Procedures which take procedures as arguments:
Assume the following definition of sum

\[
\text{sum} \left( \lambda (\text{term-proc} \ a \ \text{next-proc} \ b) \ (\text{if} \ (> \ a \ b) \\
0 \\
(+ \ (\text{term-proc} \ a) \ (\text{sum} \ \text{term-proc} \ (\text{next-proc} \ a) \ \text{next-proc} \ b))) \right)
\]

Work out what happens when we evaluate

\[
(\text{sum} \ (\lambda (x) \ (x)) \ 1 \ (\lambda (x) \ (+ \ x \ 1)) \ 10)
\]

How do we compute the sum for \( i = 1 \) to \( 10 \) of \( 1/i \)?

How do we compute the sum for \( i = \{1, 3, 5, 7, 9\} \) of \( (i^3 + 2i) \)?

Now consider another definition of sum:

\[
\text{sum} \left( \lambda (\text{term-proc} \ a \ \text{next-proc} \ \text{condition}) \ (\text{if} \ (\text{condition} \ a) \\
0 \\
(+ \ (\text{term-proc} \ a) \ (\text{sum} \ \text{term-proc} \ (\text{next-proc} \ a) \ \text{next-proc} \ \text{condition}))) \right)
\]

How do we sum the integers between 1 and 10 inclusive using this definition?

Procedures which return procedures:
write a procedure \((\text{sum-func} \ f \ g)\) which takes two procedures \( f(x) \) and \( g(x) \)
and returns a new procedure which is \( f(x) + g(x) \)

What is the type of this procedure?

Procedures which return procedures:
write a procedure \((\text{max-func} \ f \ g)\) which takes two procedures \( f(x) \) and \( g(x) \)
and returns a new procedure \( h \) which is \( h(x) = \max\{f(x), g(x)\} \)

Procedures which return procedures:
Assume we have defined ‘‘\text{compose}’’ as follows

\[
(\text{define} \ \text{compose} \ (\lambda (f \ g) \ (\lambda (x) \ (f \ (g \ x)))))
\]

Now define a new procedure \( \text{compose-n} \) which composes a function with itself \( n \) times
Assume the following definitions for "map", "accumulate" and "filter":

\[
\begin{align*}
\text{(define (map proc seq)} & \text{ (if (null? seq)} \\
& \text{ nil) } \\
& \text{ (cons (proc (car seq))} \\
& \text{ (map proc (cdr seq)))))}
\end{align*}
\]

\[
\begin{align*}
\text{(define (accumulate op init seq)} & \text{ (if (null? seq)} \\
& \text{ init) } \\
& \text{ (op (car seq) (accumulate op init (cdr seq)))))}
\end{align*}
\]

\[
\begin{align*}
\text{(define (filter pred seq)} & \text{ (if (null? seq)} \\
& \text{ nil) } \\
& \text{ (let ((rest (filter pred (cdr seq))))} \\
& \text{ (if (pred (car seq))} \\
& \text{ (cons (car seq) rest) } \\
& \text{ rest)))})
\end{align*}
\]

What are the values for the following expressions?:

\[
\begin{align*}
\text{(map (lambda (x) (- x 5)) (list 1 2 3 4))} \\
\text{(accumulate * 1 (list 1 2 3 4))} \\
\text{(filter (lambda (x) (< x 3)) (list 1 2 3 4))}
\end{align*}
\]

Now write functions using map, filter or accumulate which:

1) calculates the sum of elements in a list, e.g. (list 1 2 3 4) => 10
2) calculates the length of a list

A final (tough) question: say we define ls to be a list of *procedures*:

\[
\begin{align*}
\text{(define (square x) (* x x))} \\
\text{(define (double x) (* x 2))} \\
\text{(define (inc x) (+ x 1))}
\end{align*}
\]

\[
\begin{align*}
\text{(define ls (list square double inc))}
\end{align*}
\]

Now say we want a function "apply-procs" that behaves as follows:

\[
\begin{align*}
\text{(apply-procs ls 4)} & \Rightarrow ((\text{square 4}) (\text{double 4}) (\text{inc 4})) = (16 \ 8 \ 5) \\
\text{(apply-procs ls 3)} & \Rightarrow ((\text{square 3}) (\text{double 3}) (\text{inc 3})) = (9 \ 6 \ 4)
\end{align*}
\]

How do we achieve this using map?