` = ..., **exclude_side_effects**: `Literal[False]` = ...)

Changed in version 1.0.0: Added the `exclude_side_effects` parameter.

discover_entry_points(*group_name*, *match_func=None*)

Returns a list of entry points in the given category, optionally filtered by `match_func`.

New in version 1.1.0.

Parameters

- **group_name** (`str`) – The entry point group name, e.g. `'entry_points'`.
- **match_func** (`Optional[Callable[[Any], bool]]`) – Function taking an object and returning `True` if the object is to be included in the output. Default `None`, which includes all objects.

Return type `List[Any]`

Returns List of matching objects.

discover_entry_points_by_name(*group_name*, *name_match_func=None*, *object_match_func=None*)

Returns a mapping of entry point names to the entry points in the given category, optionally filtered by `name_match_func` and `object_match_func`.

New in version 2.5.0.

Parameters

- **group_name** (`str`) – The entry point group name, e.g. `'entry_points'`.
- **name_match_func** (`Optional[Callable[[Any], bool]]`) – Function taking the entry point name and returning `True` if the entry point is to be included in the output. Default `None`, which includes all entry points.
- **object_match_func** (`Optional[Callable[[Any], bool]]`) – Function taking an object and returning `True` if the object is to be included in the output. Default `None`, which includes all objects.

Return type `Dict[str, Any]`

discover_in_module(*module*, *match_func=None*, *exclude_side_effects=True*)

Returns a list of objects in the given module, optionally filtered by `match_func`.

New in version 2.6.0.

Parameters

- **module** (`ModuleType`) – A Python module.
- **match_func** (`Optional[Callable[[Any], bool]]`) – Function taking an object and returning `True` if the object is to be included in the output. Default `None`, which includes all objects.
- **exclude_side_effects** (`bool`) – Don't include objects that are only there because of an import side effect. Default `True`.

Return type `List[Any]`

iter_submodules(*module*)

Returns an iterator over the names of the submodules and subpackages of the given module.

New in version 2.6.0.

Parameters **module** (`str`)

Return type `Iterator[str]`

Chapter

TEN

iterative

Functions for iteration, looping etc.

New in version 1.4.0.

Data:

AnyNum	Invariant `TypeVar` constrained to `float` and `complex`.

Functions:

Len(obj[, start, step])	Shorthand for `range(len(obj))`.
chunks(l, n)	Yield successive `n`-sized chunks from `l`.
count([start, step])	Make an iterator which returns evenly spaced values starting with number `start`.
double_chain(iterable)	Flatten a list of lists of lists into a single list.
extend(sequence, minsize)	Extend `sequence` by repetition until it is at least as long as `minsize`.
extend_with(sequence, minsize, with_)	Extend `sequence` by adding `with\_` to the right hand end until it is at least as long as `minsize`.
extend_with_none(sequence, minsize)	Extend `sequence` by adding `None` to the right hand end until it is at least as long as `minsize`.
flatten(iterable[, primitives])	Flattens a mixed list of primitive types and iterables of those types into a single list, regardless of nesting.
groupfloats(iterable[, step])	Returns an iterator over the discrete ranges of values in `iterable`.
make_tree(tree)	Returns the string representation of a mixed list of strings and lists of strings, similar to *tree(1)*.
natmax(seq[, key, alg])	Returns the maximum value from `seq` when sorted naturally.
natmin(seq[, key, alg])	Returns the minimum value from `seq` when sorted naturally.
permutations(data[, n])	Return permutations containing `n` items from `data` without any reverse duplicates.
ranges_from_iterable(iterable[, step])	Returns an iterator over the minimum and maximum values for each discrete ranges of values in `iterable`.
split_len(string, n)	Split `string` every `n` characters.

AnyNum = TypeVar(AnyNum, float, complex)

Type: `TypeVar`

Invariant `TypeVar` constrained to `float` and `complex`.

Len(*obj*, *start=0*, *step=1*)
Shorthand for `range(len(obj))`.

Returns an object that produces a sequence of integers from `start` (inclusive) to `len(obj)` (exclusive) by `step`.

New in version 0.4.7.

Parameters

- **obj** (`Sized`) – The object to iterate over the length of.
- **start** (`int`) – The start value of the range. Default `0`.
- **step** (`int`) – The step of the range. Default `1`.

Return type `range`

Changed in version 1.4.0: Moved from *domdf_python_tools.utils*

chunks(*l*, *n*)
Yield successive `n`-sized chunks from `l`.

Parameters

- **l** (`Sequence`[~_T]) – The objects to yield chunks from.
- **n** (`int`) – The size of the chunks.

Return type `Iterator`[`Sequence`[~_T]]

Changed in version 1.4.0: Moved from *domdf_python_tools.utils*

count(*start=0*, *step=1*)
Make an iterator which returns evenly spaced values starting with number `start`.

Often used as an argument to `map()` to generate consecutive data points. Can also be used with `zip()` to add sequence numbers.

New in version 2.7.0.

Parameters

- **start** (*~AnyNum*) – Default `0`.
- **step** (*~AnyNum*) – The step between values. Default `1`.

Return type `Iterator`[*~AnyNum*]

See also:

`itertools.count()`.

The difference is that this returns more exact floats, whereas the values from `itertools.count()` drift.

double_chain(*iterable*)

Flatten a list of lists of lists into a single list.

Literally just:

```
chain.from_iterable(chain.from_iterable(iterable))
```

Will convert

```
[[(1, 2), (3, 4)], [(5, 6), (7, 8)]]
```

to

```
[1, 2, 3, 4, 5, 6, 7, 8]
```

New in version 0.4.7.

Parameters **iterable** (Iterable[Iterable[Iterable[~_T]]]) – The iterable to chain.

Return type Iterator[~_T]

Changed in version 1.4.0: Moved from *domdf_python_tools.utils*

extend(*sequence*, *minsize*)

Extend `sequence` by repetition until it is at least as long as `minsize`.

New in version 2.3.0.

Parameters

- **sequence** (Iterable[~_T])
- **minsize** (int)

Return type List[~_T]

See also: *extend_with()* and *extend_with_none()*

extend_with(*sequence*, *minsize*, *with_*)

Extend `sequence` by adding `with\_` to the right hand end until it is at least as long as `minsize`.

New in version 2.3.0.

Parameters

- **sequence** (Iterable[~_T])
- **minsize** (int)
- **with_** (~_T)

Return type List[~_T]

See also: *extend()* and *extend_with_none()*

extend_with_none(*sequence*, *minsize*)

Extend `sequence` by adding `None` to the right hand end until it is at least as long as `minsize`.

New in version 2.3.0.

Parameters

- **sequence** (`Iterable[~_T]`)
- **minsize** (`int`)

Return type `Sequence[Optional[~_T]]`

See also: *extend()* and *extend_with()*

flatten(*iterable*, *primitives=(<class 'str'>, <class 'int'>, <class 'float'>)*)

Flattens a mixed list of primitive types and iterables of those types into a single list, regardless of nesting.

New in version 1.4.0.

Parameters

- **iterable** (`Iterable[~_T]`)
- **primitives** (`Tuple[Type, ...]`) – The primitive types to allow. Default `(<class 'str'>, <class 'int'>, <class 'float'>)`.

Return type `Iterator[~_T]`

groupfloats(*iterable*, *step=1*)

Returns an iterator over the discrete ranges of values in `iterable`.

For example:

```
>>> list(groupfloats(
...     [170.0, 170.05, 170.1, 170.15, 171.05, 171.1, 171.15, 171.2],
...     step=0.05,
... ))
[(170.0, 170.05, 170.1, 170.15), (171.05, 171.1, 171.15, 171.2)]
>>> list(groupfloats([1, 2, 3, 4, 5, 7, 8, 9, 10]))
[(1, 2, 3, 4, 5), (7, 8, 9, 10)]
```

New in version 2.0.0.

Parameters

- **iterable** (`Iterable[float]`)
- **step** (`float`) – The step between values in `iterable`. Default `1`.

Return type `Iterable[Tuple[float, ...]]`

See also: *ranges_from_iterable()*, which returns an iterator over the min and max values for each range.

make_tree(*tree*)

Returns the string representation of a mixed list of strings and lists of strings, similar to *tree(1)*.

New in version 1.4.0.

Parameters **tree** (`Union[Sequence[str], Sequence[Union[Sequence[str], Sequence]]]`)

Return type `Iterator[str]`

natmax(*seq*, *key=None*, *alg=<ns.DEFAULT: 0>*)

Returns the maximum value from `seq` when sorted naturally.

New in version 1.8.0.

Parameters

- **seq** (`Iterable[~_T]`)
- **key** (`Optional[Callable[[Any], Any]]`) – A key used to determine how to sort each element of the iterable. It is **not** applied recursively. The callable should accept a single argument and return a single value. Default `None`.
- **alg** (`int`) – This option is used to control which algorithm `natsort` uses when sorting. Default `<ns.DEFAULT: 0>`.

Return type `~_T`

natmin(*seq*, *key=None*, *alg=<ns.DEFAULT: 0>*)

Returns the minimum value from `seq` when sorted naturally.

New in version 1.8.0.

Parameters

- **seq** (`Iterable[~_T]`)
- **key** (`Optional[Callable[[Any], Any]]`) – A key used to determine how to sort each element of the iterable. It is **not** applied recursively. The callable should accept a single argument and return a single value. Default `None`.
- **alg** (`int`) – This option is used to control which algorithm `natsort` uses when sorting. Default `<ns.DEFAULT: 0>`.

Return type `~_T`

permutations(*data*, *n=2*)

Return permutations containing `n` items from `data` without any reverse duplicates.

If `n` is equal to or greater than the length of the data an empty list of returned.

Parameters

- **data** (`Iterable[~_T]`)
- **n** (`int`) – Default `2`.

Return type `List[Tuple[~_T, ...]]`

Changed in version 1.4.0: Moved from *domdf_python_tools.utils*

See also: `itertools.permutations()` and `itertools.combinations()`

ranges_from_iterable(*iterable*, *step=1*)

Returns an iterator over the minimum and maximum values for each discrete ranges of values in `iterable`.

For example:

```
>>> list(ranges_from_iterable([170.0, 170.05, 170.1, 170.15, 171.05, 171.1, 171.
↪15, 171.2], step=0.05))
[(170.0, 170.15), (171.05, 171.2)]
>>> list(ranges_from_iterable([1, 2, 3, 4, 5, 7, 8, 9, 10]))
[(1, 5), (7, 10)]
```

Parameters

- **iterable** (Iterable[float])
- **step** (float) – The step between values in `iterable`. Default `1`.

Return type Iterable[Tuple[float, float]]

split_len(*string*, *n*)

Split `string` every `n` characters.

Parameters

- **string** (str)
- **n** (int) – The number of characters to split after

Return type List[str]

Returns The split string

Changed in version 1.4.0: Moved from *domdf_python_tools.utils*

Chapter

ELEVEN

paths

Functions for paths and files.

Changed in version 1.0.0: Removed `relpath2`. Use *domdf_python_tools.paths.relpath()* instead.

Classes:

DirComparator(a, b[, ignore, hide])	Compare the content of `a` and `a`.
PathPlus(*args, **kwargs)	Subclass of `pathlib.Path` with additional methods and a default encoding of UTF-8.
PosixPathPlus(*args, **kwargs)	*PathPlus* subclass for non-Windows systems.
TemporaryPathPlus([suffix, prefix, dir])	Securely creates a temporary directory using the same rules as `tempfile.mkdtemp()`.
WindowsPathPlus(*args, **kwargs)	*PathPlus* subclass for Windows systems.

Data:

_P	Invariant `TypeVar` bound to `pathlib.Path`.
_PP	Invariant `TypeVar` bound to *domdf_python_tools.paths.PathPlus*.
unwanted_dirs	A list of directories which will likely be unwanted when searching directory trees for files.

