:mod:`typing` --- Support for type hints

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2); *backlink*

Unknown interpreted text role "mod".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 5)

Unknown directive type "module".

```
.. module:: typing
   :synopsis: Support for type hints (see :pep:`484`).
```

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

Unknown directive type "versionadded".

```
.. versionadded:: 3.5
```

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

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

Unknown interpreted text role "source".

Note

The Python runtime does not enforce function and variable type annotations. They can be used by third party tools such as type checkers, IDEs, linters, etc.

This module provides runtime support for type hints. The most fundamental support consists of the types :data:`Any`, :data:`Union`, :data:`Callable`, :class:`TypeVar`, and :class:`Generic`. For a full specification, please see PEP 484. For a simplified introduction to type hints, see PEP 483.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 20); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 20); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 20); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 20); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 20); *backlink*

Unknown interpreted text role "class".

The function below takes and returns a string and is annotated as follows:

```
def greeting(name: str) -> str:
    return 'Hello ' + name
```

In the function `greeting`, the argument `name` is expected to be of type :class:`str` and the return type :class:`str`. Subtypes are accepted as arguments.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 31); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 31); *backlink*

Unknown interpreted text role "class".

New features are frequently added to the `typing` module. The typing_extensions package provides backports of these new features to older versions of Python.

Relevant PEPs

Since the initial introduction of type hints in PEP 484 and PEP 483, a number of PEPs have modified and enhanced Python's framework for type annotations. These include:

- PEP 526: Syntax for Variable Annotations

  Introducing syntax for annotating variables outside of function definitions, and :data:`ClassVar`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 49); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 544: Protocols: Structural subtyping (static duck typing)

  Introducing :class:`Protocol` and the :func:`@runtime_checkable<runtime_checkable>` decorator

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 52); *backlink***
  >
  > Unknown interpreted text role "class".

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 52); *backlink***
  >
  > Unknown interpreted text role "func".

- PEP 585: Type Hinting Generics In Standard Collections

  Introducing :class:`types.GenericAlias` and the ability to use standard library classes as :ref:`generic types<types-genericalias>`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 55); *backlink***
  >
  > Unknown interpreted text role "class".

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 55); *backlink***
  >
  > Unknown interpreted text role "ref".

- PEP 586: Literal Types

  Introducing :data:`Literal`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 58); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 589: TypedDict: Type Hints for Dictionaries with a Fixed Set of Keys

  Introducing :class:`TypedDict`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 60); *backlink***
  >
  > Unknown interpreted text role "class".

- PEP 591: Adding a final qualifier to typing

  Introducing :data:`Final` and the :func:`@final<final>` decorator

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 62); *backlink***
  >
  > Unknown interpreted text role "data".

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 62); *backlink***
  >
  > Unknown interpreted text role "func".

- PEP 593: Flexible function and variable annotations

Introducing :data:`Annotated`

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 64); *backlink***
>
> Unknown interpreted text role "data".

- PEP 604: Allow writing union types as `X | Y`

  Introducing :data:`types.UnionType` and the ability to use the binary-or operator `|` to signify a :ref:`union of types<types-union>`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 66); *backlink***
  >
  > Unknown interpreted text role "data".

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 66); *backlink***
  >
  > Unknown interpreted text role "ref".

- PEP 612: Parameter Specification Variables

  Introducing :class:`ParamSpec` and :data:`Concatenate`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 70); *backlink***
  >
  > Unknown interpreted text role "class".

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 70); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 613: Explicit Type Aliases

  Introducing :data:`TypeAlias`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 72); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 646: Variadic Generics

  Introducing :data:`TypeVarTuple`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 74); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 647: User-Defined Type Guards

  Introducing :data:`TypeGuard`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 76); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 673: Self type

  Introducing :data:`Self`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 78); *backlink***
  >
  > Unknown interpreted text role "data".

- PEP 675: Arbitrary Literal String Type

  Introducing :data:`LiteralString`

  > **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 80); *backlink***

Unknown interpreted text role "data".

Type aliases

A type alias is defined by assigning the type to the alias. In this example, `Vector` and `list[float]` will be treated as interchangeable synonyms:

```
Vector = list[float]

def scale(scalar: float, vector: Vector) -> Vector:
    return [scalar * num for num in vector]

# typechecks; a list of floats qualifies as a Vector.
new_vector = scale(2.0, [1.0, -4.2, 5.4])
```

Type aliases are useful for simplifying complex type signatures. For example:

```
from collections.abc import Sequence

ConnectionOptions = dict[str, str]
Address = tuple[str, int]
Server = tuple[Address, ConnectionOptions]

def broadcast_message(message: str, servers: Sequence[Server]) -> None:
    ...

# The static type checker will treat the previous type signature as
# being exactly equivalent to this one.
def broadcast_message(
        message: str,
        servers: Sequence[tuple[tuple[str, int], dict[str, str]]]) -> None:
    ...
```

Note that `None` as a type hint is a special case and is replaced by `type(None)`.

NewType

Use the :class:`NewType` helper class to create distinct types:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 124); *backlink*
>
> Unknown interpreted text role "class".

```
from typing import NewType

UserId = NewType('UserId', int)
some_id = UserId(524313)
```

The static type checker will treat the new type as if it were a subclass of the original type. This is useful in helping catch logical errors:

```
def get_user_name(user_id: UserId) -> str:
    ...

# typechecks
user_a = get_user_name(UserId(42351))

# does not typecheck; an int is not a UserId
user_b = get_user_name(-1)
```

You may still perform all `int` operations on a variable of type `UserId`, but the result will always be of type `int`. This lets you pass in a `UserId` wherever an `int` might be expected, but will prevent you from accidentally creating a `UserId` in an invalid way:

```
# 'output' is of type 'int', not 'UserId'
output = UserId(23413) + UserId(54341)
```

Note that these checks are enforced only by the static type checker. At runtime, the statement `Derived = NewType('Derived', Base)` will make `Derived` a class that immediately returns whatever parameter you pass it. That means the expression `Derived(some_value)` does not create a new class or introduce much overhead beyond that of a regular function call.

More precisely, the expression `some_value is Derived(some_value)` is always true at runtime.

It is invalid to create a subtype of `Derived`:

```
from typing import NewType

UserId = NewType('UserId', int)

# Fails at runtime and does not typecheck
class AdminUserId(UserId): pass
```

However, it is possible to create a :class:`NewType` based on a 'derived' `NewType`:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 169); *backlink*
>
> Unknown interpreted text role "class".

```
from typing import NewType

UserId = NewType('UserId', int)

ProUserId = NewType('ProUserId', UserId)
```

and typechecking for `ProUserId` will work as expected.

See PEP 484 for more details.

Note

Recall that the use of a type alias declares two types to be *equivalent* to one another. Doing `Alias = Original` will make the static type checker treat `Alias` as being *exactly equivalent* to `Original` in all cases. This is useful when you want to simplify complex type signatures.

In contrast, `NewType` declares one type to be a *subtype* of another. Doing `Derived = NewType('Derived', Original)` will make the static type checker treat `Derived` as a *subclass* of `Original`, which means a value of type `Original` cannot be used in places where a value of type `Derived` is expected. This is useful when you want to prevent logic errors with minimal runtime cost.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 195)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 197)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   ``NewType`` is now a class rather than a function.  There is some additional
   runtime cost when calling ``NewType`` over a regular function.  However, this
   cost will be reduced in 3.11.0.
```

Callable

Frameworks expecting callback functions of specific signatures might be type hinted using `Callable[[Arg1Type, Arg2Type], ReturnType]`.

For example:

```
from collections.abc import Callable

def feeder(get_next_item: Callable[[], str]) -> None:
    # Body

def async_query(on_success: Callable[[int], None],
                on_error: Callable[[int, Exception], None]) -> None:
    # Body
```

It is possible to declare the return type of a callable without specifying the call signature by substituting a literal ellipsis for the list of arguments in the type hint: `Callable[..., ReturnType]`.

Callables which take other callables as arguments may indicate that their parameter types are dependent on each other using :class:`ParamSpec`. Additionally, if that callable adds or removes arguments from other callables, the :data:`Concatenate` operator may be used. They take the form `Callable[ParamSpecVariable, ReturnType]` and `Callable[Concatenate[Arg1Type, Arg2Type, ..., ParamSpecVariable], ReturnType]` respectively.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 224); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 224); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 232)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   ``Callable`` now supports :class:`ParamSpec` and :data:`Concatenate`.
   See :pep:`612` for more information.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 236)

Unknown directive type "seealso".

```
.. seealso::
   The documentation for :class:`ParamSpec` and :class:`Concatenate` provide
   examples of usage in ``Callable``.
```

Generics

Since type information about objects kept in containers cannot be statically inferred in a generic way, abstract base classes have been extended to support subscription to denote expected types for container elements.

```
from collections.abc import Mapping, Sequence
```

```
def notify_by_email(employees: Sequence[Employee],
                    overrides: Mapping[str, str]) -> None: ...
```

Generics can be parameterized by using a factory available in typing called :class:`TypeVar`.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 256); *backlink*
>
> Unknown interpreted text role "class".

```
from collections.abc import Sequence
from typing import TypeVar

T = TypeVar('T')      # Declare type variable

def first(l: Sequence[T]) -> T:   # Generic function
    return l[0]
```

User-defined generic types

A user-defined class can be defined as a generic class.

```
from typing import TypeVar, Generic
from logging import Logger

T = TypeVar('T')

class LoggedVar(Generic[T]):
    def __init__(self, value: T, name: str, logger: Logger) -> None:
        self.name = name
        self.logger = logger
        self.value = value

    def set(self, new: T) -> None:
        self.log('Set ' + repr(self.value))
        self.value = new

    def get(self) -> T:
        self.log('Get ' + repr(self.value))
        return self.value

    def log(self, message: str) -> None:
        self.logger.info('%s: %s', self.name, message)
```

`Generic[T]` as a base class defines that the class `LoggedVar` takes a single type parameter `T` . This also makes `T` valid as a type within the class body.

The :class:`Generic` base class defines :meth:`~object.__class_getitem__` so that `LoggedVar[t]` is valid as a type:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 304); *backlink*
>
> Unknown interpreted text role "class".

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 304); *backlink*
>
> Unknown interpreted text role "meth".

