:mod:`dataclasses` --- Data Classes

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 1); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 4)

Unknown directive type "module".

```
.. module:: dataclasses
    :synopsis: Generate special methods on user-defined classes.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 7)

Unknown directive type "moduleauthor".

```
.. moduleauthor:: Eric V. Smith <eric@trueblade.com>
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 8)

Unknown directive type "sectionauthor".

```
.. sectionauthor:: Eric V. Smith <eric@trueblade.com>
```

Source code: :source:`Lib/dataclasses.py`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 10); *backlink*

Unknown interpreted text role "source".

This module provides a decorator and functions for automatically adding generated :term:`special method`s such as :meth:`__init__` and :meth:`__repr__` to user-defined classes. It was originally described in PEP 557.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 14); *backlink*

Unknown interpreted text role "term".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 14); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 14); *backlink*

Unknown interpreted text role "meth".

The member variables to use in these generated methods are defined using PEP 526 type annotations. For example, this code:

```
from dataclasses import dataclass

@dataclass
class InventoryItem:
    """Class for keeping track of an item in inventory."""
    name: str
    unit_price: float
    quantity_on_hand: int = 0

    def total_cost(self) -> float:
        return self.unit_price * self.quantity_on_hand
```

will add, among other things, a :meth:`__init__` that looks like:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 34); *backlink*

Unknown interpreted text role "meth".

```
def __init__(self, name: str, unit_price: float, quantity_on_hand: int = 0):
    self.name = name
    self.unit_price = unit_price
    self.quantity_on_hand = quantity_on_hand
```

Note that this method is automatically added to the class: it is not directly specified in the `InventoryItem` definition shown above.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 44)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7
```

Module contents

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 49)

Unknown directive type "decorator".

```
.. decorator:: dataclass(*, init=True, repr=True, eq=True, order=False, unsafe_hash=False, frozen=False, match_args=True, kw_only=

   This function is a :term:`decorator` that is used to add generated
   :term:`special method`\s to classes, as described below.

   The :func:`dataclass` decorator examines the class to find
   ``field``\s.  A ``field`` is defined as a class variable that has a
   :term:`type annotation <variable annotation>`.  With two
   exceptions described below, nothing in :func:`dataclass`
   examines the type specified in the variable annotation.
```

```
The order of the fields in all of the generated methods is the
order in which they appear in the class definition.

The :func:`dataclass` decorator will add various "dunder" methods to
the class, described below.  If any of the added methods already
exist in the class, the behavior depends on the parameter, as documented
below. The decorator returns the same class that it is called on; no new
class is created.

If :func:`dataclass` is used just as a simple decorator with no parameters,
it acts as if it has the default values documented in this
signature.  That is, these three uses of :func:`dataclass` are
equivalent::

  @dataclass
  class C:
      ...

  @dataclass()
  class C:
      ...

  @dataclass(init=True, repr=True, eq=True, order=False, unsafe_hash=False, frozen=False, match_args=True, kw_only=False, slots
  class C:
     ...

The parameters to :func:`dataclass` are:

- ``init``: If true (the default), a :meth:`__init__` method will be
  generated.

  If the class already defines :meth:`__init__`, this parameter is
  ignored.

- ``repr``: If true (the default), a :meth:`__repr__` method will be
  generated.  The generated repr string will have the class name and
  the name and repr of each field, in the order they are defined in
  the class.  Fields that are marked as being excluded from the repr
  are not included.  For example:
  ``InventoryItem(name='widget', unit_price=3.0, quantity_on_hand=10)``.

  If the class already defines :meth:`__repr__`, this parameter is
  ignored.

- ``eq``: If true (the default), an :meth:`__eq__` method will be
  generated.  This method compares the class as if it were a tuple
  of its fields, in order.  Both instances in the comparison must
  be of the identical type.

  If the class already defines :meth:`__eq__`, this parameter is
  ignored.