Functions:

append(var, filename, **kwargs)	Append `var` to the file `filename` in the current directory.
clean_writer(string, fp)	Write string to `fp` without trailing spaces.
compare_dirs(a, b)	Compare the content of two directory trees.
copytree(src, dst[, symlinks, ignore])	Alternative to `shutil.copytree()` to support copying to a directory that already exists.
delete(filename, **kwargs)	Delete the file in the current directory.
in_directory(directory)	Context manager to change into the given directory for the duration of the `with` block.
make_executable(filename)	Make the given file executable.
matchglob(filename, pattern[, matchcase])	Given a filename and a glob pattern, return whether the filename matches the glob.
maybe_make(directory[, mode, parents])	Create a directory at the given path, but only if the directory does not already exist.
parent_path(path)	Returns the path of the parent directory for the given file or directory.
read(filename, **kwargs)	Read a file in the current directory (in text mode).
relpath(path[, relative_to])	Returns the path for the given file or directory relative to the given directory or, if that would require path traversal, returns the absolute path.

continues on next page

Table 3 – continued from previous page

sort_paths(*paths)	Sort the `paths` by directory, then by file.
traverse_to_file(base_directory, *filename)	Traverse the parents of the given directory until the desired file is found.
write(var, filename, **kwargs)	Write a variable to file in the current directory.

class DirComparator(*a*, *b*, *ignore=None*, *hide=None*)

Bases: `dircmp`

Compare the content of `a` and `a`.

In contrast with `filecmp.dircmp`, this subclass compares the content of files with the same path.

New in version 2.7.0.

Parameters

- **a** (`Union`[`str`, `Path`, `PathLike`]) – The "left" directory to compare.
- **b** (`Union`[`str`, `Path`, `PathLike`]) – The "right" directory to compare.
- **ignore** (`Optional`[`Sequence`[`str`]]) – A list of names to ignore. Default `filecmp.DEFAULT_IGNORES`.
- **hide** (`Optional`[`Sequence`[`str`]]) – A list of names to hide. Default `[ os.curdir, os.pardir ]`.

class PathPlus(**args*, ***kwargs*)

Bases: `Path`

Subclass of `pathlib.Path` with additional methods and a default encoding of UTF-8.

Path represents a filesystem path but, unlike `pathlib.PurePath`, also offers methods to do system calls on path objects. Depending on your system, instantiating a *PathPlus* will return either a *PosixPathPlus* or a *WindowsPathPlus*. object. You can also instantiate a *PosixPathPlus* or `WindowsPath` directly, but cannot instantiate a *WindowsPathPlus* on a POSIX system or vice versa.

New in version 0.3.8.

Changed in version 0.5.1: Defaults to Unix line endings (`LF`) on all platforms.

Methods:

abspath()	Return the absolute version of the path.
append_text(string[, encoding, errors])	Open the file in text mode, append the given string to it, and close the file.
dump_json(data[, encoding, errors, ...])	Dump `data` to the file as JSON.
from_uri(uri)	Construct a *PathPlus* from a `file` URI returned by `pathlib.PurePath.as_uri()`.
iterchildren([exclude_dirs, match, matchcase])	Returns an iterator over all children (files and directories) of the current path object.
load_json([encoding, errors, json_library, ...])	Load JSON data from the file.
make_executable()	Make the file executable.
maybe_make([mode, parents])	Create a directory at this path, but only if the directory does not already exist.
move(dst)	Recursively move `self` to `dst`.

continues on next page

Table 4 – continued from previous page

open([mode, buffering, encoding, errors, ...])	Open the file pointed by this path and return a file object, as the built-in *open()* function does.
read_lines([encoding, errors])	Open the file in text mode, return a list containing the lines in the file, and close the file.
read_text([encoding, errors])	Open the file in text mode, read it, and close the file.
stream([chunk_size])	Stream the file in `chunk_size` sized chunks.
write_clean(string[, encoding, errors])	Write to the file without trailing whitespace, and with a newline at the end of the file.
write_lines(data[, encoding, errors, ...])	Write the given list of lines to the file without trailing whitespace.
write_text(data[, encoding, errors, newline])	Open the file in text mode, write to it, and close the file.

abspath()
Return the absolute version of the path.

New in version 1.3.0.

Return type *PathPlus*

append_text(*string*, *encoding='UTF-8'*, *errors=None*)
Open the file in text mode, append the given string to it, and close the file.

New in version 0.3.8.

Parameters

- **string** (`str`)
- **encoding** (`Optional`[`str`]) – The encoding to write to the file in. Default `'UTF-8'`.
- **errors** (`Optional`[`str`]) – Default `None`.

dump_json(*data*, *encoding='UTF-8'*, *errors=None*, *json_library=<module 'json'>*, ***, *compress=False*, ***kwargs*)
Dump `data` to the file as JSON.

New in version 0.5.0.

Parameters

- **data** (`Any`) – The object to serialise to JSON.
- **encoding** (`Optional`[`str`]) – The encoding to write to the file in. Default `'UTF-8'`.
- **errors** (`Optional`[`str`]) – Default `None`.
- **json_library** (*JsonLibrary*) – The JSON serialisation library to use. Default `json`.
- **compress** (`bool`) – Whether to compress the JSON file using gzip. Default `False`.
- ****kwargs** – Keyword arguments to pass to the JSON serialisation function.

Changed in version 1.0.0: Now uses *PathPlus.write_clean* rather than *PathPlus.write_text*, and as a result returns `None` rather than `int`.

Changed in version 1.9.0: Added the `compress` keyword-only argument.

classmethod from_uri(*uri*)
Construct a *PathPlus* from a `file` URI returned by `pathlib.PurePath.as_uri()`.

New in version 2.9.0.

Parameters **uri** (`str`)

Return type *PathPlus*

iterchildren(*exclude_dirs=('.git', '.hg', 'venv', '.venv', '.mypy_cache', '__pycache__', '.pytest_cache', '.tox', '.tox4', '.nox', '__pypackages__')*, *match=None*, *matchcase=True*)
Returns an iterator over all children (files and directories) of the current path object.

New in version 2.3.0.

Parameters

- **exclude_dirs** (`Optional[Iterable[str]]`) – A list of directory names which should be excluded from the output, together with their children. Default `('.git', '.hg', 'venv', '.venv', '.mypy_cache', '__pycache__', '.pytest_cache', '.tox', '.tox4', '.nox', '__pypackages__')`.
- **match** (`Optional[str]`) – A pattern to match filenames against. The pattern should be in the format taken by *matchglob()*. Default `None`.
- **matchcase** (`bool`) – Whether the filename's case should match the pattern. Default `True`.

Return type `Iterator[~_PP]`

Changed in version 2.5.0: Added the `matchcase` option.

load_json(*encoding='UTF-8'*, *errors=None*, *json_library=<module 'json'>*, ***, *decompress=False*, ***kwargs*)
Load JSON data from the file.

New in version 0.5.0.

Parameters

- **encoding** (`Optional[str]`) – The encoding to write to the file in. Default `'UTF-8'`.
- **errors** (`Optional[str]`) – Default `None`.
- **json_library** (*JsonLibrary*) – The JSON serialisation library to use. Default `json`.
- **decompress** (`bool`) – Whether to decompress the JSON file using gzip. Will raise an exception if the file is not compressed. Default `False`.
- ****kwargs** – Keyword arguments to pass to the JSON deserialisation function.

Return type `Any`

Returns The deserialised JSON data.

Changed in version 1.9.0: Added the `compress` keyword-only argument.

make_executable()
Make the file executable.

New in version 0.3.8.

maybe_make(*mode=511*, *parents=False*)

Create a directory at this path, but only if the directory does not already exist.

New in version 0.3.8.

Parameters

- **mode** (int) – Combined with the process' umask value to determine the file mode and access flags. Default 511.
- **parents** (bool) – If False (the default), a missing parent raises a FileNotFoundError. If True, any missing parents of this path are created as needed; they are created with the default permissions without taking mode into account (mimicking the POSIX mkdir -p command).

Changed in version 1.6.0: Removed the 'exist_ok' option, since it made no sense in this context.

> **Attention:** This will fail silently if a file with the same name already exists. This appears to be due to the behaviour of os.mkdir().

move(*dst*)

Recursively move self to dst.

self may be a file or a directory.

See shutil.move() for more details.

New in version 3.2.0.

Parameters **dst** (Union[str, Path, PathLike])

Returns The new location of self.

Return type *PathPlus*

open(*mode='r'*, *buffering=- 1*, *encoding='UTF-8'*, *errors=None*, *newline=NEWLINE_DEFAULT*)

Open the file pointed by this path and return a file object, as the built-in *open()* function does.

New in version 0.3.8.

Parameters

- **mode** (str) – The mode to open the file in. Default 'r' (read only).
- **buffering** (int) – Default -1.
- **encoding** (Optional[str]) – Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.
- **newline** (Optional[str]) – Default universal newlines for reading, Unix line endings (LF) for writing.

Return type IO[Any]

Changed in version 0.5.1: Defaults to Unix line endings (LF) on all platforms.

read_lines(*encoding='UTF-8'*, *errors=None*)
Open the file in text mode, return a list containing the lines in the file, and close the file.

New in version 0.5.0.

Parameters

- **encoding** (Optional[str]) – The encoding to write to the file in. Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.

Return type List[str]

Returns The content of the file.

read_text(*encoding='UTF-8'*, *errors=None*)
Open the file in text mode, read it, and close the file.

New in version 0.3.8.

Parameters

- **encoding** (Optional[str]) – The encoding to write to the file in. Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.

Return type str

Returns The content of the file.

stream(*chunk_size=1024*)
Stream the file in chunk_size sized chunks.

Parameters **chunk_size** (int) – The chunk size, in bytes. Default 1024.

New in version 3.2.0.

Return type Iterator[bytes]

write_clean(*string*, *encoding='UTF-8'*, *errors=None*)
Write to the file without trailing whitespace, and with a newline at the end of the file.

New in version 0.3.8.

Parameters

- **string** (str)
- **encoding** (Optional[str]) – The encoding to write to the file in. Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.

write_lines(*data*, *encoding='UTF-8'*, *errors=None*, ***, *trailing_whitespace=False*)
Write the given list of lines to the file without trailing whitespace.

New in version 0.5.0.

Parameters

- **data** (Iterable[str])
- **encoding** (Optional[str]) – The encoding to write to the file in. Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.

- **trailing_whitespace** (bool) – If True trailing whitespace is preserved. Default False.

Changed in version 2.4.0: Added the trailing_whitespace option.

write_text(*data*, *encoding='UTF-8'*, *errors=None*, *newline=NEWLINE_DEFAULT*)
Open the file in text mode, write to it, and close the file.

New in version 0.3.8.

Parameters

- **data** (str)
- **encoding** (Optional[str]) – The encoding to write to the file in. Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.
- **newline** (Optional[str]) – Default universal newlines for reading, Unix line endings (LF) for writing.

Changed in version 3.1.0: Added the newline argument to match Python 3.10. (see python/cpython#22420)

Return type int

class PosixPathPlus(**args*, ***kwargs*)
Bases: *PathPlus*, PurePosixPath

PathPlus subclass for non-Windows systems.

On a POSIX system, instantiating a *PathPlus* object should return an instance of this class.

New in version 0.3.8.

class TemporaryPathPlus(*suffix=None*, *prefix=None*, *dir=None*)
Bases: TemporaryDirectory

Securely creates a temporary directory using the same rules as tempfile.mkdtemp(). The resulting object can be used as a context manager. On completion of the context or destruction of the object the newly created temporary directory and all its contents are removed from the filesystem.

Unlike tempfile.TemporaryDirectory() this class is based around a *PathPlus* object.

New in version 2.4.0.

Methods:

cleanup()	Cleanup the temporary directory by removing it and its contents.

Attributes:

name	The temporary directory itself.

cleanup()
Cleanup the temporary directory by removing it and its contents.

If the *TemporaryPathPlus* is used as a context manager this is called when leaving the with block.

name
Type: *PathPlus*

The temporary directory itself.

