```
import pytest

from swarms.utils import print_class_parameters


class TestObject:
    def __init__(self, value1, value2: int):
        pass


class TestObject2:
    def __init__(self: "TestObject2", value1, value2: int = 5):
        pass


def test_class_with_complex_parameters():
    class ComplexArgs:
        def __init__(self, value1: list, value2: dict = {}):
            pass

    output = {"value1": "<class 'list'>", "value2": "<class 'dict'>"}
    assert (
        print_class_parameters(ComplexArgs, api_format=True) == output
    )
```

```
def test_empty_class():
    class Empty:
        pass

    with pytest.raises(Exception):
        print_class_parameters(Empty)



def test_class_with_no_annotations():
    class NoAnnotations:
        def __init__(self, value1, value2):
            pass

    output = {
        "value1": "<class 'inspect._empty'>",
        "value2": "<class 'inspect._empty'>",
    }
    assert (
        print_class_parameters(NoAnnotations, api_format=True)
        == output
    )



def test_class_with_partial_annotations():
    class PartialAnnotations:
        def __init__(self, value1, value2: int):
```

```
        pass


    output = {
        "value1": "<class 'inspect._empty'>",
        "value2": "<class 'int'>",
    }
    assert (
        print_class_parameters(PartialAnnotations, api_format=True)
        == output
    )



@pytest.mark.parametrize(
    "obj, expected",
    [
        (
            TestObject,
            {
                "value1": "<class 'inspect._empty'>",
                "value2": "<class 'int'>",
            },
        ),
        (
            TestObject2,
            {
                "value1": "<class 'inspect._empty'>",
```

```
                "value2": "<class 'int'>",
            },
        ),
    ],
)
def test_parametrized_class_parameters(obj, expected):
    assert print_class_parameters(obj, api_format=True) == expected



@pytest.mark.parametrize(
    "value",
    [
        int,
        float,
        str,
        list,
        set,
        dict,
        bool,
        tuple,
        complex,
        bytes,
        bytearray,
        memoryview,
        range,
        frozenset,
```

```
        slice,
        object,
    ],
)
def test_not_class_exception(value):
    with pytest.raises(Exception):
        print_class_parameters(value)



def test_api_format_flag():
    assert print_class_parameters(TestObject2, api_format=True) == {
        "value1": "<class 'inspect._empty'>",
        "value2": "<class 'int'>",
    }
    print_class_parameters(TestObject)
    # TODO: Capture printed output and assert correctness.
```