Do not turn this page until you have received the signal to start.
(In the meantime, please fill out the identification section above.)

This test consists of 4 questions on 6 pages (including this one).
When you receive the signal to start, please make sure that your copy of the test is complete.

Good Luck!
A program is a sequence of expressions and statements.

**Expression Forms**

**Literal Value**

- function-name : function by name from a definition or from our language
- true #false : boolean
- list literal-value etc : list

**Variable Reference**

- variable-name : from a definition or from our language

**Function Call**

- (function-name argument-expression etc) : function name applied to arguments

**Statement Forms**

**Definition of a Variable or Function**

- (define variable-name value-expression) : "header" (body-expression)

**Assertion/Test**

- (same expression expression) : true! condition-expression
- (false! condition-expression) : false! condition-expression

**Reveal Algebraic Evaluation**

- (step expression) : 

**Equation/Pattern/Rule**

- For a function from a definition: copy its body and substitute the arguments in place of the parameter names wherever the names occur in the body.
- For combine or map: match its rule's first pattern to determine a b c...
- Substitute those into its rule's second pattern (the "template").
- If the expression doesn't match its pattern: an error

- (map function (list a b c etc))
- (combine function (list a b c etc))

**Goal Example**

- (same (function-name example-argument etc) literal-result)

**Full Design**

- (same (function-name example-argument etc) fully-generalizable-expression)

**Partial Design**

- (same (function-name example-argument etc) partially-generalizable-expression)

**Type Predicates**

- (true! (function? flip))
- (true! (list? (list "z") (true! (number? -12)))
- (true! (text? "Hi"))
- (true! (image? ("")))

**Function Predicates**

- (true! (unary? flip))
- (false! (binary? flip))

**List Functions**

- (same! (list (filled-triangle 9) (zero? 0) (+ 2 3) "hi") (list #true 5 "hi"))
- (same! (map - (list 3 1 7)) (list -3 -1 -7))
- (same! (combine + (list 3 1 7)) 11)
- (true! (empty? (list)))
- (same! (length (list #true 5 "hi")) 4)
- (same! (first (list #true 5 "hi")) #true 5 "hi")
- (same! (rest (list #true 5 "hi")) (list #true 5 "hi"))
- (same! (reverse (list #true 5 "hi")) (list "hi" 5 #true))

**Image Functions**

- (same! (mirror "hi") "hi")
- (same! (flip "hi") "hi")
- (same! (rotate "hi" 30) "hi")
- (same! (scale "hi" 1.5) "hi")
- (same! (shrink "hi") "hi")
- (same! (enlarge "hi") "hi")
- (same! (filled-triangle 9) (triangle 9))
- (same! (filled-circle 9) (circle 9))
- (same! (filled-square 9) (square 9))
- (same! (filled-oval 9 15) (oval 9 15))
- (same! (filled-rectangle 9 15) (rectangle 9 15))
- (same! (width (oval 9 15)) 9)
- (same! (height (oval 9 15)) 15)

- (same! (above "hi") "hi")
- (same! (beside "hi") "hi")
- (same! (align-lefts "hi") "hi")
- (same! (align-tops "hi") "hi")
- (same! (align-rights "hi") "hi")
- (same! (align-bottoms "hi") "hi")

**Numeric Functions**

- (same! (+ 2 10 3) 15)
- (same! (+ 2 10 3 60) 85)
- (true! (zero? 0))
- (true! (positive? 12))

**Text Functions**

- (same! (text-length "one") 3)
- (same! (text-join "Hi" "human!"))
- (same! (text->image "Hi")) "Hi human!")
- (same! (list "Hi" "Hi" "Hi")) (list "Hi" "Hi" "Hi")

... variable-name ...
+ literal-value ...
- Substitute the value that was computed when the variable was defined.

**Function Design**

**Goal Example**

- (same! (function-name example-argument etc) literal-result)

**Full Design**

- (same! (function-name example-argument etc) fully-generalizable-expression)

- The generalizable expression contains and only uses the argument(s) as-is.

**Partial Design**

- (same! (function-name example-argument etc) partially-generalizable-expression)

- The partially general expression is not fully generalizable, but not just literal.
: Question 1 : 12 Marks

: Assume the following definition has been entered/run:
(define x 9)

: Show the following steps:
(step (beside (wider (oval (text-length "hello friend")
24))
   (rotate (square (width (circle 10)))
5 4.5 (inc x))))
Question 2: 2 + 6 = 8 marks

Assume the following two definitions have been entered/run:

```scheme
(define (pre t)
  (text-join ",", "t))

(define (sep texts)
  (text-join (first texts)
    (combine text-join (map pre (rest texts)))))
```

Show the following steps:

```scheme
(step (pre "pear"))

(step (sep (list "pear" "plum" "kiwi")))
```
Question 3: 5 + 5 + 5 = 15 Marks

For each of the following function descriptions ...

A. For each test/assertion, write down which kind of assertion/test it is:
   - Example,
   - Partial Design, or
   - Full Design

B. Write another correct assertion/test for the function (it can be any kind).

C. Define the function.

- age : for a list with a birth year, surname, and given name, produce the age based on the birth year

A. The following is a/an ______________________ for the function.
(same! (age (list 1912 "Turing" "Alan") 107); Alan Turing was born in 1912, so would be 107 this year.

A. The following is a/an ______________________ for the function.
(same! (age (list 1975 "Furler" "Sia") 44); Sia will be 44 this year.

A. The following is a/an ______________________ for the function.
(same! (age (list 1975 "Furler" "Sia")
         (- 2019 1975))

B.

C. (define (age person)

- versus : from two texts, form a question asking which one

A. The following is a/an ______________________ for the function.
(same! (versus "cat" "dog") "cat or dog?")

B.

C.

- has-zero? : for a list of numbers, determine whether at least one element is zero

A. The following is a/an ______________________ for the function.
(same! (has-zero? (list 1 0 4)) #true)

A. The following is a/an ______________________ for the function.
(same! (has-zero? (range -5 -2)) (zero? (combine * (range -5 -2))))

B.

C.
Question 4: $2 + 2 + 2 + 2 + 3 = 11$ Marks

Design and implement a function point by doing the tasks indicated with a "✓" below.

```
(same! (point △) △)  (same! (point •) •)
```

DO NOT draw any images by hand — use the following variable pan instead of drawing the panda.

```
(define pan 🐼)
```

```
(same! (width △) (height pan))  (same! (height △) (height pan))
```

✓ Write an expression, using variable pan, to produce that triangle △:

✓ Write an expression, using variable pan, to produce that triangle △:

✓ Write an expression, using variable pan, to produce that triangle △:

✓ Write an expression, using variable pan, to produce the image 🐼:

✓ Fill in the body of point:

```
(define (point an-image)
```

✓ Show the following steps:

```
(step (point (rectangle 10 20)))
```