This will be assigned to the target of the `as` clause if the *TemporaryPathPlus* is used as a context manager.

class WindowsPathPlus(**args*, ***kwargs*)
Bases: *PathPlus*, PureWindowsPath

PathPlus subclass for Windows systems.

On a Windows system, instantiating a *PathPlus* object should return an instance of this class.

New in version 0.3.8.

The following methods are unsupported on Windows:

- `group()`
- `is_mount()`
- `owner()`

_P = TypeVar(_P, bound=Path)
Type: TypeVar

Invariant TypeVar bound to `pathlib.Path`.

New in version 0.11.0.

Changed in version 1.7.0: Now bound to `pathlib.Path`.

_PP = TypeVar(_PP, bound=PathPlus)
Type: TypeVar

Invariant TypeVar bound to *domdf_python_tools.paths.PathPlus*.

New in version 2.3.0.

append(*var*, *filename*, ***kwargs*)
Append `var` to the file `filename` in the current directory.

Parameters

- **var** (`str`) – The value to append to the file
- **filename** (Union[`str`, `Path`, `PathLike`]) – The file to append to

Return type `int`

clean_writer(*string*, *fp*)
Write string to `fp` without trailing spaces.

Parameters

- **string** (`str`)
- **fp** (`IO`)

compare_dirs(*a*, *b*)
Compare the content of two directory trees.

New in version 2.7.0.

Parameters

- **a** (Union[str, Path, PathLike]) – The "left" directory to compare.
- **b** (Union[str, Path, PathLike]) – The "right" directory to compare.

Return type bool

Returns False if they differ, True is they are the same.

copytree(*src*, *dst*, *symlinks=False*, *ignore=None*)
Alternative to shutil.copytree() to support copying to a directory that already exists.

Based on https://stackoverflow.com/a/12514470 by https://stackoverflow.com/users/23252/atzz

In Python 3.8 and above shutil.copytree() takes a dirs_exist_ok argument, which has the same result.

Parameters

- **src** (Union[str, Path, PathLike]) – Source file to copy
- **dst** (Union[str, Path, PathLike]) – Destination to copy file to
- **symlinks** (bool) – Whether to represent symbolic links in the source as symbolic links in the destination. If false or omitted, the contents and metadata of the linked files are copied to the new tree. When symlinks is false, if the file pointed by the symlink doesn't exist, an exception will be added in the list of errors raised in an Error exception at the end of the copy process. You can set the optional ignore_dangling_symlinks flag to true if you want to silence this exception. Notice that this option has no effect on platforms that don't support os.symlink(). Default False.
- **ignore** (Optional[Callable]) – A callable that will receive as its arguments the source directory, and a list of its contents. The ignore callable will be called once for each directory that is copied. The callable must return a sequence of directory and file names relative to the current directory (i.e. a subset of the items in its second argument); these names will then be ignored in the copy process. shutil.ignore_patterns() can be used to create such a callable that ignores names based on glob-style patterns. Default None.

Return type Union[str, Path, PathLike]

delete(*filename*, ***kwargs*)
Delete the file in the current directory.

Parameters **filename** (Union[str, Path, PathLike]) – The file to delete

in_directory(*directory*)
Context manager to change into the given directory for the duration of the with block.

Parameters **directory** (Union[str, Path, PathLike])

make_executable(*filename*)
Make the given file executable.

Parameters **filename** (Union[str, Path, PathLike])

matchglob(*filename*, *pattern*, *matchcase=True*)
Given a filename and a glob pattern, return whether the filename matches the glob.

New in version 2.3.0.

Parameters

- **filename** (Union[str, Path, PathLike])
- **pattern** (str) – A pattern structured like a filesystem path, where each element consists of the glob syntax. Each element is matched by fnmatch. The special element ** matches zero or more files or directories.
- **matchcase** (bool) – Whether the filename's case should match the pattern. Default True.

Return type bool

See also: Glob (programming)#Syntax on Wikipedia

Changed in version 2.5.0: Added the matchcase option.

maybe_make(*directory*, *mode=511*, *parents=False*)
Create a directory at the given path, but only if the directory does not already exist.

> **Attention:** This will fail silently if a file with the same name already exists. This appears to be due to the behaviour of os.mkdir().

Parameters

- **directory** (Union[str, Path, PathLike]) – Directory to create
- **mode** (int) – Combined with the process's umask value to determine the file mode and access flags. Default 511.
- **parents** (bool) – If False (the default), a missing parent raises a FileNotFoundError. If True, any missing parents of this path are created as needed; they are created with the default permissions without taking mode into account (mimicking the POSIX mkdir -p command).

Changed in version 1.6.0: Removed the 'exist_ok' option, since it made no sense in this context.

parent_path(*path*)
Returns the path of the parent directory for the given file or directory.

Parameters **path** (Union[str, Path, PathLike]) – Path to find the parent for

Return type Path

Returns The parent directory

read(*filename*, ***kwargs*)
Read a file in the current directory (in text mode).

Parameters **filename** (Union[str, Path, PathLike]) – The file to read from.

Return type str

Returns The contents of the file.

relpath(*path*, *relative_to=None*)
Returns the path for the given file or directory relative to the given directory or, if that would require path traversal, returns the absolute path.

Parameters

- **path** (Union[str, Path, PathLike]) – Path to find the relative path for
- **relative_to** (Union[str, Path, PathLike, None]) – The directory to find the path relative to. Defaults to the current directory.

Return type Path

sort_paths(**paths*)
Sort the paths by directory, then by file.

New in version 2.6.0.

Parameters **paths**

Return type List[*PathPlus*]

traverse_to_file(*base_directory*, **filename*, *height=- 1*)
Traverse the parents of the given directory until the desired file is found.

New in version 1.7.0.

Parameters

- **base_directory** (*~_P*) – The directory to start searching from
- ***filename** (Union[str, Path, PathLike]) – The filename(s) to search for
- **height** (int) – The maximum height to traverse to. Default -1.

Return type *~_P*

unwanted_dirs = ('.git', '.hg', 'venv', '.venv', '.mypy_cache', '__pycache__', '.pytest_ca
Type: tuple

A list of directories which will likely be unwanted when searching directory trees for files.

New in version 2.3.0.

Changed in version 2.9.0: Added .hg (mercurial)

Changed in version 3.0.0: Added __pypackages__ (**PEP 582**)

Changed in version 3.2.0: Added .nox (https://nox.thea.codes/)

write(*var*, *filename*, ***kwargs*)
Write a variable to file in the current directory.

Parameters

- **var** (str) – The value to write to the file.
- **filename** (Union[str, Path, PathLike]) – The file to write to.

Chapter

TWELVE

pretty_print

Functions and classes for pretty printing.

New in version 0.10.0.

Classes:

FancyPrinter([indent, width, depth, stream, ...])	Subclass of `PrettyPrinter` with different formatting.

Functions:

simple_repr(*attributes[, show_module])	Adds a simple `__repr__` method to the decorated class.

class FancyPrinter(*indent=1*, *width=80*, *depth=None*, *stream=None*, *, *compact=False*, *sort_dicts=True*)

Bases: `PrettyPrinter`

Subclass of `PrettyPrinter` with different formatting.

Parameters

- **indent** (`int`) – Number of spaces to indent for each level of nesting. Default `1`.
- **width** (`int`) – Attempted maximum number of columns in the output. Default `80`.
- **depth** (`Optional[int]`) – The maximum depth to print out nested structures. Default `None`.
- **stream** (`Optional[IO[str]]`) – The desired output stream. If omitted (or `False`), the standard output stream available at construction will be used. Default `None`.
- **compact** (`bool`) – If `True`, several items will be combined in one line. Default `False`.
- **sort_dicts** (`bool`) – If `True`, dict keys are sorted. Only takes effect on Python 3.8 and later, or if pprint36 is installed. Default `True`.

simple_repr(**attributes*, *show_module=False*, ***kwargs*)

Adds a simple `__repr__` method to the decorated class.

Parameters

- **attributes** – The attributes to include in the `__repr__`.
- **show_module** (`bool`) – Whether to show the name of the module in the `__repr__`. Default `False`.
- ****kwargs** – Keyword arguments passed on to `pprint.PrettyPrinter`.

Chapter

THIRTEEN

secrets

Functions for working with secrets, such as API tokens.

New in version 0.4.6.

Classes:

Secret(value)	Subclass of `str` that guards against accidentally printing a secret to the terminal.

class Secret (*value*)

Bases: `str`

Subclass of `str` that guards against accidentally printing a secret to the terminal.

The actual value of the secret is accessed via the `.value` attribute.

The protection should be maintained even when the secret is in a list, tuple, set or dict, but you should still refrain from printing objects containing the secret.

The secret overrides the *__eq__()* method of `str`, so:

```
>>> Secret("Barry as FLUFL") == "Barry as FLUFL"
True
```

New in version 0.4.6.

Methods:

__eq__(other)	Return `self == other`.

Attributes:

value	The actual value of the secret.

__eq__ (*other*)

Return `self == other`.

Return type `bool`

value

Type: `str`

The actual value of the secret.

Chapter

FOURTEEN

stringlist

A list of strings that represent lines in a multiline string.

Changed in version 1.0.0: *String* should now be imported from *domdf_python_tools.typing*.

Classes:

DelimitedList([iterable])	Subclass of `list` that supports custom delimiters in format strings.
Indent([size, type])	Represents an indent, having a symbol/type and a size.
StringList([iterable, convert_indents])	A list of strings that represent lines in a multiline string.

Data:

_SL	Invariant `TypeVar` bound to *domdf_python_tools.stringlist.StringList*.

Functions:

joinlines(lines)	Given a list of two-element tuples, each containing a line and a newline character (or empty string), return a single string.
splitlines(string)	Split `string` into a list of two-element tuples, containing the line content and the newline character(s), if any.

class DelimitedList(*iterable=()*, */*)

Bases: `List`[~_S]

Subclass of `list` that supports custom delimiters in format strings.

Example:

```
>>> l = DelimitedList([1, 2, 3, 4, 5])
>>> format(l, ", ")
'1, 2, 3, 4, 5'
>>> f"Numbers: {l:, }"
'Numbers: 1, 2, 3, 4, 5'
```

New in version 1.1.0.

__format__(*format_spec*)

Default object formatter.

Return type `str`

class Indent(*size=0*, *type='\t'*)

Bases: object

Represents an indent, having a symbol/type and a size.

Parameters

- **size** (int) – The indent size. Default 0.
- **type** (str) – The indent character. Default '\t'.

Methods:

__eq__(other)	Return self == other.
__iter__()	Returns the size and type of the *Indent*.
__repr__()	Returns the string representation of the *Indent*.
__str__()	Returns the *Indent* as a string.

Attributes:

size	The indent size.
type	The indent character.

__eq__(*other*)

Return self == other.

Return type bool

__iter__()

Returns the size and type of the *Indent*.

Return type Iterator[Union[str, Any]]

__repr__()

Returns the string representation of the *Indent*.

Return type str

__str__()

Returns the *Indent* as a string.

Return type str

property size

The indent size.

Return type int

property type

The indent character.

Return type str

class StringList(*iterable=()*, *convert_indents=False*)

Bases: List[str]

A list of strings that represent lines in a multiline string.

Parameters

- **iterable** (Iterable[*String*]) – Content to populate the StringList with. Default ().
- **convert_indents** (bool) – Whether indents at the start of lines should be converted. Default False.

Methods:

__bytes__()	Returns the *StringList* as bytes.
__eq__(other)	Returns whether the other object is equal to this *StringList*.
__getitem__(index)	Returns the line with the given index.
__setitem__(index, line)	Replaces the given line with new content.
__str__()	Returns the *StringList* as a string.
append(line)	Append a line to the end of the *StringList*.
blankline([ensure_single])	Append a blank line to the end of the *StringList*.
copy()	Returns a shallow copy of the *StringList*.
count_blanklines()	Returns a count of the blank lines in the *StringList*.
extend(iterable)	Extend the *StringList* with lines from iterable.
insert(index, line)	Insert a line into the *StringList* at the given position.
set_indent(indent[, size])	Sets the indent to insert at the beginning of new lines.
set_indent_size([size])	Sets the size of the indent to insert at the beginning of new lines.
set_indent_type([indent_type])	Sets the type of the indent to insert at the beginning of new lines.
splitlines([keepends])	Analagous to str.splitlines().
with_indent(indent[, size])	Context manager to temporarily use a different indent.
with_indent_size([size])	Context manager to temporarily use a different indent size.
with_indent_type([indent_type])	Context manager to temporarily use a different indent type.

