UNIVERSDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO

PROGRAMACIÓN DECLARATIVA

INGENIGERÍA NFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

Tema 7.- Lectura y escritura

Primera parte: Scheme

- Tema 1.- Introducción al lenguaje Scheme
- Tema 2.- Expresiones y funciones
- Tema 3.- Predicados y sentencias condicionales
- Tema 4.- Iteración y recursión
- Tema 5.- Tipos de datos compuestos
- Tema 6.- Abstracción de datos
- Tema 7.- Lectura y escritura

Segunda parte: Prolog

- Tema 8.- Introducción al lenguaje Prolog
- Tema 9.- Elementos básicos de Prolog
- Tema 10.- Listas
- Tema 11.- Reevaluación y el “corte”
- Tema 12.- Entrada y salida

Primera parte: Scheme

Tema 1.- Introducción al lenguaje Scheme

Tema 2.- Expresiones y funciones

Tema 3.- Predicados y sentencias condicionales

Tema 4.- Iteración y recursión

Tema 5.- Tipos de datos compuestos

Tema 6.- Abstracción de datos

Tema 7.- Lectura y escritura

Índice

Índice

1. Interacción con el sistema

- Ejecución de un programa
- Transcripción de una sesión interactiva

1. Interacción con el sistema
   - Ejecución de un programa
   - Transcripción de una sesión interactiva

1. Interacción con el sistema
   - Ejecución de un programa
     - Ejecución usando la sentencia ***load***
     - Ejecución usando la interfaz gráfica

1. Interacción con el sistema
   - Ejecución de un programa
     - ***load***
       - **Sintaxis**

         *(**load** fichero)*

         ❑ ***fichero***: nombre de un fichero existente.
       - **Significado**

         ❑ Interpreta **secuencialmente** las instrucciones del *fichero*.

         ❑ Si se definen variables o funciones entonces pueden ser utilizadas posteriormente.

1. Interacción con el sistema
   - Ejecución de un programa
     - ***load***
       - **Ejemplo1**

         *(**load** "hanoi.rkt")*
       - **Ejemplo 2**

         *(**load** "newton.rkt")*

         ;; se ejecuta la función definida en el fichero

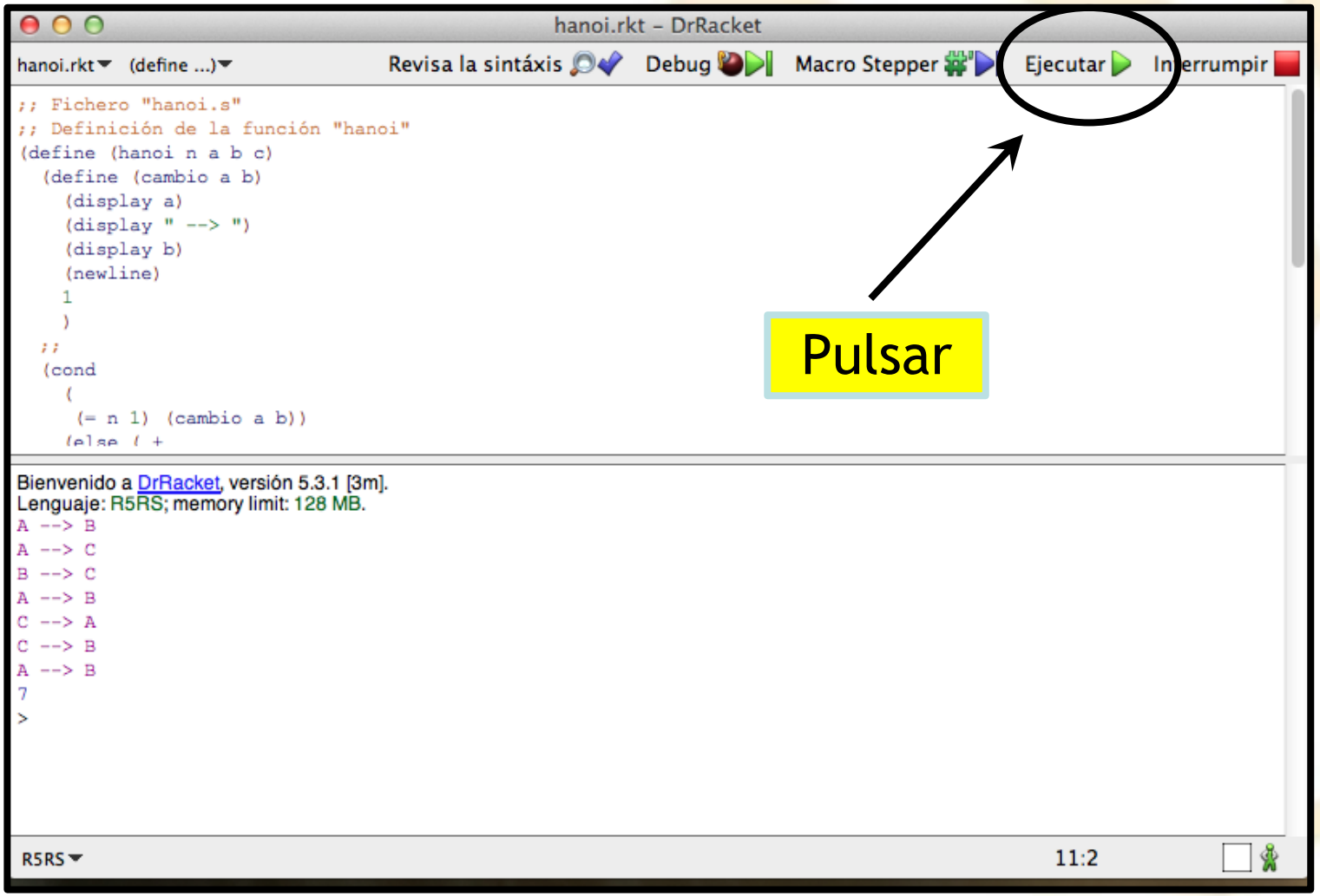
         *(newton (**lambda** (x) (- (* x x) 2)) 1)*

