

TRABAJO DE TEORÍA

Representación Gráfica en Scheme
Asignatura de Programación Declarativa

Francisco Javier Rodríguez Lozano

Escuela Politécnica Superior
Universidad de Córdoba

Curso académico 2013 - 2014



Índice

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Índice

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Índice

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Índice

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Índice

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Índice

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Introducción

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Introducción

Origen

1 Introducción

- Origen
- Características del lenguaje
- Características Gráficas

Introducción

Origen

Surgió por el interrogante:

- ¿Cómo sería **LISP** con reglas de Ámbito Léxico o Estático?

Nota

*Originalmente llamado **Schemer**, su nombre actual se debe a que el S.O.ITS limitaba la longitud de los nombres de fichero a 6 caracteres.*

Introducción

Origen

Surgió por el interrogante:

- ¿Cómo sería **LISP** con reglas de Ámbito Léxico o Estático?

Nota

*Originalmente llamado **Schemer**, su nombre actual se debe a que el S.O.ITS limitaba la longitud de los nombres de fichero a 6 caracteres.*

Introducción

Características del lenguaje

- 1 Introducción
 - Origen
 - Características del lenguaje
 - Características Gráficas

Introducción

Características del lenguaje

A destacar:

- Implementación más eficiente de la *recursión*
- Funciones de *primera clase*.
- Reglas *semánticas* rigurosas.

Introducción

Características del lenguaje

A destacar:

- Implementación más eficiente de la *recursión*
- Funciones de *primera clase*.
- Reglas *semánticas* rigurosas.

Introducción

Características del lenguaje

A destacar:

- Implementación más eficiente de la *recursión*
- Funciones de *primera clase*.
- Reglas *semánticas* rigurosas.

Introducción

Características Gráficas

1 Introducción

- Origen
- Características del lenguaje
- Características Gráficas

Introducción

Características Gráficas

Algunas características

- Dibujar figuras sencillas
- Crear ventanas
- Interactuar con las ventanas
- Representar funciones complejas de forma sencilla
- ...

Nota

Para que sea posible realizar tareas gráficas, es necesario cargar la librería gráfica por medio de la orden:
`(require (lib "graphics.ssgraphics"))`

Introducción

Características Gráficas

Algunas características

- Dibujar figuras sencillas
- Crear ventanas
- Interactuar con las ventanas
- Representar funciones complejas de forma sencilla
- ...

Nota

Para que sea posible realizar tareas gráficas, es necesario cargar la librería gráfica por medio de la orden:
`(require (lib "graphics.ssgraphics"))`

Introducción

Características Gráficas

Algunas características

- Dibujar figuras sencillas
- Crear ventanas
- Interactuar con las ventanas
- Representar funciones complejas de forma sencilla
- ...

Nota

Para que sea posible realizar tareas gráficas, es necesario cargar la librería gráfica por medio de la orden:
`(require (lib "graphics.ssgraphics"))`

Introducción

Características Gráficas

Algunas características

- Dibujar figuras sencillas
- Crear ventanas
- Interactuar con las ventanas
- Representar funciones complejas de forma sencilla
- ...

Nota

Para que sea posible realizar tareas gráficas, es necesario cargar la librería gráfica por medio de la orden:
`(require (lib "graphics.ssgraphics"))`

Introducción

Características Gráficas

Algunas características

- Dibujar figuras sencillas
- Crear ventanas
- Interactuar con las ventanas
- Representar funciones complejas de forma sencilla
- ...

Nota

Para que sea posible realizar tareas gráficas, es necesario cargar la librería gráfica por medio de la orden:
`(require (lib "graphics.ssgraphics"))`

Introducción

Características Gráficas

Algunas características

- Dibujar figuras sencillas
- Crear ventanas
- Interactuar con las ventanas
- Representar funciones complejas de forma sencilla
- ...

Nota

Para que sea posible realizar tareas gráficas, es necesario cargar la librería gráfica por medio de la orden:
`(require (lib "graphics.ssgraphics"))`

Comandos

- 1 Introducción
- 2 Comandos**
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Comandos

Apertura y de gráficos

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- Posiciones
- Colores
- Dibujos
- Texto
- Pixmap

Comandos

Apertura y de gráficos

Inicialización:

- *(open-graphics)*
- Inicializa las rutinas de la librería gráfica.

