

# Fundamentos de programación

## Procesamiento de datos simples II

Facultad de Ingeniería. Universidad del Valle

Agosto de 2018

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

- 1 Información simbólica
- 2 Estructuras
- 3 Diseño de funciones con estructuras
- 4 Estructuras dentro de estructuras
- 5 Validación de datos
- 6 Dibujar en Dr Racket

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

1 Información simbólica

2 Estructuras

3 Diseño de funciones con estructuras

4 Estructuras dentro de estructuras

5 Validación de datos

6 Dibujar en Dr Racket

"holo"    'holo x    "holo mundo"

## Definición

La información simbólica se refiere a palabras, direcciones o imágenes. Un símbolo es una secuencia de caracteres de teclado precedido por una comilla

```
'the 'dog 'ate 'a 'chocolate 'cat!
```

## Definición

Los símbolos al igual que los números son datos, por lo que se pueden realizar operaciones con ellos. La función **Symbol?** nos permite comparar dos símbolos.

```
(symbol=? 'Hello 'Hello)  
(symbol=? 'Hello 'hello)
```

Number?

## Definición

Un ejemplo es la siguiente función:

```
;; Autor:  Docente curso Fundamentos de programación
;; Fecha de creación: 11-Agosto-2018
;; Contrato: responder-saludo: simbolo -> simbolo
;; Propósito: Función para responder un saludo
;; Ejemplo: (responder-saludo 'Hola) -> 'Hola
;; Ejemplo: (responder-saludo 'Hi) -> 'RepiteLoDicho
;; Definición

(define (responder-saludo sim)
  (cond
    [(symbol=? sim 'Hola) 'Hola]
    [(symbol=? sim 'ComoEstas) 'BienYTU?]
    [(symbol=? sim 'HolaQUeHace) 'NadaYTU]
    [else 'RepiteLoDicho]
  )
)
(check-expect (responder-saludo 'Hola) 'Hola)
(check-expect (responder-saludo 'Hi) 'RepiteLoDicho)
```

## Cadenas de texto

Las cadenas de texto es una segunda forma de datos . A diferencia de la forma anterior estos permiten espacios.

```
"the dog"  
"mode of"  
"chocolate"
```

## Ejercicio en clase

Desarrolle una función **chequear-diferencia** que:

- Reciba como entrada dos números
- Si los números son iguales se retorna 'iguales
- Si la diferencia entre estos números es mayor que 0 y menor o igual que 10, se retorna 'pequeño
- Si la diferencia entre estos dos números es mayor que 10, se retorna 'grande

abs



```
;; Autor:  Docente curso Fundamentos de programación
;; Fecha de creación: 11-Agosto-2018
;; Contrato: chequear-diferencia: numero,numero -> simbolo
;; Propósito: Esta función recibe dos números y de acuerdo a su diferencia
              retorna un símbolo
;; Ejemplo: (chequear-diferencia 1000 200) -> 'grande
;; Ejemplo: (chequear-diferencia 1000 1000) -> 'iguales
;; Definición
(define (chequear-diferencia numA numB)
  (cond
    [(= numA numB) 'iguales ]
    [(and (> (abs (- numA numB)) 0) (<= (abs (- numA numB)) 10)) 'pequeño]
    [else 'grande]
  )
)
;; Pruebas
(check-expect (chequear-diferencia 1000 200) 'grande)
(check-expect (chequear-diferencia 1000 1000) 'iguales)
(check-expect (chequear-diferencia 1 10) 'pequeño)
```

## Ejercicio en clase

- 1 chequear-ecuacion**, recibe tres números  $a$ ,  $b$  y  $c$ . Estos corresponden a los parámetros de una ecuación cuadrática  $ax^2 + bx + c = 0$ , determine si la ecuación tiene 'una' dos o 'noTiene raíces reales.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\begin{array}{ll} b^2 - 4ac = 0 & \text{'una} \\ b^2 - 4ac > 0 & \text{'dos} \\ b^2 - 4ac < 0 & \text{'noTiene} \end{array}$$

```
;; Autor: Docente curso Fundamentos de programación
;; Fecha de creación: 11-Agosto-2018
;; Contrato: chequear-diferencia: numero,numero,numero ->
           simbolo
;; Propósito: Esta función recibe tres números que
           corresponde a una ecuación cuadrática, se determina
           si tiene 0, 1 o 2 raíces reales.
;; Ejemplo: (chequear-ecuacion 1 2 1) -> 'una
;; Ejemplo: (chequear-ecuacion 1 2 4) -> 'noTiene
;; Ejemplo: (chequear-ecuacion 1 4 2) -> 'dos
;; Definición
(define (chequear-ecuacion a b c)
  ;;Escribe aquí la función
)
```