```
from collections.abc import Iterable

def zero_all_vars(vars: Iterable[LoggedVar[int]]) -> None:
    for var in vars:
        var.set(0)
```

A generic type can have any number of type variables. All varieties of :class:`TypeVar` are permissible as parameters for a generic type:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 313); *backlink*
>
> Unknown interpreted text role "class".

```
from typing import TypeVar, Generic, Sequence

T = TypeVar('T', contravariant=True)
B = TypeVar('B', bound=Sequence[bytes], covariant=True)
S = TypeVar('S', int, str)

class WeirdTrio(Generic[T, B, S]):
    ...
```

Each type variable argument to :class:`Generic` must be distinct. This is thus invalid:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 325); *backlink*
>
> Unknown interpreted text role "class".

```
from typing import TypeVar, Generic
...

T = TypeVar('T')

class Pair(Generic[T, T]):   # INVALID
    ...
```

You can use multiple inheritance with :class:`Generic`:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 336); *backlink*
>
> Unknown interpreted text role "class".

```
from collections.abc import Sized
from typing import TypeVar, Generic

T = TypeVar('T')

class LinkedList(Sized, Generic[T]):
    ...
```

When inheriting from generic classes, some type variables could be fixed:

```
from collections.abc import Mapping
from typing import TypeVar

T = TypeVar('T')

class MyDict(Mapping[str, T]):
    ...
```

In this case `MyDict` has a single parameter, `T`.

Using a generic class without specifying type parameters assumes :data:`Any` for each position. In the following example, `MyIterable` is not generic but implicitly inherits from `Iterable[Any]`:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 358); *backlink*
>
> Unknown interpreted text role "data".

```
from collections.abc import Iterable

class MyIterable(Iterable): # Same as Iterable[Any]
```

User defined generic type aliases are also supported. Examples:

```
from collections.abc import Iterable
from typing import TypeVar
S = TypeVar('S')
Response = Iterable[S] | int

# Return type here is same as Iterable[str] | int
def response(query: str) -> Response[str]:
    ...

T = TypeVar('T', int, float, complex)
Vec = Iterable[tuple[T, T]]

def inproduct(v: Vec[T]) -> T: # Same as Iterable[tuple[T, T]]
    return sum(x*y for x, y in v)
```

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 383)
>
> Unknown directive type "versionchanged".
>
> ```
> .. versionchanged:: 3.7
>    :class:`Generic` no longer has a custom metaclass.
> ```

User-defined generics for parameter expressions are also supported via parameter specification variables in the form `Generic[P]`. The behavior is consistent with type variables' described above as parameter specification variables are treated by the typing module as a specialized type variable. The one exception to this is that a list of types can be used to substitute a :class:`ParamSpec`:

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 386); *backlink*
>
> Unknown interpreted text role "class".

```
>>> from typing import Generic, ParamSpec, TypeVar

>>> T = TypeVar('T')
>>> P = ParamSpec('P')

>>> class Z(Generic[T, P]): ...
...
>>> Z[int, [dict, float]]
__main__.Z[int, (<class 'dict'>, <class 'float'>)]
```

Furthermore, a generic with only one parameter specification variable will accept parameter lists in the forms `X[[Type1, Type2, ...]]` and also `X[Type1, Type2, ...]` for aesthetic reasons. Internally, the latter is converted to the former and are thus equivalent:

```
>>> class X(Generic[P]): ...
...
>>> X[int, str]
__main__.X[(<class 'int'>, <class 'str'>)]
>>> X[[int, str]]
__main__.X[(<class 'int'>, <class 'str'>)]
```

Do note that generics with :class:`ParamSpec` may not have correct `__parameters__` after substitution in some cases because they are intended primarily for static type checking.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 415); *backlink*

Unknown interpreted text role "class".

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

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   :class:`Generic` can now be parameterized over parameter expressions.
   See :class:`ParamSpec` and :pep:`612` for more details.
```

A user-defined generic class can have ABCs as base classes without a metaclass conflict. Generic metaclasses are not supported. The outcome of parameterizing generics is cached, and most types in the typing module are hashable and comparable for equality.

The :data:`Any` type

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 429); *backlink*

Unknown interpreted text role "data".

A special kind of type is :data:`Any`. A static type checker will treat every type as being compatible with :data:`Any` and :data:`Any` as being compatible with every type.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 432); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 432); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 432); *backlink*

Unknown interpreted text role "data".

This means that it is possible to perform any operation or method call on a value of type :data:`Any` and assign it to any variable:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 436); *backlink*

Unknown interpreted text role "data".

```
from typing import Any

a: Any = None
a = []          # OK
a = 2           # OK

s: str = ''
s = a           # OK

def foo(item: Any) -> int:
    # Typechecks; 'item' could be any type,
    # and that type might have a 'bar' method
    item.bar()
    ...
```

Notice that no typechecking is performed when assigning a value of type :data:`Any` to a more precise type. For example, the static type checker did not report an error when assigning `a` to `s` even though `s` was declared to be of type :class:`str` and receives an :class:`int` value at runtime!

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 454); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 454); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 454); *backlink*

Unknown interpreted text role "class".

Furthermore, all functions without a return type or parameter types will implicitly default to using :data:`Any`:

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

main\Doc\library\[cpython-main][Doc][library]typing.rst, line 460); *backlink*

Unknown interpreted text role "data".

```
def legacy_parser(text):
    ...
    return data

# A static type checker will treat the above
# as having the same signature as:
def legacy_parser(text: Any) -> Any:
    ...
    return data
```

This behavior allows :data:`Any` to be used as an *escape hatch* when you need to mix dynamically and statically typed code.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 473); *backlink*

Unknown interpreted text role "data".

Contrast the behavior of :data:`Any` with the behavior of :class:`object`. Similar to :data:`Any`, every type is a subtype of :class:`object`. However, unlike :data:`Any`, the reverse is not true: :class:`object` is *not* a subtype of every other type.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 476); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 476); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 476); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 476); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 476); *backlink*

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 476); *backlink*

Unknown interpreted text role "class".

That means when the type of a value is :class:`object`, a type checker will reject almost all operations on it, and assigning it to a variable (or using it as a return value) of a more specialized type is a type error. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 481); *backlink*

Unknown interpreted text role "class".

```
def hash_a(item: object) -> int:
    # Fails; an object does not have a 'magic' method.
    item.magic()
    ...

def hash_b(item: Any) -> int:
    # Typechecks
    item.magic()
    ...

# Typechecks, since ints and strs are subclasses of object
hash_a(42)
hash_a("foo")

# Typechecks, since Any is compatible with all types
hash_b(42)
hash_b("foo")
```

Use :class:`object` to indicate that a value could be any type in a typesafe manner. Use :data:`Any` to indicate that a value is dynamically typed.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 503); *backlink*

Unknown interpreted text role "class".

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]typing.rst, line 503); *backlink*

Unknown interpreted text role "data".

Nominal vs structural subtyping

Initially PEP 484 defined Python static type system as using *nominal subtyping*. This means that a class `A` is allowed where a class `B` is expected if and only if `A` is a subclass of `B`.

This requirement previously also applied to abstract base classes, such as :class:`~collections.abc.Iterable`. The problem with this approach is that a class had to be explicitly marked to support them, which is unpythonic and unlike what one would normally do in idiomatic dynamically typed Python code. For example, this conforms to PEP 484:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 514); *backlink*

Unknown interpreted text role "class".

```
from collections.abc import Sized, Iterable, Iterator

class Bucket(Sized, Iterable[int]):
    ...
    def __len__(self) -> int: ...
    def __iter__(self) -> Iterator[int]: ...
```

PEP 544 allows to solve this problem by allowing users to write the above code without explicit base classes in the class definition, allowing `Bucket` to be implicitly considered a subtype of both `Sized` and `Iterable[int]` by static type checkers. This is known as *structural subtyping* (or static duck-typing):

```
from collections.abc import Iterator, Iterable

class Bucket:  # Note: no base classes
    ...
    def __len__(self) -> int: ...
    def __iter__(self) -> Iterator[int]: ...

def collect(items: Iterable[int]) -> int: ...
result = collect(Bucket())  # Passes type check
```

Moreover, by subclassing a special class :class:`Protocol`, a user can define new custom protocols to fully enjoy structural subtyping (see examples below).

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 543); *backlink*

Unknown interpreted text role "class".

Module contents

The module defines the following classes, functions and decorators.

Note

This module defines several types that are subclasses of pre-existing standard library classes which also extend :class:`Generic` to support type variables inside `[]`. These types became redundant in Python 3.9 when the corresponding pre-existing classes were enhanced to support `[]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 554); *backlink*

Unknown interpreted text role "class".

The redundant types are deprecated as of Python 3.9 but no deprecation warnings will be issued by the interpreter. It is expected that type checkers will flag the deprecated types when the checked program targets Python 3.9 or newer.

The deprecated types will be removed from the :mod:`typing` module in the first Python version released 5 years after the release of Python 3.9.0. See details in :pep:`585`â€”*Type Hinting Generics In Standard Collections*.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 565); *backlink*

Unknown interpreted text role "mod".

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 565); *backlink*

Inline interpreted text or phrase reference start-string without end-string.

Special typing primitives

Special types

These can be used as types in annotations and do not support `[]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 578)

Unknown directive type "data".

```
.. data:: Any

   Special type indicating an unconstrained type.

   * Every type is compatible with :data:`Any`.
   * :data:`Any` is compatible with every type.

    .. versionchanged:: 3.11
       :data:`Any` can now be used as a base class. This can be useful for
       avoiding type checker errors with classes that can duck type anywhere or
       are highly dynamic.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 590)

Unknown directive type "data".

