```
from abc import ABC, abstractmethod
import numpy as np
from math import sqrt
from src.rectangle import Rectangle
from src.source import area, middle_val, middle_val_grad


class BaseFem(ABC):
    def __init__(self, n, shape):
        # omega setup
        self.shape = shape
        self.omega = Rectangle(self.shape)
        self.omega.triangulate(n)

        # function set up manually
        self.function = lambda x, y: x * (1 - x) * y * (1 - y)
        self.grad = lambda x, y: ((1 - 2 * x) * (y - y ** 2) + (1 - 2 * y) * (x - x ** 2)
)

        self.is_grad = False
        self.node_values = []

    def error(self, is_grad=False):
        ans = 0

        self.set_error_type(is_grad)
        for _ in self.omega.triangles.triangles:
            ans += self.int_triang(_)

        return sqrt(ans)

    def set_error_type(self, is_grad):
        self.is_grad = is_grad

    def get_coords(self, tr):
        omega = self.omega
        trc = np.zeros((3, 3))
        trc[0][0], trc[0][1], trc[0][2] = omega.x_coords[tr[0]], omega.y_coords[tr[0]], s
elf.node_values[tr[0]]
        trc[1][0], trc[1][1], trc[1][2] = omega.x_coords[tr[1]], omega.y_coords[tr[1]], s
elf.node_values[tr[1]]
        trc[2][0], trc[2][1], trc[2][2] = omega.x_coords[tr[2]], omega.y_coords[tr[2]], s
elf.node_values[tr[2]]
        return trc

    def middle(self, trc, a, b):
        x = (trc[b][0] + trc[a][0]) / 2
        y = (trc[b][1] + trc[a][1]) / 2
        if self.is_grad:
            mid_val = middle_val_grad(trc)
            return (abs(mid_val - self.grad(x, y))) ** 2
        else:
            mid_val = middle_val(trc, a, b, x, y)
            return (abs(mid_val - self.function(x, y))) ** 2

    def int_triang(self, tr):
        trc = self.get_coords(tr)
        f01 = self.middle(trc, 0, 1)
        f12 = self.middle(trc, 1, 2)
        f20 = self.middle(trc, 2, 0)
        return area(trc) / 3 * (f01 + f12 + f20)

    @abstractmethod
    def set_node_values(self):
        pass
```