Se debe chequear la condición de  $\sqrt{b^2 - 4ac}$ , si

- Si  $b^2 - 4ac$  es 0 se tiene una sola raíz real repetida
- Si  $b^2 - 4ac$  es menor que 0, no se tienen raíces reales
- En otro caso se tienen dos raíces reales diferentes

```
;; Autor: Docente curso Fundamentos de programación
;; Fecha de creación: 11-Agosto-2018
;; Contrato: chequear-diferencia: numero,numero,numero -> simbolo
;; Propósito: Esta función recibe tres números que corresponde a una ecuación
               cuadrática, se determina si tiene 0, 1 o 2 raíces reales.
;; Ejemplo: (chequear-ecuacion 1 2 1) -> 'una
;; Ejemplo: (chequear-ecuacion 1 2 4) -> 'noTiene
;; Ejemplo: (chequear-ecuacion 1 4 2) -> 'dos
;; Definición
(define (chequear-ecuacion a b c)
  ;;Escribe aquí la función
)

;;Función auxiliar
;;Autor: Docente curso Fundamentos de programación
;;Contrato: número, , numero -> numero
;;Propósito: Chequear la condición de la raíz de la ecuación cuadrática
;;Ejemplo (chequear-raiz 1 2 1) -> 0
(define (chequear-raiz a b c)
  (- (sqr b) (* 4 a c))
)
;;Pruebas
(check-expect (chequear-raiz 1 2 1) 0)
```

```
;; Autor:  Docente curso Fundamentos de programación
;; Fecha de creación: 11-Agosto-2018
;; Contrato: chequear-diferencia: numero,numero,numero -> simbolo
;; Propósito: Esta función recibe tres números que corresponde a una ecuación
              cuadrática, se determina si tiene 0, 1 o 2 raíces reales.
;; Ejemplo: (chequear-ecuacion 1 2 1) -> 'una
;; Ejemplo: (chequear-ecuacion 1 2 4) -> 'noTiene
;; Ejemplo: (chequear-ecuacion 1 4 2) -> 'dos
;; Definición
(define (chequear-ecuacion a b c)
  (cond
    [(= (chequear-raiz a b c) 0) 'una]
    [(< (chequear-raiz a b c) 0) 'noTiene]
    [else 'dos]
  )
)
;; Pruebas
(check-expect (chequear-ecuacion 1 2 1) 'una)
(check-expect (chequear-ecuacion 1 2 4) 'noTiene)
(check-expect (chequear-ecuacion 1 4 2) 'dos)
```

Desarrolle la función chequear-numeros, esta recibe dos números. Dependiendo de la diferencia (prográmalo a tu gusto) se retorna 'Pequeña, 'Iguales o 'Grande . Después de este proceso agrega el paquete de enseñanza guess.rkt en el menú Lenguaje y da clic en ejecutar. Después ejecuta lo siguiente:

```
( guess-with-gui chequear-numeros )
```

**Pista:** El juego trabaja comparando el número que ingresa con un número aleatorio entre 0 y 99999, por lo que tenga esto en cuenta para la programación de su función.

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

- 1 Información simbólica

2 Estructuras

3 Diseño de funciones con estructuras

4 Estructuras dentro de estructuras

5 Validación de datos

6 Dibujar en Dr Racket



Simbólica

## Definición

La entrada de una función no siempre es un tipo de dato simple: número, booleano o texto, si no que puede ser una estructura que tenga más información. Por ejemplo, usted puede tener una función que reciba la información de un carro (marca, modelo, precio) y la procese.

## Definición

Un caso particular es la representación de puntos de un plano cartesiano en nuestro computador. Para ubicar un punto se requieren las coordenadas  $x$  e  $y$ : Para esto tenemos la siguiente estructura.

```
(make-posn 2 3)
(make-posn 1 2)
(make-posn 3 2)
```

De esta forma podemos representar puntos de un plano cartesiano.

