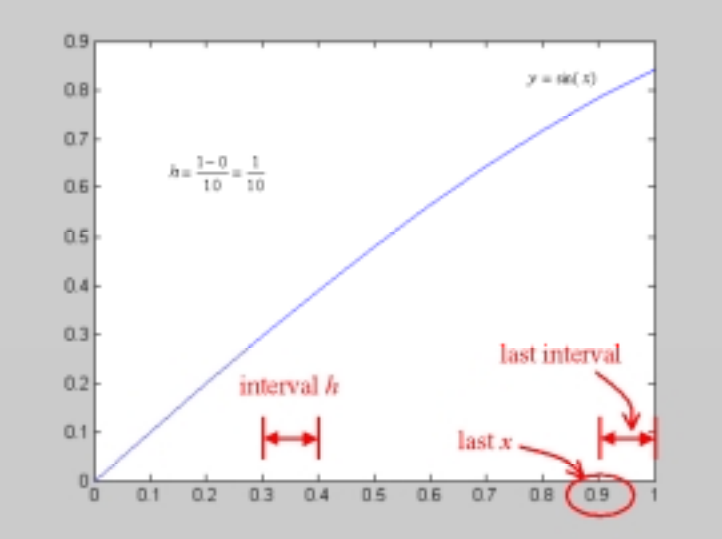
All of the functions that you write for this homework should be defined in the same source file, hw2.scm. Each question is worth 2 points, for a total of 20.

1. In the "Lambda" lecture I gave a version of the sum procedure that takes two procedures `term` and `next` and a range defined by `a` and `b`, and returns a summation of the terms over the range. As given, the procedure generates a process that is linear recursive. Rewrite the procedure to generate an iterative process. You can use the following skeleton.

```
(define (sum term a next b)
  (define (iter a result)
  (if ⟨??⟩)
      ⟨??⟩
      (iter ⟨??⟩ ⟨??⟩)))
  (iter ⟨??⟩ ⟨??⟩))
```

2. Simpson's rule is one way of numerically integrating a function over a range. Recall the drawing in the "More Lambda" lecture of a function over a range with some intervals.



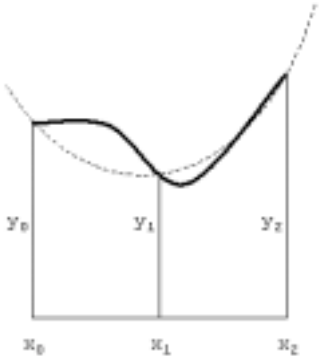
The integral of any function f(x) = y between a and b is approximated by

$$\int_a^b f(x) \approx \frac{h}{3}\left[y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \cdots + 2y_{n-2} + 4y_{n-1} + y_n\right]$$

where $h = \frac{b-a}{n}$ for some even integer *n* and $y_k = f(a+kh)$.

Write a procedure `simp1` that takes as arguments `f`, `a`, `b`, and `n`, and uses your `sum` procedure to calculate and return a value for the approximate integral. You can assume that `n` is even. Note that you need to be able to recognize which interval you are working on in order to multiply by the correct coefficient. Also, due to roundoff errors, `a+n·h` may not exactly equal `b`. It may be easier to sum over the integers k in the interval 0 to `n`.

3. The summation given in problem 2 is the result of adding up second order approximations for the area under the curve on successive intervals $[x_0,x_2]$, $[x_2,x_4]$, …, $[x_{n-2},x_n]$ and expressing the resulting sum as one formula. The approximation for an individual interval is given by $Area_{[x_i,x_{i+2}]} = \frac{h}{3}\cdot\left[y_i + 4y_{i+1} + y_{i+2}\right]$, where $x_k = a + kh$ and *h*, *y*, and *n* are the same as above.



If you add these terms up for i = 0 to n-2, you will see that you have generated the formula in problem 2. Again you can assume that n is even.

