```
import sympy as sp
import numpy as np
import functools

x, y = sp.symbols('x y', real=True)
f = 3 * (x - 5)**4 + (10 * ((y - 9)**2))
g = sp.Max(x - 5, 0) + (10 * sp.Abs(y - 9))
relu = sp.Max(x,0)

def f_real(x, y):
    return 3 * (x - 5)**4 + 10 * (y - 9)**2


def g_real(x, y):
    return np.maximum(x - 5, 0) + 10 * np.abs(y - 9)

def relu_real(x):
    return np.maximum(x,0)


def apply_sym(x, f):
    for x_sym, x_val in zip(f.free_symbols, x):
        f = f.subs(x_sym, x_val)
    return f

config = {
    "f": {
        "sym": f,
        "real": f_real,
        "name": "f",
    },
    "g": {
        "sym": g,
        "real": g_real,
        "name": "g",
    },
    "relu": {
        "sym": relu,
        "real": lambda x: max(x, 0),
        "name": "relu",
    }
}

class GradientDescent():
    def __init__(self):
        self._max_iter = 1000
        self._debug = False
        self._converged = lambda x1, x2: False
        self._epsilon = 0.0001
        self._dimension = None
        self._beta = 0
        self._algorithm = None
        self._iteration = None
        self._function = None
        self._sum = None
        self._x_value = None
        self._step_coeff = None
        self._converged_value = None
        self._grad_value = None
        self._m = None
        self._v = None
        self._adam_grad = None
        self._beta = None
        self._beta2 = None
        self._step_size = None
        self._z = None
        self._f_star = None

    def step_size(self, a):
        self._step_size = a
        return self

    def beta(self, b):
        self._beta = b
        return self

    def beta2(self, b):
        self._beta2 = b
        return self

    def epsilon(self, e):
        self._epsilon = e
        return self

    def function(self, f, function_name=None, dimension=None):
        self._function = f
        self.function_name = function_name
        self._dimension = dimension
        return self

    def sym_function(self, function, function_name=None):
        self.function_name = function_name
        self._dimension = len(function.free_symbols)
        def fn(x):
            return apply_sym(x, function)

        diffs = [function.diff(var) for var in function.free_symbols]

        def grad(x):
            return np.array([
                apply_sym(x, diff) for diff in diffs])

        self._function = fn
        self._gradient = grad
        return self

    def gradient(self, g):
        self._gradient = g
        return self

    def max_iter(self, m):
        self._max_iter = m
        return self

    def start(self, s):
        self._start = s
        self._x_value = s
        return self

    def debug(self, d):
        self._debug = d
        return self

    def converged(self, c):
        self._converged = c
        return self

    def set_iterate(self, f):
        self.iterate = functools.partial(f, self)
        return self

    def algorithm(self, alg):
        self._algorithm = alg
        if self._algorithm == "rmsprop":
            import rmsprop
            self.set_iterate(rmsprop.iterate)
        elif self._algorithm == "adam":
            import adam
            self.set_iterate(adam.iterate)
        elif self._algorithm == "heavy_ball":
            import heavy_ball
            self.set_iterate(heavy_ball.iterate)
        else:
            raise Exception("Unknown algorithm:" + alg)
        return self

    def state_dict(self):
        print(self._function(self._x_value))
        return {
            "alg": self._algorithm,
            "function_name": self.function_name,
            "iteration": self._iteration,
            "step_coeff": self._step_coeff,
            "adam_grad": self._adam_grad,
            "f(x)": self._function(self._x_value),
            "epsilon": self._epsilon,
            "converged": self._converged_value,
            "gradient": self._grad_value,
            "m": self._m,
            "v": self._v,
            "beta1": self._beta,
            "beta2": self._beta2,
            "alpha": self._step_size,
            "sum": self._sum,
            "z": self._z,
            **{"x" + str(i): self._x_value[i] for i in range(len(self._x_value))},
        }

    def run2csv(self, fname, summarise=True):
        import pandas as pd
        iterations = list(self.iterate())
        df = pd.DataFrame(iterations)
        df.to_csv(fname)
        if(summarise):
            with open(fname + ".summary", "w") as f:
                print(f"iterations: {len(df)}", file=f)
                print(f"start: {df[’x0’][0]} {df[’x1’][0]}", file=f)
                print(f"final: {df[’x0’][len(df) − 1]} {df[’x1’][len(df) − 1]}", file=f)


if __name__ == "__main__":
    print(f.diff(x), f.diff(y))
    print(g.diff(x), g.diff(y))
```