         ➔ *1,4142…*

1. Interacción con el sistema

• Ejecución de un programa

○ Ejecución usando una interfaz gráfica



1. Interacción con el sistema
   - Ejecución de un programa
   - Transcripción de una sesión interactiva

1. Interacción con el sistema

- Transcripción de una sesión interactiva
  - Se utiliza para **depurar** el código
  - Tipos
    - Transcripción desde la ventana de interacciones.
    - Transcripción usando una interfaz gráfica.

1. Interacción con el sistema
   - Transcripción de una sesión interactiva
     - **Transcripción desde la ventana de interacciones**
     - *transcript-on* y *transcript-off*
       - **Sintaxis**

```
(transcript-on fichero)
…
;; Interacciones
….
(transcript-off)
```

1. Interacción con el sistema

- Transcripción de una sesión interactiva
  - **Transcripción desde la ventana de interacciones**
  - *transcript-on* y *transcript-off*
    - **Significado**
      - Almacena en el fichero las **interacciones** del usuario con el intérprete.
      - *(transcript-on fichero)* crea el fichero en el que se almacenarán las interacciones.
      - (*transcrip-off*) cierra el fichero activo y finaliza la transcripción.

1. Interacción con el sistema

- Transcripción de una sesión interactiva
  - **Transcripción desde la ventana de interacciones**
  - *transcript-on* y *transcript-off*
    - **Observación**
      - Solamente puede haber un fichero activo en un mismo instante,
      - aunque algunos intérpretes pueden ser más permisivos.

1. Interacción con el sistema
   - Transcripción de una sesión interactiva
     - **Transcripción desde la ventana de interacciones**
     - *transcript-on* y *transcript-off*
       - **Ejemplo**

```
(transcript-on  "salida.txt")
(newton  (lambda (x) (- (* x x) 2)) 1)
1.4142165798805022
(transcript-off)
```

1. Interacción con el sistema

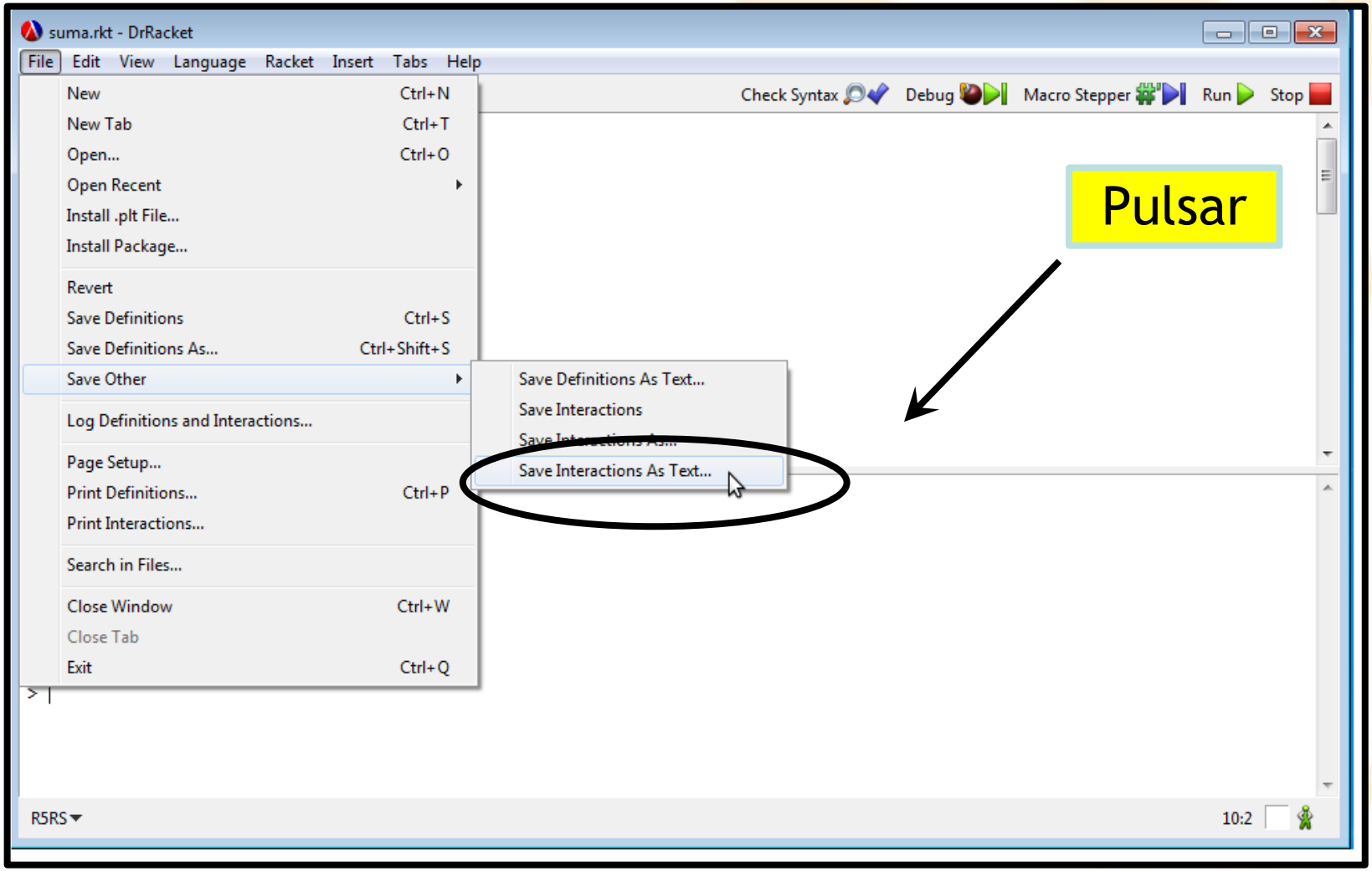
- Transcripción de una sesión interactiva
  - **Transcripción desde la ventana de interacciones**
  - *transcript-on* y *transcript-off*
    - **Ejemplo**

