```python
#!/usr/bin/env python3
# -*-coding:Utf-8 -*


# The MIT License (MIT)
#
# Copyright (c) 2016 Rémi Blaise <remi.blaise@gmx.fr> "http://php-zzortell.rhcloud.com/"
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in all
# copies or substantial portions of the Software.
#
# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
# SOFTWARE.


def inject_arguments(in_function):
    """Inject arguments of a method as attributes

    To use as decorator.
    """

    def out_function(*args, **kwargs):
        _self = args[0]

        # Get all of argument's names of the in_function
        all_names = in_function.__code__.co_varnames[1:in_function.__code__.co_argcount]

        ## Add default values for non-specified arguments
        defaults = in_function.__defaults__
        if defaults:
            _self.__dict__.update(zip(all_names[-len(defaults):], defaults))

        ## Add kwargs
        _self.__dict__.update(kwargs)

        ## Add args
        # Get only the names that don't belong to kwargs
        names = [n for n in all_names if not n in kwargs]
        # Match argument names with values
        _self.__dict__.update(zip(names, args[1:]))

        return in_function(*args, **kwargs)

    return out_function


if __name__=='__main__':
    import unittest

    class ArgumentInjectionTest(unittest.TestCase):
        def test(self):
            class Test:
                @inject_arguments
                def __init__(self, name, surname, default = 'lol'):
                    pass

            t = Test('mickey', surname='mouse')
            self.assertEqual('mickey', t.name)
            self.assertEqual('mouse', t.surname)
            self.assertEqual('lol', t.default)

        def test_defaultAlone(self):
            class Test:
                @inject_arguments
                def __init__(self, default='lol'):
                    pass

            t = Test('given')
            self.assertEqual('given', t.default)

        def test_inheritance(self):
            class A():
                @inject_arguments
                def __init__(self, a1):
                    pass

            class B(A):
                @inject_arguments
                def __init__(self, b1 = None, b2 = None, *args, **kwargs):
                    super().__init__(*args, **kwargs)

            b = B(0, 1, 2)
            self.assertEqual(0, b.b1)
            self.assertEqual(1, b.b2)
            self.assertEqual(2, b.a1)

        def test_defaultInheritance(self):
```

```
            class Test:
                @inject_arguments
                def __init__(self, default='lol'):
                    pass

            class Child(Test):
                @inject_arguments
                def __init__(self, minus = None, malus = None, *args, **kwargs):
                    super().__init__(*args, **kwargs)

            c = Child(1, -1)
            self.assertEqual(1, c.minus)
            self.assertEqual(-1, c.malus)
            self.assertEqual('lol', c.default)

            c = Child(1, -1, 'hey')
            self.assertEqual(1, c.minus)
            self.assertEqual(-1, c.malus)
            self.assertEqual('hey', c.default)

        def test_giveLastDefaultArgument (self):
            class TestLastGivenDefault :
                @inject_arguments
                def __init__(self, default1=1, default2=2):
                    pass

            t = TestLastGivenDefault (default2=3)
            self.assertEqual(1, t.default1)
            self.assertEqual(3, t.default2)

    unittest.main()
```