Attributes:

convert_indents	Whether indents at the start of lines should be converted.
indent	The indent to insert at the beginning of new lines.
indent_size	The current indent size.
indent_type	The current indent type.

__bytes__()

Returns the *StringList* as bytes.

New in version 2.1.0.

Return type bytes

__eq__(*other*)

Returns whether the other object is equal to this *StringList*.

Return type bool

__getitem__(*index*)
Returns the line with the given index.

Parameters **index** (Union[*SupportsIndex*, slice])

Return type Union[str, *StringList*]

Overloads

- *__getitem__*(**index**: *SupportsIndex*) -> str
- *__getitem__*(**index**: slice) -> *~_SL*

Changed in version 1.8.0: Now returns a *StringList* when index is a slice.

Changed in version 3.2.0: Changed int in the type annotation to *SupportsIndex*.

__setitem__(*index*, *line*)
Replaces the given line with new content.

If the new content consists of multiple lines subsequent content in the *StringList* will be shifted down.

Parameters

- **index** (Union[*SupportsIndex*, slice])
- **line** (Union[*String*, Iterable[*String*]])

Changed in version 3.2.0: Changed int in the type annotation to *SupportsIndex*.

Overloads

- *__setitem__*(**index**: *SupportsIndex*, **line**: *String*) -> None
- *__setitem__*(**index**: slice, **line**: Iterable[*String*]) -> None

__str__()
Returns the *StringList* as a string.

Return type str

append(*line*)
Append a line to the end of the *StringList*.

Parameters **line** (*String*)

blankline(*ensure_single=False*)
Append a blank line to the end of the *StringList*.

Parameters **ensure_single** (bool) – Ensure only a single blank line exists after the previous line of text. Default False.

convert_indents
Type: bool

Whether indents at the start of lines should be converted.

Only applies to lines added after this is enabled/disabled.

Can only be used when the indent is '\t' or '␣'.

copy()
Returns a shallow copy of the *StringList*.

Equivalent to `a[:]`.

Return type *StringList*

count_blanklines()
Returns a count of the blank lines in the *StringList*.

New in version 0.7.1.

Return type int

extend(*iterable*)
Extend the *StringList* with lines from `iterable`.

Parameters **iterable** (Iterable[*String*]) – An iterable of string-like objects to add to the end of the *StringList*.

indent
Type: *Indent*

The indent to insert at the beginning of new lines.

property indent_size
The current indent size.

Return type int

property indent_type
The current indent type.

Return type str

insert(*index*, *line*)
Insert a line into the *StringList* at the given position.

Parameters

- **index** (*SupportsIndex*)
- **line** (*String*)

Changed in version 3.2.0: Changed `int` in the type annotation to *SupportsIndex*.

set_indent(*indent*, *size=0*)
Sets the indent to insert at the beginning of new lines.

Parameters

- **indent** (Union[*String*, *Indent*]) – The *Indent* to use for new lines, or the indent type.
- **size** (int) – If `indent` is an indent type, the indent size to use for new lines. Default `0`.

set_indent_size(*size=0*)
Sets the size of the indent to insert at the beginning of new lines.

Parameters **size** (int) – The indent size to use for new lines. Default 0.

set_indent_type(*indent_type='\t'*)
Sets the type of the indent to insert at the beginning of new lines.

Parameters **indent_type** (str) – The type of indent to use for new lines. Default '\t'.

splitlines(*keepends=False*)
Analagous to str.splitlines().

New in version 3.8.0.

Return type List[str]

with_indent(*indent*, *size=0*)
Context manager to temporarily use a different indent.

```
>>> sl = StringList()
>>> with sl.with_indent("    ", 1):
...     sl.append("Hello World")
```

Parameters

- **indent** (Union[*String*, *Indent*]) – The *Indent* to use within the with block, or the indent type.
- **size** (int) – If indent is an indent type, the indent size to use within the with block. Default 0.

with_indent_size(*size=0*)
Context manager to temporarily use a different indent size.

```
>>> sl = StringList()
>>> with sl.with_indent_size(1):
...     sl.append("Hello World")
```

Parameters **size** (int) – The indent size to use within the with block. Default 0.

with_indent_type(*indent_type='\t'*)
Context manager to temporarily use a different indent type.

```
>>> sl = StringList()
>>> with sl.with_indent_type("    "):
...     sl.append("Hello World")
```

Parameters **indent_type** (str) – The type of indent to use within the with block. Default '\t'.

splitlines(*string*)

Split `string` into a list of two-element tuples, containing the line content and the newline character(s), if any.

New in version 3.2.0.

Parameters **string** (`str`)

Return type `List`[`Tuple`[`str`, `str`]]

See also: `str.splitlines()` and *joinlines()*

joinlines(*lines*)

Given a list of two-element tuples, each containing a line and a newline character (or empty string), return a single string.

New in version 3.2.0.

Parameters **lines** (`List`[`Tuple`[`str`, `str`]])

Return type `str`

See also: *splitlines()*

_SL = TypeVar(_SL, bound=StringList)

Type: `TypeVar`

Invariant `TypeVar` bound to *domdf_python_tools.stringlist.StringList*.

Chapter

FIFTEEN

terminal

Useful functions for terminal-based programs.

Changed in version 2.0.0: `domdf_python_tools.terminal.get_terminal_size()` was removed. Use `shutil.get_terminal_size()` instead.

Classes:

Echo([indent])	Context manager for echoing variable assignments (in CPython).

Functions:

br()	Prints a blank line.
clear()	Clears the display.
interrupt()	Print the key combination needed to abort the script; dynamic depending on OS.
overtype(*objects[, sep, end, file, flush])	Print `*objects` to the text stream `file`, starting with `'\\r'`, separated by `sep` and followed by `end`.

class Echo(*indent=''*)

Bases: `object`

Context manager for echoing variable assignments (in CPython).

Parameters **indent** (`str`) – The indentation of the dictionary of variable assignments. Default `'␣␣'`.

Methods:

__enter__()	Called when entering the context manager.
__exit__(*args, **kwargs)	Called when exiting the context manager.

__enter__()

Called when entering the context manager.

__exit__(**args*, ***kwargs*)

Called when exiting the context manager.

br()

Prints a blank line.

clear()

Clears the display.

Works for Windows and POSIX, but does not clear the Python Interpreter or PyCharm's Console.

interrupt()
Print the key combination needed to abort the script; dynamic depending on OS.

Useful when you have a long-running script that you might want to interrupt part way through.

Example:

```
>>> interrupt()
(Press Ctrl-C to quit at any time)
```

overtype(**objects*, *sep=' '*, *end=''*, *file=None*, *flush=False*)
Print `*objects` to the text stream `file`, starting with `'\\r'`, separated by `sep` and followed by `end`.

All non-keyword arguments are converted to strings like `str` does and written to the stream, separated by `sep` and followed by `end`.

If no objects are given, *`overtype()`* will just write `"\\r"`.

Parameters

- ***objects** – A list of strings or string-like objects to write to the terminal.
- **sep** (`str`) – The separator between values. Default `'␣'`.
- **end** (`str`) – The final value to print. Default `''`.
- **file** (`Optional`[`IO`]) – An object with a `write(string)` method. If not present or `None`, `sys.stdout` will be used.
- **flush** (`bool`) – If `True` the stream is forcibly flushed after printing. Default `False`.

Chapter

SIXTEEN

typing

Various type annotation aids.

16.1 Type Hints

PathLike	Type hint for objects that represent filesystem paths.
PathType	Invariant TypeVar constrained to str, pathlib.Path and os.PathLike.
AnyNumber	Type hint for numbers.
WrapperDescriptorType	The type of methods of some built-in data types and base classes.
MethodWrapperType	The type of *bound* methods of some built-in data types and base classes.
MethodDescriptorType	The type of methods of some built-in data types.
ClassMethodDescriptorType	The type of *unbound* class methods of some built-in data types.

PathLike

Type hint for objects that represent filesystem paths.

See also: *domdf_python_tools.typing.PathType*

Alias of Union[str, Path, PathLike]

PathType = TypeVar(PathType, str, Path, PathLike)

Type: TypeVar

Invariant TypeVar constrained to str, pathlib.Path and os.PathLike.

Type variable for objects that represent filesystem paths.

New in version 2.2.0.

See also: *domdf_python_tools.typing.PathLike*

AnyNumber

Type hint for numbers.

Changed in version 0.4.6: Moved from *domdf_python_tools.pagesizes*

Alias of Union[float, int, Decimal]

WrapperDescriptorType
The type of methods of some built-in data types and base classes, such as `object.__init__()` or `object.__lt__()`.

New in version 0.8.0.

MethodWrapperType
The type of *bound* methods of some built-in data types and base classes. For example, it is the type of `object().__str__`.

New in version 0.8.0.

MethodDescriptorType
The type of methods of some built-in data types, such as `str.join()`.

New in version 0.8.0.

ClassMethodDescriptorType
The type of *unbound* class methods of some built-in data types, such as `dict.__dict__['fromkeys']`.

New in version 0.8.0.

16.2 Protocols

JsonLibrary	`typing.Protocol` for libraries that implement the same API as `json`.
HasHead	`typing.Protocol` for classes that have a `head` method.
String	`Protocol` for classes that implement `__str__`.
FrameOrSeries	Invariant `TypeVar` constrained to `pandas.Series` and `pandas.DataFrame`.
SupportsIndex	`typing.Protocol` for classes that support `__index__`.
SupportsLessThan	`typing.Protocol` for classes that support `__lt__`.
SupportsLessEqual	`typing.Protocol` for classes that support `__le__`.
SupportsGreaterThan	`typing.Protocol` for classes that support `__gt__`.
SupportsGreaterEqual	`typing.Protocol` for classes that support `__ge__`.

protocol JsonLibrary
Bases: `Protocol`

`typing.Protocol` for libraries that implement the same API as `json`.

Useful for annotating functions which take a JSON serialisation-deserialisation library as an argument.

Classes that implement this protocol must have the following methods / attributes:

static dumps(*obj*, ***, *skipkeys=...*, *ensure_ascii=...*, *check_circular=...*, *allow_nan=...*, *cls=...*, *indent=...*, *separators=...*, *default=...*, *sort_keys=...*, ***kwds*)

Serialize `obj` to a JSON formatted `str`.

Parameters

- **obj** (`Any`)
- **skipkeys** (`bool`)
- **ensure_ascii** (`bool`)
- **check_circular** (`bool`)
- **allow_nan** (`bool`)
- **cls** (`Optional[Type[JSONEncoder]]`)
- **indent** (`Union[None, int, str]`)
- **separators** (`Optional[Tuple[str, str]]`)
- **default** (`Optional[Callable[[Any], Any]]`)
- **sort_keys** (`bool`)
- **kwds**

Return type `str`

static loads(*s*, ***, *cls=...*, *object_hook=...*, *parse_float=...*, *parse_int=...*, *parse_constant=...*, *object_pairs_hook=...*, ***kwds*)

Deserialize `s` to a Python object.

Parameters

- **s** (`Union[str, bytes]`)
- **cls** (`Optional[Type[JSONDecoder]]`)
- **object_hook** (`Optional[Callable[[Dict[Any, Any]], Any]]`)
- **parse_float** (`Optional[Callable[[str], Any]]`)
- **parse_int** (`Optional[Callable[[str], Any]]`)
- **parse_constant** (`Optional[Callable[[str], Any]]`)
- **object_pairs_hook** (`Optional[Callable[[List[Tuple[Any, Any]]], Any]]`)
- **kwds**

Return type `Any`

protocol HasHead

Bases: Protocol

typing.Protocol for classes that have a head method.

