




Type Hinting in PyCharm

In this section:

- [Basics](#)
- [Type syntax](#)
- [Specifying types of parameters](#)
- [Specifying return types](#)
- [Specifying types of local variables](#)
- [Specifying types of fields](#)
- [Annotating lists and dictionaries](#)

Basics


Python is dynamically typed. That's why one doesn't need to specify variable types explicitly. However, it is possible to use *docstrings* ([reStructuredText](#) , [epytext](#) ) and [py3 annotations](#)  to specify information about the expected types of the following values:

- parameters passed to a function
- return values
- local variables
- fields

PyCharm suggests [code completion](#) with these expected types.

Type syntax

Type syntax in Python docstrings is not defined by any standard. Thus, PyCharm suggests the following notation:

- `Foo` # Class `Foo` visible in the current scope
- `x.y.Bar` # Class `Bar` from `x.y` module
- `Foo | Bar` # `Foo` or `Bar`
- `(Foo, Bar)` # Tuple of `Foo` and `Bar`
- `list[Foo]` # List of `Foo` elements
- `dict[Foo, Bar]` # Dict from `Foo` to `Bar`
- `T` # Generic type (`T-Z` are reserved for generics)
- `T <= Foo` # Generic type with upper bound `Foo`
- `Foo[T]` # `Foo` parameterized with `T`
- `(Foo, Bar) -> Baz` # Function of `Foo` and `Bar` that returns `Baz`
- `list[dict[str, datetime]]` # List of dicts from `str` to `datetime` (nested arguments)
- `:param "type_name" "param_name": "param_description"` # Combining parameter type and documentation in a single line. See [Sphinx documentation](#)  for details.)

Specifying types of parameters

Consider adding information about the expected parameter type. This information is specified using `:type` or `@type` docstrings:

```

1 def f(param):
2     """
3     @type param: int
4     """
5     param.

```

m	bit_length(self)	int
m	conjugate(self, args, kwargs)	int
f	denominator	int
f	imag	int
f	numerator	int

```

1 def f(param):
2     """
3     :type param: int
4     """
5     param.

```

m	bit_length(self)	int
m	conjugate(self, args, kwargs)	int
f	denominator	int
f	imag	int
f	numerator	int

When Python 3 is specified as the project interpreter, you can also use annotations to specify the expected parameter type:

```

7 def f(param: int):
8     param.

```

m	bit_length(self)	int
m	conjugate(self, args, kwargs)	int
f	denominator	int

Specifying return types

Use docstrings `:rtype` or `@rtype` to specify the expected return type:

```

7 def f():
8     """
9     :rtype: int
10    """
11    a=f()
12    a.

```

m	bit_length(self)	int
m	conjugate(self, args, kwargs)	int
f	denominator	int

- `:rtype: collections.Iterable[int]` # return type: 'items' is of type generator or `collections.Iterable`, 'a' is of type `int`, see the following code:

```

def my_iter():
    for i in range(10):
        yield i
items = my_iter()
for a in items:
    print a

```

- `:rtype: list[int]` for `my_iter` # return type: 'a' is of type `int`, see the following code:

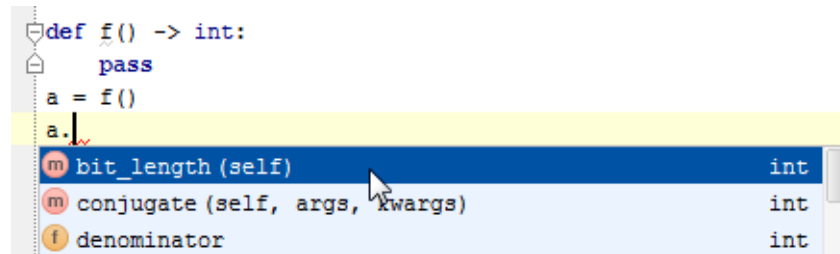
```

def my_iter():
    for i in range(10):
        yield i
for a in my_iter():
    print a

