```python
from math import sin, radians, degrees, pi

class Neville:
    def __init__(self, p):
        """p(x) := (x; f; [x_0, ... , x_n])"""
        self.x = p[0]
        self.func = p[1]
        # for each x calculate the corresponding y and eliminate duplicates
        self.grid_points = [(x, self.func(x)) for x in p[2]]
        self.pik = [
            [None for _ in range(k)] if k != 0
            else [x_y[1] for x_y in self.grid_points]
            for k in range(len(self.grid_points))
        ]

    def compute(self):
        for k in range(len(self.pik) - 1):
            for i in range(k, len(self.pik[k]) - 1):
                next_pik = self.pik[k][i+1]
                next_pik += (self.x - self.grid_points[i+1][0]) /
                            (self.grid_points[i+1][0] - self.grid_points[i-k][0]) *
                            (self.pik[k][i+1] - self.pik[k][i])
                self.pik[k+1].append(next_pik)
        return self.pik[len(self.pik)-1][len(self.pik[len(self.pik)-1]) - 1]

def main():
    x_n = [0, 30, 60, 90]
    # transform the values from degree to radian
    x_n = [radians(x) for x in x_n]

    # p(x) := (x; f; [x_0, ... , x_n])
    px = [radians(45), sin, x_n]
    sin_interpol = Neville(px)
    print("Result: ", sin_interpol.compute(), "\n")

    # create a scaled version of the sin() function
    scaled_sin = lambda n: lambda x: sin(n*x)
    for i in [2, 4, 8]:
        px = [radians(45), scaled_sin(i), x_n]
        sin_interpol_b = Neville(px)
        print(sin_interpol_b.compute())

if __name__ == "__main__":
    main()
```