This includes pandas.DataFrame and pandas.Series.

New in version 0.8.0.

This protocol is runtime checkable.

Classes that implement this protocol must have the following methods / attributes:

__non_callable_proto_members__ = {}

Type: set

head(*n=5*)

Return the first n rows.

Parameters **n** (int) – Number of rows to select. Default 5.

Return type *HasHead*

Returns The first n rows of the caller object.

to_string(**args*, ***kwargs*)

Render the object to a console-friendly tabular output.

Return type Optional[str]

protocol String

Bases: Protocol

Protocol for classes that implement __str__.

Changed in version 0.8.0: Moved from *domdf_python_tools.stringlist*.

This protocol is runtime checkable.

Classes that implement this protocol must have the following methods / attributes:

__non_callable_proto_members__ = {}

Type: set

__str__()

Return str(self).

Return type str

protocol FrameOrSeries

New in version 1.0.0.

Classes that implement this protocol must have the following methods / attributes:

protocol SupportsIndex
Bases: Protocol

typing.Protocol for classes that support __index__.

New in version 2.0.0.

Classes that implement this protocol must have the following methods / attributes:

__index__()

Return type int

protocol SupportsLessThan
Bases: Protocol

typing.Protocol for classes that support __lt__.

New in version 3.0.0.

Classes that implement this protocol must have the following methods / attributes:

__lt__(*_SupportsLessThan__other*)
Return self < value.

Return type bool

protocol SupportsLessEqual
Bases: Protocol

typing.Protocol for classes that support __le__.

New in version 3.0.0.

Classes that implement this protocol must have the following methods / attributes:

__le__(*_SupportsLessEqual__other*)
Return self <= value.

Return type bool

protocol SupportsGreaterThan
Bases: Protocol

typing.Protocol for classes that support __gt__.

New in version 3.0.0.

Classes that implement this protocol must have the following methods / attributes:

__gt__(*_SupportsGreaterThan__other*)
Return self > value.

Return type bool

protocol SupportsGreaterEqual

Bases: Protocol

typing.Protocol for classes that support __ge__.

New in version 3.0.0.

Classes that implement this protocol must have the following methods / attributes:

__ge__(*_SupportsGreaterEqual__other*)

Return self >= value.

Return type bool

16.3 Utility Functions

check_membership(*obj*, *type_*)

Check if the type of `obj` is one of the types in a `typing.Union`, `typing.Sequence` etc.

Parameters

- **obj** (`Any`) – The object to check the type of
- **type_** (`Union`[`Type`, `object`]) – A `Type` that has members, such as a `typing.List`, `typing.Union` or `typing.Sequence`.

Return type `bool`

Chapter

SEVENTEEN

utils

General utility functions.

Changed in version 1.0.0:

- Removed `tuple2str` and `list2string`. Use *domdf_python_tools.utils.list2str()* instead.
- Removed `as_text` and `word_join`. Import from *domdf_python_tools.words* instead.
- Removed `splitLen`. Use *domdf_python_tools.iterative.split_len()* instead.

Changed in version 2.0.0: *chunks()*, *permutations()*, *split_len()*, *Len()*, and *double_chain()* moved to *domdf_python_tools.iterative()*.

Changed in version 2.3.0: Removed `domdf_python_tools.utils.deprecated()`. Use the new deprecation-alias package instead.

Data:

SPACE_PLACEHOLDER	The ␣ character.
etc	Object that provides an ellipsis string
pyversion	The current major python version.

Functions:

cmp(x, y)	Implementation of `cmp` for Python 3.
convert_indents(text[, tab_width, from_, to])	Convert indentation at the start of lines in `text` from tabs to spaces.
divide(string, sep)	Divide a string into two parts, about the given string.
double_repr_string(string)	Like `repr(str)`, but tries to use double quotes instead.
enquote_value(value)	Adds single quotes (`'`) to the given value, suitable for use in a templating system such as Jinja2.
head(obj[, n])	Returns the head of the given object.
list2str(the_list[, sep])	Convert an iterable, such as a list, to a comma separated string.
magnitude(x)	Returns the magnitude of the given value.
posargs2kwargs(args, posarg_names[, kwargs])	Convert the positional args in `args` to kwargs, based on the relative order of `args` and `posarg_names`.
printe(*values[, sep, end])	Alias of *stderr_writer()*
printr(obj, *values[, sep, end, file, flush])	Print the `repr()` of an object.
printt(obj, *values[, sep, end, file, flush])	Print the type of an object.
redirect_output([combine])	Context manager to redirect stdout and stderr to two `io.StringIO` objects.
redivide(string, pat)	Divide a string into two parts, splitting on the given regular expression.

continues on next page

Table 2 – continued from previous page

replace_nonprinting(string[, exclude])	Replace nonprinting (control) characters in `string` with `^` and `M-` notation.
stderr_writer(*values[, sep, end])	Print `*values` to `sys.stderr`, separated by `sep` and followed by `end`.
str2tuple(input_string[, sep])	Convert a comma-separated string of integers into a tuple.
strtobool(val)	Convert a string representation of truth to `True` or `False`.
trim_precision(value[, precision])	Trim the precision of the given floating point value.
unique_sorted(elements, *[, key, reverse])	Returns an ordered list of unique items from `elements`.

SPACE_PLACEHOLDER = '␣'

Type: `str`

The ␣ character.

cmp(*x*, *y*)

Implementation of `cmp` for Python 3.

Compare the two objects x and y and return an integer according to the outcome.

The return value is negative if `x < y`, zero if `x == y` and strictly positive if `x > y`.

Return type `int`

convert_indents(*text*, *tab_width=4*, *from_='\t'*, *to='␣'*)

Convert indentation at the start of lines in `text` from tabs to spaces.

Parameters

- **text** (`str`) – The text to convert indents in.
- **tab_width** (`int`) – The number of spaces per tab. Default `4`.
- **from_** (`str`) – The indent to convert from. Default `'\t'`.
- **to** (`str`) – The indent to convert to. Default `'␣'`.

Return type `str`

divide(*string*, *sep*)

Divide a string into two parts, about the given string.

New in version 2.7.0.

Parameters

- **string** (`str`)
- **sep** (`str`) – The separator to split at.

Return type `Tuple[str, str]`

double_repr_string(*string*)

Like `repr(str)`, but tries to use double quotes instead.

New in version 2.5.0.

Parameters **string** (`str`)

Return type `str`

enquote_value(*value*)

Adds single quotes (`'`) to the given value, suitable for use in a templating system such as Jinja2.

`Floats`, `integers`, `booleans`, `None`, and the strings `'True'`, `'False'` and `'None'` are returned as-is.

Parameters **value** (`Any`) – The value to enquote

Return type `Union`[`str`, `bool`, `float`]

etc = ...

Type: `_Etcetera`

Object that provides an ellipsis string

New in version 0.8.0.

head(*obj*, *n=10*)

Returns the head of the given object.

New in version 0.8.0.

Parameters

- **obj** (`Union`[`Tuple`, `List`, `DataFrame`, `Series`, *`String`*, *`HasHead`*])
- **n** (`int`) – Show the first `n` items of `obj`. Default `10`.

See also:

- `textwrap.shorten()`, which truncates a string to fit within a given number of characters.
- `itertools.islice()`, which returns the first `n` elements from an iterator.

Return type `Optional`[`str`]

list2str(*the_list*, *sep=','*)

Convert an iterable, such as a list, to a comma separated string.

Parameters

- **the_list** (`Iterable`[`Any`]) – The iterable to convert to a string.
- **sep** (`str`) – Separator to use for the string. Default `','`.

Return type `str`

Returns Comma separated string

magnitude(*x*)

Returns the magnitude of the given value.

- For negative numbers the absolute magnitude is returned.
- For decimal numbers below `1` the magnitude will be negative.

New in version 2.0.0.

Parameters **x** (`float`) – Numerical value to find the magnitude of.

Return type `int`

posargs2kwargs(*args*, *posarg_names*, *kwargs=None*)

Convert the positional args in `args` to kwargs, based on the relative order of `args` and `posarg_names`.

Important: Python 3.8's Positional-Only Parameters (**PEP 570**) are not supported.

New in version 0.4.10.

Parameters

- **args** (`Iterable`[`Any`]) – List of positional arguments provided to a function.
- **posarg_names** (`Union`[`Iterable`[`str`], `Callable`]) – Either a list of positional argument names for the function, or the function object.
- **kwargs** (`Optional`[`Dict`[`str`, `Any`]]) – Optional mapping of keyword argument names to values. The arguments will be added to this dictionary if provided. Default `{}`.

Return type `Dict`[`str`, `Any`]

Returns Dictionary mapping argument names to values.

Changed in version 2.8.0: The "self" argument for bound methods is ignored. For unbound methods (which are just functions) the behaviour is unchanged.

printe(**values*, *sep=' '*, *end='\n'*)

Alias of *stderr_writer()*

printr(*obj*, **values*, *sep=' '*, *end='\n'*, *file=None*, *flush=False*)

Print the `repr()` of an object.

If no objects are given, *printr()* will just write `end`.

Parameters

- **obj** (`Any`)
- ***values** (`object`) – Additional values to print. These are printed verbatim.
- **sep** (`Optional`[`str`]) – The separator between values. Default `'␣'`.
- **end** (`Optional`[`str`]) – The final value to print. Setting to `''` will leave the insertion point at the end of the printed text. Default `'\n'`.
- **file** (`Optional`[`IO`]) – The file to write to. If not present or `None`, `sys.stdout` will be used.
- **flush** (`bool`) – If `True` the stream is forcibly flushed after printing. Default `False`.

printt(*obj*, **values*, *sep=' '*, *end='\n'*, *file=None*, *flush=False*)
Print the type of an object.

If no objects are given, *printt()* will just write `end`.

Parameters

- **obj** (`Any`)
- ***values** (`object`) – Additional values to print. These are printed verbatim.
- **sep** (`Optional[str]`) – The separator between values. Default `'␣'`.
- **end** (`Optional[str]`) – The final value to print. Setting to `''` will leave the insertion point at the end of the printed text. Default `'\n'`.
- **file** (`Optional[IO]`) – The file to write to. If not present or `None`, `sys.stdout` will be used.
- **flush** (`bool`) – If `True` the stream is forcibly flushed after printing. Default `False`.

pyversion = 3
Type: `int`

The current major python version.

redirect_output(*combine=False*)
Context manager to redirect stdout and stderr to two `io.StringIO` objects.

These are assigned (as a `tuple`) to the target the `as` expression.

Example:

```
with redirect_output() as (stdout, stderr):
    ...
```

New in version 2.6.0.

Parameters **combine** (`bool`) – If `True` `stderr` is combined with `stdout`. Default `False`.

Return type `Iterator[Tuple[StringIO, StringIO]]`

redivide(*string*, *pat*)
Divide a string into two parts, splitting on the given regular expression.

New in version 2.7.0.

Parameters

- **string** (`str`)
- **pat** (`Union[str, Pattern]`)

Return type `Tuple[str, str]`

replace_nonprinting(*string*, *exclude=None*)

Replace nonprinting (control) characters in `string` with `^` and `M-` notation.

New in version 3.3.0.

Parameters

- **string** (`str`)
- **exclude** (`Optional[Set[int]]`) – A set of codepoints to exclude. Default `None`.

Return type `str`

See also: C0 and C1 control codes on Wikipedia

stderr_writer(**values*, *sep=' '*, *end='\n'*)

Print `*values` to `sys.stderr`, separated by `sep` and followed by `end`.

`sys.stdout` is flushed before printing, and `sys.stderr` is flushed afterwards.

If no objects are given, *`stderr_writer()`* will just write `end`.

Parameters

- ***values** (`object`)
- **sep** (`Optional[str]`) – The separator between values. Default `'␣'`.
- **end** (`Optional[str]`) – The final value to print. Setting to `''` will leave the insertion point at the end of the printed text. Default `'\n'`.