- ``order``: If true (the default is ``False``), :meth:`__lt__`,
  :meth:`__le__`, :meth:`__gt__`, and :meth:`__ge__` methods will be
  generated.  These compare the class as if it were a tuple of its
  fields, in order.  Both instances in the comparison must be of the
  identical type.  If ``order`` is true and ``eq`` is false, a
  :exc:`ValueError` is raised.

  If the class already defines any of :meth:`__lt__`,
  :meth:`__le__`, :meth:`__gt__`, or :meth:`__ge__`, then
  :exc:`TypeError` is raised.

- ``unsafe_hash``: If ``False`` (the default), a :meth:`__hash__` method
  is generated according to how ``eq`` and ``frozen`` are set.

  :meth:`__hash__` is used by built-in :meth:`hash()`, and when objects are
  added to hashed collections such as dictionaries and sets.  Having a
  :meth:`__hash__` implies that instances of the class are immutable.
  Mutability is a complicated property that depends on the programmer's
  intent, the existence and behavior of :meth:`__eq__`, and the values of
  the ``eq`` and ``frozen`` flags in the :func:`dataclass` decorator.

  By default, :func:`dataclass` will not implicitly add a :meth:`__hash__`
  method unless it is safe to do so.  Neither will it add or change an
  existing explicitly defined :meth:`__hash__` method.  Setting the class
  attribute ``__hash__ = None`` has a specific meaning to Python, as
  described in the :meth:`__hash__` documentation.

  If :meth:`__hash__` is not explicitly defined, or if it is set to ``None``,
  then :func:`dataclass` *may* add an implicit :meth:`__hash__` method.
  Although not recommended, you can force :func:`dataclass` to create a
  :meth:`__hash__` method with ``unsafe_hash=True``. This might be the case
  if your class is logically immutable but can nonetheless be mutated.
  This is a specialized use case and should be considered carefully.

  Here are the rules governing implicit creation of a :meth:`__hash__`
  method.  Note that you cannot both have an explicit :meth:`__hash__`
  method in your dataclass and set ``unsafe_hash=True``; this will result
  in a :exc:`TypeError`.

  If ``eq`` and ``frozen`` are both true, by default :func:`dataclass` will
  generate a :meth:`__hash__` method for you.  If ``eq`` is true and
  ``frozen`` is false, :meth:`__hash__` will be set to ``None``, marking it
  unhashable (which it is, since it is mutable).  If ``eq`` is false,
  :meth:`__hash__` will be left untouched meaning the :meth:`__hash__`
  method of the superclass will be used (if the superclass is
  :class:`object`, this means it will fall back to id-based hashing).

- ``frozen``: If true (the default is ``False``), assigning to fields will
  generate an exception.  This emulates read-only frozen instances.  If
  :meth:`__setattr__` or :meth:`__delattr__` is defined in the class, then
  :exc:`TypeError` is raised.  See the discussion below.

- ``match_args``: If true (the default is ``True``), the
  ``__match_args__`` tuple will be created from the list of
  parameters to the generated :meth:`__init__` method (even if
  :meth:`__init__` is not generated, see above).  If false, or if
  ``__match_args__`` is already defined in the class, then
  ``__match_args__`` will not be generated.

  .. versionadded:: 3.10

- ``kw_only``: If true (the default value is ``False``), then all
  fields will be marked as keyword-only.  If a field is marked as
  keyword-only, then the only effect is that the :meth:`__init__`
  parameter generated from a keyword-only field must be specified
  with a keyword when :meth:`__init__` is called.  There is no
  effect on any other aspect of dataclasses.  See the
  :term:`parameter` glossary entry for details.  Also see the
  :const:`KW_ONLY` section.

  .. versionadded:: 3.10

- ``slots``: If true (the default is ``False``), :attr:`__slots__` attribute
  will be generated and new class will be returned instead of the original one.
  If :attr:`__slots__` is already defined in the class, then :exc:`TypeError`
  is raised.