```
.. data:: LiteralString

   Special type that includes only literal strings. A string
   literal is compatible with ``LiteralString``, as is another
   ``LiteralString``, but an object typed as just ``str`` is not.
   A string created by composing ``LiteralString``-typed objects
   is also acceptable as a ``LiteralString``.

   Example::

      def run_query(sql: LiteralString) -> ...
          ...

      def caller(arbitrary_string: str, literal_string: LiteralString) -> None:
          run_query("SELECT * FROM students")  # ok
          run_query(literal_string)  # ok
          run_query("SELECT * FROM " + literal_string)  # ok
          run_query(arbitrary_string)  # type checker error
          run_query(  # type checker error
              f"SELECT * FROM students WHERE name = {arbitrary_string}"
          )

   This is useful for sensitive APIs where arbitrary user-generated
   strings could generate problems. For example, the two cases above
   that generate type checker errors could be vulnerable to an SQL
   injection attack.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 617)

Unknown directive type "data".

```
.. data:: Never

   The `bottom type <https://en.wikipedia.org/wiki/Bottom_type>`_,
   a type that has no members.

   This can be used to define a function that should never be
   called, or a function that never returns::

     from typing import Never

     def never_call_me(arg: Never) -> None:
         pass

     def int_or_str(arg: int | str) -> None:
         never_call_me(arg)  # type checker error
         match arg:
             case int():
                 print("It's an int")
             case str():
                 print("It's a str")
             case _:
                 never_call_me(arg)  # ok, arg is of type Never

   .. versionadded:: 3.11

      On older Python versions, :data:`NoReturn` may be used to express the
      same concept. ``Never`` was added to make the intended meaning more explicit.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 645)

Unknown directive type "data".

```
.. data:: NoReturn

   Special type indicating that a function never returns.
   For example::

      from typing import NoReturn

      def stop() -> NoReturn:
          raise RuntimeError('no way')

   ``NoReturn`` can also be used as a
```

```
   `bottom type <https://en.wikipedia.org/wiki/Bottom_type>`_, a type that
   has no values. Starting in Python 3.11, the :data:`Never` type should
   be used for this concept instead. Type checkers should treat the two
   equivalently.

   .. versionadded:: 3.5.4
   .. versionadded:: 3.6.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 664)

Unknown directive type "data".

```
.. data:: Self

   Special type to represent the current enclosed class.
   For example::

      from typing import Self

      class Foo:
         def returns_self(self) -> Self:
            ...
            return self


   This annotation is semantically equivalent to the following,
   albeit in a more succinct fashion::

      from typing import TypeVar

      Self = TypeVar("Self", bound="Foo")

      class Foo:
         def returns_self(self: Self) -> Self:
            ...
            return self

   In general if something currently follows the pattern of::

      class Foo:
         def return_self(self) -> "Foo":
            ...
            return self

   You should use use :data:`Self` as calls to ``SubclassOfFoo.returns_self`` would have
   ``Foo`` as the return type and not ``SubclassOfFoo``.

   Other common use cases include:

      - :class:`classmethod`\s that are used as alternative constructors and return instances
        of the ``cls`` parameter.
      - Annotating an :meth:`object.__enter__` method which returns self.

   For more information, see :pep:`673`.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 709)

Unknown directive type "data".

```
.. data:: TypeAlias

   Special annotation for explicitly declaring a :ref:`type alias <type-aliases>`.
   For example::

    from typing import TypeAlias

    Factors: TypeAlias = list[int]

   See :pep:`613` for more details about explicit type aliases.

   .. versionadded:: 3.10
```

Special forms

These can be used as types in annotations using `[]`, each having a unique syntax.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 727)

Unknown directive type "data".

```
.. data:: Tuple

   Tuple type; ``Tuple[X, Y]`` is the type of a tuple of two items
   with the first item of type X and the second of type Y. The type of
   the empty tuple can be written as ``Tuple[()]``.

   Example: ``Tuple[T1, T2]`` is a tuple of two elements corresponding
   to type variables T1 and T2.  ``Tuple[int, float, str]`` is a tuple
   of an int, a float and a string.

   To specify a variable-length tuple of homogeneous type,
   use literal ellipsis, e.g. ``Tuple[int, ...]``. A plain :data:`Tuple`
   is equivalent to ``Tuple[Any, ...]``, and in turn to :class:`tuple`.
```

```
   .. deprecated:: 3.9
      :class:`builtins.tuple <tuple>` now supports ``[]``. See :pep:`585` and
      :ref:`types-genericalias`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 745)

Unknown directive type "data".

```
.. data:: Union

   Union type; ``Union[X, Y]`` is equivalent to ``X | Y`` and means either X or Y.

   To define a union, use e.g. ``Union[int, str]`` or the shorthand ``int | str``. Using that shorthand is recomm

   * The arguments must be types and there must be at least one.

   * Unions of unions are flattened, e.g.::

       Union[Union[int, str], float] == Union[int, str, float]

   * Unions of a single argument vanish, e.g.::

       Union[int] == int  # The constructor actually returns int

   * Redundant arguments are skipped, e.g.::

       Union[int, str, int] == Union[int, str] == int | str

   * When comparing unions, the argument order is ignored, e.g.::

       Union[int, str] == Union[str, int]

   * You cannot subclass or instantiate a ``Union``.

   * You cannot write ``Union[X][Y]``.

   .. versionchanged:: 3.7
      Don't remove explicit subclasses from unions at runtime.

   .. versionchanged:: 3.10
      Unions can now be written as ``X | Y``. See
      :ref:`union type expressions<types-union>`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 780)

Unknown directive type "data".

```
.. data:: Optional

   Optional type.

   ``Optional[X]`` is equivalent to ``X | None`` (or ``Union[X, None]``).

   Note that this is not the same concept as an optional argument,
   which is one that has a default.  An optional argument with a
   default does not require the ``Optional`` qualifier on its type
   annotation just because it is optional. For example::

      def foo(arg: int = 0) -> None:
          ...

   On the other hand, if an explicit value of ``None`` is allowed, the
   use of ``Optional`` is appropriate, whether the argument is optional
   or not. For example::

      def foo(arg: Optional[int] = None) -> None:
          ...

   .. versionchanged:: 3.10
      Optional can now be written as ``X | None``. See
      :ref:`union type expressions<types-union>`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 805)

Unknown directive type "data".

```
.. data:: Callable

   Callable type; ``Callable[[int], str]`` is a function of (int) -> str.

   The subscription syntax must always be used with exactly two
   values: the argument list and the return type.  The argument list
   must be a list of types or an ellipsis; the return type must be
   a single type.

   There is no syntax to indicate optional or keyword arguments;
   such function types are rarely used as callback types.
   ``Callable[..., ReturnType]`` (literal ellipsis) can be used to
   type hint a callable taking any number of arguments and returning
   ``ReturnType``.  A plain :data:`Callable` is equivalent to
   ``Callable[..., Any]``, and in turn to
   :class:`collections.abc.Callable`.

   Callables which take other callables as arguments may indicate that their
```

```
   parameter types are dependent on each other using :class:`ParamSpec`.
   Additionally, if that callable adds or removes arguments from other
   callables, the :data:`Concatenate` operator may be used.  They
   take the form ``Callable[ParamSpecVariable, ReturnType]`` and
   ``Callable[Concatenate[Arg1Type, Arg2Type, ..., ParamSpecVariable], ReturnType]``
   respectively.

   .. deprecated:: 3.9
      :class:`collections.abc.Callable` now supports ``[]``. See :pep:`585` and
      :ref:`types-genericalias`.

   .. versionchanged:: 3.10
      ``Callable`` now supports :class:`ParamSpec` and :data:`Concatenate`.
      See :pep:`612` for more information.

   .. seealso::
      The documentation for :class:`ParamSpec` and :class:`Concatenate` provide
      examples of usage with ``Callable``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 842)

Unknown directive type "data".

```
.. data:: Concatenate

   Used with :data:`Callable` and :class:`ParamSpec` to type annotate a higher
   order callable which adds, removes, or transforms parameters of another
   callable.  Usage is in the form
   ``Concatenate[Arg1Type, Arg2Type, ..., ParamSpecVariable]``. ``Concatenate``
   is currently only valid when used as the first argument to a :data:`Callable`.
   The last parameter to ``Concatenate`` must be a :class:`ParamSpec`.

   For example, to annotate a decorator ``with_lock`` which provides a
   :class:`threading.Lock` to the decorated function,  ``Concatenate`` can be
   used to indicate that ``with_lock`` expects a callable which takes in a
   ``Lock`` as the first argument, and returns a callable with a different type
   signature.  In this case, the :class:`ParamSpec` indicates that the returned
   callable's parameter types are dependent on the parameter types of the
   callable being passed in::

      from collections.abc import Callable
      from threading import Lock
      from typing import Concatenate, ParamSpec, TypeVar

      P = ParamSpec('P')
      R = TypeVar('R')

      # Use this lock to ensure that only one thread is executing a function
      # at any time.
      my_lock = Lock()

      def with_lock(f: Callable[Concatenate[Lock, P], R]) -> Callable[P, R]:
          '''A type-safe decorator which provides a lock.'''
          global my_lock
          def inner(*args: P.args, **kwargs: P.kwargs) -> R:
              # Provide the lock as the first argument.
              return f(my_lock, *args, **kwargs)
          return inner

      @with_lock
      def sum_threadsafe(lock: Lock, numbers: list[float]) -> float:
          '''Add a list of numbers together in a thread-safe manner.'''
          with lock:
              return sum(numbers)

      # We don't need to pass in the lock ourselves thanks to the decorator.
      sum_threadsafe([1.1, 2.2, 3.3])
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 887)

Unknown directive type "versionadded".

```
.. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 889)

Unknown directive type "seealso".