## Definición

Estas estructuras se pueden acceder de la siguiente forma:

```
(define p1 (make-posn 2 3))  
(define p2 (make-posn 3 2))  
(posn-x p1)  
(posn-y p2)
```

## Ejemplo

Se desea desarrollar una función que calcule la distancia entre dos puntos  $(x_0, y_0)$  y  $(x_1, y_1)$  cualquiera. Para este problema requiere conocer la formula de distancia entre dos puntos:

$$\sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2} \quad (1)$$

## Ejemplo

```
;; Autor: Docente fundamentos de programación
;; Fecha de creación: 11 de Agosto de 2018
;; Contrato: calcularDistancia: posn, posn -> numero
;; Propósito: Este programa recibe dos puntos y retorna la distancia entre
              ellos
;; Ejemplo: (calcularDistancia (make-posn 1 2) (make-posn 2 3)) -> 1.41...
;;           (calcularDistancia (make-posn 5 2) (make-posn 2 9)) -> 7.65...
;; Definición
(define (calcularDistancia p1 p2)
  (sqrt (+
    (sqr (- (posn-x p1) (posn-x p2)))
    (sqr (- (posn-y p1) (posn-y p2)))
  )
)
)
;; Pruebas
(calcularDistancia (make-posn 1 2) (make-posn 2 3))
(calcularDistancia (make-posn 5 2) (make-posn 2 9))
;; No se utiliza check-expect, ya que se tienen números irracionales y no se
   puede comparar.
;; En DrRacket los números irracionales se representa como #i<numero>
```

## Ejercicio

Se desea desarrollar una función que reciba 3 puntos  $p_0 = (x_0, y_0)$ ,  $p_1 = (x_1, y_1)$  y  $p_2 = (x_2, y_2)$  cualquiera. Se desea conocer cual de los dos puntos  $p_1$  o  $p_2$  está más cerca de  $p_0$

**Corregir para indicar que se retorna un punto**

<https://pastebin.com/K5ucYX8F>

Carro

- Marca
- Color
- Modelo
- Velocidad

Crear una variable por cada atributo del carro.

3 Carros necesitarían  
12 variables!!!!

## Definición

Como hemos visto anteriormente las estructuras nos permiten manejar datos un poco más complejos. En este caso hemos explorado la estructura **posn**, que nos provee la representación de un punto en 2 dimensiones. Ahora, podremos definir nuestras propias estructuras así:


```
(define-struct  
  <nombre-estructura>  
  (atributo1 atributo2 ... atributoN)  
)
```



## Definición

Para el caso de **posn** la estructura se define así:

```
(define-struct
  posn
  (x y)
)
```



Para acceder a los campos debemos utilizar **posn-x** y **posn-y**

(make-posn 5 4) ← creación  
 nombre nombre x y  
 (posn-x ...) ← atributo

## Definición

Ahora, intentemos realizar nuestra primera estructura. Se desea almacenar el nombre de una persona, su número de teléfono y email.

```
(define-struct  
  datos-persona  
  (nombre telefono email)  
)
```

## Definición

Para crear datos de esa estructura utilizamos  
**make-datos-persona**

```
(define persona1 (make-datos-persona 'Pedro 1254545 'pedro@gmail.com))  
(define persona2 (make-datos-persona 'Mario 1254545 'mario@hotmail.com))  
(define persona3 (make-datos-persona 'Ana 1254545 'ana@latinmail.com))
```

¿Como podríamos acceder al los datos de esa estructura?

## Definición

Los selectores permiten acceder a la información. Su formato es **estructura-campo**

```
(datos-persona-nombre persona1)  
(datos-persona-telefono persona2)  
(datos-persona-email persona3)
```

¿Que observa?

## Ejercicios en clase

Considere las siguiente estructuras

- 1 (define-struct película (titulo productor))
- 2 (define-struct amigo (nombre edad color-pelo color-ojos))
- 3 (define-struct carro (marca color costo))
- 4 (define-struct anime (nombre genero))

Defina 3 estructuras de cada una (almacenarlas en una variable) y consulte sus campos.

