Reminders, including some intermediate steps:

(map f (list a b c ...))

(list (f a) (f b) (f c) ...)

(map f (list a b c ...)
(list x y z ...))

(list (f a x) (f b y) (f c z) ...)

(apply f (list a b c ...))

(f a b c ...)

; repeated : function any number -> list
(define (repeated f a n)
  (cond [(= n 1) (list a)]
       [else (list* a (repeated f (f a) (- n 1)))]))

(repeated f a n)

(list a (f a) (f (f a)) ...) ; with n elements.

; make-string : number char -> string

(check-expect (make-string 5 #\a) "aaaaa")
**Question 1.** [7 marks]

Part (a) [4 marks]
Add the binary numbers 110110 and 101011. Carry out the addition algorithm in binary notation, showing all the steps (including carries), then convert the result to our familiar decimal (base 10).

sample solution:

\[
\begin{array}{c}
110110 \\
+ 101011 \\
\hline
\end{array}
\]

\[
\begin{array}{c}
\text{carries: 11111} \\
\hline
1100001 \\
\end{array}
\]

In decimal: 54 + 43 = 97.

Part (b) [3 marks]
The binary representation of our decimal 100 is 1100100. Use this to figure out the binary representation of 25 and the binary representation of 400. Briefly explain the connection between the three binary representations.

sample solution: The binary representation of 25 is 11001, since removing the two right-most zeros from the representation of 100 divides it by 4. The binary representation of 400 is 110010000, since by adding to more zeros on the right we multiply by 4.

**Question 2.** [6 marks]

Function sierpinski-count gives the number of triangle components for the sierpinski fractal of depth n. However, it does it unacceptably slowly beyond about (sierpinski-count 16):

\[
; \text{sierpinski-count : number -> number} \\
(\text{define (sierpinski-count n)}) \\
(\text{cond}) \\
[(= n 0) 1] \\
[else] \\
(+ (sierpinski-count (- n 1)) \\
(sierpinski-count (- n 1)) \\
(sierpinski-count (- n 1))))]]
\]
Part (a)  [3 MARKS]

Briefly explain why sierpinski-count takes so long to calculate (sierpinski-count 16).

**sample solution:** The three calls to (sierpinski-count ...) each generate three calls or 9 calls altogether, each of those generate 3 calls or 27 calls altogether, and this exponential growth in calls ends up taking a huge amount of time.

Part (b)  [3 MARKS]

Suggest a small change in the [else ...] clause of the body of sierpinski-count that would return the same result while greatly speeding up the calculation. Explain how your change helps.

**sample solution:** Replace the sum of three calls to (sierpinski-count (- n 1)) with one call multiplied by 3.

**Question 3.**  [7 MARKS]

Consider function S:

```
(require picturing-programs)

(define (S n)
  (cond
   [(= n 0) (square 10 "solid" "black")]
   [else (beside
     (S (- n 1))
     (above (S (- n 1)) (S (- n 1)))
     (S (- n 1)))]))
```

Part (a)  [3 MARKS]

Draw what is produced by (S 0) and (S 1):

```
   
```

Part (b)  [2 MARKS]

Complete the design check-expect for (S 2) using (S 1) — no drawings:

```
(check-expect (S 2)
  (beside
   (S 1)
   (above (S 1) (S 1)))
  (S 1)))
```
Part (c) [2 MARKS]
Draw the value produced by (S 2):

[Diagram of value]

Question 4. [9 MARKS]
Part (a) [4 MARKS]
Write the definition of function \( q? \):

\[
; q? : \text{string number number} \rightarrow \text{boolean} \\
; \text{Produce #true if the length of a-string is greater} \\
; \text{than x and less than y, #false otherwise.}
\]

sample solution:

\[
\begin{align*}
(\text{define } (q? \text{ a-string x y}) & \\
& (< x (\text{string-length a-string}) y))
\end{align*}
\]

Part (b) [5 MARKS]
Write the definition of function \( p? \):

\[
; p? : \text{number image image} \rightarrow \text{boolean} \\
; \text{Produce #true if width of image-1 is more than n times} \\
; \text{the width of image-2 and the height of image-1 is more} \\
; \text{than n times the height of image-2, #false otherwise}
\]

sample solution:

\[
\begin{align*}
(\text{define } (p? n \text{ image-1 image-2}) & \\
& (\text{and} \\
& (> (\text{image-width image-1}) (* n (\text{image-width image-2}))) \\
& (> (\text{image-height image-1}) (* n (\text{image-height image-2})))
\end{align*}
\]

Question 5. [6 MARKS]
Recall functions \text{list->string} and \text{string->list}:

(\text{check-expect} (\text{string->list} "abcde") (\text{list} #\a #\b #\c #\d #\e))
(\text{check-expect} (\text{list->string} (\text{list} #\a #\b #\c #\d #\e)) "abcde")

Now show the intermediate steps and result for the following expression:

\[
(\text{apply append} \\
(\text{map rest} \\
(\text{map string->list} \\
(\text{list} "winter" "of" "angst")))
)
sample solution:

(apply append
  (map rest
    (list (string->list "winter") (string->list "of") (string->list "angst"))))

(apply append
  (map rest
    (list (list #\w #\i #\n #\t #\e #\r) (list #\o #\f) (list #\a #\n #\g #\s #\t))))