```
.. seealso::

   * :pep:`612` -- Parameter Specification Variables (the PEP which introduced
     ``ParamSpec`` and ``Concatenate``).
   * :class:`ParamSpec` and :class:`Callable`.
```

A variable annotated with `C` may accept a value of type `C`. In contrast, a variable annotated with `Type[C]` may accept values that are classes themselves -- specifically, it will accept the *class object* of `C`. For example:

```
a = 3         # Has type 'int'
b = int       # Has type 'Type[int]'
c = type(a)   # Also has type 'Type[int]'
```

Note that `Type[C]` is covariant:

```
class User: ...
class BasicUser(User): ...
class ProUser(User): ...
class TeamUser(User): ...

# Accepts User, BasicUser, ProUser, TeamUser, ...
def make_new_user(user_class: Type[User]) -> User:
    # ...
    return user_class()
```

The fact that `Type[C]` is covariant implies that all subclasses of `C` should implement the same constructor signature and class method signatures as `C`. The type checker should flag violations of this, but should also allow constructor calls in subclasses that match the constructor calls in the indicated base class. How the type checker is required to handle this particular case may change in future revisions of PEP 484.

The only legal parameters for :class:`Type` are classes, :data:`Any`, :ref:`type variables <generics>`, and unions of any of these types. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 927); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 927); backlink

Unknown interpreted text role "data".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 927); backlink

Unknown interpreted text role "ref".

```
def new_non_team_user(user_class: Type[BasicUser | ProUser]): ...
```

`Type[Any]` is equivalent to `Type` which in turn is equivalent to `type`, which is the root of Python's metaclass hierarchy.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 936)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 938)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`builtins.type <type>` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 942)

Unknown directive type "data".

```
.. data:: Literal

   A type that can be used to indicate to type checkers that the
   corresponding variable or function parameter has a value equivalent to
   the provided literal (or one of several literals). For example::

      def validate_simple(data: Any) -> Literal[True]:  # always returns True
          ...

      MODE = Literal['r', 'rb', 'w', 'wb']
      def open_helper(file: str, mode: MODE) -> str:
          ...

      open_helper('/some/path', 'r')  # Passes type check
      open_helper('/other/path', 'typo')  # Error in type checker

   ``Literal[...]`` cannot be subclassed. At runtime, an arbitrary value
   is allowed as type argument to ``Literal[...]``, but type checkers may
   impose restrictions. See :pep:`586` for more details about literal types.

   .. versionadded:: 3.8

   .. versionchanged:: 3.9.1
      ``Literal`` now de-duplicates parameters.  Equality comparisons of
      ``Literal`` objects are no longer order dependent. ``Literal`` objects
      will now raise a :exc:`TypeError` exception during equality comparisons
      if one of their parameters are not :term:`hashable`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 970)

Unknown directive type "data".

```
.. data:: ClassVar
```

```
   Special type construct to mark class variables.

   As introduced in :pep:`526`, a variable annotation wrapped in ClassVar
   indicates that a given attribute is intended to be used as a class variable
   and should not be set on instances of that class. Usage::

      class Starship:
          stats: ClassVar[dict[str, int]] = {} # class variable
          damage: int = 10                     # instance variable

   :data:`ClassVar` accepts only types and cannot be further subscribed.

   :data:`ClassVar` is not a class itself, and should not
   be used with :func:`isinstance` or :func:`issubclass`.
   :data:`ClassVar` does not change Python runtime behavior, but
   it can be used by third-party type checkers. For example, a type checker
   might flag the following code as an error::

      enterprise_d = Starship(3000)
      enterprise_d.stats = {} # Error, setting class variable on instance
      Starship.stats = {}     # This is OK

   .. versionadded:: 3.5.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 996)

Unknown directive type "data".

```
.. data:: Final

   A special typing construct to indicate to type checkers that a name
   cannot be re-assigned or overridden in a subclass. For example::

      MAX_SIZE: Final = 9000
      MAX_SIZE += 1  # Error reported by type checker

      class Connection:
          TIMEOUT: Final[int] = 10

      class FastConnector(Connection):
          TIMEOUT = 1  # Error reported by type checker

   There is no runtime checking of these properties. See :pep:`591` for
   more details.

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1015)

Unknown directive type "data".

```
.. data:: Annotated

   A type, introduced in :pep:`593` (``Flexible function and variable
   annotations``), to decorate existing types with context-specific metadata
   (possibly multiple pieces of it, as ``Annotated`` is variadic).
   Specifically, a type ``T`` can be annotated with metadata ``x`` via the
   typehint ``Annotated[T, x]``. This metadata can be used for either static
   analysis or at runtime. If a library (or tool) encounters a typehint
   ``Annotated[T, x]`` and has no special logic for metadata ``x``, it
   should ignore it and simply treat the type as ``T``. Unlike the
   ``no_type_check`` functionality that currently exists in the ``typing``
   module which completely disables typechecking annotations on a function
   or a class, the ``Annotated`` type allows for both static typechecking
   of ``T`` (e.g., via mypy or Pyre, which can safely ignore ``x``)
   together with runtime access to ``x`` within a specific application.

   Ultimately, the responsibility of how to interpret the annotations (if
   at all) is the responsibility of the tool or library encountering the
   ``Annotated`` type. A tool or library encountering an ``Annotated`` type
   can scan through the annotations to determine if they are of interest
   (e.g., using ``isinstance()``).

   When a tool or a library does not support annotations or encounters an
   unknown annotation it should just ignore it and treat annotated type as
   the underlying type.

   It's up to the tool consuming the annotations to decide whether the
   client is allowed to have several annotations on one type and how to
   merge those annotations.

   Since the ``Annotated`` type allows you to put several annotations of
   the same (or different) type(s) on any node, the tools or libraries
   consuming those annotations are in charge of dealing with potential
   duplicates. For example, if you are doing value range analysis you might
   allow this::

       T1 = Annotated[int, ValueRange(-10, 5)]
       T2 = Annotated[T1, ValueRange(-20, 3)]

   Passing ``include_extras=True`` to :func:`get_type_hints` lets one
   access the extra annotations at runtime.

   The details of the syntax:

   * The first argument to ``Annotated`` must be a valid type

   * Multiple type annotations are supported (``Annotated`` supports variadic
```

```
  arguments)::

    Annotated[int, ValueRange(3, 10), ctype("char")]

* ``Annotated`` must be called with at least two arguments (
  ``Annotated[int]`` is not valid)

* The order of the annotations is preserved and matters for equality
  checks::

    Annotated[int, ValueRange(3, 10), ctype("char")] != Annotated[
        int, ctype("char"), ValueRange(3, 10)
    ]

* Nested ``Annotated`` types are flattened, with metadata ordered
  starting with the innermost annotation::

    Annotated[Annotated[int, ValueRange(3, 10)], ctype("char")] == Annotated[
        int, ValueRange(3, 10), ctype("char")
    ]

* Duplicated annotations are not removed::

    Annotated[int, ValueRange(3, 10)] != Annotated[
        int, ValueRange(3, 10), ValueRange(3, 10)
    ]

* ``Annotated`` can be used with nested and generic aliases::

    T = TypeVar('T')
    Vec = Annotated[list[tuple[T, T]], MaxLen(10)]
    V = Vec[int]

    V == Annotated[list[tuple[int, int]], MaxLen(10)]

.. versionadded:: 3.9
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1100)

Unknown directive type "data".

```
.. data:: TypeGuard

   Special typing form used to annotate the return type of a user-defined
   type guard function.  ``TypeGuard`` only accepts a single type argument.
   At runtime, functions marked this way should return a boolean.

   ``TypeGuard`` aims to benefit *type narrowing* -- a technique used by static
   type checkers to determine a more precise type of an expression within a
   program's code flow.  Usually type narrowing is done by analyzing
   conditional code flow and applying the narrowing to a block of code.  The
   conditional expression here is sometimes referred to as a "type guard"::

      def is_str(val: str | float):
          # "isinstance" type guard
          if isinstance(val, str):
              # Type of ``val`` is narrowed to ``str``
              ...
          else:
              # Else, type of ``val`` is narrowed to ``float``.
              ...

   Sometimes it would be convenient to use a user-defined boolean function
   as a type guard.  Such a function should use ``TypeGuard[...]`` as its
   return type to alert static type checkers to this intention.

   Using  ``-> TypeGuard`` tells the static type checker that for a given
   function:

   1. The return value is a boolean.
   2. If the return value is ``True``, the type of its argument
      is the type inside ``TypeGuard``.

      For example::

         def is_str_list(val: list[object]) -> TypeGuard[list[str]]:
             '''Determines whether all objects in the list are strings'''
             return all(isinstance(x, str) for x in val)

         def func1(val: list[object]):
             if is_str_list(val):
                 # Type of ``val`` is narrowed to ``list[str]``.
                 print(" ".join(val))
             else:
                 # Type of ``val`` remains as ``list[object]``.
                 print("Not a list of strings!")

   If ``is_str_list`` is a class or instance method, then the type in
   ``TypeGuard`` maps to the type of the second parameter after ``cls`` or
   ``self``.

   In short, the form ``def foo(arg: TypeA) -> TypeGuard[TypeB]: ...``,
   means that if ``foo(arg)`` returns ``True``, then ``arg`` narrows from
   ``TypeA`` to ``TypeB``.

   .. note::

      ``TypeB`` need not be a narrower form of ``TypeA`` -- it can even be a
      wider form. The main reason is to allow for things like
      narrowing ``list[object]`` to ``list[str]`` even though the latter
```

```
          is not a subtype of the former, since ``list`` is invariant.
          The responsibility of writing type-safe type guards is left to the user.

       ``TypeGuard`` also works with type variables.  For more information, see
       :pep:`647` (User-Defined Type Guards).

