Talleres 6 y 7
Fundamentos de Programación

Carlos Andres Delgado S, Msc
carlos.andres.delgado@correounivalle.edu.co

Febrero de 2020

Aclaraciones

1. Este taller debe ser trabajado en clase en grupos 3 estudiantes seleccionados por el docente
2. Debe entregar un archivo comprimidos en el campus virtual, el cual contendrá 3 archivos de Racket, uno para cada puntos
3. Debe respetar los nombres de funciones especificadas, el no hacerlo traerá sanción a la nota
4. El grupo debe realizar una sola entrega, si hay más de una entrega se seleccionará aleatoriamente una
5. El código debe seguir la metodología vista en el curso, no seguirla tendrá sanción en la nota
6. Agregue los ejemplos solicitados en el código, use la función check-expect siempre que sea posible.
7. En los puntos use funciones auxiliares para evitar códigos muy extensos.
8. El taller debe ser entregado el 23 de Febrero a las 23:55 por el campus virtua.
9. No incluir imágenes ni comentarios en caja, para que el archivo de racket resultante sea de texto.

Lectura de archivos

Investigue el paquete de enseñanza batch-io `https://docs.racket-lang.org/teachpack/2htdpbatch-io.html`

1. Enunciado del taller

1.1. Estructuras de datos

1. **(10 puntos)** Define las siguientes estructuras:

   - Defina una estructura Empleado que tenga como elementos (CC, nombre, sueldo, cargo, teléfono), donde CC, sueldo, teléfono son números y nombre, cargo son cadenas de texto.
   - Defina una estructura llamada Vehículo que tenga como elementos (Placa, tipo, ciudad, cantidadGalones, velocidadMaxima, distanciaXgalon), donde cantidadGalones, velocidadMaxima, distanciaXgalon son números y Placa, tipo, ciudad son cadenas de texto.
   - Defina una estructura Empresa que tenga como elementos (NIT, nombre, Empleados, dirección, dineroActual, Vehículos), donde NIT, nombre, dirección son cadenas de texto; dineroActual es un número; Empleados es una lista de Empleado y Vehículos es una lista de Vehículo.
   - Defina una estructura llamada Multinacional que tenga como elementos: (NIT, nombre, EmpresasAsociadas), donde EmpresasAsociadas es una lista de empresas, NIT y nombre son cadenas de texto.

1.2. Listas y estructuras

2. **(10 puntos)** Defina la función list-list->list-empleado, que tome como argumento una lista de listas de string y retorne una lista de empleados.

```
(define (list->list->list-empleado lis)
... )
```

Ejemplo:

Dada una lista que contiene una o varias listas cuyo contenido son los datos de los empleados

```
(list
   (list "1143843823" "Roberto_Rivaz" "620000" "Aseo" "3485698")
   (list "11145876834" "Maria_Valencia" "1000000" "Asesor" "6789043")
   (list "1234678432" "Patricia_olaya" "279000" "Mantenimiento" "
      8765423"))
```

Al aplicar la función:

```
(list->list->list-empleado
  (list
   (list "1143843823" "Roberto_Rivaz" "620000" "Aseo" "3485698")
```

```
   (list "11145876834" "Maria_Valencia" "1000000" "Asesor" "6789043")
   (list "1234678432" "Patricia_olaya" "279000" "Mantenimiento" "
      8765423")))
```

Se debe obtener una lista de estructuras Empleado:

```
(list
  (make-Empleado 1143843823 "Roberto_Rivaz" 620000 "Aseo" 3485698)
  (make-Empleado 11145876834 "Maria_Valencia" 1000000 "Asesor"
     6789043)
  (make-Empleado 1234678432 "Patricia_olaya" 279000 "Mantenimiento"
     8765423))
```

Puede usar la función check-expect para verificar:

```
(check-expect
 (list-list->list-empleado
  (list (list "1143843823" "Roberto_Rivaz" "620000" "Aseo" "3485698")
     ))
 (list (make-Empleado 1143843823 "Roberto_Rivaz" 620000 "Aseo"
    3485698)))
(check-expect
 (list-list->list-empleado
  (list
   (list "1143843823" "Roberto_Rivaz" "620000" "Aseo" "3485698")
   (list "11145876834" "Maria_Valencia" "1000000" "Asesor" "6789043")
   (list "1234678432" "Patricia_olaya" "279000" "Mantenimiento" "
      8765423")))
 (list
  (make-Empleado 1143843823 "Roberto_Rivaz" 620000 "Aseo" 3485698)
  (make-Empleado 11145876834 "Maria_Valencia" 1000000 "Asesor"
     6789043)
  (make-Empleado 1234678432 "Patricia_olaya" 279000 "Mantenimiento"
     8765423)))
```