```

When Python 3 is specified as the project interpreter, you can use annotations to specify the expected return type:

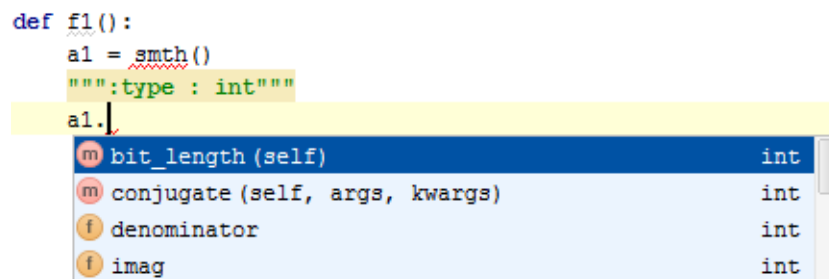
```
def f() -> int:
    pass
a = f()
a.
```



Specifying types of local variables

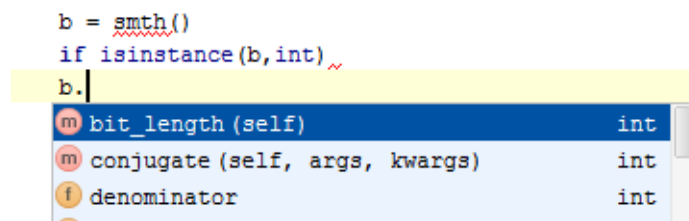
Consider adding information about the expected type of a local variable using `:type` or `@type` docstrings:

```
def f1():
    a1 = smth()
    """ :type : int """
    a1.
```

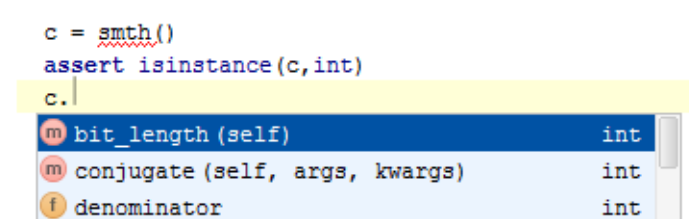


It is also possible to use `isinstance` to define the expected local variable type:

```
b = smth()
if isinstance(b, int):
    b.
```



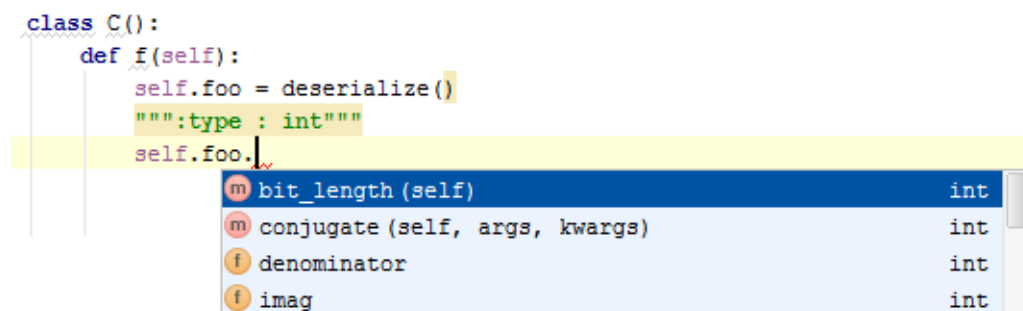
```
c = smth()
assert isinstance(c, int)
c.
```



Specifying types of fields

Finally, you can use type hinting to specify the expected type of fields:

```
class C():
    def f(self):
        self.foo = deserialize()
        """ :type : int """
        self.foo.
```



Annotating lists and dictionaries

```
:type: dict of [str, C]
:type: list of [str]
```

See Also

Procedures:

- [Auto-Completing Code](#)

External Links:

- [reStructuredText](#) 
- [epytext](#) 

Web Resources:

- [Developer Community](#) 