```
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:
```

```
101:             raise Exception("Unknown algorithm:" + alg)
102:         return self
103:
104:     def state_dict(self):
105:         print(self._function(self._x_value))
106:         return {
107:             "alg": self._algorithm,
108:             "function_name": self.function_name,
109:             "iteration": self._iteration,
110:             "step_coeff": self._step_coeff,
111:             "adam_grad": self._adam_grad,
112:             "f(x)": self._function(self._x_value),
113:             "epsilon": self._epsilon,
114:             "converged": self._converged_value,
115:             "gradient": self._grad_value,
116:             "m": self._m,
117:             "v": self._v,
118:             "beta1": self._beta,
119:             "beta2": self._beta2,
120:             "alpha": self._step_size,
121:             "sum": self._sum,
122:             "z": self._z,
123:             **{"x" + str(i): self._x_value[i] for i in range(len(self._x_value))},
124:         }
125:
126:     def run2csv(self, fname, summarise=True):
127:         import pandas as pd
128:         iterations = list(self.iterate())
129:         df = pd.DataFrame(iterations)
130:         df.to_csv(fname)
131:         if(summarise):
132:             with open(fname + ".summary", "w") as f:
133:                 print(f"iterations: {len(df)}", file=f)
134:                 print(f"start: {df['x0'][0]} {df['x1'][0]}", file=f)
135:                 print(f"final: {df['x0'][len(df) - 1]} {df['x1'][len(df) - 1]}", file=f)
```