Nota

Debe llamarse antes que ningún otro procedimiento.

Comandos

Apertura y de gráficos

Cierre:

- *(close-graphics)*
- Cierra todas las ventanas

Nota

*Después de su llamada para acceder a otro procedimiento gráfico es necesario volverlos a inicializar con *(open-graphics)*.*

Comandos

Creación y cierre de ventanas

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- Posiciones
- Colores
- Dibujos
- Texto
- Pixmap

Comandos

Creación y cierre de ventanas

Creación:

- (*open-viewport* *nombreVentana* *ancho* *alto*)
 - + *nombreVentana*: es el nombre que le daremos a la ventana .
 - + *ancho*: especifica el ancho en píxeles de la ventana.
 - + *alto*: especifica el alto en píxeles de la ventana.

Comandos

Creación y cierre de ventanas

Cierre:

- *(close-viewport nombreVentana)*
+ **nombreVentana**: es el nombre de la ventana que queremos cerrar.

Nota

Con este procedimiento cerramos solo la ventana que indicamos, es decir, las demás ventanas no sufren cambio alguno.

Comandos

Creación y cierre de ventanas

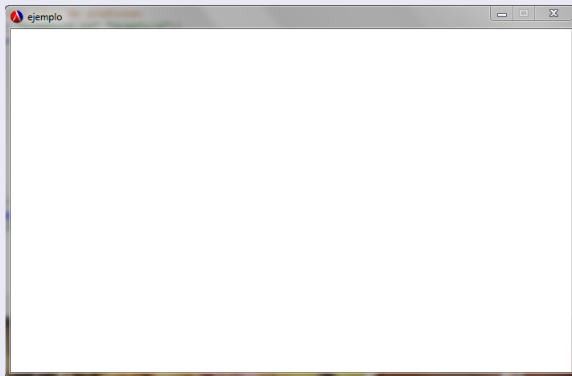
Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;; Inicializar los gráficos:  
(open-graphics)  
;; Anchura de la ventana de gráficos  
(define horizontal 700)  
;; Altura de la ventana de gráficos  
(define vertical 600)  
;; Se crea una ventana gráfica  
(define ventana (open-viewport "ejemplo" horizontal vertical) )
```

Comandos

Creación y cierre de ventanas

Ejemplo



Uso de open-viewport

Comandos

Posiciones

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- **Posiciones**
- Colores
- Dibujos
- Texto
- Pixmap

Comandos

Posiciones

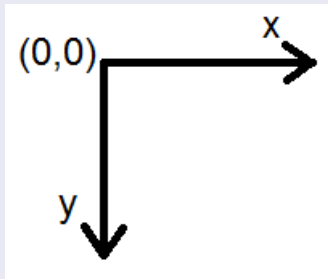
Ejes de coordenadas:

- En Scheme la posición de un objeto en la ventana es un tanto especial ya que:
 - + No existen coordenadas **negativas**.
 - + El **eje** de **abscisas** va desde el origen(lado **izquierdo** de la ventana), hasta el final de la ventana(lado **derecho** de la ventana).
 - + El **eje** de **ordenadas** es exactamente igual pero de **arriba** a **abajo**.

Comandos

Posiciones

Ejemplo



Punto origen en Scheme

Comandos

Posiciones

Objetos posición:

- *(make-posn coord_X coord_Y)*
 - + *coord_X*: coordenada X del objeto
 - + *coord_Y*: coordenada Y del objeto

Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;;cramos un objeto de tipo posición  
(define objeto (make-posn 100.0 30))
```

Comandos

Posiciones

Predicados de objetos posición:

- *(posn? objeto)*
- Devuelve *#t* si *objeto* es de tipo posición
- Devuelve *#f* en caso contrario

Obtención de coordenadas:

- *(posn-x objeto)*
 - + Devuelve la coordenada X del *objeto* de tipo posición.
- *(posn-y objeto)*
 - + Devuelve la coordenada Y del *objeto* de tipo posición.