      Contenido del fichero "salida.txt"

```
> (newton  (lambda (x) (- (* x x) 2)) 1)
1.4142165798805022
(transcript-off)
```

1. Interacción con el sistema

- Transcripción de una sesión interactiva
  - **Transcripción usando una interfaz gráfica**



Índice

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
- Funciones de ficheros y directorios
- Cierre de un puerto
- Predicados sobre puertos de entrada y salida

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
- Funciones de ficheros y directorios
- Cierre de un puerto
- Predicados sobre puertos de entrada y salida

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
  - *open-input-file*
  - *open-output-file*

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
  - ***open-input-file***
    - **Sintaxis**

      *(**open-input-file** fichero)*

      - ***fichero***: nombre de un fichero que debe existir.
      - Devuelve un **puerto de entrada**.
    - **Significado**
      - Abre el fichero para **lectura** y lo asocia a un puerto de entrada.
      - Las **operaciones de lectura** en el fichero se realizarán a través del puerto de entrada.

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
  - ***open-input-file***
    - **Ejemplo**

*(**define** p1 (**open-input-file** "datos.txt"))*

p1

➔ *#<input-port: … datos.txt>*

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
  - ***open-output-file***
    - **Sintaxis**

      *(**open-output-file** fichero)*

      ❑ ***fichero***: nombre de un fichero que se va crear.

      ❑ Devuelve un **puerto de salida**.

    - **Significado**

      ❑ **Crea** el fichero para escribir y lo **asocia** a un puerto de salida.

      ❑ Las **operaciones de escritura** en el fichero se realizarán a través del puerto de salida.

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
  - ***open-output-file***
    - **Ejemplo**

*(**define** p2 (**open-output-file** "salida.txt"))*

p2

➔ *#<output-port: … salida.txt>*

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
- Funciones de ficheros y directorios
- Cierre de un puerto
- Predicados sobre puertos de entrada y salida

2. Apertura de ficheros y cierre de puertos

- Funciones de ficheros y directorios
  - *file-exists?*
  - *directory-exist?*
  - Otras funciones

2. Apertura de ficheros y cierre de puertos

- Funciones de ficheros y directorios
  - ***file-exists?***
    - **Sintaxis**

      *(**file-exists?** **nombre**)*

    - **Significado**

      ❑ Devuelve *#t* si ***nombre*** es un fichero (no un directorio).

2. Apertura de ficheros y cierre de puertos

- Funciones de ficheros y directorios
  - ***directory-exists?***
    - **Sintaxis**

      (directory-exists? nombre)

    - **Significado**
      - ❑ Devuelve *#t* si ***nombre*** es un directorio.

2. Apertura de ficheros y cierre de puertos

- Funciones de ficheros y directorios
  - ***Otras funciones:***
    - Ficheros
    - Directorios

Más información:

http://docs.racket-lang.org/reference/Filesystem.html?q=file

2. Apertura de ficheros y cierre de puertos

- Funciones de ficheros y directorios
  - ***Otras funciones***
    - ***Ficheros***
      - *link-exists?*
      - *delete-file*
      - *rename-file-or-directory*
      - *file-or-directory-modify-seconds*
      - *file-or-directory-permissions*
      - *file-or-directory-identity*
      - *file-size*
      - *copy-file*
      - *make-file-or-directory-link*

2. Apertura de ficheros y cierre de puertos
   - Funciones de ficheros y directorios
     - ***Otras funciones:***
       - ***Directorios***
         - ❑ *current-directory*
         - ❑ *current-drive*
         - ❑ *make-directory*
         - ❑ *delete-directory*
         - ❑ *directory-list*
         - ❑ *filesystem-root-list*

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
- Funciones de ficheros y directorios
- Cierre de un puerto
- Predicados sobre puertos de entrada y salida

2. Apertura de ficheros y cierre de puertos

- Cierre de un puerto
  - *close-input-port*
  - *close-output-port*

2. Apertura de ficheros y cierre de puertos

- Cierre de un puerto
  - ***close-input-port***
    - **Sintaxis**

      *(**close-input-port** puerto)*

      ❑ ***puerto***: nombre de un puerto de entrada.
    - **Significado**

      ❑ **Cierra** el fichero asociado al puerto de entrada.