       .. versionadded:: 3.10
```

Building generic types

These are not used in annotations. They are building blocks for creating generic types.

Abstract base class for generic types.

A generic type is typically declared by inheriting from an instantiation of this class with one or more type variables. For example, a generic mapping type might be defined as:

```
class Mapping(Generic[KT, VT]):
    def __getitem__(self, key: KT) -> VT:
        ...
        # Etc.
```

This class can then be used as follows:

```
X = TypeVar('X')
Y = TypeVar('Y')

def lookup_name(mapping: Mapping[X, Y], key: X, default: Y) -> Y:
    try:
        return mapping[key]
    except KeyError:
        return default
```

Type variable.

Usage:

```
T = TypeVar('T')  # Can be anything
S = TypeVar('S', bound=str)  # Can be any subtype of str
A = TypeVar('A', str, bytes)  # Must be exactly str or bytes
```

Type variables exist primarily for the benefit of static type checkers. They serve as the parameters for generic types as well as for generic function definitions. See :class:`Generic` for more information on generic types. Generic functions work as follows:

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1207); *backlink***
>
> Unknown interpreted text role "class".

```
def repeat(x: T, n: int) -> Sequence[T]:
    """Return a list containing n references to x."""
    return [x]*n


def print_capitalized(x: S) -> S:
    """Print x capitalized, and return x."""
    print(x.capitalize())
    return x


def concatenate(x: A, y: A) -> A:
    """Add two strings or bytes objects together."""
    return x + y
```

Note that type variables can be *bound*, *constrained*, or neither, but cannot be both bound *and* constrained.

Bound type variables and constrained type variables have different semantics in several important ways. Using a *bound* type variable means that the `TypeVar` will be solved using the most specific type possible:

```
x = print_capitalized('a string')
reveal_type(x)  # revealed type is str

class StringSubclass(str):
    pass

y = print_capitalized(StringSubclass('another string'))
reveal_type(y)  # revealed type is StringSubclass

z = print_capitalized(45)  # error: int is not a subtype of str
```

Type variables can be bound to concrete types, abstract types (ABCs or protocols), and even unions of types:

```
U = TypeVar('U', bound=str|bytes)  # Can be any subtype of the union str|bytes
V = TypeVar('V', bound=SupportsAbs)  # Can be anything with an __abs__ method
```

Using a *constrained* type variable, however, means that the `TypeVar` can only ever be solved as being exactly one of the constraints given:

```
a = concatenate('one', 'two')
reveal_type(a)  # revealed type is str

b = concatenate(StringSubclass('one'), StringSubclass('two'))
reveal_type(b)  # revealed type is str, despite StringSubclass being passed in

c = concatenate('one', b'two')  # error: type variable 'A' can be either str or bytes in a function call, but not both
```

At runtime, `isinstance(x, T)` will raise :exc:`TypeError`. In general, :func:`isinstance` and :func:`issubclass` should not be used with types.

> **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]typing.rst, line 1262); *backlink*

Unknown interpreted text role "exc".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1262); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1262); *backlink*

Unknown interpreted text role "func".

Type variables may be marked covariant or contravariant by passing `covariant=True` or `contravariant=True`. See PEP 484 for more details. By default, type variables are invariant.

Type variable tuple. A specialized form of :class:`Type variable <TypeVar>` that enables *variadic* generics.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1271); *backlink*

Unknown interpreted text role "class".

A normal type variable enables parameterization with a single type. A type variable tuple, in contrast, allows parameterization with an *arbitrary* number of types by acting like an *arbitrary* number of type variables wrapped in a tuple. For example:

```
T = TypeVar('T')
Ts = TypeVarTuple('Ts')

def remove_first_element(tup: tuple[T, *Ts]) -> tuple[*Ts]:
    return tup[1:]

# T is bound to int, Ts is bound to ()
# Return value is (), which has type tuple[()]
remove_first_element(tup=(1,))

# T is bound to int, Ts is bound to (str,)
# Return value is ('spam',), which has type tuple[str]
remove_first_element(tup=(1, 'spam'))

# T is bound to int, Ts is bound to (str, float)
# Return value is ('spam', 3.0), which has type tuple[str, float]
remove_first_element(tup=(1, 'spam', 3.0))
```

Note the use of the unpacking operator `*` in `tuple[T, *Ts]`. Conceptually, you can think of `Ts` as a tuple of type variables `(T1, T2, ...)`. `tuple[T, *Ts]` would then become `tuple[T, *(T1, T2, ...)]`, which is equivalent to `tuple[T, T1, T2, ...]`. (Note that in older versions of Python, you might see this written using :data:`Unpack <Unpack>` instead, as `Unpack[Ts]`.)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1297); *backlink*

Unknown interpreted text role "data".

Type variable tuples must *always* be unpacked. This helps distinguish type variable types from normal type variables:

```
x: Ts          # Not valid
x: tuple[Ts]   # Not valid
x: tuple[*Ts]  # The correct way to to do it
```

Type variable tuples can be used in the same contexts as normal type variables. For example, in class definitions, arguments, and return types:

```
Shape = TypeVarTuple('Shape')
class Array(Generic[*Shape]):
    def __getitem__(self, key: tuple[*Shape]) -> float: ...
    def __abs__(self) -> Array[*Shape]: ...
    def get_shape(self) -> tuple[*Shape]: ...
```

Type variable tuples can be happily combined with normal type variables:

```
DType = TypeVar('DType')

class Array(Generic[DType, *Shape]):  # This is fine
    pass

class Array2(Generic[*Shape, DType]):  # This would also be fine
    pass

float_array_1d: Array[float, Height] = Array()     # Totally fine
int_array_2d: Array[int, Height, Width] = Array()  # Yup, fine too
```

However, note that at most one type variable tuple may appear in a single list of type arguments or type parameters:

```
x: tuple[*Ts, *Ts]                     # Not valid
class Array(Generic[*Shape, *Shape]):  # Not valid
    pass
```

Finally, an unpacked type variable tuple can be used as the type annotation of `*args`:

```
def call_soon(
        callback: Callable[[*Ts], None],
        *args: *Ts
) -> None:
    ...
    callback(*args)
```

In contrast to non-unpacked annotations of `*args` - e.g. `*args: int`, which would specify that *all* arguments are `int` - `*args: *Ts` enables reference to the types of the *individual* arguments in `*args`. Here, this allows us to ensure the types of the `*args` passed to `call_soon` match the types of the (positional) arguments of `callback`.

For more details on type variable tuples, see PEP 646.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1360)

Unknown directive type "versionadded".

```
.. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1362)

Unknown directive type "data".

```
.. data:: Unpack

    A typing operator that conceptually marks an object as having been
    unpacked. For example, using the unpack operator ``*`` on a
    :class:`type variable tuple <TypeVarTuple>` is equivalent to using ``Unpack``
    to mark the type variable tuple as having been unpacked::

        Ts = TypeVarTuple('Ts')
        tup: tuple[*Ts]
        # Effectively does:
        tup: tuple[Unpack[Ts]]

    In fact, ``Unpack`` can be used interchangeably with ``*`` in the context
    of types. You might see ``Unpack`` being used explicitly in older versions
    of Python, where ``*`` couldn't be used in certain places::

        # In older versions of Python, TypeVarTuple and Unpack
        # are located in the `typing_extensions` backports package.
        from typing_extensions import TypeVarTuple, Unpack

        Ts = TypeVarTuple('Ts')
        tup: tuple[*Ts]         # Syntax error on Python <= 3.10!
        tup: tuple[Unpack[Ts]]  # Semantically equivalent, and backwards-compatible
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1386)

Invalid class attribute value for "class" directive: "ParamSpec(name, *, bound=None, covariant=False, contravariant=False)".

```
.. class:: ParamSpec(name, *, bound=None, covariant=False, contravariant=False)

   Parameter specification variable.  A specialized version of
   :class:`type variables <TypeVar>`.

   Usage::

      P = ParamSpec('P')

   Parameter specification variables exist primarily for the benefit of static
   type checkers.  They are used to forward the parameter types of one
   callable to another callable -- a pattern commonly found in higher order
   functions and decorators.  They are only valid when used in ``Concatenate``,
   or as the first argument to ``Callable``, or as parameters for user-defined
   Generics.  See :class:`Generic` for more information on generic types.

   For example, to add basic logging to a function, one can create a decorator
   ``add_logging`` to log function calls.  The parameter specification variable
   tells the type checker that the callable passed into the decorator and the
   new callable returned by it have inter-dependent type parameters::

      from collections.abc import Callable
      from typing import TypeVar, ParamSpec
      import logging

      T = TypeVar('T')
      P = ParamSpec('P')

      def add_logging(f: Callable[P, T]) -> Callable[P, T]:
          '''A type-safe decorator to add logging to a function.'''
          def inner(*args: P.args, **kwargs: P.kwargs) -> T:
              logging.info(f'{f.__name__} was called')
              return f(*args, **kwargs)
          return inner

      @add_logging
      def add_two(x: float, y: float) -> float:
          '''Add two numbers together.'''
          return x + y

   Without ``ParamSpec``, the simplest way to annotate this previously was to
   use a :class:`TypeVar` with bound ``Callable[..., Any]``.  However this
   causes two problems:

      1. The type checker can't type check the ``inner`` function because
         ``*args`` and ``**kwargs`` have to be typed :data:`Any`.
      2. :func:`~cast` may be required in the body of the ``add_logging``
         decorator when returning the ``inner`` function, or the static type
         checker must be told to ignore the ``return inner``.

   .. attribute:: args
```

```
      .. attribute:: kwargs

         Since ``ParamSpec`` captures both positional and keyword parameters,
         ``P.args`` and ``P.kwargs`` can be used to split a ``ParamSpec`` into its
         components.  ``P.args`` represents the tuple of positional parameters in a
         given call and should only be used to annotate ``*args``.  ``P.kwargs``
         represents the mapping of keyword parameters to their values in a given call,
         and should be only be used to annotate ``**kwargs``.  Both
         attributes require the annotated parameter to be in scope. At runtime,
         ``P.args`` and ``P.kwargs`` are instances respectively of
         :class:`ParamSpecArgs` and :class:`ParamSpecKwargs`.