## Ejercicios en clase

```
(define-struct pelicula (titulo productor))
```

```
(define-struct pelicula (titulo productor))  
;; Crear  
(define peliculaA (make-pelicula "Buscando a Dory" "  
    Disney"))  
(define peliculaB (make-pelicula "Civil Wars" "Marvel"))  
(define peliculaC (make-pelicula "Martha vs Martha" "DC  
    Comics"))  
;; Consultar  
(pelicula-titulo peliculaA)  
(pelicula-titulo peliculaB)  
(pelicula-productor peliculaC)
```

## Ejercicios en clase

(define-struct amigo (nombre edad color-pelo color-ojos))

```
(define-struct amigo (nombre edad color-pelo color-ojos))  
;; Crear  
(define amigoA (make-amigo "Pedro" 10 "Verde" "Azul"))  
(define amigoB (make-amigo "Martha" 12 "Azul" "Verde"))  
(define amigoC (make-amigo "Pablo" 80 "Violeta" "Violeta")  
))  
;; Consultar  
(amigo-nombre amigoA)  
(amigo-edad amigoB)  
(amigo-color-pelo amigoC)
```

## Definición

Para preguntar si un tipo de dato es una estructura, se utiliza el nombre de la estructura seguido de un ?

```
(define-struct
  datos-persona
  (nombre apellidos email)
)

(define juan (make-datos-persona "Juan" "Perez" "
  juan@gmail.com"))

(datos-persona? juan)
(datos-persona? 2)
```



## 1) Datos complejos

1 Información simbólica

2 Estructuras

3 Diseño de funciones con estructuras

4 Estructuras dentro de estructuras

5 Validación de datos

6 Dibujar en Dr Racket

→ (define-struct  
    <nombre>  
    (component  
      ...  
      component))

(make-<nombre>

    —  
    component)

(<nombre> -<component> 1)

fix

## Definición

Para el diseño de funciones que trabajen con estructuras debe tenerse en cuenta:

- 1 El contrato debe considerar las estructuras como entrada o salida de una función
- 2 Debe tener en cuenta que campos contiene la estructura que se desea trabajar
- 3 Utilizar `make-estructura` para crear una estructura
- 4 Para acceder a la información usar `estructura-campo`

<https://pastebin.com/SntvuR9h>

## Ejemplo

Usando la estructura datos-persona diseñe una función llamada actualizar-datos que:

- 1 Reciba un número, una estructura datos-persona y un símbolo. *Salida dato - persona*
- 2 Si el número es 1, cambia el nombre de la persona por el símbolo ingresado.
- 3 Si el número es 2, cambia el apellido de la persona por el símbolo ingresado.
- 4 Si el número es 3, cambia el correo de la persona por el símbolo ingresado.
- 5 La función retorna la información de la persona actualizada

Tenemos un amigo, el cual tiene los siguientes atributos, nombre: simbolo, edad: numero, altura: numero (cm)

Diseñe una función actualizar-amigo  
Esta función recibe amigo, num(a), num(b), simbolo(sym)

Si  $a = 1$  Se actualiza el nombre por sym

Si  $a = 2$  se actualiza la edad por b

En otro caso se actualiza la altura por b

# Diseño de funciones con estructuras

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

## Ejemplo

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 11-Agosto-2018
;;Contrato: actualizar-datos: numero, datos-persona, simbolo ->
           datos-persona
;;Propósito: Permite actualizar los datos de una persona contenidos en una
           estructura datos-persona
;;Ejemplo: Dado (define persona (make-datos-persona 'Pedro 'Perez '
           Perez@hotmail.com))
;; (actualizar-datos 1 persona 'Jaime) --> (make-datos-persona 'Jaime 'Perez
           'Perez@hotmail.com)
;; (actualizar-datos 2 persona 'Hernandez) --> (make-datos-persona 'Pedro '
           Hernandez 'Perez@hotmail.com)
;; (actualizar-datos 3 persona 'PedroElPro@hotmail.com) --> (
           make-datos-persona 'Pedro 'Perez 'PedroElPro@hotmail.com)

(define actualizar-datos (opcion persona datoNuevo)
  ;;Escribe tu función aquí
)
```

## Ejemplo

### Completa el ejemplo

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 11-Agosto-2018
;;Contrato: actualizar-datos: numero, datos-persona, simbolo ->
           datos-persona
;;Propósito: Permite actualizar los datos de una persona contenidos en una
           estructura datos-persona
;;Ejemplo: ....

(define (actualizar-datos opcion persona datoNuevo)
  (cond
    [(= opcion 1) (make-datos-persona datoNuevo
                                       (datos-persona-apellidos persona)
                                       (datos-persona-email persona)
                                       )]
    [(= opcion 2) ;;Completa el código]
    [else ;;Completa el código]
  )
)
;;Pruebas (usa check-expect)
```

## Ejemplo

Usando las siguientes definiciones de estructuras:

```
(define-struct movie (title producer))  
(define-struct boyfriend (name hair eyes phone))  
(define-struct CD (artist title price))  
(define-struct sweater (material size producer))
```

El código lo encuentras aquí:

<https://pastebin.com/zizfb8QU>

## Ejemplo

- Diseña una función **crear-pelicula** que reciba dos símbolos que corresponden al titulo y productor de una película, se retorna una estructura **movie**
- Diseña una función **cambiar-nombre-novio** que recibe una estructura **boyfriend** y un símbolo. Esta retorna la misma estructura **boyfriend** pero con el campo **name** cambiado por el símbolo ingresado.