  .. versionadded:: 3.10
```

```
   .. versionchanged:: 3.11
      If a field name is already included in the ``__slots__``
      of a base class, it will not be included in the generated ``__slots__``
      to prevent `overriding them <https://docs.python.org/3/reference/datamodel.html#notes-on-using-slots>`_.
      Therefore, do not use ``__slots__`` to retrieve the field names of a
      dataclass. Use :func:`fields` instead.
      To be able to determine inherited slots,
      base class ``__slots__`` may be any iterable, but *not* an iterator.


   ``field``\s may optionally specify a default value, using normal
   Python syntax::

     @dataclass
     class C:
         a: int       # 'a' has no default value
         b: int = 0   # assign a default value for 'b'

   In this example, both ``a`` and ``b`` will be included in the added
   :meth:`__init__` method, which will be defined as::

     def __init__(self, a: int, b: int = 0):

   :exc:`TypeError` will be raised if a field without a default value
   follows a field with a default value.  This is true whether this
   occurs in a single class, or as a result of class inheritance.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 218)

Unknown directive type "function".

```
.. function:: field(*, default=MISSING, default_factory=MISSING, init=True, repr=True, hash=None, compare=True, metadata=None, kw_

   For common and simple use cases, no other functionality is
   required.  There are, however, some dataclass features that
   require additional per-field information.  To satisfy this need for
   additional information, you can replace the default field value
   with a call to the provided :func:`field` function.  For example::

     @dataclass
     class C:
         mylist: list[int] = field(default_factory=list)

     c = C()
     c.mylist += [1, 2, 3]

   As shown above, the :const:`MISSING` value is a sentinel object used to
   detect if some parameters are provided by the user. This sentinel is
   used because ``None`` is a valid value for some parameters with
   a distinct meaning.  No code should directly use the :const:`MISSING` value.

   The parameters to :func:`field` are:

   - ``default``: If provided, this will be the default value for this
     field.  This is needed because the :meth:`field` call itself
     replaces the normal position of the default value.

   - ``default_factory``: If provided, it must be a zero-argument
     callable that will be called when a default value is needed for
     this field.  Among other purposes, this can be used to specify
     fields with mutable default values, as discussed below.  It is an
     error to specify both ``default`` and ``default_factory``.

   - ``init``: If true (the default), this field is included as a
     parameter to the generated :meth:`__init__` method.

   - ``repr``: If true (the default), this field is included in the
     string returned by the generated :meth:`__repr__` method.

   - ``hash``: This can be a bool or ``None``.  If true, this field is
     included in the generated :meth:`__hash__` method.  If ``None`` (the
     default), use the value of ``compare``: this would normally be
     the expected behavior.  A field should be considered in the hash
     if it's used for comparisons.  Setting this value to anything
     other than ``None`` is discouraged.

     One possible reason to set ``hash=False`` but ``compare=True``
     would be if a field is expensive to compute a hash value for,
     that field is needed for equality testing, and there are other
     fields that contribute to the type's hash value.  Even if a field
     is excluded from the hash, it will still be used for comparisons.

   - ``compare``: If true (the default), this field is included in the
     generated equality and comparison methods (:meth:`__eq__`,
     :meth:`__gt__`, et al.).

   - ``metadata``: This can be a mapping or None. None is treated as
     an empty dict.  This value is wrapped in
     :func:`~types.MappingProxyType` to make it read-only, and exposed
     on the :class:`Field` object. It is not used at all by Data
     Classes, and is provided as a third-party extension mechanism.
     Multiple third-parties can each have their own key, to use as a
     namespace in the metadata.

   - ``kw_only``: If true, this field will be marked as keyword-only.
     This is used when the generated :meth:`__init__` method's
     parameters are computed.

    .. versionadded:: 3.10

   If the default value of a field is specified by a call to
   :func:`field()`, then the class attribute for this field will be
   replaced by the specified ``default`` value.  If no ``default`` is
   provided, then the class attribute will be deleted.  The intent is
   that after the :func:`dataclass` decorator runs, the class
   attributes will all contain the default values for the fields, just
   as if the default value itself were specified.  For example,
   after::

     @dataclass
     class C:
         x: int
         y: int = field(repr=False)
         z: int = field(repr=False, default=10)
         t: int = 20

   The class attribute ``C.z`` will be ``10``, the class attribute
   ``C.t`` will be ``20``, and the class attributes ``C.x`` and
   ``C.y`` will not be set.
```

:class:`Field` objects describe each defined field. These objects are created internally, and are returned by the :func:`fields` module-level method (see below). Users should never instantiate a :class:`Field` object directly. Its documented attributes are:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 309); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 309); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 309); *backlink*

Unknown interpreted text role "class".