      Parameter specification variables created with ``covariant=True`` or
      ``contravariant=True`` can be used to declare covariant or contravariant
      generic types.  The ``bound`` argument is also accepted, similar to
      :class:`TypeVar`.  However the actual semantics of these keywords are yet to
      be decided.

      .. versionadded:: 3.10

      .. note::
         Only parameter specification variables defined in global scope can
         be pickled.

      .. seealso::
         * :pep:`612` -- Parameter Specification Variables (the PEP which introduced
           ``ParamSpec`` and ``Concatenate``).
         * :class:`Callable` and :class:`Concatenate`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1466)

Unknown directive type "data".

```
   .. data:: ParamSpecArgs
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1467)

Unknown directive type "data".

```
   .. data:: ParamSpecKwargs

      Arguments and keyword arguments attributes of a :class:`ParamSpec`. The
      ``P.args`` attribute of a ``ParamSpec`` is an instance of ``ParamSpecArgs``,
      and ``P.kwargs`` is an instance of ``ParamSpecKwargs``. They are intended
      for runtime introspection and have no special meaning to static type checkers.

      Calling :func:`get_origin` on either of these objects will return the
      original ``ParamSpec``::

         P = ParamSpec("P")
         get_origin(P.args)  # returns P
         get_origin(P.kwargs)  # returns P

      .. versionadded:: 3.10
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1484)

Unknown directive type "data".

```
   .. data:: AnyStr

      ``AnyStr`` is a :class:`constrained type variable <TypeVar>` defined as
      ``AnyStr = TypeVar('AnyStr', str, bytes)``.

      It is meant to be used for functions that may accept any kind of string
      without allowing different kinds of strings to mix. For example::

         def concat(a: AnyStr, b: AnyStr) -> AnyStr:
             return a + b

         concat(u"foo", u"bar")  # Ok, output has type 'unicode'
         concat(b"foo", b"bar")  # Ok, output has type 'bytes'
         concat(u"foo", b"bar")  # Error, cannot mix unicode and bytes
```

Base class for protocol classes. Protocol classes are defined like this:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing), for example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with :func:`runtime_checkable` (described later) act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1519); *backlink*

Unknown interpreted text role "func".

Protocol classes can be generic, for example:

```
class GenProto(Protocol[T]):
    def meth(self) -> T:
        ...
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1530)

Unknown directive type "versionadded".

```
.. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1532)

Unknown directive type "decorator".

```
.. decorator:: runtime_checkable

   Mark a protocol class as a runtime protocol.

   Such a protocol can be used with :func:`isinstance` and :func:`issubclass`.
   This raises :exc:`TypeError` when applied to a non-protocol class.  This
   allows a simple-minded structural check, very similar to "one trick ponies"
   in :mod:`collections.abc` such as :class:`~collections.abc.Iterable`.  For example:

      @runtime_checkable
      class Closable(Protocol):
          def close(self): ...

      assert isinstance(open('/some/file'), Closable)

   .. note::

        :func:`runtime_checkable` will check only the presence of the required
        methods, not their type signatures. For example, :class:`ssl.SSLObject`
        is a class, therefore it passes an :func:`issubclass`
        check against :data:`Callable`.  However, the
        :meth:`ssl.SSLObject.__init__` method exists only to raise a
        :exc:`TypeError` with a more informative message, therefore making
        it impossible to call (instantiate) :class:`ssl.SSLObject`.

   .. versionadded:: 3.8
```

Other special directives

These are not used in annotations. They are building blocks for declaring types.

Typed version of :func:`collections.namedtuple`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1566); *backlink*

Unknown interpreted text role "func".

Usage:

```
class Employee(NamedTuple):
    name: str
    id: int
```

This is equivalent to:

```
Employee = collections.namedtuple('Employee', ['name', 'id'])
```

To give a field a default value, you can assign to it in the class body:

```
class Employee(NamedTuple):
    name: str
    id: int = 3

employee = Employee('Guido')
assert employee.id == 3
```

Fields with a default value must come after any fields without a default.

The resulting class has an extra attribute `__annotations__` giving a dict that maps the field names to the field types. (The field names are in the `_fields` attribute and the default values are in the `_field_defaults` attribute both of which are part of the namedtuple API.)

`NamedTuple` subclasses can also have docstrings and methods:

```
class Employee(NamedTuple):
    """Represents an employee."""
    name: str
    id: int = 3

    def __repr__(self) -> str:
        return f'<Employee {self.name}, id={self.id}>'
```

Backward-compatible usage:

```
Employee = NamedTuple('Employee', [('name', str), ('id', int)])
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1609)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6
   Added support for :pep:`526` variable annotation syntax.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1612)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.6.1
   Added support for default values, methods, and docstrings.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1615)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.8
   The ``_field_types`` and ``__annotations__`` attributes are
   now regular dictionaries instead of instances of ``OrderedDict``.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1619)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.9
   Removed the ``_field_types`` attribute in favor of the more
   standard ``__annotations__`` attribute which has the same information.
```

A helper class to indicate a distinct type to a typechecker, see :ref`distinct`. At runtime it returns an object that returns its argument when called. Usage:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1625); *backlink*

Unknown interpreted text role "ref".

```
UserId = NewType('UserId', int)
first_user = UserId(1)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1633)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1635)

Unknown directive type "versionchanged".

```
.. versionchanged:: 3.10
   ``NewType`` is now a class rather than a function.
```

Special construct to add type hints to a dictionary. At runtime it is a plain :class:`dict`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1640); *backlink*

Unknown interpreted text role "class".

`TypedDict` declares a dictionary type that expects all of its instances to have a certain set of keys, where each key is associated with a value of a consistent type. This expectation is not checked at runtime but is only enforced by type checkers. Usage:

```
class Point2D(TypedDict):
    x: int
    y: int
    label: str

a: Point2D = {'x': 1, 'y': 2, 'label': 'good'}  # OK
b: Point2D = {'z': 3, 'label': 'bad'}           # Fails type check

assert Point2D(x=1, y=2, label='first') == dict(x=1, y=2, label='first')
```

To allow using this feature with older versions of Python that do not support PEP 526, `TypedDict` supports two additional equivalent syntactic forms:

- Using a literal :class:`dict` as the second argument:

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

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1663); backlink

Unknown interpreted text role "class".

```
Point2D = TypedDict('Point2D', {'x': int, 'y': int, 'label': str})
```

- Using keyword arguments:

```
Point2D = TypedDict('Point2D', x=int, y=int, label=str)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1671)

Unknown directive type "deprecated-removed".

```
.. deprecated-removed:: 3.11 3.13
   The keyword-argument syntax is deprecated in 3.11 and will be removed
   in 3.13. It may also be unsupported by static type checkers.
```

The functional syntax should also be used when any of the keys are not valid :ref:`identifiers`, for example because they are keywords or contain hyphens. Example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1675); backlink

Unknown interpreted text role "ref".

```
# raises SyntaxError
class Point2D(TypedDict):
    in: int  # 'in' is a keyword
    x-y: int  # name with hyphens

# OK, functional syntax
Point2D = TypedDict('Point2D', {'in': int, 'x-y': int})
```

By default, all keys must be present in a `TypedDict`. It is possible to override this by specifying totality. Usage:

```
class Point2D(TypedDict, total=False):
    x: int
    y: int

# Alternative syntax
Point2D = TypedDict('Point2D', {'x': int, 'y': int}, total=False)
```

This means that a `Point2D` `TypedDict` can have any of the keys omitted. A type checker is only expected to support a literal `False` or `True` as the value of the `total` argument. `True` is the default, and makes all items defined in the class body required.

It is possible for a `TypedDict` type to inherit from one or more other `TypedDict` types using the class-based syntax. Usage:

```
class Point3D(Point2D):
    z: int
```

`Point3D` has three items: `x`, `y` and `z`. It is equivalent to this definition:

```
class Point3D(TypedDict):
    x: int
    y: int
    z: int
```

A `TypedDict` cannot inherit from a non-TypedDict class, notably including :class:`Generic`. For example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1718); backlink

Unknown interpreted text role "class".

```
class X(TypedDict):
    x: int

class Y(TypedDict):
    y: int

class Z(object): pass  # A non-TypedDict class

class XY(X, Y): pass  # OK

class XZ(X, Z): pass  # raises TypeError

T = TypeVar('T')
class XT(X, Generic[T]): pass  # raises TypeError
```

A `TypedDict` can be introspected via annotations dicts (see :ref:`annotations-howto` for more information on annotations best practices), :attr:`__total__`, :attr:`__required_keys__`, and :attr:`__optional_keys__`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1736); backlink

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1736); backlink

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1736); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1736); *backlink*

Unknown interpreted text role "attr".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1740)

Unknown directive type "attribute".

```
.. attribute:: __total__

   ``Point2D.__total__`` gives the value of the ``total`` argument.
   Example::

      >>> from typing import TypedDict
      >>> class Point2D(TypedDict): pass
      >>> Point2D.__total__
      True
      >>> class Point2D(TypedDict, total=False): pass
      >>> Point2D.__total__
      False
      >>> class Point3D(Point2D): pass
      >>> Point3D.__total__
      True
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1756)

Unknown directive type "attribute".