Changed in version 3.0.0: The only permitted keyword arguments are `sep` and `end`. Previous versions allowed other keywords arguments supported by `print()` but they had no effect.

str2tuple(*input_string*, *sep=','*)

Convert a comma-separated string of integers into a tuple.

Important: The input string must represent a comma-separated series of integers.

Parameters

- **input_string** (`str`) – The string to be converted into a tuple
- **sep** (`str`) – The separator in the string. Default `','`.

Return type `Tuple[int, ...]`

strtobool(*val*)

Convert a string representation of truth to `True` or `False`.

If val is an integer then its boolean representation is returned. If val is a boolean it is returned as-is.

`True` values are `'y'`, `'yes'`, `'t'`, `'true'`, `'on'`, `'1'`, and `1`.

`False` values are `'n'`, `'no'`, `'f'`, `'false'`, `'off'`, `'0'`, and `0`.

Raises `ValueError` if `val` is anything else.

Return type `bool`

trim_precision(*value*, *precision=4*)

Trim the precision of the given floating point value.

For example, if you have the value *170.10000000000002* but really only care about it being ≈ `179.1`:

```
>>> trim_precision(170.10000000000002, 2)
170.1
>>> type(trim_precision(170.10000000000002, 2))
<class 'float'>
```

New in version 2.0.0.

Parameters

- **value** (`float`)
- **precision** (`int`) – The number of decimal places to leave in the output. Default `4`.

Return type `float`

unique_sorted(*elements*, *, *key=None*, *reverse=False*)

Returns an ordered list of unique items from `elements`.

New in version 3.0.0.

Parameters

- **elements** (`Iterable`)
- **key** (`Optional`[`Callable`]) – A function of one argument used to extract a comparison key from each item when sorting. For example, `key=str.lower`. The default value is `None`, which will compare the elements directly. Default `None`.
- **reverse** (`bool`) – If `True` the list elements are sorted as if each comparison were reversed. Default `False`.

See also: `set` and `sorted()`

Return type `List`

Overloads

- *unique_sorted*(**elements**: `Iterable`[`~SupportsLessThanT`], **key**: `None` = ..., **reverse** = ...) -> `List`[`~SupportsLessThanT`]
- *unique_sorted*(**elements**: `Iterable`[`~_T`], **key**: `Callable`[[`~_T`], *SupportsLessThan*], **reverse** = ...) -> `List`[`~_T`]

Chapter

EIGHTEEN

versions

NamedTuple-like class to represent a version number.

New in version 0.4.4.

Classes:

Version([major, minor, patch])	NamedTuple-like class to represent a version number.

namedtuple Version(*major=0*, *minor=0*, *patch=0*)

Bases: Tuple[int, int, int]

NamedTuple-like class to represent a version number.

Fields

0) **major** (int) – The major version number.
1) **minor** (int) – The minor version number.
2) **patch** (int) – The patch version number.

Changed in version 1.4.0: Implemented the same interface as a collections.namedtuple().

__eq__(*other*)

Returns whether this version is equal to the other version.

Return type bool

__final__ = True

Type: bool

__float__()

Return the major and minor version number as a float.

Return type float

__ge__(*other*)

Returns whether this version is greater than or equal to the other version.

Return type bool

__gt__(*other*)

Returns whether this version is greater than the other version.

Return type bool

__int__()
Return the major version number as an integer.

Return type int

__le__(*other*)
Returns whether this version is less than or equal to the other version.

Return type bool

__lt__(*other*)
Returns whether this version is less than the other version.

Return type bool

__repr__()
Return the representation of the version.

Return type str

__slots__ = ()
Type: tuple

__str__()
Return version as a string.

Return type str

_asdict()
Return a new dict which maps field names to their corresponding values.

New in version 1.4.0.

Return type Dict[str, int]

_field_defaults = {'major': 0, 'minor': 0, 'patch': 0}
Type: Dict[str, int]

Dictionary mapping field names to default values.

New in version 1.4.0.

_fields = ('major', 'minor', 'patch')
Type: Tuple[str, str, str]

Tuple of strings listing the field names.

Useful for introspection and for creating new named tuple types from existing named tuples.

New in version 1.4.0.

classmethod _make(*iterable*)
Class method that makes a new instance from an existing sequence or iterable.

New in version 1.4.0.

Parameters **iterable** (Iterable[Union[str, int]])

Return type ~_V

_replace(***kwargs*)
Return a new instance of the named tuple replacing specified fields with new values.

New in version 1.4.0.

Parameters **kwargs**

Return type ~_V

classmethod from_float(*version_float*)
Create a *Version* from a float.

Parameters **version_float** (float) – The version number.

Return type ~_V

Returns The created *Version*.

classmethod from_str(*version_string*)
Create a *Version* from a str.

Parameters **version_string** (str) – The version number.

Return type ~_V

Returns The created *Version*.

classmethod from_tuple(*version_tuple*)
Create a *Version* from a tuple.

Parameters **version_tuple** (Tuple[Union[int, str], ...]) – The version number.

Return type ~_V

Returns The created *Version*.

Changed in version 0.9.0: Tuples with more than three elements are truncated. Previously a TypeError was raised.

Chapter

NINETEEN

words

Functions for working with (English) words.

New in version 0.4.5.

19.1 Constants

Data:

CR	The carriage return character (\r) for use in f-strings.
LF	The newline character (\n) for use in f-strings.
TAB	A literal TAB (\t) character for use in f-strings.
ascii_digits	ASCII numbers.
greek_lowercase	Lowercase Greek letters.
greek_uppercase	Uppercase Greek letters.

CR = '\r'
Type: str

The carriage return character (\r) for use in f-strings.

New in version 1.3.0.

LF = '\n'
Type: str

The newline character (\n) for use in f-strings.

New in version 1.3.0.

TAB = '\t'
Type: str

A literal TAB (\t) character for use in f-strings.

New in version 1.3.0.

ascii_digits
Type: str

ASCII numbers.

New in version 0.7.0.

greek_uppercase
Type: str

Uppercase Greek letters.

New in version 0.7.0.

greek_lowercase
Type: str

Lowercase Greek letters.

New in version 0.7.0.

19.2 Fonts

Data:

DOUBLESTRUCK_LETTERS	Doublestruck *Font*.
FRAKTUR_LETTERS	Fraktur *Font*.
MONOSPACE_LETTERS	Monospace *Font*.
SANS_SERIF_BOLD_ITALIC_LETTERS	Bold and Italic Sans-Serif *Font*.
SANS_SERIF_BOLD_LETTERS	Bold Sans-Serif *Font*.
SANS_SERIF_ITALIC_LETTERS	Italic Sans-Serif *Font*.
SANS_SERIF_LETTERS	Normal Sans-Serif *Font*.
SCRIPT_LETTERS	Script *Font*.
SERIF_BOLD_ITALIC_LETTERS	Bold and Italic Serif *Font*.
SERIF_BOLD_LETTERS	Bold Serif *Font*.
SERIF_ITALIC_LETTERS	Italic Serif *Font*.

Classes:

Font	Represents a Unicode pseudo-font.

Functions:

make_font(uppers, lowers[, digits, ...])	Returns a dictionary mapping ASCII alphabetical characters and digits to the Unicode equivalents in a different pseudo-font.

make_font (*uppers*, *lowers*, *digits=None*, *greek_uppers=None*, *greek_lowers=None*)

Returns a dictionary mapping ASCII alphabetical characters and digits to the Unicode equivalents in a different pseudo-font.

New in version 0.7.0.

Parameters

- **uppers** (Iterable[str]) – Iterable of uppercase letters (A-Z, 26 characters).
- **lowers** (Iterable[str]) – Iterable of lowercase letters (a-z, 26 characters).
- **digits** (Optional[Iterable[str]]) – Optional iterable of digits (0-9). Default None.
- **greek_uppers** (Optional[Iterable[str]]) – Optional iterable of uppercase Greek letters (A-Ω, 25 characters). Default None.
- **greek_lowers** (Optional[Iterable[str]]) – Optional iterable of lowercase Greek letters (α-ω, 32 characters). Default None.

Return type *Font*

class Font

Bases: Dict[str, str]

Represents a Unicode pseudo-font.

Mapping of ASCII letters to their equivalents in the pseudo-font.

Individual characters can be converted using the *Font.get* method or the getitem syntax. Entire strings can be converted by calling the *Font* object and passing the string as the first argument.

Methods:

__call__(text)	Returns the given text in this font.
__getitem__(char)	Returns the given character in this font.
get(char[, default])	Returns the given character in this font.

__call__ (*text*)
Returns the given text in this font.

Parameters **text** (str)

Return type str

__getitem__ (*char*)
Returns the given character in this font.

If the character is not found in this font the character is returned unchanged.

Parameters **char** (str) – The character to convert.

Return type str

get (*char*, *default=None*)
Returns the given character in this font.

If the character is not found in this font the character is returned unchanged or, if a value for default is provided, that is returned instead.

Parameters

- **char** (str) – The character to convert.
- **default** (Optional[str]) – Optional default value. Default None.

Return type str

SERIF_BOLD_LETTERS
Bold Serif *Font*.

This font includes numbers and Greek letters.

New in version 0.7.0.

SERIF_ITALIC_LETTERS
Italic Serif *Font*.

This font includes Greek letters.

New in version 0.7.0.

SERIF_BOLD_ITALIC_LETTERS
Bold and Italic Serif *Font*.

This font includes Greek letters.

New in version 0.7.0.

SANS_SERIF_LETTERS
Normal Sans-Serif *Font*.

This font includes numbers.

New in version 0.7.0.

SANS_SERIF_BOLD_LETTERS
Bold Sans-Serif *Font*.

This font includes numbers.

New in version 0.7.0.

SANS_SERIF_ITALIC_LETTERS
Italic Sans-Serif *Font*.

New in version 0.7.0.

SANS_SERIF_BOLD_ITALIC_LETTERS
Bold and Italic Sans-Serif *Font*.

This font includes Greek letters.

New in version 0.7.0.

SCRIPT_LETTERS
Script *Font*.

New in version 0.7.0.

FRAKTUR_LETTERS
Fraktur *Font*.

New in version 0.7.0.

MONOSPACE_LETTERS
Monospace *Font*.

This font includes numbers.

New in version 0.7.0.

DOUBLESTRUCK_LETTERS
Doublestruck *Font*.

This font includes numbers.

New in version 0.7.0.

19.3 Functions

Functions:

alpha_sort(iterable, alphabet[, reverse])	Sorts a list of strings using a custom alphabet.
as_text(value)	Convert the given value to a string.
get_random_word([min_length, max_length])	Returns a random word, optionally only one whose length is between `min_length` and `max_length`.
get_words_list([min_length, max_length])	Returns the list of words, optionally only those whose length is between `min_length` and `max_length`.
word_join(iterable[, use_repr, oxford, ...])	Join the given list of strings in a natural manner, with 'and' to join the last two elements.

alpha_sort(*iterable*, *alphabet*, *reverse=False*)
Sorts a list of strings using a custom alphabet.

New in version 0.7.0.

Parameters

- **iterable** (Iterable[str]) – The strings to sort.
- **alphabet** (Iterable[str]) – The custom alphabet to use for sorting.
- **reverse** (bool) – Default False.

Return type List[str]

as_text(*value*)
Convert the given value to a string. None is converted to ''.

Parameters **value** (Any) – The value to convert to a string.

Return type str

Changed in version 0.8.0: Moved from *domdf_python_tools.utils*.

get_words_list(*min_length=0*, *max_length=- 1*)
Returns the list of words, optionally only those whose length is between `min_length` and `max_length`.

New in version 0.4.5.

Parameters

- **min_length** (`int`) – The minimum length of the words to return. Default `0`.
- **max_length** (`int`) – The maximum length of the words to return. A value of `-1` indicates no upper limit.

