Exercise 1.29. Simpson's Rule is a more accurate method of numerical integration than the method illustrated above. Using Simpson's Rule, the integral of a function f between a and b is approximated as

$$\frac{h}{3}[y_0+4\,y_1+2\,y_2+4\,y_3+2\,y_4+\cdots+2\,y_{n-2}+4\,y_{n-1}+y_n]$$

where $h=(b-a)/n$, for some even integer n, and $y_k=f(a+k\,h)$. (Increasing n increases the accuracy of the approximation.) Define a procedure that takes as arguments f, a, b, and n and returns the value of the integral, computed using Simpson's Rule. Use your procedure to integrate `cube` between 0 and 1 (with $n=100$ and $n=1000$), and compare the results to those of the `integral` procedure shown above.

Answer. In order to write down our procedure effectively, we are supposed to dig out the patterns submerged in the formula above. We start with analyzing its structure hoping that we could get some intuition for designing our procedure. The following questions will probably help us capture those underlying patterns:

1. Is the formula able to be decomposed into several parts among which we can use the `sum` procedure?

2. What are the lower limit and the upper limit of the integral?

3. How to represent terms in the summation of y in our procedure?

4. What is the incremental step here?

At a glance of the formula, one might immediately find out that it can be expressed with the product of $\frac{h}{3}$ and a summation which probably can be depicted by our `sum` procedure. By observing the subscript of each term in the summation of y, we see that the lower limit of the intergral is 0 and the upper limit is n. Thus, by wishful thiking, we can put down the prototype of our procedure for approaching definite integral in terms of p `sum` as follow:

```
(define (simp-int f a b n)
  (* (/ h 3)
     (sum simp-term 0 inc n)))
```

Now what we have to do is to clarify two procedures which we left above: `simp-term` and `next`. Well, by looking at the patterns in coefficients of terms in the summation series

$$\text{Coefficient of } y_k \;=\; \begin{cases} 1 & k=1 \text{ or } k=n \\ 4 & k \text{ is an odd integer} \\ 2 & k \text{ is an even integer} \end{cases}$$

and that $y_k=f(a+k\,h)$, we are now able to express our `simp-term` procedure as:

```
(define simp-term
    (lambda (k) (* (coef k)
                   (y k))))
  (define (coef k)
    (cond ((or (= k 0) (= k n)) 1)
          ((even? k) 2)
          (else 4)))
  (define (y k)
    (f (+ a (* k h))))
```

The incremental step `inc` here is obvious, just increase the subscript of y_k by 1:

```
(define inc (lambda (x) (+ x 1)))
```

So far, we have create all the building blocks for our `simp-int` procedure. All we have to do now is to assemble them into a complete module:

```
(define (simp-int f a b n)
  (define h (/ (- b a) n))
  (define simp-term
    (lambda (k) (* (coef k)
                   (y k))))
  (define (coef k)
      (cond ((or (= k 0) (= k n)) 1)
            ((even? k) 2)
            (else 4)))
    (define (y k)
      (f (+ a (* k h))))
    (define inc (lambda (x) (+ x 1)))
    (* (/ h 3)
       (sum simp-term 0 inc n)))
```

Finally, let's use our procedure to integrate **cube** between 0 and 1 (with $n = 100$ and $n = 1000$), and compare the results to those of the **integral** procedure.

```
(simp-int cube 0 1 100.00)
;Value: .24999999999999992

(simp-int cube 0 1 1000.00)
;Value: .2500000000000003
```

Obiviously, comparing with that performance of the original **integral** procedure, our procedure here integrates more accurately.