- `name`: The name of the field.
- `type`: The type of the field.
- `default`, `default_factory`, `init`, `repr`, `hash`, `compare`, `metadata`, and `kw_only` have the identical meaning and values as they do in the :func:`field` function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 318); *backlink*

Unknown interpreted text role "func".

Other attributes may exist, but they are private and must not be inspected or relied on.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 325)

Unknown directive type "function".

```
.. function:: fields(class_or_instance)

   Returns a tuple of :class:`Field` objects that define the fields for this
   dataclass.  Accepts either a dataclass, or an instance of a dataclass.
   Raises :exc:`TypeError` if not passed a dataclass or instance of one.
   Does not return pseudo-fields which are ``ClassVar`` or ``InitVar``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 332)

Unknown directive type "function".

```
.. function:: asdict(obj, *, dict_factory=dict)

   Converts the dataclass ``obj`` to a dict (by using the
   factory function ``dict_factory``).  Each dataclass is converted
   to a dict of its fields, as ``name: value`` pairs.  dataclasses, dicts,
   lists, and tuples are recursed into.  Other objects are copied with
   :func:`copy.deepcopy`.

   Example of using :func:`asdict` on nested dataclasses::

     @dataclass
     class Point:
          x: int
          y: int

     @dataclass
     class C:
          mylist: list[Point]

     p = Point(10, 20)
     assert asdict(p) == {'x': 10, 'y': 20}

     c = C([Point(0, 0), Point(10, 4)])
     assert asdict(c) == {'mylist': [{'x': 0, 'y': 0}, {'x': 10, 'y': 4}]}

   To create a shallow copy, the following workaround may be used::

     dict((field.name, getattr(obj, field.name)) for field in fields(obj))

   :func:`asdict` raises :exc:`TypeError` if ``obj`` is not a dataclass
   instance.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 364)

Unknown directive type "function".

```
.. function:: astuple(obj, *, tuple_factory=tuple)

   Converts the dataclass ``obj`` to a tuple (by using the
   factory function ``tuple_factory``).  Each dataclass is converted
   to a tuple of its field values.  dataclasses, dicts, lists, and
   tuples are recursed into. Other objects are copied with
   :func:`copy.deepcopy`.

   Continuing from the previous example::

     assert astuple(p) == (10, 20)
     assert astuple(c) == ([(0, 0), (10, 4)],)

   To create a shallow copy, the following workaround may be used::

     tuple(getattr(obj, field.name) for field in dataclasses.fields(obj))

   :func:`astuple` raises :exc:`TypeError` if ``obj`` is not a dataclass
   instance.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 384)

Unknown directive type "function".

```
.. function:: make_dataclass(cls_name, fields, *, bases=(), namespace=None, init=True, repr=True, eq=True, order=False, unsafe_hash

   Creates a new dataclass with name ``cls_name``, fields as defined
   in ``fields``, base classes as given in ``bases``, and initialized
   with a namespace as given in ``namespace``.  ``fields`` is an
   iterable whose elements are each either ``name``, ``(name, type)``,
   or ``(name, type, Field)``.  If just ``name`` is supplied,
   ``typing.Any`` is used for ``type``.  The values of ``init``,
   ``repr``, ``eq``, ``order``, ``unsafe_hash``, ``frozen``,
   ``match_args``, ``kw_only``, and  ``slots`` have the same meaning as
   they do in :func:`dataclass`.
```