Return type `List[str]`

Returns The list of words meeting the above specifiers.

get_random_word(*min_length=0*, *max_length=- 1*)
Returns a random word, optionally only one whose length is between `min_length` and `max_length`.

New in version 0.4.5.

Parameters

- **min_length** (`int`) – The minimum length of the words to return. Default `0`.
- **max_length** (`int`) – The maximum length of the words to return. A value of `-1` indicates no upper limit.

Return type `str`

Returns A random word meeting the above specifiers.

word_join(*iterable*, *use_repr=False*, *oxford=False*, *delimiter=','*, *connective='and'*)
Join the given list of strings in a natural manner, with 'and' to join the last two elements.

Parameters

- **iterable** (`Iterable[str]`)
- **use_repr** (`bool`) – Whether to join the `repr` of each object. Default `False`.
- **oxford** (`bool`) – Whether to use an oxford comma when joining the last two elements. Default `False`. Always `False` if there are fewer than three elements.
- **delimiter** (`str`) – A single character to use between the words. Default `','`.
- **connective** (`str`) – The connective to join the final two words with. Default `'and'`.

Return type `str`

Changed in version 0.11.0: Added `delimiter` and `connective` arguments.

truncate_string(*string*, *max_length*, *ending=...*)

Truncate a string to `max_length` characters, and put `ending` on the end.

The truncated string is further truncated by the length of `ending` so the returned string is no more then `max_length`.

New in version 3.3.0.

Parameters

- **string** (`str`)
- **max_length** (`int`)
- **ending** (`str`) – Default `'...'`.

Return type `str`

19.4 Classes

Classes:

Plural(singular, plural)	Represents a word as its singular and plural.
PluralPhrase(template, words)	Represents a phrase which varies depending on a numerical count.

class Plural(*singular*, *plural*)

Bases: `partial`

Represents a word as its singular and plural.

New in version 2.0.0.

Parameters

- **singular** (`str`) – The singular form of the word.
- **plural** (`str`) – The plural form of the word.

```
>>> cow = Plural("cow", "cows")
>>> n = 1
>>> print(f"The farmer has {n} {cow(n)}.")
The farmer has 1 cow.
>>> n = 2
>>> print(f"The farmer has {n} {cow(n)}.")
The farmer has 2 cows.
>>> n = 3
>>> print(f"The farmer has {n} {cow(n)}.")
The farmer has 3 cows.
```

Methods:

__call__(n)	Returns either the singular or plural form of the word depending on the value of `n`.
__repr__()	Return a string representation of the *Plural*.

__call__(*n*)
Returns either the singular or plural form of the word depending on the value of `n`.

Parameters **n** (`int`)

Return type `str`

__repr__()
Return a string representation of the *Plural*.

Return type `str`

namedtuple PluralPhrase(*template*, *words*)
Bases: `NamedTuple`

Represents a phrase which varies depending on a numerical count.

New in version 3.3.0.

Fields

0) **template** (`str`) – The phrase template.
1) **words** (`Tuple[Plural, ...]`) – The words to insert into the template.

For example, consider the phase:

```
The proposed changes are to ...
```

The "phrase template" would be:

```
"The proposed {} {} to ..."
```

and the two words to insert are:

```
Plural("change", "changes")
Plural("is", "are")
```

The phrase is constructed as follows:

```
>>> phrase = PluralPhrase(
...     "The proposed {} {} to ...",
...     (Plural("change", "changes"), Plural("is", "are"))
... )
>>> phrase(1)
'The proposed change is to ...'
>>> phrase(2)
'The proposed changes are to ...'
```

The phrase template can use any valid syntax for `str.format()`, except for keyword arguments. The exception if the keyword `n`, which is replaced with the count (e.g. `2`) passed in when the phrase is constructed. For example:

```
>>> phrase2 = PluralPhrase("The farmer has {n} {0}.", (Plural("cow", "cows"), ))
>>> phrase2(2)
'The farmer has 2 cows.'
```

__call__(*n*)
Construct the phrase based on the value of n.

Parameters **n** (int)

Return type str

__repr__()
Return a string representation of the *PluralPhrase*.

Return type str

Chapter

TWENTY

pagesizes

List of common pagesizes and some tools for working with them.

This module defines a few common page sizes in points (1/72 inch).

20.1 classes

Classes representing pagesizes.

Classes:

BaseSize(width, height)	Base class namedtuple representing a page size, in point.
PageSize(width, height[, unit])	Represents a pagesize in point.
Size_cm(width, height)	Represents a pagesize in centimeters.
Size_inch(width, height)	Represents a pagesize in inches.
Size_mm(width, height)	Represents a pagesize in millimeters.
Size_pica(width, height)	Represents a pagesize in pica.
Size_um(width, height)	Represents a pagesize in micrometers.

namedtuple BaseSize(*width*, *height*)

Bases: NamedTuple

Base class namedtuple representing a page size, in point.

Fields

0) **width** (*Unit*) – The page width.
1) **height** (*Unit*) – The page height.

static __new__(*cls*, *width*, *height*)

Create a new *BaseSize* object.

Parameters

- **width** (Union[float, int, Decimal]) – The page width.
- **height** (Union[float, int, Decimal]) – The page height.

classmethod from_pt(*size*)

Create a *BaseSize* object from a page size in point.

Parameters **size** (Tuple[float, float]) – The size, in point, to convert from.

Return type A subclass of *BaseSize*

classmethod from_size(*size*)
Create a *BaseSize* object from a tuple.

Return type *BaseSize*

is_landscape()
Returns whether the page is in the landscape orientation.

Return type bool

is_portrait()
Returns whether the page is in the portrait orientation.

Return type bool

is_square()
Returns whether the given pagesize is square.

Return type bool

landscape()
Returns the pagesize in landscape orientation.

Return type *BaseSize*

portrait()
Returns the pagesize in portrait orientation.

Return type *BaseSize*

to_pt()
Returns the page size in point.

Return type *PageSize*

namedtuple PageSize(*width*, *height*, *unit=<Unit '1.000 pt': 1.000pt>*)
Bases: *domdf_python_tools.pagesizes.classes.BaseSize*

Represents a pagesize in point.

Fields

0) **width** (*Unit*) – The page width
1) **height** (*Unit*) – The page height

The pagesize can be converted to other units using the properties below.

static __new__(*cls*, *width*, *height*, *unit=<Unit '1.000 pt': 1.000pt>*)
Create a new *PageSize* object.

Parameters

- **width** (Union[float, int, Decimal]) – The page width.
- **height** (Union[float, int, Decimal]) – The page height.
- **unit** (Union[float, int, Decimal]) – Default <Unit '1.000 pt': 1.000pt>.

property cm
Returns the pagesize in centimeters.

Return type *Size_cm*

property inch
Returns the pagesize in inches.

Return type *Size_inch*

property mm
Returns the pagesize in millimeters.

Return type *Size_mm*

property pc
Returns the pagesize in pica.

Return type *Size_pica*

property pica
Returns the pagesize in pica.

Return type *Size_pica*

property pt
Returns the pagesize in pt.

Return type *PageSize*

property um
Returns the pagesize in micrometers.

Return type *Size_um*

property µm
Returns the pagesize in micrometers.

Return type *Size_um*

namedtuple Size_cm(*width*, *height*)
Bases: *domdf_python_tools.pagesizes.classes.BaseSize*

Represents a pagesize in centimeters.

Fields

0) **width** (*Unit*) – The page width.
1) **height** (*Unit*) – The page height.

namedtuple Size_inch(*width*, *height*)

Bases: *domdf_python_tools.pagesizes.classes.BaseSize*

Represents a pagesize in inches.

Fields

0) **width** (*Unit*) – The page width.
1) **height** (*Unit*) – The page height.

namedtuple Size_mm(*width*, *height*)

Bases: *domdf_python_tools.pagesizes.classes.BaseSize*

Represents a pagesize in millimeters.

Fields

0) **width** (*Unit*) – The page width.
1) **height** (*Unit*) – The page height.

namedtuple Size_pica(*width*, *height*)

Bases: *domdf_python_tools.pagesizes.classes.BaseSize*

Represents a pagesize in pica.

Fields

0) **width** (*Unit*) – The page width.
1) **height** (*Unit*) – The page height.

namedtuple Size_um(*width*, *height*)

Bases: *domdf_python_tools.pagesizes.classes.BaseSize*

Represents a pagesize in micrometers.

Fields

0) **width** (*Unit*) – The page width.
1) **height** (*Unit*) – The page height.

20.2 sizes

Common pagesizes in point/pt.

Each pagesize is an instance of `domdf_python_tools.pagesizes.PageSize`. The following sizes are available:

A0	ISO 216 A0 Paper
A1	ISO 216 A1 Paper
A10	ISO 216 A10 Paper
A2	ISO 216 A2 Paper
A2EXTRA	A2 Extra Paper
A3	ISO 216 A3 Paper
A3EXTRA	A3 Extra Paper

continues on next page

Table 2 – continued from previous page

A3SUPER	A3 Super Paper (different to (Super A3)
A4	ISO 216 A3 Paper
A4EXTRA	A4 Extra Paper
A4LONG	A4 Long Paper
A4SUPER	A4 Super Paper (different to (Super A4)
A5	ISO 216 A5 Paper
A5EXTRA	A4 Extra Paper
A6	ISO 216 A6 Paper
A7	ISO 216 A7 Paper
A8	ISO 216 A8 Paper
A9	ISO 216 A0 Paper
ANTIQUARIAN	
ATLAS	
B0	ISO 216 B0 Paper
B1	ISO 216 B1 Paper
B10	ISO 216 B10 Paper
B2	ISO 216 B2 Paper
B3	ISO 216 B3 Paper
B4	ISO 216 B4 Paper
B5	ISO 216 B5 Paper
B6	ISO 216 B6 Paper
B7	ISO 216 B7 Paper
B8	ISO 216 B8 Paper
B9	ISO 216 B9 Paper
BRIEF	
BROADSHEET	
C0	ISO 216 C0 Paper
C1	ISO 216 C1 Paper
C10	ISO 216 C10 Paper
C2	ISO 216 C2 Paper
C3	ISO 216 C3 Paper
C4	ISO 216 C4 Paper
C5	ISO 216 C5 Paper
C6	ISO 216 C6 Paper
C7	ISO 216 C7 Paper
C8	ISO 216 C8 Paper
C9	ISO 216 C9 Paper
CARTRIDGE	
COLOMBIER	
COPY_DRAUGHT	
CROWN	
DEMY	
DOUBLE_DEMY	
DOUBLE_DEMY_UK	
DOUBLE_DEMY_US	
DOUBLE_ELEPHANT	
DOUBLE_LARGE_POST	
DOUBLE_POST	
DOUBLE_ROYAL	
DUKES	

continues on next page

Table 2 – continued from previous page

ELEPHANT	
EMPEROR	Emperor
EXECUTIVE	
FOLIO	
FOOLSCAP_FOLIO	
FOOLSCAP_UK	
FOOLSCAP_US	
GOV_LEGAL	Government Legal
GOV_LETTER	Government Letter
GRAND_EAGLE	
HALF_LETTER	Half Letter
HALF_POST	
ID_000	SIM cards
ID_1	Most banking cards and ID cards
ID_2	French and other ID cards; Visas
ID_3	US government ID cards
IMPERIAL	
JUNIOR_LEGAL	Junior Legal
KINGS	
LARGE_POST_UK	
LARGE_POST_US	
LEDGER	Ledger
LEGAL	North American “Legal” Paper
LETTER	North American “Letter” Paper
MEDIUM_UK	
MEDIUM_US	
MONARCH	
PINCHED_POST	
POST_UK	
POST_US	
POTT	
PRINCESS	
QUAD_DEMY	
QUAD_ROYAL	Quad Royal
QUARTO	
QUARTO_UK	
QUARTO_US	
ROYAL	
SHEET	
SMALL_FOOLSCAP	
SOB5EXTRA	SO B5 Extra Paper
SUPERA3	Super A3 Paper (different to A3 Super)
SUPERA4	Super A3 Paper (different to A4 Super)
SUPER_ROYAL	

20.3 units

Provides a variety of units for use with pagesizes.