    ❑ **Ejemplo**

      *(**close-input-port** p1)*

2. Apertura de ficheros y cierre de puertos

- Cierre de un puerto
  - ***close-output-port***
    - **Sintaxis**

      *(**close-output-port** puerto)*

      ❑ ***puerto***: nombre de un puerto de salida.
    - **Significado**

      ❑ **Cierra** el fichero asociado al puerto de salida.
    - ❑ **Ejemplo**

      *(**close-output-port** p2)*

2. Apertura de ficheros y cierre de puertos

- Apertura de un fichero
- Funciones de ficheros y directorios
- Cierre de un puerto
- Predicados sobre puertos de entrada y salida

2. Apertura de ficheros y cierre de puertos

- Predicados sobre puertos de entrada y salida
  - *input-port?*
  - *output-port?*

2. Apertura de ficheros y cierre de puertos

- Predicados sobre puertos de entrada y salida
  - ***input-port?***
    - **Sintaxis**

      *(**input-port?** objeto)*
    - **Significado**

      ❑ Comprueba si *objeto* es un puerto de **entrada**
    - **Ejemplo**

      *(**input-port?** p1)*

      ➔ *#t*

2. Apertura de ficheros y cierre de puertos

- Predicados sobre puertos de entrada y salida
  - ***output-port?***
    - **Sintaxis**

      *(**output-port?** objeto)*
    - **Significado**

      ❑ Comprueba si *objeto* es un puerto de **salida**
    - **Ejemplo**

      *(**output-port?** p2)*

      ➔ *#t*

Índice

3. Lectura

- *current-input-port*
- *read*
- *read-char*
- *peek-char*
- *char-ready?*
- *eof-object?*

3. Lectura

- ***current-input-port***
  - **Significado**
    - Devuelve el ***puerto de entrada actual***
    - Generalmente, este **puerto** está **asociado** al **teclado.**
    - El procedimiento ***with-input-from-file*** puede asociar un fichero al puerto de entrada actual (véase el apartado nº 6).

3. Lectura

- ***read***
  - **Sintaxis**

    (read [puerto])

    - ***puerto***: puerto de **entrada** asociado a un fichero de **lectura**; es opcional.
    - Si se omite, se utiliza el puerto de entrada actual
  - **Significado**
    - Lee un **objeto** del fichero o dispositivo asociado al puerto.
    - Se **actualiza** el puerto para que se coloque al principio del siguiente objeto.

3. Lectura

- ***read***
  - **Ejemplo**

```
(display "Escriba un número --> ")
(define dato (read))
```

Escriba un número --> 9 **eof**

dato

➔ *9*

3. Lectura

- ***read***
  - **Ejemplo**

```
(define p1 (open-input-file "datos.txt"))
(display "Se lee un número del fichero ")
(define dato (read p1))
(display dato)
...
```

3. Lectura

- ***read-char***
  - **Sintaxis**

    *(**read-char** **[puerto]**)*

    - ***puerto***: puerto de **entrada** asociado a un fichero de **lectura**; es opcional.
    - Si se omite, se utiliza el puerto de entrada actual.
  - **Significado**
    - Lee un **carácter** del fichero o dispositivo asociado al puerto.
    - Se **actualiza** el puerto para que se coloque en el siguiente carácter.

3. Lectura

- ***read-char***
  - **Ejemplo**

```
(display "Escriba un carácter--> ")
(define tecla (read-char))
```

Escriba un carácter--> a **eof**

tecla

➔ *#\a*

3. Lectura

- ***read-char***
  - **Ejemplo**

```
(define  p1 (open-input-file "datos.txt"))
(display "Se lee un carácter del fichero ")
(define dato (read-char   p1))
(display dato)
 ...
```

3. Lectura

- ***peek-char***
  - **Sintaxis**

    *(**peek-char** [**puerto**])*

    - ***puerto***: puerto de **entrada** asociado a un fichero de **lectura**; es opcional.
    - Si se omite, se utiliza el puerto de entrada actual.
  - **Significado**
    - Lee un **carácter** del fichero o dispositivo asociado al puerto de entrada actual.
    - **No** se actualiza el puerto para que se coloque en el siguiente carácter.

3. Lectura

- ***peek-char***
  - **Ejemplo**

```
(display "Escriba un carácter--> ")
(define tecla (peek-char))
```

Escriba un carácter--> *a* **eof**

tecla

➔ *#\a*

3. Lectura

- ***peek-char***
  - **Ejemplo**

```
(define p1 (open-input-file "datos.txt"))
(display "Se lee un carácter del fichero ")
(define dato (peek-char p1))
(display dato)
 …
```

3. Lectura

- ***char-ready?***
  - **Sintaxis**

    *(**char-ready?** [puerto])*

    - ***puerto***: puerto de **entrada** asociado a un fichero de **lectura**; es opcional.
    - Si se omite, se utiliza el puerto de entrada actual.
  - **Significado**
    - Devuelve verdadero #t si hay un carácter preparado en el *puerto*.
    - En caso contrario, devuelve falso *#f*.

3. Lectura

- ***char-ready?***
  - **Ejemplo**

```
(define p1 (open-input-file "datos.txt"))
 (if (char-ready? p1)
      ...
```

3. Lectura

- ***eof-object?***
  - **Sintaxis**

    *(**eof-object?** objeto)*
  - **Significado**
    - Devuelve *#t* si y solamente si el argumento es el **objeto fin de fichero #<eof>.**
    - En caso contario, devuelve *#f*.

Índice

4. Escritura

- *current-output-port*
- *display*
- *write*
- *write-char*
- *newline*

4. Escritura

- ***current-output-port***
  - **Significado**
    - Devuelve el ***puerto de salida actual***.
    - Generalmente, este **puerto** está **asociado** a la **pantalla**.
    - El procedimiento ***with-output-to-file*** puede asociar un fichero al puerto de salida actual (véase el apartado nº 6).

4. Escritura

- ***display***
  - **Sintaxis**

    *(**display** objeto [puerto])*

    - ***puerto***: puerto de **salida** asociado a un fichero de **escritura**; es opcional.
    - Si se omite, se utiliza el puerto de salida actual.

4. Escritura

- ***display***
  - **Significado**
    - Escribe una **representación** del ***objeto*** en el fichero o dispositivo asociado al puerto de salida actual.
    - Genera una salida **legible por personas**.
    - Las cadenas se escriben **sin** comillas delimitadoras y **no** contienen secuencias de escape .
    - Un carácter se escribe como si se hubiera hecho con *write-char*.