Comandos

Posiciones

Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;; creamos un objeto de tipo posición  
(define objeto (make-posn 100.0 30))  
;; devuelve #t  
(posn? objeto)  
;; devuelve 100.0  
(posn-x objeto)  
;; devuelve 30  
(posn-y objeto)
```

Comandos

Colores

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- Posiciones
- **Colores**
- Dibujos
- Texto
- Pixmap

Comandos

Colores

Tres formas de crear colores:

- Como un *índice* de color *[0-299]*, ambos incluidos.
- Como una *cadena* que simboliza su color.
 - + p.e. "red", "blue", "green", etc.
- Como un objeto de *tipo RGB*.

Comandos

Colores

Como un objeto RGB:

- Es necesario utilizar el método *(make-rgb rojo verde azul)*
 - + Los parámetros *rojo*, *verde* y *azul* simbolizan su color con un valor comprendido entre [0-1].
- Si queremos comprobar que un objeto es de tipo RGB, solo tenemos que utilizar el predicado *(rgb? objeto)* dónde:
 - + Devuelve *#t* si *objeto* es de tipo RGB
 - + Devuelve *#f* en caso contrario

Comandos

Colores

Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;; Inicializar los gráficos:  
(open-graphics)  
;; Define un nuevo color RGB  
(define color (make-rgb 0.1 0.2 1))  
;; Devuelve #t, ya que color es RGB  
(rgb? color)
```

Comandos

Colores

Obtención de colores:

- *(rgb-red objeto)*, *(rgb-green objeto)*, *(rgb-blue objeto)*
 - + Obtiene el valor rojo, verde y azul respectivamente del objeto RGB
- *((get-pixel ventana) pixel)*:
 - + *ventana*: ventana creada con *open-viewport*.
 - + *pixel*: objeto de tipo posición creado con *make-posn*.
 - + Este predicado devuelve: (0) para denotar que el color del píxel es blanco, y (1) para representar que no es blanco

Comandos

Colores

Obtención de colores[continuación]:

- *((get-color-pixel ventana) pixel)*:
 - + Los argumentos son los mismos de antes, pero en este caso devuelve un objeto de tipo **RGB** con los colores del píxel
- *((test-pixel ventana) color)*:
 - + Devuelve un objeto de tipo **RGB**, con el color que actualmente se utiliza para dibujar.
- *(change-color indice rgb)*:
 - + Cambia el color del **indice** en la tabla de colores con el color especificado en **rgb**

Comandos

Colores

Obtención de colores[continuación]:

- *(default-display-is-color?)*:
 - + Devuelve `#t` si el dispositivo por defecto donde se mostrarán las ventanas es en color
 - + Devuelve `#f` en caso contrario

Comandos

Colores

Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;; Inicializar los gráficos:  
(open-graphics)  
;; Anchura de la ventana de gráficos  
(define horizontal 700)  
;; Altura de la ventana de gráficos  
(define vertical 600)  
;; Se verá en las secciones siguientes  
(define ventana (open-viewport "ejemplo" horizontal vertical) )  
;;....
```

Comandos

Colores

Ejemplo

```
;;....  
;;Se verá en las secciones siguientes  
((draw-line ventana)(make-posn 100.0 30) (make-posn 200 60) "red")  
;;obtenemos un píxel  
(define pixel ((get-color-pixel ventana) (make-posn 200 60)))  
;;devuelve 1  
(rgb-red pixel)  
;;devuelve 0  
(rgb-green pixel)  
;;devuelve 0  
(rgb-blue pixel)
```

Comandos

Colores

Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;;Devuelve #t, ya que la ventana es en color  
(default-display-is-color?)
```

Comandos

Colores

Colores RGB a través de índices en tablas:
























































