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:

```
              110110
            + 101011
            --------
carries:     11111
            --------
             1100001
```

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)**:

```
; sierpinski-count : number -> number
(define (sierpinski-count n)
  (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)**:

Question 4. [9 MARKS]

Part (a) [4 MARKS]

Write the definition of function **q?**:

```
; q? : string number number -> boolean
; Produce #true if the length of a-string is greater
; than x and less than y, #false otherwise.
```

sample solution:
```
(define (q? a-string x y)
  (< x (string-length a-string) y))
```

Part (b) [5 MARKS]

Write the definition of function **p?**:

```
; p? : number image image -> boolean
; Produce #true if width of image-1 is more than n times
; the width of image-2 and the height of image-1 is more
; than n times the height of image-2, #false otherwise
```

sample solution:
```
(define (p? n image-1 image-2)
  (and
   (> (image-width image-1) (* n (image-width image-2)))
   (> (image-height image-1) (* n (image-height image-2)))))
```

Question 5. [6 MARKS]

Recall functions `list->string` and `string->list`:

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

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

```
(apply append
   (map rest
      (map string->list
         (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:

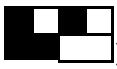
```
(require picturing-programs)

(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!**

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:

```
(SN 2)


(D (list 2 3))


(F (list 3 5))
```

sample solution:

```
(SN 2)
"xx"

(D (list 2 3))
"xx-xxx"

(F (list 3 5))
(list 5 8)
```

Part (b) [4 MARKS]

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

```
(repeated F (list 0 1) 5)
```

sample solution:

```
(repeated F (list 0 1) 5)

(list (list 0 1) (F (list 0 1))
      (F (F (list 0 1))) (F (F (F (list 0 1)))) (F (F (F (F (list 0 1))))))
(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:

```
(map D (repeated F (list 0 1) 5))
```

sample solution:

```
(map D (repeated F (list 0 1) 5))

(map D (list (list 0 1) (F (list 0 1)) (F (F (list 0 1)))
       (F (F (F (list 0 1)))) (F (F (F (F (list 0 1)))))))
(map D (list (list 0 1) (list 1 1) (list 1 2) (list 2 3) (list 3 5)))
(list (D (list 0 1)) (D (list 1 1)) (D (list 1 2)) (D (list 2 3)) (D (list 3 5)))
(list "-x" "x-x" "x-xx" "xx-xxx" "xxx-xxxxx")
```

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.

Total Marks = 69