4. Escritura

- ***display***
  - **Ejemplo (1/2)**

*(**display** "Cadena \"sin\" comillas iniciales ni finales")*

➔ *Cadena "sin" comillas iniciales ni finales*

4. Escritura

- ***display***
  - **Ejemplo (2/2)**

```
(define p2 (open-output-file "salida.txt"))
(display "Cadena \"sin\" comillas iniciales ni finales"  p2)
```

Contenido de "salida.txt"

```
Cadena "sin" comillas iniciales ni finales
```

4. Escritura

- ***write***
  - **Sintaxis**

    *(**write** objeto [puerto])*

    - ***puerto***: puerto de **salida** asociado a un fichero de **escritura**; es opcional.
    - Si se omite, se utiliza el puerto de salida actual.

4. Escritura

- ***write***
  - **Significado**
    - Escribe una **representación** del ***objeto*** en el fichero o dispositivo asociado al puerto de salida actual.
    - Genera una **salida orientada al ordenador.**
    - Las **cadenas** se escriben
      - ✓ con comillas delimitadoras
      - ✓ y contienen secuencias de escape.

4. Escritura

- ***write***
  - **Ejemplo (1/2)**

*(**write** "Cadena \"con\" comillas iniciales y finales")*

➔ *"Cadena \"con\" comillas iniciales y finales"*

4. Escritura

- ***write***
  - **Ejemplo (2/2)**

```
(define p2 (open-output-file "salida.txt"))
(write  "Cadena \"con\" comillas iniciales y finales" p2 )
```

Contenido de "salida.txt

```
"Cadena \"con\" comillas iniciales y finales"
```

4. Escritura

- ***write-char***
  - **Sintaxis**

    *(**write -char** objeto [puerto])*

    - ***puerto***: puerto de **salida** asociado a un fichero de **escritura**; es opcional.
    - Si se omite, se utiliza el puerto de salida actual.
  - **Significado**
    - Escribe el *carácter* en el fichero o dispositivo asociado al puerto de salida actual.

4. Escritura

- ***write-char***
  - **Ejemplo (1/2)**

```
(define tecla (read-char))
(write-char tecla)
```

4. Escritura

- ***write-char***
  - **Ejemplo (2/2)**

```
(define p2 (open-output-file "salida.txt"))
(define tecla (read-char) )
(write-char tecla p2 )
```

4. Escritura

- ***newline***
  - **Sintaxis**

    *(**newline** [puerto])*

    - ***puerto***: puerto de **salida** asociado a un fichero de **escritura**; es opcional.
    - Si se omite, se utiliza el puerto de salida actual.
  - **Significado**
    - Genera un **salto de línea** en el fichero o dispositivo asociado al puerto de salida actual.

4. Escritura

- ***newline***
  - **Ejemplo (1/2)**

*(**display** "uno")*

*(**newline**)*

*(**display** "dos")*

➔ *uno*

dos

4. Escritura

- ***newline***
  - **Ejemplo (2/2)**

```
(define p2 (open-output-file "salida.txt"))
(display "uno" p2)
(newline p2)
(display "dos" p2)
```

Índice

5. Ejemplos de lectura y escritura

- Programa con menú de opciones
- Programa de gestión de donantes de sangre

5. Ejemplos de lectura y escritura

- Programa con menú de opciones
- Programa de gestión de donantes de sangre

Ejemplo: programa con menú de opciones (1/3)

```
(define (programa)
    (define (pedir-opcion)
        (display "elige una opcion")
        (newline)

        (display "1 →   raíz cuadrada")
        (newline)

        (display "2 →   seno")
        (newline)

        (display "0 →    salir")
        (newline)
        (newline)
        (read)
    )
      ...
```

Ejemplo: programa con menú de opciones (2/3)

```
...
;; cuerpo de programa

(do
    (
      (opcion   (pedir-opcion)   (pedir-opcion))
    )

   ;; condición de salida
  ( (= opcion 0)     (display "fin del programa") )

   ;; cuerpo del bucle do

 ...
```

Ejemplo: programa con menú de opciones (3/3)

```
...
;; cuerpo del bucle do
  (cond
     ((= opcion 1)  (display "escribe un numero positivo → ")
                    (display (sqrt  (read)))
                    (newline)
     )
    ((= opcion 2)  (display "escribe un numero → ")
                   (display (sin (read)))
                   (newline)
     )
     (else  (display “error”)  (newline) )
    )
    (newline)
  )
)
;; Llamada al programa
(programa)
```

5. Ejemplos de lectura y escritura

- Programa con menú de opciones
- Programa de gestión de donantes de sangre

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;    FUNCIONES DEL TIPO ABSTRACTO DE DATOS: DONANTE
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


; FUNCIÓN DE CREACIÓN
;=====================
(define (crear-donante nombre  num_donaciones grupo rh)
    (list   (list 'nombre nombre)
            (list 'numero_donaciones num_donaciones)
            (list 'grupo grupo)
            (list 'rh rh)
    )
)
```

Programa de gestión de donantes de sangre (2/21)

```
;; FUNCIONES DE CONSULTA O ACCESO
;================================
(define (nombre donante)
    (cadr (assoc 'nombre donante))
)
(define (numero_donaciones donante)
    (cadr (assoc 'numero_donaciones donante))
)
(define (grupo donante)
    (cadr (assoc 'grupo donante))
)
(define (rh donante)
    (cadr (assoc 'rh donante))
```

Programa de gestión de donantes de sangre (3/21)