Write a second procedure simp2 that takes the same arguments `f`, `a`, `b`, and `n`, and uses the `sum` procedure to calculate and return a value for the approximate integral. The difference here is that your `term` procedure calculates $\left[y_i + 4y_{i+1} + y_{i+2}\right]$ each time it is called, rather than a single y_i value as in problem 2. Your `next` function must operate accordingly.

4. The procedure `sum` is only one of many similar abstractions. Write a procedure named `product` that takes as arguments two procedures `term` and `next` and a range defined by `a` and `b`, and returns the product of the terms over the range. Note the slightly odd order of the arguments: `(product term a next b)`

5. Write a procedure named `factorial` that uses `product` to calculate the factorial of a positive integer.

6. Notice that sum and product are both special cases of a more general notion that we could call accumulate that combines a collection of terms using some more general accumulation procedure.

Accumulate takes as arguments the same term and range specifications as sum and product, plus a combiner function and a base value. The combiner function takes two arguments and combines the current term with the accumulation of the preceding terms. The base value specify the value to use when the terms run out. Again, note the order of the arguments.

```
(accumulate comb null-value term a next b)
```

a. Write the `accumulate` procedure.

b. Write a procedure named `acc-product` that takes as arguments two procedures `term` and `next` and a range defined by `a` and `b`, and returns the product of the terms over the range. `acc-product` should use `accumulate` to do the actual accumulation task. The arguments should be in the same order as in problem 4 above.

7. The product of terms $\frac{2i \cdot (2i+2)}{(2i+1)(2i+1)}, i = 1,2,\ldots n$ is approximately equal to $\frac{\pi}{4}$.

In other words, $\frac{\pi}{4} \approx \frac{2 \cdot 4 \cdot 4 \cdot 6 \cdot 6 \cdot 8 \cdots (2n) \cdot (2n+2)}{3 \cdot 3 \cdot 5 \cdot 5 \cdot 7 \cdot 7 \cdots (2n+1)(2n+1)}$

Use `acc-product` to implement procedure `wallis-pi` that takes one argument `n` and returns an approximate value for pi.

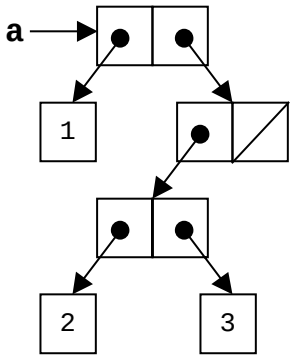
8a. Write a procedure double that takes a procedure of one argument as its argument and returns a procedure that applies the original procedure twice. For example, if inc is a procedure that that adds 1 to its argument, then (double inc) should evaluate to a procedure that adds 2.

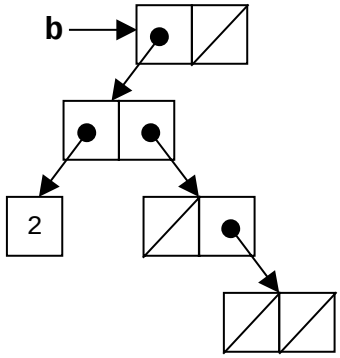
8b. Let f and g be two one-argument functions. The composition f after g is written f∘g or (f(g(x)). Write a procedure `compose` that implements composition. For example, if `inc` is a procedure that adds 1 to its argument and `square` takes the square of its argument, then `((compose square inc) 6)` returns 49.

9. Provide `cons` statements

```
(define a (cons foo bar))
(define a (cons baz faz))
```

that will produce the structures shown here. Refer to the Pairs lecture for examples of similar `cons` statements.





10. Show that we can represent pairs of non-negative integers using only numbers and arithmetic operations if we represent the pair *a* and *b* as the integer that is the product 2^a3^b. Define three procedures i-cons, i-car, and i-cdr that implement pairs this way. In other words, i-cons takes the arguments a and b, and returns an integer; i-car takes an integer and returns a; and i-cdr takes an integer and returns b.