Powering Innovation That Drives Human Advancement

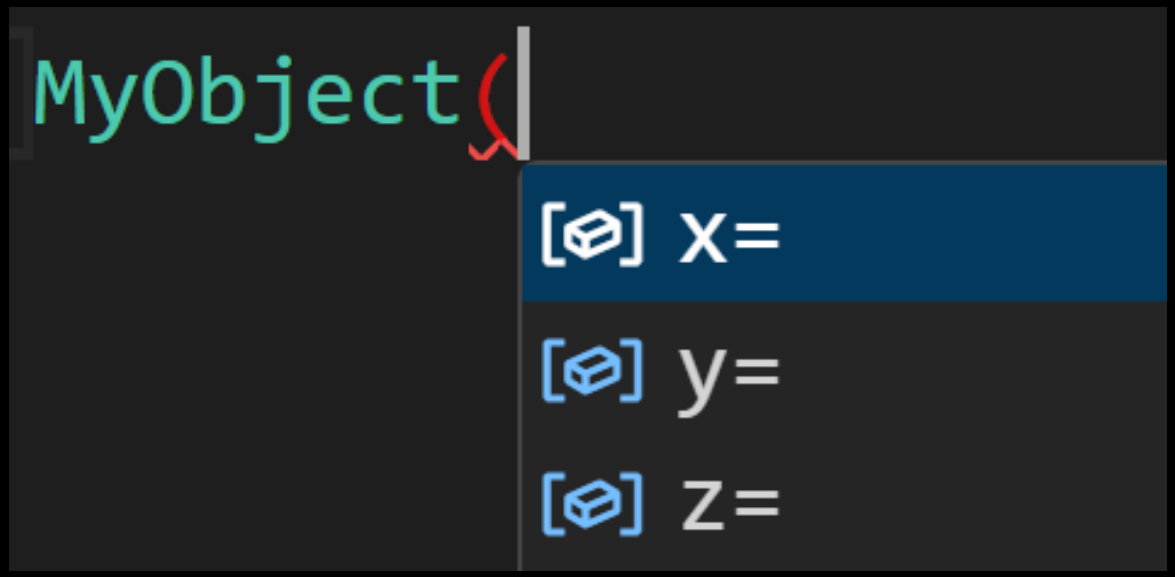
Better Autocomplete with Type Hints

Dominik Gresch

```python
x: int = 4
def foo(y: str) -> int:
```

Static code checks… but can they do more?

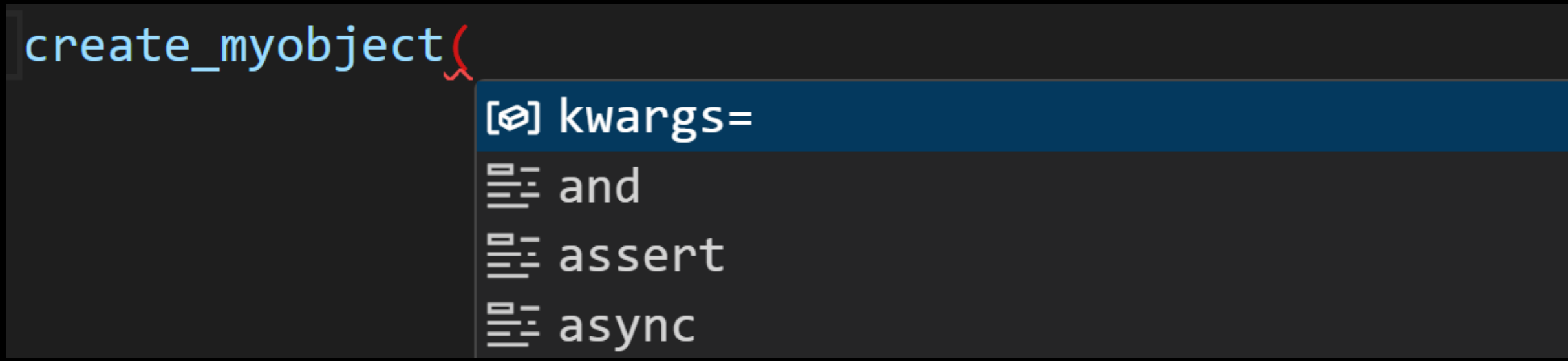
```
class MyObject:
    def __init__(self, x, y, z):
        self.x = x
        self.y = y
        self.z = z
```



```python
def define_object_factory(object_type):
    def inner(*args, **kwargs):
        return object_type(*args, **kwargs)

    return inner

create_myobject = define_object_factory(MyObject)
```



```python
from collections.abc import Collection

class MyCollection(Collection):
    def __init__(self, values):
        self._values = list(values)

    def __contains__(self, value):
        return value in self._values

    def __iter__(self):
        return iter(self._values)

    def __len__(self):
        return len(self._values)


def create_collection(values):
    return MyCollection(values=values)
```

```
collection = create_collection(values=[MyObject(x=1, y="a", z=False)])

obj = next(iter(collection))
✨
obj.
```

No suggestions.