3. **(10 puntos)** Defina la función list->vehiculo, que tome como argumento una lista de string y retorne una estructura Vehiculo. (TIP: para convertir un string en number puede usar string-¿number).

```
define (list->vehiculo lis)
  (make-Vehiculo ....))
```

Ejemplo:

Dada una lista que contiene los atributos de un vehiculos como string.

```
(list "WBC-896" "147_1.6_T_Spark_Eco_Impression" "Cali" "16.56" "180"
    "16.56")
```

Al aplicar la función

```
(list->vehiculo (list "WBC-896" "147_1.6_T_Spark_Eco_Impression" "
   Cali" "16.56" "180" "16.56"))
;;Retorna
(make-Vehiculo "WBC-896" "147_1.6_T_Spark_Eco_Impression" "Cali"
   16.56 180 16.56)
```

Puede usar los siguientes check-expect para probar el funcionamiento de su función:

```
(check-expect (list->vehiculo (list "WBC-896" "147_1.6
   _T_Spark_Eco_Impression" "Cali" "16.56" "180" "16.56"))
              (make-Vehiculo "WBC-896" "147_1.6
                 _T_Spark_Eco_Impression" "Cali" 16.56 180 16.56))
(check-expect (list->vehiculo (list "ABC-561" "147_GTA" "Bogota" "
   16.56" "230" "12.94"))
              (make-Vehiculo "ABC-561" "147_GTA" "Bogota" 16.56 230
                 12.94))
```

4. **(10 puntos)** Defina la función list-list->list-vehiculo, que tome como argumento una lista de listas de string y retorne una lista de vehículos.

```
(define (list-list->list->vehiculo lis)
  ........)
```

Ejemplo: Dada una lista que contiene una o varias listas cuyo contenido son los datos de los vehiculos.

```
(list    (list "WBC-896" "147_1.6_T_Spark_Eco_Impression" "Cali" "
   16.56" "180" "16.56")
         (list "ABC-561" "147_GTA" "Bogota" "16.56" "230" "12.94"))
;Al aplicar la función:
(list-list->list->vehiculo
                  (list  (list "WBC-896" "147_1.6
                     _T_Spark_Eco_Impression" "Cali" "16.56" "180" "
                     16.56")
                     (list "ABC-561" "147_GTA" "Bogota" "16.56" "230
                        " "12.94")))

;Se debe obtener una lista de estructuras Vehiculo:
```

```
(list (make-Vehiculo "WBC-896" "147_1.6_T_Spark_Eco_Impression" "Cali
   " 16.56 180 16.56)
       (make-Vehiculo "ABC-561" "147_GTA" "Bogota" 16.56 230 12.94))
```

Puede usar los siguientes check-expect para probar el funcionamiento de su función:

```
(check-expect
 (list-list->list->vehiculo (list (list "HVC-987" "GTO" "California"
    "15.4" "137" "39.2")))
 (list (make-Vehiculo "HVC-987" "GTO" "California" 15.4 137 39.2)))

(check-expect
 (list-list->list->vehiculo (list
                      (list "WBC-896" "147_1.6_T_Spark_Eco_Impression
                         " "Cali" "16.56" "180" "16.56")
                      (list "ABC-561" "147_GTA" "Bogota" "16.56" "230
                         " "12.94")))
 (list (make-Vehiculo "WBC-896" "147_1.6_T_Spark_Eco_Impression" "
    Cali" 16.56 180 16.56)
       (make-Vehiculo "ABC-561" "147_GTA" "Bogota" 16.56 230 12.94)))
```

5. (5 puntos) Las funciones que creó anteriormente le permitirán pasar el contenido de un archivo a estructuras y listas en Racket, copie y pegue las siguientes sentencias en su código fuente para cargar los archivos que se encuentran en el directorio actual.

```
(define Archivo_Empleados1 (read-words/line "Data/Empleados1.txt"))
(define Archivo_Empleados2 (read-words/line "Data/Empleados2.txt"))
(define Archivo_Empleados3 (read-words/line "Data/Empleados3.txt"))
(define Archivo_Vehiculos1 (read-words/line "Data/Vehiculos1.txt"))
(define Archivo_Vehiculos2 (read-words/line "Data/Vehiculos2.txt"))
(define Archivo_Vehiculos3 (read-words/line "Data/Vehiculos3.txt"))
```

Notara que al escribir en el modulo de interacciones en Racket los siguiente. Luego de haber pegado el código anterior en su código fuente:

```
> Archivo_Empleados1
(list (list "1143843823" "Roberto_Rivaz" "620000" "Aseo" "3485698") (
   list "11145876834" "Maria_Valencia" "1000000" "Asesor" "6789043")
   (list "1234678432" "Patricia_olaya" "279000"

"Mantenimiento" "8765423"))
> (read-words/line "Empleados1.txt")
(list (list "1143843823" "Roberto_Rivaz" "620000" "Aseo" "3485698") (
   list "11145876834" "Maria_Valencia" "1000000" "Asesor" "6789043")
   (list "1234678432" "Patricia_olaya" "279000" "Mantenimiento" "
   8765423"))
```

```
>
```

La función read-words/line toma el nombre de un archivo y retorna una lista con cada linea del archivo, cada linea del archivo es una lista de cada palabra del archivo.