- Una forma de pasar colores de cualquier tabla que nos den, a un valor en RGB, sería la división entre:

$$\frac{\textit{Valor_RGB_de_la_tabla}}{\textit{Valor_maximo_RGB_en_la_tabla}}$$

Comandos

Colores

Ejemplo

Interior	Font	HTML	RED	GREEN	BLUE	COLOR	Interior	Font	HTML	RED	GREEN	BLUE	COLOR
	[Color 0]	#FFFFFF	255	255	255	[Color 0]		[Color 28]	#00FFFF	0	255	255	[Color 28]
	[Color 1]	#000000	0	0	0	[Color 1]		[Color 29]	#800080	128	0	128	[Color 29]
	[Color 3]	#FF0000	255	255	255	[Color 2]		[Color 30]	#000000	128	0	0	[Color 30]
	[Color 4]	#00FF00	0	255	0	[Color 3]		[Color 31]	#008080	0	128	128	[Color 31]
	[Color 5]	#0000FF	0	0	255	[Color 4]		[Color 32]	#0000FF	0	0	255	[Color 32]
	[Color 6]	#FFFF00	255	255	0	[Color 5]		[Color 33]	#00CCFF	0	204	255	[Color 33]
	[Color 7]	#FF00FF	255	0	255	[Color 6]		[Color 34]	#CCFFFF	204	255	255	[Color 34]
	[Color 8]	#00FFFF	0	255	255	[Color 7]		[Color 35]	#CCFFCC	204	255	204	[Color 35]
	[Color 9]	#800000	128	0	0	[Color 8]		[Color 36]	#FFFF99	255	255	153	[Color 36]
	[Color 10]	#008000	0	128	0	[Color 9]		[Color 37]	#99CCFF	153	204	255	[Color 37]
	[Color 11]	#000080	0	0	128	[Color 10]		[Color 38]	#FF99CC	255	153	204	[Color 38]
	[Color 12]	#800000	128	128	0	[Color 11]		[Color 39]	#CC99FF	204	153	255	[Color 39]
	[Color 13]	#000080	128	0	128	[Color 12]		[Color 40]	#FFCC99	255	204	153	[Color 40]
	[Color 14]	#008080	0	128	128	[Color 13]		[Color 41]	#3366FF	51	102	255	[Color 41]
	[Color 15]	#C0C0C0	192	192	192	[Color 14]		[Color 42]	#33CCCC	51	204	204	[Color 42]
	[Color 16]	#808080	128	128	128	[Color 15]		[Color 43]	#99CC00	153	204	0	[Color 43]
	[Color 17]	#9999FF	153	153	255	[Color 16]		[Color 44]	#FFCC00	255	204	0	[Color 44]
	[Color 18]	#993366	153	51	102	[Color 17]		[Color 45]	#FF9900	255	153	0	[Color 45]
	[Color 19]	#00FFCC	255	255	204	[Color 18]		[Color 46]	#FF6600	255	102	0	[Color 46]
	[Color 20]	#CCFFFF	204	255	255	[Color 19]		[Color 47]	#666699	102	102	153	[Color 47]
	[Color 21]	#660066	102	0	102	[Color 20]		[Color 48]	#996699	150	150	150	[Color 48]
	[Color 22]	#FF8080	255	128	128	[Color 21]		[Color 49]	#003366	0	51	102	[Color 49]
	[Color 23]	#0066CC	0	102	204	[Color 22]		[Color 50]	#339966	51	153	102	[Color 50]
	[Color 24]	#CCCCFF	204	204	255	[Color 23]		[Color 51]	#003300	0	51	0	[Color 51]
	[Color 25]	#000080	0	0	128	[Color 24]		[Color 52]	#333300	51	51	0	[Color 52]
	[Color 26]	#FF00FF	255	0	255	[Color 25]		[Color 53]	#993300	153	51	0	[Color 53]
	[Color 27]	#FFFF00	255	255	0	[Color 26]		[Color 54]	#993366	153	51	102	[Color 54]
						[Color 27]		[Color 55]	#333399	51	51	153	[Color 55]

Ejemplo tabla colores con índices

Comandos

Colores

Ejemplo

Para un color con *R:0 G:128 B:0* de la tabla anterior, obtenemos que el valor en RGB para Scheme es:

$$\frac{0}{255}, \frac{128}{255}, \frac{0}{255} = [0, 0,50, 0]$$

Comandos

Dibujos

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- Posiciones
- Colores
- Dibujos
- Texto
- Pixmap

Comandos

Dibujos

Dibujando en ventanas:

- `((draw-viewport ventana) [color])`
 - + Dibuja una *ventana* dada, del *color* especificado o si se omite se dibuja de color negro.
- `((clear-viewport ventana))`
 - + Borra el contenido de una *ventana* dada.