Add type hints!

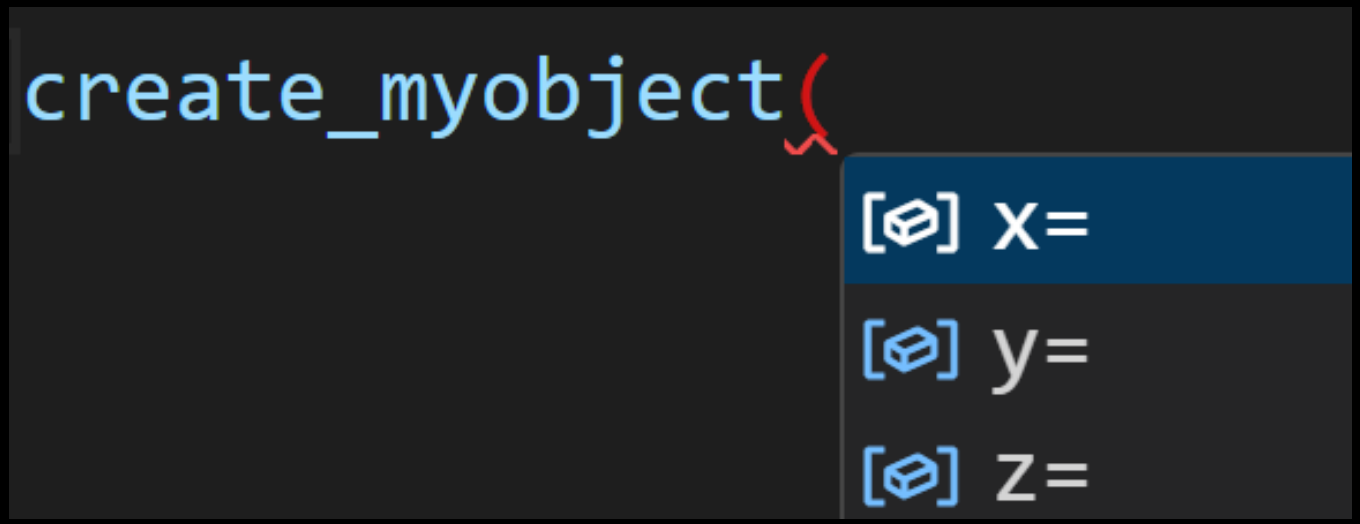
```python
class MyObject:
    def __init__(self, x: int, y: str, z: bool):
        self.x = x
        self.y = y
        self.z = z
```

```python
from typing import ParamSpec, TypeVar


T = TypeVar("T")
P = ParamSpec("P")


def define_object_factory(object_type: Callable[P, T]) ->
Callable[P, T]:
    def inner(*args: P.args, **kwargs: P.kwargs) -> T:
        return object_type(*args, **kwargs)

    return inner
```



```python
from collections.abc import Callable, Collection, Iterable, Iterator
from typing import TypeVar

T = TypeVar("T")

class MyCollection(Collection[T]):
    def __init__(self, values: Iterable[T]):
        self._values = list(values)

    def __contains__(self, value: object) -> bool:
        return value in self._values

    def __iter__(self) -> Iterator[T]:
        return iter(self._values)

    def __len__(self) -> int:
        return len(self._values)


def create_collection(values: Iterable[T] = ()) -> MyCollection[T]:
    return MyCollection(values=values)
```

```
collection = create_collection(values=[MyObject(x=1, y="a", z=False)])

obj = next(iter(collection))
obj.
```

- x
- y
- z

Happy Users!

Testing?

```
from typing_extensions import reveal_type

reveal_type(create_myobject)
```

```
Revealed type is "def (x: builtins.int, y: builtins.str, z:
builtins.bool) -> with_typehints.MyObject"
```

```
from typing import Callable
from typing_extensions import assert_type
from mypy_extensions import Arg

assert_type(
    create_myobject,
    Callable[
        [Arg(int, 'x'), Arg(str, 'y'), Arg(bool, 'z’)],
        MyObject
    ]
)
```

Happy Developers!

Ansys