Recuerda la receta de diseño. El código lo encuentras aquí:  
<https://pastebin.com/zizfb8QU>



## Ejemplo

- Genera una función **cambiar-artista** que recibe un símbolo y una estructura **CD**. Retorna la misma estructura, pero con el campo **artist** cambiado por el símbolo ingresado.
- Genera una función **mejor-sweater** que reciba dos estructuras **sweater**, esta retorna la estructura **sweater** que tenga un valor **size** mayor. En la entrada el valor del campo **size** de ambas estructuras nunca es igual.

Recuerda la receta de diseño. El código lo encuentras aquí:  
<https://pastebin.com/zizfb8QU>

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

1 Información simbólica

2 Estructuras

3 Diseño de funciones con estructuras

4 Estructuras dentro de estructuras

5 Validación de datos

6 Dibujar en Dr Racket

## Estructuras dentro de estructuras

Dentro de una estructura podemos tener otras estructuras, observemos el siguiente ejemplo:

- 1 Una habitación tiene: una cama, estantería y una lampara
- 2 Una cama tiene un largo y un ancho
- 3 Una estantería tiene un nombre y una descripción
- 4 Una lampara tiene una marca, un alto y un ancho

## Estructuras dentro de estructuras

### Definimos las estructuras

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 11-Agosto-2018
;;Propósito: Estructura dentro de estructuras, definición de estructuras

(define-struct habitacion (cama estanteria lampara))

(define-struct cama (largo ancho))

(define-struct estanteria (nombre descripcion))

(define-struct lampara (marca alto ancho))
```

## Estructuras dentro de estructuras

Podemos definir algunas habitaciones

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 11-Agosto-2018
;;Propósito: Estructura dentro de estructuras, ejemplos

(define camaA (make-cama 10 20))
(define camaB (make-cama 30 50))

(define estanteriaA (make-estanteria "Armario" "Grande"))
(define estanteriaB (make-estanteria "Mueble" "Dorado"))

(define lamparaA (make-lampara "Sony" 10 20))
(define lamparaB (make-lampara "Toshiba" 20 30))

(define habitacionA (make-habitacion camaA estanteriaB lamparaB))
(define habitacionB (make-habitacion camaB estanteriaA lamparaA))
```

- X Hasta aquí

## Estructuras dentro de estructuras

Ahora vamos a desarrollar una función llamada **revisar-cama** que reciba una habitación y nos indique si la cama es grande o pequeña. La cama es pequeña si tiene 10 o menos de largo y es grande en otro caso.

¿Que necesitamos?

## Estructuras dentro de estructuras

### Podemos definir algunas habitaciones

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 11-Agosto-2018
;;Contrato:  revisar-cama: habitacion -> cadenaDeTexto
;;Propósito: Indicar si una cama es pequeña o es grande
;;Ejemplos: (revisar-cama habitacionA) -> "pequeña"
;;           (revisar-cama habitacionB) -> "grande"
;;           (revisar-cama 'hola) -> "error"
(define (revisar-cama hab)
  ;;Escribe tu función aquí
)
```

## Estructuras dentro de estructuras

Primero verificamos si la entrada es una estructura tipo habitación

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 11-Agosto-2018
;;Contrato:  revisar-cama: habitacion -> cadenaDeTexto
;;Propósito: Indicar si una cama es pequeña o es grande
;;Ejemplos: ...
(define (revisar-cama hab)
  (if
    (habitacion? hab)
    (cond
      [(<= (cama-largo (habitacion-cama hab)) 10) "pequeña"]
      [else "grande"]
    )
    "error"
  )
)

(check-expect (revisar-cama habitacionA) "pequeña")
(check-expect (revisar-cama habitacionB) "grande")
(check-expect (revisar-cama 5) "error")
```



## Ejercicio

- 1 Construya una estructura llamada **avion** que provea la información de un avión de combate, los datos requeridos son: aceleración (numérico), velocidad máxima (numérico) y rango (estructura rango).
- 2 Construya una estructura para el rango del avión llamada **rango-avion**, este tiene tres campos: alcance mínimo, alcance máximo y precisión.
- 3 Construya 3 aviones y consulte sus datos.