```
;; FUNCIONES DE MODIFICACIÓN
;==========================
(define (cambiar-nombre! donante nuevo)
    (set-cdr! (assoc 'nombre donante) (list nuevo))
)
(define (cambiar-donaciones! donante nuevo)
    (set-cdr! (assoc 'numero_donaciones donante) (list nuevo))
)
(define (cambiar-grupo! donante nuevo)
    (set-cdr! (assoc 'grupo donante) (list nuevo))
)
(define (cambiar-rh! donante nuevo)
    (set-cdr! (assoc 'rh donante) (list nuevo))
)
```

Programa de gestión de donantes de sangre (4/21)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;          PROGRAMA
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (programa)
   ;; FUNCIONES AUXILIARES
   ;; FUNCIÓN QUE MUESTRA LAS OPCIONES DEL MENÚ
   ;; Y PERMITE ELEGIR UNA DE ELLAS

    (define (pedir-opcion)
          …
```

Programa de gestión de donantes de sangre (5/21)

```
(define (pedir-opcion)
  (display "Elige una opcion" )
  (newline)
  (display "1 -> Introducir un donante desde el teclado" )
  (newline)
  (display "2 -> Cargar donantes desde un fichero" )
  (newline)
  (display "3 -> Grabar los donantes en un fichero" )
  (newline)
  (display "4 -> Mostrar los donantes por pantalla" )
  (newline)
  (display "5 -> Mostrar la estructura interna de la lista de donantes" )
  (newline)
  (display "0 -> Salir" )
  (newline)
  (newline)
   (display " --> ")
  ;; lee la opción elegida
   (read)
  )
```

Programa de gestión de donantes de sangre (6/21)

```
;; LEE UN DATO DEL TECLADO Y LO DEVUELVE
;; Parámetro:
;;      mensaje: cadena de texto que indica el dato que se solicita
;;

(define (leer-teclado mensaje)
 (display mensaje)
 (display " --> ")
 (read)
 )
```

Programa de gestión de donantes de sangre (7/21)

```
;; LEE UNA CADENA DEL TECLADO Y LA DEVUELVE
;; Parámetro:
;;      mensaje: cadena de texto que indica el dato que se solicita

(define (leer-teclado-cadena mensaje)
    (display mensaje)
    (display " --> ")

    ;; Elimina el carácter de salto de línea #\newline, si existe
    (if (char=? (peek-char) #\newline)
        (read-char)
    )

    ;; Lee los caracteres
    ;; hasta que encuentra el carácter de salto de línea #\newline

        ...
```

Programa de gestión de donantes de sangre (8/21)

```
;; Lee los caracteres
;; hasta que encuentra el carácter de salto de línea #\newline

  (do
     (
      (cadena   (make-string 0)  (string-append cadena (string caracter)))
      (caracter (read-char)      (read-char))
      )
    ;; condición de salida
    ((char=? #\newline   caracter)
    ;; devuelve la cadena leída
       cadena
      )
    ;; no hay cuerpo del bucle do
    )
  )
```

Programa de gestión de donantes de sangre (8/21)

```
;; Lee los caracteres
;; hasta que encuentra el carácter de salto de línea #\newline

  (do
     (
      (cadena   (make-string 0)  (string-append cadena (string caracter)))
      (caracter (read-char)      (read-char))
      )
    ;; condición de salida
    ((char=? #\newline   caracter)
    ;; devuelve la cadena leída
       cadena
      )
    ;; no hay cuerpo del bucle do
    )
  )
```

Funciones de manejo de cadenas

Programa de gestión de donantes de sangre (9/21)

```
;; INTRODUCIR LOS DATOS DE UN DONANTE DESDE EL TECLADO
 (define (leer-donante-teclado)
    (crear-donante
      (leer-teclado-cadena "Nombre del donante (escribe con comillas): ")
     (leer-teclado "Numero de donaciones: ")
     (leer-teclado "Grupo Sanguíneo: ")
    (leer-teclado "Factor rh: ")
     )
  )
```

Programa de gestión de donantes de sangre (10/21)

```
;; FUNCIÓN QUE PONE LAS COMILLAS INICIALES Y FINALES A UN TEXTO
;; Y LO DEVUELVE COMO CADENA
;; Parámetro:
;;      texto: caracteres a los que se les van a poner las comillas
;;
(define (poner-comillas texto)
  (string-append (string #\") texto (string #\"))
  )
```

Programa de gestión de donantes de sangre (11/21)

```
; CARGAR LOS DONANTES DESDE UN FICHERO
(define (leer-donantes-fichero)
 (define puerto
        (open-input-file
            (leer-teclado-cadena "nombre del fichero (sin comillas)")
        )
  )

  ...
```

Programa de gestión de donantes de sangre (12/21)

```
...
 (do
   (
    (lista_donantes '() (append lista_donantes
                                (list (crear-donante (poner-comillas nombre)
                                                     (read puerto)
                                                     (read puerto)
                                                     (read puerto)
                                                     )
                                      )
                                )
                    )
    (nombre    (read puerto)  (read puerto))
    )
   ;; Condición de salida del bucle
   ...
```

Programa de gestión de donantes de sangre (13/21)

```
...
  ;; Condición de salida del bucle
  ((eof-object? nombre)
   ;; Se cierra el puerto asociado al fichero de entrada
      (close-input-port puerto)
   ;; Se devuelve la lista de donantes
    lista_donantes
    )
  ;; No hay cuerpo del bucle
  )
)
```