```
   This function is not strictly required, because any Python
   mechanism for creating a new class with ``__annotations__`` can
   then apply the :func:`dataclass` function to convert that class to
   a dataclass.  This function is provided as a convenience.  For
   example::

     C = make_dataclass('C',
                        [('x', int),
                          'y',
                         ('z', int, field(default=5))],
                        namespace={'add_one': lambda self: self.x + 1})

   Is equivalent to::

     @dataclass
     class C:
         x: int
         y: 'typing.Any'
         z: int = 5

         def add_one(self):
             return self.x + 1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 419)

Unknown directive type "function".

```
.. function:: replace(obj, /, **changes)

   Creates a new object of the same type as ``obj``, replacing
   fields with values from ``changes``.  If ``obj`` is not a Data
   Class, raises :exc:`TypeError`.  If values in ``changes`` do not
   specify fields, raises :exc:`TypeError`.

   The newly returned object is created by calling the :meth:`__init__`
   method of the dataclass.  This ensures that
   :meth:`__post_init__`, if present, is also called.

   Init-only variables without default values, if any exist, must be
   specified on the call to :func:`replace` so that they can be passed to
   :meth:`__init__` and :meth:`__post_init__`.

   It is an error for ``changes`` to contain any fields that are
   defined as having ``init=False``.  A :exc:`ValueError` will be raised
   in this case.

   Be forewarned about how ``init=False`` fields work during a call to
   :func:`replace`.  They are not copied from the source object, but
   rather are initialized in :meth:`__post_init__`, if they're
   initialized at all.  It is expected that ``init=False`` fields will
   be rarely and judiciously used.  If they are used, it might be wise
   to have alternate class constructors, or perhaps a custom
   ``replace()`` (or similarly named) method which handles instance
   copying.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 447)

Unknown directive type "function".

```
.. function:: is_dataclass(obj)

   Return ``True`` if its parameter is a dataclass or an instance of one,
   otherwise return ``False``.

   If you need to know if a class is an instance of a dataclass (and
   not a dataclass itself), then add a further check for ``not
   isinstance(obj, type)``::

     def is_dataclass_instance(obj):
         return is_dataclass(obj) and not isinstance(obj, type)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 459)

Unknown directive type "data".

```
.. data:: MISSING

   A sentinel value signifying a missing default or default_factory.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 463)

Unknown directive type "data".

```
.. data:: KW_ONLY

   A sentinel value used as a type annotation.  Any fields after a
   pseudo-field with the type of :const:`KW_ONLY` are marked as
   keyword-only fields.  Note that a pseudo-field of type
   :const:`KW_ONLY` is otherwise completely ignored.  This includes the
   name of such a field.  By convention, a name of ``_`` is used for a
   :const:`KW_ONLY` field.  Keyword-only fields signify
   :meth:`__init__` parameters that must be specified as keywords when
   the class is instantiated.

   In this example, the fields ``y`` and ``z`` will be marked as keyword-only fields::

    @dataclass
    class Point:
      x: float
      _: KW_ONLY
      y: float
      z: float

    p = Point(0, y=1.5, z=2.0)

   In a single dataclass, it is an error to specify more than one
   field whose type is :const:`KW_ONLY`.

   .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 490)

Unknown directive type "exception".

```
.. exception:: FrozenInstanceError

   Raised when an implicitly defined :meth:`__setattr__` or
   :meth:`__delattr__` is called on a dataclass which was defined with
   ``frozen=True``. It is a subclass of :exc:`AttributeError`.
```

Post-init processing

The generated :meth:`__init__` code will call a method named :meth:`__post_init__`, if :meth:`__post_init__` is defined on the class. It will normally be called as `self.__post_init__()`. However, if any `InitVar` fields are defined, they will also be passed to :meth:`__post_init__` in the order they were defined in the class. If no :meth:`__init__` method is generated, then :meth:`__post_init__` will not automatically be called.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 499**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 499**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 499**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 499**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 499**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 499**); ***backlink***

Unknown interpreted text role "meth".