```
.. attribute:: __required_keys__
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1757)

Unknown directive type "attribute".

```
.. attribute:: __optional_keys__

   ``Point2D.__required_keys__`` and ``Point2D.__optional_keys__`` return
   :class:`frozenset` objects containing required and non-required keys, respectively.
   Currently the only way to declare both required and non-required keys in the
   same ``TypedDict`` is mixed inheritance, declaring a ``TypedDict`` with one value
   for the ``total`` argument and then inheriting it from another ``TypedDict`` with
   a different value for ``total``.
   Usage::

      >>> class Point2D(TypedDict, total=False):
      ...     x: int
      ...     y: int
      ...
      >>> class Point3D(Point2D):
      ...     z: int
      ...
      >>> Point3D.__required_keys__ == frozenset({'z'})
      True
      >>> Point3D.__optional_keys__ == frozenset({'x', 'y'})
      True
```

See PEP 589 for more examples and detailed rules of using `TypedDict`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1781)

Unknown directive type "versionadded".

```
.. versionadded:: 3.8
```

Generic concrete collections

Corresponding to built-in types

A generic version of :class:`dict`. Useful for annotating return types. To annotate arguments it is preferred to use an abstract collection type such as :class:`Mapping`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1791); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1791); *backlink*

Unknown interpreted text role "class".

This type can be used as follows:

```
def count_words(text: str) -> Dict[str, int]:
    ...
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1800)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`builtins.dict <dict>` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

Generic version of :class:`list`. Useful for annotating return types. To annotate arguments it is preferred to use an abstract collection type such as :class:`Sequence` or :class:`Iterable`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1806); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1806); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1806); *backlink*

Unknown interpreted text role "class".

This type may be used as follows:

```
T = TypeVar('T', int, float)

def vec2(x: T, y: T) -> List[T]:
    return [x, y]

def keep_positives(vector: Sequence[T]) -> List[T]:
    return [item for item in vector if item > 0]
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1821)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`builtins.list <list>` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`builtins.set <set>`. Useful for annotating return types. To annotate arguments it is preferred to use an abstract collection type such as :class:`AbstractSet`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1827); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1827); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1831)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`builtins.set <set>` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`builtins.frozenset <frozenset>`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1837); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1839)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`builtins.frozenset <frozenset>` now supports ``[]``. See
```

```
:pep:`585` and :ref:`types-genericalias`.
```

Note

:data:`Tuple` is a special form.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1843); *backlink*

Unknown interpreted text role "data".

Corresponding to types in :mod:`collections`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1845); *backlink*

Unknown interpreted text role "mod".

A generic version of :class:`collections.defaultdict`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1850); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1852)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1854)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.defaultdict` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`collections.OrderedDict`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1860); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1862)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1864)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.OrderedDict` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`collections.ChainMap`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1870); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1872)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1873)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1875)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.ChainMap` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`collections.Counter`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1881); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1883)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1884)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1886)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.Counter` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`collections.deque`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1892); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1894)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1895)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1897)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.deque` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

Other concrete types

Generic type `IO[AnyStr]` and its subclasses `TextIO(IO[str])` and `BinaryIO(IO[bytes])` represent the types of I/O streams such as returned by :func:`open`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1908); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1913)

Unknown directive type "deprecated-removed".

```
.. deprecated-removed:: 3.8 3.12
   The ``typing.io`` namespace is deprecated and will be removed.
   These types should be directly imported from ``typing`` instead.
```

These type aliases correspond to the return types from :func:`re.compile` and :func:`re.match`. These types (and the corresponding functions) are generic in `AnyStr` and can be made specific by writing `Pattern[str]`, `Pattern[bytes]`, `Match[str]`, or `Match[bytes]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1920); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1920); *backlink*

Unknown interpreted text role "func".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1927)

Unknown directive type "deprecated-removed".

```
.. deprecated-removed:: 3.8 3.12
   The ``typing.re`` namespace is deprecated and will be removed.
   These types should be directly imported from ``typing`` instead.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1931)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   Classes ``Pattern`` and ``Match`` from :mod:`re` now support ``[]``.
   See :pep:`585` and :ref:`types-genericalias`.
```

`Text` is an alias for `str`. It is provided to supply a forward compatible path for Python 2 code: in Python 2, `Text` is an alias for `unicode`.

Use `Text` to indicate that a value must contain a unicode string in a manner that is compatible with both Python 2 and Python 3:

```
def add_unicode_checkmark(text: Text) -> Text:
    return text + u' \u2713'
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1947)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

Abstract Base Classes

Corresponding to collections in :mod:`collections.abc`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1952); *backlink*

Unknown interpreted text role "mod".

A generic version of :class:`collections.abc.Set`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1957); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1959)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Set` now supports ``[]``. See :pep:`585` and
   :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.ByteString`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1965); *backlink*

Unknown interpreted text role "class".

This type represents the types :class:`bytes`, :class:`bytearray`, and :class:`memoryview` of byte sequences.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1967); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1967); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1967); *backlink*

Unknown interpreted text role "class".

As a shorthand for this type, :class:`bytes` can be used to annotate arguments of any of the types mentioned above.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1970); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1973)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.ByteString` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.Collection`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1979); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1981)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.0
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1983)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Collection` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.Container`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1989); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1991)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Container` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.ItemsView`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1997); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 1999)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.ItemsView` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.KeysView`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2005); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2007)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.KeysView` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.Mapping`. This type can be used as follows:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2013); *backlink*

Unknown interpreted text role "class".

```
def get_position_in_index(word_list: Mapping[str, int], word: str) -> int:
    return word_list[word]
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2019)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Mapping` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.MappingView`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2025); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2027)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.MappingView` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.MutableMapping`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2033); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2035)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.MutableMapping` now supports ``[]``. See
   :pep:`585` and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.MutableSequence`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2041); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2043)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
```

```
:class:`collections.abc.MutableSequence` now supports ``[]``. See
:pep:`585` and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.MutableSet`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2049); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2051)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.MutableSet` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.Sequence`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2057); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2059)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Sequence` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.ValuesView`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2065); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2067)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.ValuesView` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

Corresponding to other types in :mod:`collections.abc`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2071); *backlink*

Unknown interpreted text role "mod".

A generic version of :class:`collections.abc.Iterable`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2076); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2078)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Iterable` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.Iterator`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2084); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2086)

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2101); backlink

Unknown interpreted text role "class".

If your generator will only yield values, set the `SendType` and `ReturnType` to `None`:

```
def infinite_stream(start: int) -> Generator[int, None, None]:
    while True:
        yield start
        start += 1
```

Alternatively, annotate your generator as having a return type of either `Iterable[YieldType]` or `Iterator[YieldType]`:

```
def infinite_stream(start: int) -> Iterator[int]:
    while True:
        yield start
        start += 1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2121)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Generator` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

An alias to :class:`collections.abc.Hashable`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2127); backlink

Unknown interpreted text role "class".

A generic version of :class:`collections.abc.Reversible`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2131); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2133)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Reversible` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

An alias to :class:`collections.abc.Sized`

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2139); backlink

Unknown interpreted text role "class".

Asynchronous programming

A generic version of :class:`collections.abc.Coroutine`. The variance and order of type variables correspond to those of :class:`Generator`, for example:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2146); backlink

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2146); backlink

Unknown interpreted text role "class".

```
from collections.abc import Coroutine
c: Coroutine[list[str], str, int]  # Some coroutine defined elsewhere
x = c.send('hi')                   # Inferred type of 'x' is list[str]
async def bar() -> None:
    y = await c                    # Inferred type of 'y' is int
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2156)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.3
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2158)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Coroutine` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

An async generator can be annotated by the generic type `AsyncGenerator[YieldType, SendType]`. For example:

```
async def echo_round() -> AsyncGenerator[int, float]:
    sent = yield 0
    while sent >= 0.0:
        rounded = await round(sent)
        sent = yield rounded
```

Unlike normal generators, async generators cannot return a value, so there is no `ReturnType` type parameter. As with :class:`Generator`, the `SendType` behaves contravariantly.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2173); *backlink*

Unknown interpreted text role "class".

If your generator will only yield values, set the `SendType` to `None`:

```
async def infinite_stream(start: int) -> AsyncGenerator[int, None]:
    while True:
        yield start
        start = await increment(start)
```

Alternatively, annotate your generator as having a return type of either `AsyncIterable[YieldType]` or `AsyncIterator[YieldType]`:

```
async def infinite_stream(start: int) -> AsyncIterator[int]:
    while True:
        yield start
        start = await increment(start)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2193)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.1
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2195)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.AsyncGenerator` now supports ``[]``. See
   :pep:`585` and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.AsyncIterable`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2201); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2203)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2205)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.AsyncIterable` now supports ``[]``. See :pep:`585`
```

```
and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.AsyncIterator`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2211); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2213)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2215)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.AsyncIterator` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

A generic version of :class:`collections.abc.Awaitable`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2221); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2223)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2225)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`collections.abc.Awaitable` now supports ``[]``. See :pep:`585`
   and :ref:`types-genericalias`.
```

Context manager types

A generic version of :class:`contextlib.AbstractContextManager`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2235); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2237)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2238)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.0
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2240)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`contextlib.AbstractContextManager` now supports ``[]``. See
   :pep:`585` and :ref:`types-genericalias`.
```

A generic version of :class:`contextlib.AbstractAsyncContextManager`.

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]typing.rst, line 2246); *backlink*

Unknown interpreted text role "class".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2248)

Unknown directive type "versionadded".

```
.. versionadded:: 3.5.4
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2249)

Unknown directive type "versionadded".

```
.. versionadded:: 3.6.2
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2251)

Unknown directive type "deprecated".

```
.. deprecated:: 3.9
   :class:`contextlib.AbstractAsyncContextManager` now supports ``[]``. See
   :pep:`585` and :ref:`types-genericalias`.
```

Protocols

These protocols are decorated with :func:`runtime_checkable`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2258); *backlink*

Unknown interpreted text role "func".

An ABC with one abstract method `__abs__` that is covariant in its return type.

An ABC with one abstract method `__bytes__`.

An ABC with one abstract method `__complex__`.

An ABC with one abstract method `__float__`.

An ABC with one abstract method `__index__`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2281)

Unknown directive type "versionadded".

```
.. versionadded:: 3.8
```

An ABC with one abstract method `__int__`.