¡Dentro de estructuras podemos tener otras estructuras!

## Ejercicio

Para apoyarse en este trabajo los siguiente debe funcionar:

```
;;Definiciones
(define rangoAvionA (make-rango-avion 400 600 80))
(define avionA (make-avion 100 200 rangoAvionA))
;;Consultas
(avion-rango avionA)
→ (make-avion 100 200 rangoAvionA)
(avion-aceleracion avionA)
→ 100
(rango-avion-alcance-minimo (avion-rango avionA))
→ 400
```

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

1 Información simbólica

2 Estructuras

3 Diseño de funciones con estructuras

4 Estructuras dentro de estructuras

5 Validación de datos

6 Dibujar en Dr Racket

## Introducción

En el diseño de funciones, no estamos libres de que el usuario incumpla el contrato enviando entradas no esperadas, para esto contamos con las funciones denominadas como **predicados**. Estas funciones validan si un tipo de dato es el esperado

```
;; f : Scheme-value -> ???  
(define (f v)  
  (cond  
    [(number? v) ...]  
    [(boolean? v) ...]  
    [(symbol? v) ...]  
    [(struct? v) ...]))
```

Estas funciones suelen terminar en ?

## Validación estructuras de datos

Para el caso de estructuras que hemos construido, **define-struct** crea automáticamente una función **nombre-estructura?** sobre la cual puedes validar si un dato es una estructura dada:

```
(define-struct
  datos-persona
  (nombre apellidos email)
)

(define juan (make-datos-persona "Juan" "Perez" "
  juan@gmail.com"))

(datos-persona? juan)
(datos-persona? 2)
```

Este ejemplo lo vimos anteriormente.

## Ejemplo

Para el caso de estructuras que hemos construido, **define-struct** crea automáticamente una función **nombre-estructura?** sobre la cual puedes validar si un dato es una estructura dada:

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 2-Septiembre-2018
;;Contrato: area-cuadrado numero-> numero
;;Propósito: Calcula el área de un cuadrado a partir del valor de su lado
(define (area-cuadrado l)
  (cond
    [(number? l) (sqr l)]
    [(boolean? l) (error 'area-cuadrado "l es un número")]
    [(symbol? l) (error 'area-cuadrado "l es un número")]
    [(struct? l) (error 'area-cuadrado "l es un número")]))
```

Sin embargo esto lo podemos simplificar.

## Ejemplo

Basta con validar el dato que esperamos, en otro caso debe indicarse un mensaje de error.

```
;;Autor: Docente del curso fundamentos de programación
;;Fecha: 2-Septiembre-2018
;;Contrato: area-cuadrado numero-> numero
;;Propósito: Calcula el área de un cuadrado a partir del
              valor de su lado
(define (area-cuadrado l)
  (cond
    [(number? l) (sqr l)]
    [else (error 'area-cuadrado "l es un número")]))
```

## Ejercicio

Tomando en cuenta el siguiente contrato, diseñe una función que realice las validaciones correspondientes.

```
;; Estructura usada: (define-struct producto (nombre, codigo, precio))  
;; Autor: <Su nombre>  
;; Fecha: <Fecha de hoy>  
;; Contrato: incremento-precio: producto, numero -> producto  
;; Propósito: Esta función recibe un producto y un número, retorna el mismo  
               producto pero con el campo precio sumado con el número recibido. Si el  
               incremento es mayor que 500, se asume solo el incremento como precio.  
;; Ejemplos (incremento-precio (make-producto "Gelatina" "10012" 100) 100)  
               -> (make-producto "Gelatina" "10012" 200)  
;; (incremento-precio (make-producto "Gelatina" "10012" 100) 1000) -> (  
               make-producto "Gelatina" "10012" 1000)  
(define (incremento-precio prod inc)  
  ;; Tu código aquí.  
)
```



Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

1 Información simbólica

2 Estructuras

3 Diseño de funciones con estructuras

4 Estructuras dentro de estructuras

5 Validación de datos

6 Dibujar en Dr Racket

## Definición

Algo interesante que podemos realizar en el Dr Racket son dibujos. Este proceso nos va dar apoyo en nuestro proceso de aprendizaje de la programación, pero esta vez orientado a crear dibujos de acuerdo a nuestros deseos.

## Paso previo

Antes de seguir, debemos agregar un paquete de enseñanza. Para esto deben dirigirse al menú Lenguaje / Añadir paquete de enseñanza y allí añadir **draw.rkt**

## Iniciemos

Para iniciar debemos definir nuestro lienzo de trabajo, para esto ejecuten:

```
;;A continuación vamos a definir un lienzo o canvas de  
500 por 500  
(start 500 500)
```

El lienzo inicia en la parte superior izquierda (posición 0 0) y termina en la parte inferior derecha (posición 500 500)

## Iniciemos

La estructura **posn** que vimos anteriormente será muy útil en estos momentos

```
;;A continuación vamos a definir una linea  
;;Esta inicia en posición (0 0) y termina en la posición  
   (250 250)  
(draw-solid-line (make-posn 0 0) (make-posn 250 250))
```

¿Que observa?

## Iniciemos

```
;;A continuación vamos a definir un rectángulo de Ancho  
70 y altura 200 y de color azul  
;;La parte superior izquierda del rectángulo está en la  
posición (100 50)  
(draw-solid-rect (make-posn 100 50) 70 200 'blue)
```

¿Que observa?

## Iniciemos

```
;;A continuación vamos a definir un círculo de radio 80  
   cuya línea es de color amarillo  
;;El círculo está centrado en el punto 250 250)  
(draw-circle (make-posn 250 250) 80 'yellow)
```

¿Que observa?

## Iniciemos

```
;;A continuación vamos a definir un círculo de radio 60  
    relleno con color verde  
;;El círculo está centrado en el punto 100 300)  
(draw-solid-disk (make-posn 100 300) 60 'green)
```

¿Que observa?

## Iniciemos

Para terminar el programa.

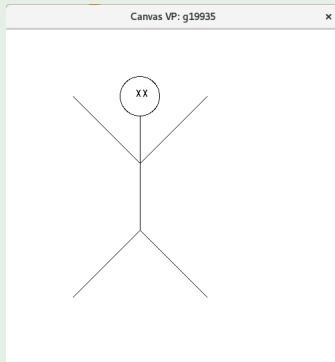
```
;;Cerrar el lienzo (canvas)  
(stop)
```

¿Que observa?



## Iniciemos

Les pongo un reto, dibujar un ser humano.



## Iniciemos

### Dibujar la cabeza y los ojos

```
;;Iniciemos el canvas  
(start 500 500)  
;;Dibujemos la cabeza  
(draw-circle (make-posn 200 100) 30 'black)  
;;Dibujemos los ojos  
(draw-solid-line (make-posn 205 90) (make-posn 210 100))  
(draw-solid-line (make-posn 210 90) (make-posn 205 100))  
(draw-solid-line (make-posn 195 90) (make-posn 200 100))  
(draw-solid-line (make-posn 200 90) (make-posn 195 100))
```

## Iniciemos

### Dibujar el cuerpo

```
;;Dibujemos el cuerpo
(draw-solid-line (make-posn 200 130) (make-posn 200 300))
;;Dibujemos los brazos
(draw-solid-line (make-posn 200 200) (make-posn 100 100))
(draw-solid-line (make-posn 200 200) (make-posn 300 100))
;;Dibujemos las piernas
(draw-solid-line (make-posn 200 300) (make-posn 100 400))
(draw-solid-line (make-posn 200 300) (make-posn 300 400))
```

## Ejercicios

- 1 Construya un vehículo, este debe ser de color azul, las llantas de color negro y las ventanas de color amarillo
- 2 Construya un semáforo, este es un rectángulo con tres círculos coloreados de rojo, amarillo y verde.

## Ejercicio en casa

Construye una figura de tu interés (es libre). Comparte la imagen en el campus virtual en el foro que se va habilitar para ello y tu código con un enlace de Pastebin (crea una cuenta).

Para la imagen no la pegues si no usa la opción de subir imagen, de lo contrario no se verá (es un bug de moodle).

Fundamentos  
de  
programación

Información  
simbólica

Estructuras

Diseño de  
funciones con  
estructuras

Estructuras  
dentro de  
estructuras

Validación de  
datos

Dibujar en Dr  
Racket