6. (5 puntos) Copiar y pegar las siguientes lineas de código en su código fuente:

```
;;EMPRESAS
(define EMPRESA1 (make-Empresa "800.198.456-1" "Electronic Arts"
   LISTA_EMPLEADOS1 "Av siempre viva 7-43" 1000000 LISTA_VEHICULOS1))
(define EMPRESA2 (make-Empresa "977.345.564-0" "Rockstar Games"
   LISTA_EMPLEADOS2 "Cr 4 # 34-8" 900000 LISTA_VEHICULOS2))
(define EMPRESA3 (make-Empresa "954.671.110-0" "Nintendo"
   LISTA_EMPLEADOS3 "Calle 8 # 98-09" 6500000 LISTA_VEHICULOS3))

(define LISTA_EMPRESAS1 (list EMPRESA1 EMPRESA2))
(define LISTA_EMPRESAS2 (list EMPRESA3))

;;MULTINACIONALES
(define MICROSOFT (make-Multinacional "894.125.678-3" "Microsoft"
   LISTA_EMPRESAS1))
(define UBISOFT (make-Multinacional "675.098.501-5" "Ubisoft"
   LISTA_EMPRESAS2))
```

Ejemplo:

En el anterior código se crean estructuras Empresa, Empresa1, Empresa2, Empresa3. A cada una se le asigna un listado de vehiculos y un listado de empleados que hacen parte de la empresa.

Finalmente se crearán estructuras Multimacional, cada una cuenta con algunos datos relevantes y una lista de empresas que hacen parte de ella.

1.3. Recorridos en listas

7. **(10 puntos)** Defina la función vehiculoMasRapido que dada una lista de vehículos determine cual es el vehículo mas rápido de la lista.

```
(define (vehiculoMasRapido lista-Vehiculos)
  ......)
```

Ejemplo: Dada una lista de vehículos:

```
(list
   (make-Vehiculo "HVC-987" "GTO" "California" 15.4 137 39.2)
   (make-Vehiculo "VCD-325" "Camaro" "California" 18.4 240 14.72)
```

```
    (make-Vehiculo "DSR-456" "Brera_2.2_JTS_Coupe" "California" 18.4
       222 24.89)
    (make-Vehiculo "ZDE-439" "147_1.6_Progression" "Washinton_DC"
       15.77 195 28.55)
    (make-Vehiculo "ABE-070" "147_1.6_Twin_Spark_Impression" "New_York
       " 15.77 300 28.25))
```

Al aplicar la función:

```
(vehiculoMasRapido
  (list
   (make-Vehiculo "HVC-987" "GTO" "California" 15.4 137 39.2)
   (make-Vehiculo "VCD-325" "Camaro" "California" 18.4 240 14.72)
   (make-Vehiculo "DSR-456" "Brera_2.2_JTS_Coupe" "California" 18.4
      222 24.89)
   (make-Vehiculo "ZDE-439" "147_1.6_Progression" "Washinton_DC"
      15.77 195 28.55)
   (make-Vehiculo "ABE-070" "147_1.6_Twin_Spark_Impression" "New_York
      " 15.77 300 28.25)))
;;Retorna
(make-Vehiculo "ABE-070" "147_1.6_Twin_Spark_Impression" "New_York"
   15.77 300 28.25)
```

Puede usar los siguientes check-expect para probar el funcionamiento de su función:

```
(check-expect
 (vehiculoMasRapido
  (list
   (make-Vehiculo "HVC-987" "GTO" "California" 15.4 137 39.2)
   (make-Vehiculo "VCD-325" "Camaro" "California" 18.4 240 14.72)
   (make-Vehiculo "DSR-456" "Brera_2.2_JTS_Coupe" "California" 18.4
      222 24.89)
   (make-Vehiculo "ZDE-439" "147_1.6_Progression" "Washinton_DC"
      15.77 195 28.55)
   (make-Vehiculo "ABE-070" "147_1.6_Twin_Spark_Impression" "New_York
      " 15.77 300 28.25)))
 (make-Vehiculo "ABE-070" "147_1.6_Twin_Spark_Impression" "New_York"
    15.77 300 28.25))
(check-expect
 (vehiculoMasRapido
  (list
   (make-Vehiculo "WBC-896" "147_1.6_T_Spark_Eco_Impression" "Cali"
      16.56 180 16.56)
   (make-Vehiculo "ABC-561" "147_GTA" "Bogota" 16.56 230 12.94)
   (make-Vehiculo "ZWQ-760" "156_2.5_V6_24V_Sportwagon" "Bogota"
      16.56 230 19.83)
```

```
    (make-Vehiculo "KLE-902" "75_2.0_Turbo_Diesel" "Medellin" 12.89
       175 26.01)
    (make-Vehiculo "BHU-451" "Alfasud_Ti" "Medellin" 13.14 500 13.14))
       )
 (make-Vehiculo "BHU-451" "Alfasud_Ti" "Medellin" 13.14 500 13.14))
```

