Example 134. Use Simpson's rule with $h=\frac{1}{2}$ to approximate $\int_1^3 \frac{1}{x}\,\mathrm{d}x = \log(3) \approx 1.09861$.

Solution.

$$\int_1^3 f(x)\,\mathrm{d}x \approx \frac{h}{3}\left[f(1)+4f\left(\frac{3}{2}\right)+2f(2)+4f\left(\frac{5}{2}\right)+f(3)\right] = \frac{1}{6}\left[1+4\cdot\frac{2}{3}+2\cdot\frac{1}{2}+4\cdot\frac{2}{5}+\frac{1}{3}\right] = \frac{11}{10} = 1.1$$

Comment. Note that $\frac{11}{10}$ is exactly what we obtained in the last part of Example 131.
Indeed, recall that Simpson's rule with $h=\frac{1}{2}$ ($n=4$) is equivalent to applying Richardson extrapolation to the trapezoidal approximations with $h=\frac{1}{2}$ ($n=4$) and $h=1$ ($n=2$).

Example 135. Python Let us compute $\int_1^3 \frac{1}{x}\,\mathrm{d}x = \log(3) \approx 1.09861$ by implementing Simpson's rule. The following code assumes that `n` is even.

```
>>> def simpson_rule(f, a, b, n):
        h = (b - a) / n
        integral = f(a) + f(b)
        for i in range(1,n):
            if i%2 == 1:
                integral += 4*f(a + i*h)
            else:
                integral += 2*f(a + i*h)
        return h/3*integral

>>> def f(x):
        return 1/x
```

With `n` set to 4, we obtain $\frac{11}{10}$ as in Examples 131 and 134:

```
>>> simpson_rule(f, 1, 3, 4)

   1.0999999999999999
```

Our approximations (here, $n=10$ and 100) quickly approach the true value:

```
>>> [simpson_rule(f, 1, 3, 10**n) for n in range(1,3)]

   [1.0986605986605984, 1.0986122939305363]
```

Indeed, the following convincingly illustrates that the error in Simpson's rule is $O(h^4)$.

```
>>> from math import log

>>> [simpson_rule(f, 1, 3, 10**n) - log(3) for n in range(1,6)]

   [4.830999248861545e-05, 5.262426494567762e-09, 5.282441151166495e-13, -
   1.5543122344752192e-15, -7.105427357601002e-15]
```

Example 136. Python Various integration methods are already implemented in `scipy`.

```
>>> from scipy import integrate
```

For instance, the following is a way to use Simpson's rule with $n=4$ (so that 5 points are used). The result matches the $\frac{11}{10}$ that we computed ourselves.

```
>>> def f(x):
        return 1/x

>>> xvalues = [1+1/2*i for i in range(5)]

>>> yvalues = [f(x) for x in xvalues]

>>> integrate.simps(yvalues, xvalues)

   1.0999999999999999
```

On the other hand, the following is a convenient way of "general purpose integration", where we only need to specify the end points:

```
>>> integrate.quad(f, 1, 3)

   (1.0986122886681096, 7.555511459798467e-14)
```

The second part of the result is an estimate for the absolute error.