An ABC with one abstract method `__round__` that is covariant in its return type.

Functions and decorators

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2295)

Unknown directive type "function".

```
.. function:: cast(typ, val)

   Cast a value to a type.

   This returns the value unchanged.  To the type checker this
   signals that the return value has the designated type, but at
   runtime we intentionally don't check anything (we want this
   to be as fast as possible).
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2304)

Unknown directive type "function".

```
.. function:: assert_type(val, typ, /)

   Ask a static type checker to confirm that *val* has an inferred type of *typ*.

   When the type checker encounters a call to ``assert_type()``, it
   emits an error if the value is not of the specified type::

       def greet(name: str) -> None:
           assert_type(name, str)  # OK, inferred type of `name` is `str`
           assert_type(name, int)  # type checker error

   At runtime this returns the first argument unchanged with no side effects.
```

```
   This function is useful for ensuring the type checker's understanding of a
   script is in line with the developer's intentions::

       def complex_function(arg: object):
           # Do some complex type-narrowing logic,
           # after which we hope the inferred type will be `int`
           ...
           # Test whether the type checker correctly understands our function
           assert_type(arg, int)

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2329)

Unknown directive type "function".

```
.. function:: assert_never(arg, /)

   Assert to the type checker that a line of code is unreachable.

   Example::

       def int_or_str(arg: int | str) -> None:
           match arg:
               case int():
                   print("It's an int")
               case str():
                   print("It's a str")
               case _ as unreachable:
                   assert_never(unreachable)

   If a type checker finds that a call to ``assert_never()`` is
   reachable, it will emit an error.

   At runtime, this throws an exception when called.

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2351)

Unknown directive type "function".

```
.. function:: reveal_type(obj)

   Reveal the inferred static type of an expression.

   When a static type checker encounters a call to this function,
   it emits a diagnostic with the type of the argument. For example::

      x: int = 1
      reveal_type(x)  # Revealed type is "builtins.int"

   This can be useful when you want to debug how your type checker
   handles a particular piece of code.

   The function returns its argument unchanged, which allows using
   it within an expression::

      x = reveal_type(1)  # Revealed type is "builtins.int"

   Most type checkers support ``reveal_type()`` anywhere, even if the
   name is not imported from ``typing``. Importing the name from
   ``typing`` allows your code to run without runtime errors and
   communicates intent more clearly.

   At runtime, this function prints the runtime type of its argument to stderr
   and returns it unchanged::

      x = reveal_type(1)  # prints "Runtime type is int"
      print(x)  # prints "1"

   .. versionadded:: 3.11
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2382)

Unknown directive type "decorator".

```
.. decorator:: overload

   The ``@overload`` decorator allows describing functions and methods
   that support multiple different combinations of argument types. A series
   of ``@overload``-decorated definitions must be followed by exactly one
   non-``@overload``-decorated definition (for the same function/method).
   The ``@overload``-decorated definitions are for the benefit of the
   type checker only, since they will be overwritten by the
   non-``@overload``-decorated definition, while the latter is used at
   runtime but should be ignored by a type checker.  At runtime, calling
   a ``@overload``-decorated function directly will raise
   :exc:`NotImplementedError`. An example of overload that gives a more
   precise type than can be expressed using a union or a type variable::

      @overload
      def process(response: None) -> None:
          ...
      @overload
```

```
        def process(response: int) -> tuple[int, str]:
            ...
        @overload
        def process(response: bytes) -> str:
            ...
        def process(response):
            <actual implementation>

    See :pep:`484` for details and comparison with other typing semantics.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2410)

Unknown directive type "decorator".

```
.. decorator:: final

   A decorator to indicate to type checkers that the decorated method
   cannot be overridden, and the decorated class cannot be subclassed.
   For example::

      class Base:
          @final
          def done(self) -> None:
              ...
      class Sub(Base):
          def done(self) -> None:  # Error reported by type checker
                ...

      @final
      class Leaf:
          ...
      class Other(Leaf):  # Error reported by type checker
          ...

   There is no runtime checking of these properties. See :pep:`591` for
   more details.

   .. versionadded:: 3.8

   .. versionchanged:: 3.11
      The decorator will now set the ``__final__`` attribute to ``True``
      on the decorated object. Thus, a check like
      ``if getattr(obj, "__final__", False)`` can be used at runtime
      to determine whether an object ``obj`` has been marked as final.
      If the decorated object does not support setting attributes,
      the decorator returns the object unchanged without raising an exception.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2444)

Unknown directive type "decorator".

```
.. decorator:: no_type_check

   Decorator to indicate that annotations are not type hints.

   This works as class or function :term:`decorator`.  With a class, it
   applies recursively to all methods and classes defined in that class
   (but not to methods defined in its superclasses or subclasses).

   This mutates the function(s) in place.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2454)

Unknown directive type "decorator".

```
.. decorator:: no_type_check_decorator

   Decorator to give another decorator the :func:`no_type_check` effect.

   This wraps the decorator with something that wraps the decorated
   function in :func:`no_type_check`.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2461)

Unknown directive type "decorator".

```
.. decorator:: type_check_only

   Decorator to mark a class or function to be unavailable at runtime.

   This decorator is itself not available at runtime. It is mainly
   intended to mark classes that are defined in type stub files if
   an implementation returns an instance of a private class::

      @type_check_only
      class Response:  # private or not available at runtime
          code: int
          def get_header(self, name: str) -> str: ...

      def fetch_response() -> Response: ...
```

```
   Note that returning instances of private classes is not recommended.
   It is usually preferable to make such classes public.
```

Introspection helpers

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2482)

Unknown directive type "function".

```
.. function:: get_type_hints(obj, globalns=None, localns=None, include_extras=False)

   Return a dictionary containing type hints for a function, method, module
   or class object.

   This is often the same as ``obj.__annotations__``. In addition,
   forward references encoded as string literals are handled by evaluating
   them in ``globals`` and ``locals`` namespaces. For a class ``C``, return
   a dictionary constructed by merging all the ``__annotations__`` along
   ``C.__mro__`` in reverse order.

   The function recursively replaces all ``Annotated[T, ...]`` with ``T``,
   unless ``include_extras`` is set to ``True`` (see :class:`Annotated` for
   more information). For example::

       class Student(NamedTuple):
           name: Annotated[str, 'some marker']

       get_type_hints(Student) == {'name': str}
       get_type_hints(Student, include_extras=False) == {'name': str}
       get_type_hints(Student, include_extras=True) == {
           'name': Annotated[str, 'some marker']
       }

   .. note::

      :func:`get_type_hints` does not work with imported
      :ref:`type aliases <type-aliases>` that include forward references.
      Enabling postponed evaluation of annotations (:pep:`563`) may remove
      the need for most forward references.

   .. versionchanged:: 3.9
      Added ``include_extras`` parameter as part of :pep:`593`.

   .. versionchanged:: 3.11
      Previously, ``Optional[t]`` was added for function and method annotations
      if a default value equal to ``None`` was set.
      Now the annotation is returned unchanged.
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2521)

Unknown directive type "function".

```
.. function:: get_args(tp)
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2522)

Unknown directive type "function".

```
.. function:: get_origin(tp)

   Provide basic introspection for generic types and special typing forms.

   For a typing object of the form ``X[Y, Z, ...]`` these functions return
   ``X`` and ``(Y, Z, ...)``. If ``X`` is a generic alias for a builtin or
   :mod:`collections` class, it gets normalized to the original class.
   If ``X`` is a union or :class:`Literal` contained in another
   generic type, the order of ``(Y, Z, ...)`` may be different from the order
   of the original arguments ``[Y, Z, ...]`` due to type caching.
   For unsupported objects return ``None`` and ``()`` correspondingly.
   Examples::

      assert get_origin(Dict[str, int]) is dict
      assert get_args(Dict[int, str]) == (int, str)

      assert get_origin(Union[int, str]) is Union
      assert get_args(Union[int, str]) == (int, str)

   .. versionadded:: 3.8
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2543)

Unknown directive type "function".

```
.. function:: is_typeddict(tp)

   Check if a type is a :class:`TypedDict`.

   For example::

      class Film(TypedDict):
          title: str
          year: int
```

```
        is_typeddict(Film)  # => True
        is_typeddict(list | str)  # => False

    .. versionadded:: 3.10
```

A class used for internal typing representation of string forward references. For example, `list["SomeClass"]` is implicitly transformed into `list[ForwardRef("SomeClass")]`. This class should not be instantiated by a user, but may be used by introspection tools.

Note

PEP 585 generic types such as `list["SomeClass"]` will not be implicitly transformed into `list[ForwardRef("SomeClass")]` and thus will not automatically resolve to `list[SomeClass]`.

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2570)

Unknown directive type "versionadded".

```
.. versionadded:: 3.7.4
```

Constant

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\cpython-main\Doc\library\[cpython-main][Doc][library]typing.rst, line 2575)

Unknown directive type "data".

```
.. data:: TYPE_CHECKING

   A special constant that is assumed to be ``True`` by 3rd party static
   type checkers. It is ``False`` at runtime. Usage::

      if TYPE_CHECKING:
          import expensive_mod

      def fun(arg: 'expensive_mod.SomeType') -> None:
          local_var: expensive_mod.AnotherType = other_fun()

   The first type annotation must be enclosed in quotes, making it a
   "forward reference", to hide the ``expensive_mod`` reference from the
   interpreter runtime.  Type annotations for local variables are not
   evaluated, so the second annotation does not need to be enclosed in quotes.

   .. note::

      If ``from __future__ import annotations`` is used in Python 3.7 or later,
      annotations are not evaluated at function definition time.
      Instead, they are stored as strings in ``__annotations__``.
      This makes it unnecessary to use quotes around the annotation.
      (see :pep:`563`).

   .. versionadded:: 3.5.2
```