Classes:

Unit([x])	Represents a unit, such as a point.
UnitInch([x])	Inch.
Unitcm([x])	Centimetres.
Unitmm([x])	Millimetres.
Unitpc([x])	Pica.
Unitpt([x])	Point.
Unitum([x])	Micrometres.

Data:

cm([value])	Centimetre
inch([value])	Inch
mm([value])	Millimetre
pc([value])	Pica
pica([value])	Pica
pt([value])	Point
um([value])	Micrometre

class Unit(*x=0, /*)

Bases: float

Represents a unit, such as a point.

Behaves much like a float (which it inherits from).

Addition

Units can be added to each other:

```
>>> (3*mm) + (7*mm)
<Unit '10.000 mm': 28.346pt>
```

When adding different *Unit* objects, the result has the type of the former unit:

```
>>> (2.54*cm) + inch
<Unit '5.080 cm': 144.000pt>
>>> inch + (2.54*cm)
<Unit '2.000 inch': 144.000pt>
```

Unit objects can also be added to float and int objects:

```
>>> (3*cm) + 7
<Unit '10.000 cm': 283.465pt>
>>> 7 + (3*cm)
<Unit '10.000 cm': 283.465pt>
```

Subtraction

Subtraction works the same as addition:

```
>>> (17*mm) - (7*mm)
<Unit '10.000 mm': 28.346pt>
>>> (2.54*cm) - inch
<Unit '0.000 cm': 0.000pt>
>>> inch - (2.54*cm)
<Unit '0.000 inch': 0.000pt>
>>> (17*cm) - 7
<Unit '10.000 cm': 283.465pt>
>>> 17 - (7*cm)
<Unit '10.000 cm': 283.465pt>
```

Multiplication

Unit objects can only be multipled by `float` and `int` objects:

```
>>> (3*mm) * 3
<Unit '9.000 mm': 25.512pt>
>>> 3 * (3*mm)
<Unit '9.000 mm': 25.512pt>
>>> 3.5 * (3*mm)
<Unit '10.500 mm': 29.764pt>
```

Multiplication works either way round.

Multiplying by another *Unit* results in a `NotImplementedError`:

```
>>> inch * (7*cm)
Traceback (most recent call last):
NotImplementedError: Multiplying a unit by another unit is not allowed.
```

Division

*Unit*s can only be divided by `float` and `int` objects:

```
>>> (3*mm) / 3
<Unit '1.000 mm': 2.835pt>
>>> (10*mm) / 2.5
<Unit '4.000 mm': 11.339pt>
```

Dividing by another unit results in a `NotImplementedError`:

```
>>> inch / (7*cm)
Traceback (most recent call last):
NotImplementedError: Dividing a unit by another unit is not allowed.
```

Likewise, trying to divide a:class:*float* and `int` object by a unit results in a `NotImplementedError`:

```
>>> 3 / (3*mm)
Traceback (most recent call last):
NotImplementedError: Dividing by a unit is not allowed.
```

Powers

Powers (using `**`) are not officially supported.

Modulo Division

Modulo division of a *Unit* by a float or int object is allowed:

```
>>> (3*mm) % 2.5
<Unit '0.500 mm': 1.417pt>
```

Dividing by a unit, or modulo division of two units, is not officially supported.

Methods:

__add__(other)	Return `self + value`.
__call__([value])	Returns an instance of the *Unit* with the given value.
__eq__(other)	Return `self == other`.
__floordiv__(other)	Return `self // value`.
__mod__(other)	Return `self % value`.
__mul__(other)	Return `self * value`.
__pow__(power[, modulo])	Return `pow(self, value, mod)`.
__radd__(other)	Return `self + value`.
__rdiv__(other)	Return `value / self`.
__repr__()	Return a string representation of the *Unit*.
__rmul__(other)	Return `self * value`.
__rsub__(other)	Return `value - self`.
__rtruediv__(other)	Return `value / self`.
__str__()	Return `str(self)`.
__sub__(other)	Return `value - self`.
__truediv__(other)	Return `self / value`.
as_pt()	Returns the unit in point.
from_pt(value)	Construct a *Unit* object from a value in point.

Attributes:

name

__add__(*other*)
Return `self + value`.

Return type *Unit*

__call__(*value=0.0*)
Returns an instance of the *Unit* with the given value.

Parameters **value** (Union[SupportsFloat, str, bytes, bytearray]) – Default `0.0`.

Return type *Unit*

__eq__(*other*)
Return `self == other`.

Return type bool

__floordiv__(*other*)
Return `self // value`.

Return type *Unit*

__mod__(*other*)
Return `self % value`.

Return type *Unit*

__mul__(*other*)
Return `self * value`.

Return type *Unit*

__pow__(*power*, *modulo=None*)
Return `pow(self, value, mod)`.

__radd__(*other*)
Return `self + value`.

Return type *Unit*

__rdiv__(*other*)
Return `value / self`.

__repr__()
Return a string representation of the *Unit*.

Return type str

__rmul__(*other*)
Return `self * value`.

Return type *Unit*

__rsub__(*other*)
Return `value - self`.

Return type *Unit*

__rtruediv__(*other*)
Return `value / self`.

__str__()
Return `str(self)`.

Return type str

__sub__(*other*)
Return `value - self`.

Return type *Unit*

__truediv__(*other*)
Return `self / value`.

Return type *Unit*

as_pt()
Returns the unit in point.

Return type *Unit*

classmethod from_pt(*value*)
Construct a *Unit* object from a value in point.

Parameters **value** (float)

Return type *Unit*

name = 'pt'
Type: str

class Unitpt(*x=0*, */*)
Bases: *Unit*

Point.

class UnitInch(*x=0*, */*)
Bases: *Unit*

Inch.

class Unitcm(*x=0*, */*)
Bases: *Unit*

Centimetres.

class Unitmm(*x=0*, */*)
Bases: *Unit*

Millimetres.

class Unitum(*x=0*, */*)
Bases: *Unit*

Micrometres.

class Unitpc(*x=0*, */*)
Bases: *Unit*

Pica.

pt(*value=0.0*) **= <Unit '1.000 pt': 1.000pt>**
Type: *Unitpt*

Point

inch(*value=0.0*) **= <Unit '1.000 inch': 72.000pt>**
Type: *UnitInch*

Inch

cm(*value=0.0*) **= <Unit '1.000 cm': 28.346pt>**
Type: *Unitcm*

Centimetre

mm(*value=0.0*) **= <Unit '1.000 mm': 2.835pt>**
Type: *Unitmm*

Millimetre

um(*value=0.0*) **= <Unit '1.000 µm': 0.003pt>**
Type: *Unitum*

Micrometre

pc(*value=0.0*) **= <Unit '1.000 pc': 12.000pt>**
Type: *Unitpc*

Pica

20.4 utils

Tools for working with pagesizes.

Functions:

convert_from(value, from_)	Convert `value` to point from the unit specified in `from_`.
parse_measurement(measurement)	Parse the given measurement.

convert_from(*value*, *from_*)
Convert `value` to point from the unit specified in `from_`.

Parameters

- **value** (Union[Sequence[Union[float, int, Decimal]], float, int, Decimal])
- **from_** (Union[float, int, Decimal]) – The unit to convert from, specified as a number of points.

Return type Union[float, Tuple[float, ...]]

Overloads

- *convert_from*(**value**: Sequence[Union[float, int, Decimal]], **from_**) -> Tuple[float, ...]
- *convert_from*(**value**: Union[float, int, Decimal], **from_**) -> float

parse_measurement(*measurement*)
Parse the given measurement.

Parameters **measurement** (str)

Return type Union[float, Tuple[float, ...]]

Part II

Contributing

Chapter

TWENTYONE

Overview

`domdf_python_tools` uses tox to automate testing and packaging, and pre-commit to maintain code quality.

Install `pre-commit` with `pip` and install the git hook:

```
$ python -m pip install pre-commit
$ pre-commit install
```

Chapter

TWENTYTWO

Coding style

formate is used for code formatting.

It can be run manually via `pre-commit`:

```
$ pre-commit run formate -a
```

Or, to run the complete autoformatting suite:

```
$ pre-commit run -a
```

Chapter

TWENTYTHREE

Automated tests

Tests are run with `tox` and `pytest`. To run tests for a specific Python version, such as Python 3.6:

```
$ tox -e py36
```

To run tests for all Python versions, simply run:

```
$ tox
```

Chapter

TWENTYFOUR

Type Annotations

Type annotations are checked using `mypy`. Run `mypy` using `tox`:

```
$ tox -e mypy
```

Chapter

TWENTYFIVE

Build documentation locally

The documentation is powered by Sphinx. A local copy of the documentation can be built with `tox`:

```
$ tox -e docs
```

Chapter

TWENTYSIX

Downloading source code

The `domdf_python_tools` source code is available on GitHub, and can be accessed from the following URL: https://github.com/domdfcoding/domdf_python_tools

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/domdfcoding/domdf_python_tools
```

```
Cloning into 'domdf_python_tools'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

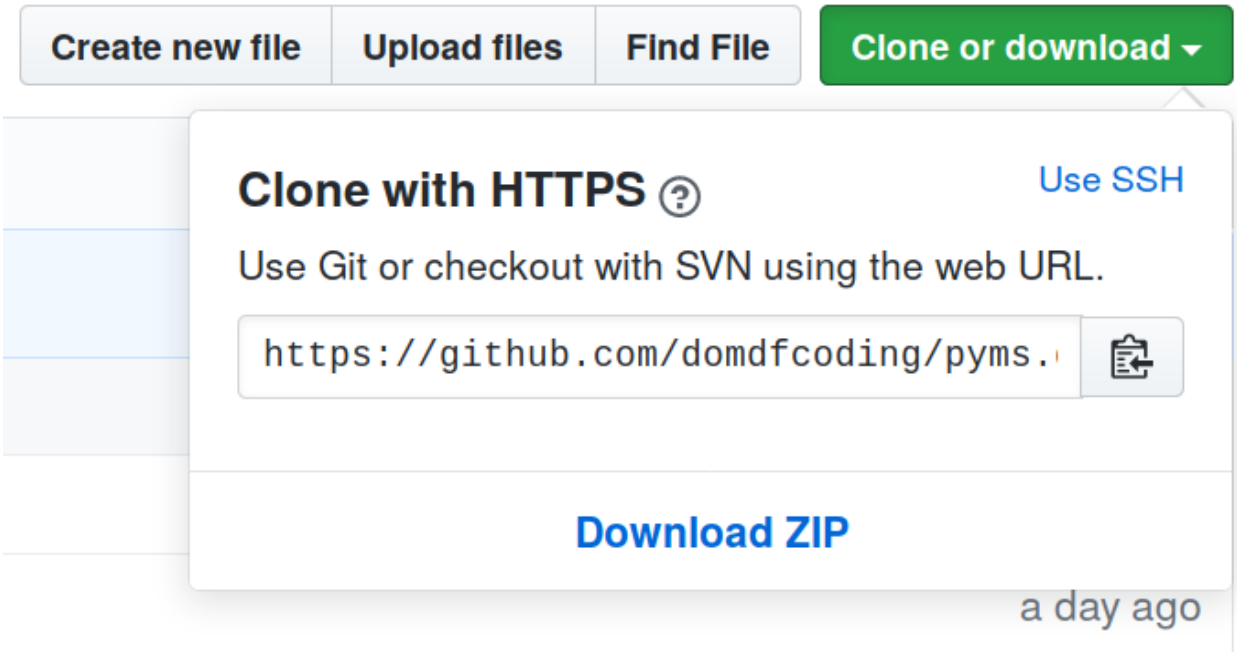

Fig. 1: Downloading a 'zip' file of the source code

26.1 Building from source

The recommended way to build `domdf_python_tools` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Chapter

TWENTYSEVEN

License

`domdf_python_tools` is licensed under the MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

Python Module Index

d

Index

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W