Among other uses, this allows for initializing field values that depend on one or more other fields. For example:

```
@dataclass
class C:
    a: float
    b: float
    c: float = field(init=False)

    def __post_init__(self):
        self.c = self.a + self.b
```

The :meth:`__init__` method generated by :func:`dataclass` does not call base class :meth:`__init__` methods. If the base class has an :meth:`__init__` method that has to be called, it is common to call this method in a :meth:`__post_init__` method:

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 519**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 519**); ***backlink***

Unknown interpreted text role "func".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 519**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 519**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 519**); ***backlink***

Unknown interpreted text role "meth".

```
@dataclass
class Rectangle:
    height: float
    width: float

@dataclass
class Square(Rectangle):
    side: float

    def __post_init__(self):
        super().__init__(self.side, self.side)
```

Note, however, that in general the dataclass-generated :meth:`__init__` methods don't need to be called, since the derived dataclass will take care of initializing all fields of any base class that is a dataclass itself.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 536**); ***backlink***

Unknown interpreted text role "meth".

See the section below on init-only variables for ways to pass parameters to :meth:`__post_init__`. Also see the warning about how :func:`replace` handles `init=False` fields.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst`, **line 540**); ***backlink***

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 540); *backlink*

Unknown interpreted text role "func".

Class variables

One of two places where :func:`dataclass` actually inspects the type of a field is to determine if a field is a class variable as defined in PEP 526. It does this by checking if the type of the field is `typing.ClassVar`. If a field is a `ClassVar`, it is excluded from consideration as a field and is ignored by the dataclass mechanisms. Such `ClassVar` pseudo-fields are not returned by the module-level :func:`fields` function.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 547); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 547); *backlink*

Unknown interpreted text role "func".

Init-only variables

The other place where :func:`dataclass` inspects a type annotation is to determine if a field is an init-only variable. It does this by seeing if the type of a field is of type `dataclasses.InitVar`. If a field is an `InitVar`, it is considered a pseudo-field called an init-only field. As it is not a true field, it is not returned by the module-level :func:`fields` function. Init-only fields are added as parameters to the generated :meth:`__init__` method, and are passed to the optional :meth:`__post_init__` method. They are not otherwise used by dataclasses.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 558); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 558); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 558); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 558); *backlink*

Unknown interpreted text role "meth".

For example, suppose a field will be initialized from a database, if a value is not provided when creating the class:

```
@dataclass
class C:
    i: int
    j: int = None
    database: InitVar[DatabaseType] = None

    def __post_init__(self, database):
        if self.j is None and database is not None:
            self.j = database.lookup('j')

c = C(10, database=my_database)
```

In this case, :func:`fields` will return :class:`Field` objects for `i` and `j`, but not for `database`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 583); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 583); *backlink*

Unknown interpreted text role "class".

Frozen instances

It is not possible to create truly immutable Python objects. However, by passing `frozen=True` to the :meth:`dataclass` decorator you can emulate immutability. In that case, dataclasses will add :meth:`__setattr__` and :meth:`__delattr__` methods to the class. These methods will raise a :exc:`FrozenInstanceError` when invoked.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 589); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 589); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 589); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 589); *backlink*

Unknown interpreted text role "exc".

There is a tiny performance penalty when using `frozen=True`: :meth:`__init__` cannot use simple assignment to initialize fields, and must use :meth:`object.__setattr__`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 595); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 595); backlink

Unknown interpreted text role "meth".

Inheritance

When the dataclass is being created by the :meth:`dataclass` decorator, it looks through all of the class's base classes in reverse MRO (that is, starting at :class:`object`) and, for each dataclass that it finds, adds the fields from that base class to an ordered mapping of fields. After all of the base class fields are added, it adds its own fields to the ordered mapping. All of the generated methods will use this combined, calculated ordered mapping of fields. Because the fields are in insertion order, derived classes override base classes. An example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 602); backlink

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 602); backlink

Unknown interpreted text role "class".

```
@dataclass
class Base:
    x: Any = 15.0
    y: int = 0

@dataclass
class C(Base):
    z: int = 10
    x: int = 15
```

The final list of fields is, in order, `x`, `y`, `z`. The final type of `x` is `int`, as specified in class `C`.