Programa de gestión de donantes de sangre (14/21)

```
;; GRABAR LOS DATOS DE LOS DONANTES EN UN FICHERO
 ;; Parámetro:
 ;;    lista_donantes: lista que contiene a los donantes que se van a grabar

(define (grabar-donantes-fichero lista_donantes)
  (do
    (
     (puerto (open-output-file
                    (leer-teclado-cadena "nombre del fichero (sin comillas")))
     (lista_auxiliar    lista_donantes    (cdr lista_auxiliar))
     )
    ;; Condición de salida del bucle

   ...
```

Programa de gestión de donantes de sangre (15/21)

```
;; Condicion de salida del bucle
   ((null? lista_auxiliar)
        ; Se cierra el puerto asociado al fichero de salida
       (close-output-port puerto)
    )
   ;; Cuerpo del bucle
    (display (nombre (car lista_auxiliar)) puerto)
    (display " " puerto)
    (display (numero_donaciones (car lista_auxiliar)) puerto)
    (display " " puerto)
    (display (grupo (car lista_auxiliar)) puerto)
    (display " " puerto)
    (display (rh (car lista_auxiliar)) puerto)
    (newline puerto)
  )
 )
```

Programa de gestión de donantes de sangre (16/21)

```
;; MOSTRAR POR PANTALLA LOS DATOS DE LOS DONANTES
 ;; Parámetro:
 ;;   lista_donantes: lista que contiene a los donantes que se van a mostrar
(define (mostrar-donantes lista_donantes)
  (do
    (
     (lista_auxiliar   lista_donantes    (cdr lista_auxiliar))
     )
    ;; Condición de salida del bucle
    ((null? lista_auxiliar)     (newline))

    ;; Cuerpo del bucle
     …
```

Programa de gestión de donantes de sangre (17/21)

```
  ;; Cuerpo del bucle
  (display "Nombre:              ")
  (display (nombre (car lista_auxiliar)))
  (newline)
  (display "Numero de donaciones: ")
  (display (numero_donaciones (car lista_auxiliar)))
  (newline)
  (display "Grupo sanguineo:      ")
  (display (grupo (car lista_auxiliar)))
  (newline)
  (display "Factor rh:            ")
  (display (rh (car lista_auxiliar)))
  (newline)
  (newline)
 )
)
```

Programa de gestión de donantes de sangre (18/21)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; CUERPO DEL PROGRAMA DE DONANTES
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(do
   ;; Variables
   (
     ;; LISTA EN LA QUE SE VAN A ALMACENAR LOS DONANTES
    (donantes    '())
    (opcion      (pedir-opcion) (pedir-opcion))
    )
   ;; condicion de salida
   ((= opcion 0)      (display "fin del programa"))
   ;; cuerpo del bucle
   ...
```

Programa de gestión de donantes de sangre (19/21)

```
;; cuerpo del bucle
(cond
  ;; INTRODUCIR UN DONANTES DESDE EL TECLADO
 ((= opcion 1)
    (display "Introduccion de datos de un donante")
    (newline)
      ;; Uso obligatorio de set!
    (set! donantes (append donantes (list (leer-donante-teclado))))
    )

  ;; CARGAR DONANTES DESDE UN FICHERO
 ((= opcion 2)
    (display "Carga de los datos de los donantes contenidos en un
fichero")
    (newline)
      ;; Uso obligatorio de set!
    (set! donantes (append donantes (leer-donantes-fichero)))
    (display "Datos cargados")
    (newline)
 )
```

Programa de gestión de donantes de sangre (20/21)

```
;; GRABAR LOS DONANTES EN UN FICHERO
      ((= opcion 3)
        (display "Grabacion de los datos de los donantes en un fichero")
        (newline)
        (grabar-donantes-fichero donantes)
        (display "Datos grabados")
          (newline)
       )
        ;; MOSTRAR LOS DONANTES POR LA PANTALLA
      ((= opcion 4)
           (mostrar-donantes donantes)
       )
        ;; MOSTRAR LA ESTRUCTURA INTERNA DE LA LISTA DE DONANTES
      ((= opcion 5)
           (display donantes)
            (newline)
       )
```

Programa de gestión de donantes de sangre (21/21)

```
;; CONTROL DE ERRORES
        (else (display "Opcion incorrecta")
            (newline)
        )
      )
        ;; Parada antes de continuar
        (leer-teclado-cadena "Pulse \"Enter\" para continuar")
        (newline)
    )
  )

;;LLAMADA AL PROGRAMA
(programa)
```

Índice

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
- **Procedimientos que operan directamente con los ficheros**

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-input-from-file***
    - **Cambia** el puerto de **entrada actual**.

      → *current-input-port*
  - ***with-output-to-file***
    - **Cambia** el puerto de **salida actual**.

      → *current-output-port*

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-input-from-file***
    - **Sintaxis**

      *(**with-input-from-file** f p)*

      - ***f***: nombre de un fichero existente.
      - ***p***: procedimiento sin argumentos.
    - **Significado**

      1. Abre el fichero ***f*** y lo asocia al puerto de entrada actual → ***current-input-port***.
      2. Llama al procedimiento ***p***, que realiza sus operaciones de lectura desde el fichero asociado al puerto de entrada actual.

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-input-from-file***
    - **Ejemplo**

```
(define (contar-palabras)
  (cond
        ((eof-object? (read))   0)
        (else              (+ 1 (contar-palabras)))
  )
)

;; LLamada
(with-input-from-file "datos.txt" contar-palabras)
```

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-input-from-file***
    - **Ejemplo**

