```
; Homework 2, Problem 1
; Students' name here

; Step 1: contract, purpose, & header
; Area : number -> number
; Purpose: takes the radius of a circle as input & produces the circle's
area as output
;(define (Area r) ...)

; Step 2: test data
;   Input      |  Result
;     0        |    0
;     1        |   3.14159 ... (pi)
;    10        |   314.159 ... (100 pi)

; Step 3: write the expression needed for the code body
;         (informed by the worked examples in step 2)
;   (* pi r r )

; Step 4: put it all together to get the program

; Area : number -> number
; Purpose: takes the radius of a circle as input & produces the circle's
area as output
(define (Area r)
  (* pi r r))
```

Welcome to DrScheme, version 208p1.
Language: Beginning Student.

```
> ; Homework 2, Problem 1
> (Area 0)
0
> (Area 1)
#i3.141592653589793
> (Area 2)
#i12.566370614359172
>
```

```
; Homework 2, Problem 2
; Student's name

; Step 1: contract, purpose, & header
; Washer: number number -> number
; Purpose: Given an outside radius and an inside radius
;          for a washer, compute its area
; (define (Washer outside inside) ...)

; Step 2: test data
;   Input            | Result
;      0 0           | 0
;      1 1           | 0
;      2 1           | 9.42477796... (3 pi)

; Step 3: write an expression for the body of the function
;
;   Two possibilities come to mind:
;     - reuse the Area computation from problem 1, which yields
;       (- (Area outside) (Area inside))
;
;     - compute it all explicitly
;       (- (* pi outside outside) (* pi inside inside))
;
;   We prefer the first solution, since it reuses code, so here
;      is the code, cut and pasted, from problem 1

; Area : number -> number
; Purpose: takes the radius of a circle as input & produces the circle's
area as output
(define (Area r)
  (* pi r r))

; Washer: number number -> number
; Purpose: Given an outside radius and an inside radius
;          for a washer, compute its area
(define (Washer outside inside)
   (- (Area outside) (Area inside)))
```

Welcome to DrScheme, version 208p1.
Language: Beginning Student.

```
> ; Homework 2, Problem 2
> (Washer 0 0)
0
> (Washer 1 1)
#i0.0
> (Washer 2 1)
#i9.42477796076938
>
```

```
; Homework 2, Problem 3
; Student's name

; Step 1: contract, purpose, & header
; Cylinder: number number -> number
; Purpose: Given a radius and a height, compute the area of the right
;          circular cylinder that those dimensions specify
; (define (Cylinder radius height) ...)

; Step 2: test data
;   Input            | Result
;      0 0           | 0
;      0 1 (or 1 0)  | 0
;      1 1           | 3.121159
;      2 1           | 12.56637... (4 pi)

; Step 3: write an expression for the body of the function
;
;   Two possibilities come to mind:
;     - reuse the Area computation from problem 1, which yields
;       (* (Area radius) height))
;
;     - compute it all explicitly
;       (* (* pi radius radius) height)
;
;   We prefer the first solution, since it reuses code, so here
;      is the code, cut and pasted, from problem 1

; Area : number -> number
; Purpose: takes the radius of a circle as input & produces the circle's
area as output
(define (Area r)
  (* pi r r))


; Cylinder : number number -> number
; Purpose: Given a radius and a height, compute the area of the right
;          circular cylinder that those dimensions specify
(define (Cylinder radius height)
     (* (Area radius) height))
```

Welcome to DrScheme, version 208p1.
Language: Beginning Student.

```
> (Cylinder 0 0)
0
> (Cylinder 0 1)
0
> (Cylinder 1 0)
0
> (Cylinder 1 1)
#i3.141592653589793
> (Cylinder 2 1)
#i12.566370614359172
>
```

```
; Homework 2, Problem 4
; Students' names

; Step 1: contract, purpose, and header
; Taxes : number -> number
; Purpose: compute a graduated tax on the income given as input
;          and return the amount of tax due
; (define (Taxes amt) ...)

; Step 2: test data
;
; The tricky part of this problem is that the income tax is incremental,
; that is, the tax on 7,001 is 7000 * 0.10 + 1 * 0.15.  Thus, the
computation
; cannot simply multiply the rate times the amount.

; Input      | Result            This takes some calculation
;  0         | 0
;  1         | 0.10
;  7000      | 700
;  7001      | 700.15
;  28400     | 3910
;  28401     | 3910.25
;  68800     | 1410
;  68801     | 1410.28
;  143500    | 34926
;  143501    | 34926.33
;  311950    | 90514.5
;  311951    | 90514.85

; Taxes : number -> number
; Purpose: compute a graduated tax on the income given as input
;          and return the amount of tax due
(define (Taxes amt)
  (cond
    ((<= amt 0) 0)
    ((<= amt 7000)    (* 0.1 amt))
    ((<= amt 28400)   (+ 700  (* (- amt 7000) 0.15)))
    ((<= amt 68800)   (+ 3910 (* (- amt 28400) 0.25)))
    ((<= amt 143500)  (+ 14010 (* (- amt 68800) 0.28)))
    ((<= amt 311950)  (+ 34926 (* (- amt 143500) 0.33)))
    ((>  amt 311950)  (+ 90514.5 (* (- amt 311950) 0.35)))
    ))
```

Welcome to DrScheme, version 208p1.
Language: Beginning Student.

```
> (Taxes 0)
0
> (Taxes 1)
0.1
> (Taxes 7000)
700
> (Taxes 701)
70.1
> (Taxes 7001)
700.15
> (Taxes 28400)
3910
> (Taxes 28401)
3910.25
> (Taxes 68800)
14010
> (Taxes 68801)
14010.28
> (Taxes 143500)
34926
> (Taxes 143501)
34926.33
> (Taxes 311950)
90514.5
> (Taxes 311951)
90514.85
> (Taxes -1)
0
>
```

```
; Homework 2, Part 5
; Students' names

; Data analysis

; a point is a structure
;  (make-point x y)
; where x and y are numbers
(define-struct point (x y))

; to simplify the examples
(define origin (make-point 0 0))

; contract, purpose, & header

; midpoint: point point -> point
; Purpose: find the midpoint of a line determined by the two input points
; (define (midpoint p1 p2) ...)


; Test data
;   Input                  | Result
; origin origin            | origin
; origin (make-point 1 1)  | (make-point 0.5 0.5)

; Derive an expression from the data and problem statement
;  (make-point (/ (+ (point-x p1) (point-x p2)) 2)
;              (/ (+ (point-y p1) (point-y p2)) 2))

; midpoint: point point -> point
; Purpose: find the midpoint of a line determined by the two input points
(define (midpoint p1 p2)
  (make-point (/ (+ (point-x p1) (point-x p2)) 2)
              (/ (+ (point-y p1) (point-y p2)) 2)))
```

Welcome to DrScheme, version 208p1.
Language: Beginning Student.

```
> (midpoint origin origin)
(make-point 0 0)
> (midpoint origin (make-point 1 1))
(make-point 0.5 0.5)
>
```