8. **(10 puntos)** Defina la función vehiculoMasRapidoMultinacional que tome como argumento una multinacional y determine cual es el vehiculo mas rapido de todas las empresas que componen la mutinacional. (TIP es libre de usar todas las funciones auxiliares que requiera para descomponer el problema, es oblicgatorio que la funcion principal se llame vehiculoMasRapidoMultinacional).

```
(define (vehiculoMasRapidoMultinacional mult)
  .........)
```

Ejemplo: Dada la multinacional MICRISOFT definida anteriorme (por simplicidad no se mostrará su composición), al aplicar la función:

```
(vehiculoMasRapidoMultinacional MICROSOFT)
;;Retorna
(make-Vehiculo "BHU-451" "Alfasud_Ti" "Medellin" 13.14 500 13.14)
```

Puede usar los siguientes check-expect para probar el funcionamiento de su función:

```
(check-expect (vehiculoMasRapidoMultinacional MICROSOFT) (
   make-Vehiculo "BHU-451" "Alfasud_Ti" "Medellin" 13.14 500 13.14))
(check-expect (vehiculoMasRapidoMultinacional UBISOFT) (make-Vehiculo
    "QIC-689" "156_2.0-TS_Lusso" "Pasto" 16.56 600 26.9))
```

9. **(10 puntos)** Defina la función sumarDineroEmpresas que reciba como argumento una lista de empresas y retorne la suma de todo el dinero actual de la lista de empresas.

```
(define (sumaDineroEmpresas lista-empresas)
  .....)
```

Ejemplo, Dada una lista de empresas LISTA_EMPRESAS1 (por simplicidad no se mostrará su composición), al aplicar la función (sumaDineroEmpresas LISTA_EMPRESAS1), deber retornar 1900000.

```
(check-expect (sumaDineroEmpresas LISTA_EMPRESAS1) 1900000)
(check-expect (sumaDineroEmpresas LISTA_EMPRESAS2) 6500000)
```

10. **(10 puntos)** Copiar y pegar las siguientes lineas de código en su código fuente.

```
(define (dineroTotal mult)
  (sumaDineroEmpresas (Multinacional-EmpresasAsociadas mult)))
```

Ejemplo: Esta función nos permite calcular el dinero total que posee una mutinacional, dado la suma del dinero actual de totas las empresas que la componen. Esta función hace uso de la función sumaDineroEmpresas para descomponer la estructra Multinacional obteniendo la lista de empresas que la componen.

11. **(10 puntos)** Defina la función multinacionalMasDinero que reciba como argumentos una lista de mutinacionales y retorne la multinacional que tiene mas dinero

```
(define (multinacionalMasDinero mults)
  ......)
```

Ejemplo: Dada una lista de multinacionales (list MICROSOF UBISOFT), MICROSOF y UBISOFT ambas definidas enteriormente. Al aplicar la funcion. (multinacionalMasDinero (list MICROSOFT UBISOFT)) debe retornar:

```
(make-Multinacional
 "675.098.501-5"
 "Ubisoft"
 (list
  (make-Empresa
   "954.671.110-0"
   "Nintendo"
   (list (make-Empleado 1143843823 "Carlos_Orozco" 900000 "Asesor"
      3485698) (make-Empleado 11145876834 "Ximena_Sanchez" 1000000 "
      Ingeniero" 6789043) (make-Empleado 1234678432 "Juan_Carlos"
      279000 "Mantenimiento" 8765423))
   "Calle 8 # 98-09"
   6500000
   (list
    (make-Vehiculo "ASD-093" "Alfasud_Berlina" "Armenia" 13.14 137
       13.14)
    (make-Vehiculo "BVR" "156_V6_Q-System" "Cali" 16.56 227 13.37)
    (make-Vehiculo "WWB-904" "156_Sportwagon_3.2" "Pereira" 16.56 250
        39.67)
    (make-Vehiculo "WEX-093" "Spider_3.2_V6" "Cali" 18.4 242 17.73)
    (make-Vehiculo "QIC-689" "156_2.0-TS_Lusso" "Pasto" 16.56 600
       26.9)))))
```

que es equivalente a decir UBISOFT, puede probar:

```
(check-expect (multinacionalMasDinero (list MICROSOFT UBISOFT))
   UBISOFT)
```