The generated :meth:`__init__` method for `C` will look like:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 625); backlink

Unknown interpreted text role "meth".

```
def __init__(self, x: int = 15, y: int = 0, z: int = 10):
```

Re-ordering of keyword-only parameters in :meth:`__init__`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 629); backlink

Unknown interpreted text role "meth".

After the parameters needed for :meth:`__init__` are computed, any keyword-only parameters are moved to come after all regular (non-keyword-only) parameters. This is a requirement of how keyword-only parameters are implemented in Python: they must come after non-keyword-only parameters.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 632); backlink

Unknown interpreted text role "meth".

In this example, `Base.y`, `Base.w`, and `D.t` are keyword-only fields, and `Base.x` and `D.z` are regular fields:

```
@dataclass
class Base:
    x: Any = 15.0
    _: KW_ONLY
    y: int = 0
    w: int = 1

@dataclass
class D(Base):
    z: int = 10
    t: int = field(kw_only=True, default=0)
```

The generated :meth:`__init__` method for `D` will look like:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 653); backlink

Unknown interpreted text role "meth".

```
def __init__(self, x: Any = 15.0, z: int = 10, *, y: int = 0, w: int = 1, t: int = 0):
```

Note that the parameters have been re-ordered from how they appear in the list of fields: parameters derived from regular fields are followed by parameters derived from keyword-only fields.

The relative ordering of keyword-only parameters is maintained in the re-ordered :meth:`__init__` parameter list.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 661); backlink

Unknown interpreted text role "meth".

Default factory functions

If a :func:`field` specifies a `default_factory`, it is called with zero arguments when a default value for the field is needed. For example, to create a new instance of a list, use:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 668); backlink

Unknown interpreted text role "func".

```
mylist: list = field(default_factory=list)
```

If a field is excluded from :meth:`__init__` (using `init=False`) and the field also specifies `default_factory`, then the default

factory function will always be called from the generated :meth:`__init__` function. This happens because there is no other way to give the field an initial value.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 674); *backlink*

Unknown interpreted text role "meth".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 674); *backlink*

Unknown interpreted text role "meth".

Mutable default values

Python stores default member variable values in class attributes. Consider this example, not using dataclasses:

```
class C:
    x = []
    def add(self, element):
        self.x.append(element)

o1 = C()
o2 = C()
o1.add(1)
o2.add(2)
assert o1.x == [1, 2]
assert o1.x is o2.x
```

Note that the two instances of class `C` share the same class variable `x`, as expected.

Using dataclasses, *if* this code was valid:

```
@dataclass
class D:
    x: List = []
    def add(self, element):
        self.x += element
```

it would generate code similar to:

```
class D:
    x = []
    def __init__(self, x=x):
        self.x = x
    def add(self, element):
        self.x += element

assert D().x is D().x
```

This has the same issue as the original example using class `C`. That is, two instances of class `D` that do not specify a value for `x` when creating a class instance will share the same copy of `x`. Because dataclasses just use normal Python class creation they also share this behavior. There is no general way for Data Classes to detect this condition. Instead, the :func:`dataclass` decorator will raise a :exc:`TypeError` if it detects an unhashable default parameter. The assumption is that if a value is unhashable, it is mutable. This is a partial solution, but it does protect against many common errors.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 720); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 720); *backlink*

Unknown interpreted text role "exc".

Using default factory functions is a way to create new instances of mutable types as default values for fields:

```
@dataclass
class D:
    x: list = field(default_factory=list)

assert D().x is not D().x
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]dataclasses.rst, line 740)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.11
   Instead of looking for and disallowing objects of type ``list``,
   ``dict``, or ``set``, unhashable objects are now not allowed as
   default values.  Unhashability is used to approximate
   mutability.
```