(apply append
  (list (rest (list #\w #\i #\n #\t #\e #\r)) (rest (list #\o #\f))
   (rest (list #\a #\n #\g #\s #\t))))

(apply append
  (list (list #\i #\n #\t #\e #\r) (list #\f) (list #\n #\g #\s #\t)))

(append (list #\i #\n #\t #\e #\r) (list #\f) (list #\n #\g #\s #\t))

(list #\i #\n #\t #\e #\r #\f #\n #\g #\s #\t)

**Question 6.**  [9 marks]

Assume the following code has been run:

(require picturing-programs)

(define (RS n) (rectangle n 10 "solid" "black"))
(define (RO n) (rectangle n 10 "outline" "black"))

**Part (a)**  [4 marks]

Read the following check-expects carefully:

(check-expect (R 1) (beside (RS 10) (RO 10)))

(check-expect (R 2))

(check-expect (R 3))

(check-expect (R 4))

Now complete (check-expect (R 4) ...) using (R 3), RS and RO and no drawing! 

Student #: 

Page 5 of 10
sample solution:

(check-expect (R 4)
  (above
   (beside
    (R (- 4 1)) (R (- 4 1)))
   (beside (RS (image-width (R (- 4 1))))
    (RO (image-width (R (- 4 1))))))))

Part (b)  [5 MARKS]

Now define function R:

; R : number -> image
(define (R n)
  (cond
   [(= n 1) (beside (RS 10) (RO 10))]
   [else
     (above
      (beside (R (- n 1)) (R (- n 1)))
      (beside (RS (image-width (R (- n 1))))
       (RO (image-width (R (- n 1))))))])

Question 7.  [13 MARKS]

Read over the three function definitions below:

; SN : number -> string
(define (SN n)
  (make-string n #\x))
(check-expect (SN 0) "")
(check-expect (SN 1) "x")

; D : list-of-numbers -> string
(define (D pair)
  (string-append
   (SN (first pair)) "-" (SN (second pair))))

; F : list-of-numbers -> list-of-numbers
(define (F pair)
  (list (second pair)
    (+ (second pair) (first pair))))
Part (a) [3 MARKS]

Show the result of the following expressions:

\[(\text{SN 2})\]

\[(\text{D (list 2 3)})\]

\[(\text{F (list 3 5)})\]

Sample solution:

\[(\text{SN 2})\]
\"xx\"

\[(\text{D (list 2 3)})\]
\"xx-xxx\"

\[(\text{F (list 3 5)})\]
\[(\text{list 5 8})\]

Part (b) [4 MARKS]

Show the intermediate step and the result of the following expression:

\[(\text{repeated F (list 0 1) 5})\]

Sample solution:

\[(\text{repeated F (list 0 1) 5})\]

\[(\text{list (list 0 1) (list 0 1))}\]

\[\text{(F (F (list 0 1))) (F (F (list 0 1)))) (F (F (F (list 0 1))))\]

\[\text{(list (list 0 1) (list 1 1) (list 1 2) (list 2 3) (list 3 5))}\]

Part (c) [6 MARKS]

Show the intermediate step and the result of the following expression:

\[(\text{map D (repeated F (list 0 1) 5})\]

Sample solution:
Question 8. [12 MARKS]

Read the following definitions:

(define LLL
  (list (list 1 2 3) (list 4 5 6) (list 7 8 9)))

; apply+ : list-of-number -> number
(define (apply+ a-list)
  (apply + a-list))
(check-expect (apply+ (list 1 2 3)) 6)

Show the intermediate steps and result for the following expressions:

(reverse (map reverse LLL))
(length (map length LLL))
(map apply+ LLL)
(apply + (map apply+ LLL))

Sample solution:

(reverse (map reverse LLL))
(reverse (map reverse (list (list 1 2 3) (list 4 5 6) (list 7 8 9))))
(reverse (list (reverse (list 1 2 3)) (reverse (list 4 5 6)) (reverse (list 7 8 9))))
(reverse (list (list 3 2 1) (list 6 5 4) (list 9 8 7)))
(list (list 9 8 7) (list 6 5 4) (list 3 2 1))

(length (map length LLL))
(length (map length (list (list 1 2 3) (list 4 5 6) (list 7 8 9))))
(length (list (length (list 1 2 3)) (length (list 4 5 6)) (length (list 7 8 9))))
(length (list 3 3 3))
3

(map apply+ LLL)
(map apply+ (list (list 1 2 3) (list 4 5 6) (list 7 8 9)))
(list (apply+ (list 1 2 3)) (apply+ (list 4 5 6)) (apply+ (list 7 8 9)))
(list (+ 1 2 3) (+ 4 5 6) (+ 7 8 9))
(list 6 15 24)

(apply + (map apply+ LLL))
(apply + (map apply+ LLL))
(apply + (map apply+ (list (list 1 2 3) (list 4 5 6) (list 7 8 9))))
(apply + (list (apply+ (list 1 2 3)) (apply+ (list 4 5 6)) (apply+ (list 7 8 9))))
(apply + (list (+ 1 2 3) (+ 4 5 6) (+ 7 8 9)))
(apply + (list 6 15 24))
(+ 6 15 24)
45

This page has been left intentionally (mostly) blank, in case you need space.
This page has been left intentionally (mostly) blank, in case you need space.