```
;; LLamada
(with-input-from-file "datos.txt" contar-palabras)
```

    - **Observación**
      - Todas las **sentencias de lectura** del *contar-palabras* van **dirigidas** al fichero “datos.txt”

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-output-to-file***
    - **Sintaxis**

      *(**with-output-to-file** f p)*

      - ***f**: nombre de un fichero que se va a crear.*
      - ***p**: procedimiento sin argumentos.*
    - **Significado**

      1. Abre el fichero ***f*** y lo asocia al puerto de salida actual → *current-output-port*.
      2. Llama al procedimiento ***p***, que realiza sus operaciones de escritura en el fichero asociado al puerto de salida actual.

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-output-to-file***
    - **Ejemplo**

```
(with-output-to-file
        "salida.txt"
        (lambda () (load "hanoi.rkt")))
```

    - **Observación**
      - el procedimiento ***p*** es

```
(lambda () (load "hanoi.rkt"))
```

      - *"hanoi.rkt"*: fichero que contiene el procedimiento que resuelve el problema de la Torres de Hanoi.

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-output-to-file***
    - **Ejemplo**

```
(with-output-to-file
        "salida.txt"
        (lambda () (load "hanoi.rkt")))
```

    - **Observación**
      - ❑ Todas las **sentencias de escritura** del *hanoi.rkt* van **dirigidas** al fichero "salida.txt"

```
;; Fichero "hanoi.rkt"
;; Definición de la función "hanoi"
(define (hanoi n a b c)
        (define (cambio a b)
                (display a)
                (display " → ")
                (display b)
                (newline)
                1
        )
        (cond ( (= n 1) (cambio a b))
                (else ( +
                        (hanoi (- n 1) a  c b)
                        (cambio a b)
                        (hanoi (- n1) c b a)
                      )
                )
        )
)

;; Llamada a la función "hanoi"
   (hanoi 3 "a" "b" "c")
```

5. Interacción entre funciones y ficheros

- **Procedimientos que cambian los puertos de entrada o salida actuales**
  - ***with-output-to-file***
    - **Ejemplo**

```
(with-output-to-file "salida.txt"
                     (lambda () (load "hanoi.rkt")))
```

❑ Contenido del fichero "salida.txt"

```
A --> B
A --> C
B --> C
A --> B
C --> A
C --> B
A --> B
```

5. Interacción entre funciones y ficheros

- **Procedimientos que operan directamente con los ficheros**
  - ***call-with-input-file***
    - Utiliza un fichero de entrada.
  - ***call-with-output-file***
    - Utiliza un fichero de salida.

5. Interacción entre funciones y ficheros

- **Procedimientos que operan directamente con los ficheros**
  - ***call-with-input-file***
    - **Sintaxis**

      (call-with-input-file f p)

      - ***f***: nombre de un fichero existente.
      - ***p***: procedimiento
        que recibe como argumento
        un puerto asociado a un fichero de entrada.

5. Interacción entre funciones y ficheros

- **Procedimientos que operan directamente con los ficheros**
  - ***call-with-input-file***
    - **Significado**

      1. Abre el fichero ***f*** y lo asocia a un puerto de entrada.

      2. Llama al procedimiento ***p***,
         que realiza sus operaciones de lectura
         desde el fichero asociado
         al puerto de entrada
         abierto en el paso anterior.

- **Ejemplo**

```
(define (sumar puerto)
  (let
      ;; variables de let
      (
        ( n  (read puerto))
      )
      ;; cuerpo de let
      (cond
              ((eof-object? n) 0)
              (else (+ n (sumar puerto) )
              )
      )
  )
)

;; Llamada
(call-with-input-file "datos.txt" sumar)
```

- **Equivalencia**

```
;; Llamada
(call-with-input-file "numeros.txt" sumar)
```

  - **es equivalente a**

```
;; Se abre el fichero de entrada
;; y se asocia al puerto de entrada
(define puerto (open-input-file "numeros.txt"))

;; Se llama a la función o procedimiento
(sumar puerto)

;; Se cierra el puerto de entrada
(close-input-port puerto)
```

5. Interacción entre funciones y ficheros

- **Procedimientos que operan directamente con los ficheros**
  - ***call-with-output-file***
    - **Sintaxis**

      (call-with-output-file f p)

      - ***f***: nombre de un fichero que se va a crear
      - ***p***: procedimiento
        que recibe como argumento
        un puerto asociado a un fichero de salida.

5. Interacción entre funciones y ficheros

- **Procedimientos que operan directamente con los ficheros**
  - ***call-with-output-file***
    - **Significado**

1. Abre el fichero ***f*** y lo asocia a un puerto de salida.

2. Llama al procedimiento ***p***,
   que realiza sus operaciones de escritura
   en el fichero asociado al puerto de salida
   abierto en el paso anterior.

- **Ejemplo**

```
(define (escribir-carta puerto)
   (do
         (
          (a     (read)      (read))
         )
         ;; condición de salida
         ((eof-object?  a) (newline puerto))

         ;; cuerpo de salida
          (display a puerto)
          (display “  ” puerto)
    )
)

;; Llamada
(call-with-output-file "nota.txt" escribir-carta)
```

- **Equivalencia**

```
;; Llamada
(call-with-output-file "nota.txt" escribir-carta)
```

  - **es equivalente a**

```
;; Se abre el fichero de salida
;; y se asocia al puerto de salida
(define puerto (open-output-file "nota.txt" ))

;; Se llama a la función o procedimiento
(escribir-carta puerto)

;; Se cierra el puerto de salida
(close-output-port puerto)
```

UNIVERSDAD DE CÓRDOBA

ESCUELA POLITÉCNICA SUPERIOR DE CÓRDOBA

DEPARTAMENTO DE
INFORMÁTICA Y ANÁLISIS NUMÉRICO

PROGRAMACIÓN DECLARATIVA

INGENIGERÍA NFORMÁTICA

CUARTO CURSO

PRIMER CUATRIMESTRE

Tema 7.- Lectura y escritura