Comandos

Dibujos

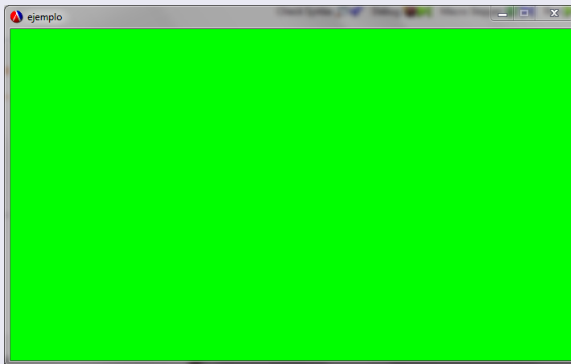
Ejemplo

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
;; Inicializar los gráficos:  
(open-graphics)  
;; Anchura de la ventana de gráficos  
(define horizontal 700)  
;; Altura de la ventana de gráficos  
(define vertical 600)  
;; Se crea una ventana gráfica  
(define ventana (open-viewport "ejemplo" horizontal vertical) )  
((draw-viewport ventana) "green")
```

Comandos

Dibujos

Ejemplo



Resultado draw-viewport

Comandos

Dibujos

Nota

A partir de ésta página, como los parámetros de dibujar y eliminar son equivalentes no se explicarán los de esta última

Nota

*Para dibujar se puede utilizar las funciones **draw**, como las que utilizan **flip** preo su resultado al dibujar y eliminar es el mismo.*

Comandos

Dibujos

Dibujando píxeles:

- `((draw-pixel ventana) posicion [color])`
 - + `ventana`: ventana creada con *open-viewport*.
 - + `posicion`: objeto de tipo posición creado con *make-posn*.
 - + `[color]`: parámetro opcional que representa el color del píxel, por defecto es negro.
- `((clear-pixel ventana) posicion)`

Comandos

Dibujos

Ejemplo

```
;;Carga y definición de la ventana como antes
```

```
;;....
```

```
;;
```

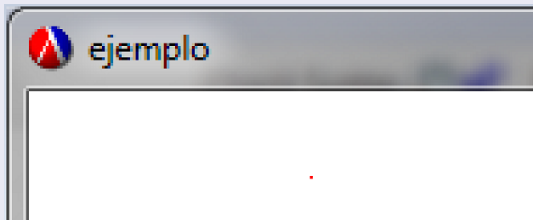
```
;;Dibujamos un pixel
```

```
((draw-pixel ventana)(make-posn 100.0 30) "red")
```

Comandos

Dibujos

Ejemplo



Píxel en Scheme

Comandos

Dibujos

Dibujando líneas:

- `((draw-line ventana) posicionA posicionB [color])`
 - + `ventana`: ventana creada con *open-viewport*.
 - + `posicionA`: objeto de tipo posición creado con *make-posn*.
 - + `posicionB`: objeto de tipo posición creado con *make-posn*.
 - + `[color]`: parámetro opcional que representa el color de la línea, por defecto es negro.
- `((clear-line ventana) posicionA posicionB)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

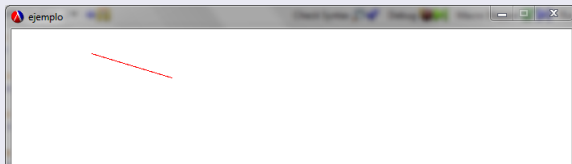
;;Dibujamos una linea

```
((draw-line ventana)(make-posn 100.0 30) (make-posn 200 60)  
"red")
```

Comandos

Dibujos

Ejemplo



Línea en Scheme

Comandos

Dibujos

Dibujando rectángulos:

- `((draw-rectangle ventana) posicion altura anchura [color])`
 - + **ventana**: ventana creada con *open-viewport*.
 - + **posicion**: objeto de tipo posición creado con *make-posn*.
 - + **altura**: dato de tipo numérico que representa la altura.
 - + **anchura**: dato de tipo numérico que representa la anchura.
 - + **[color]**: parámetro opcional que representa el color del rectángulo, por defecto es negro.
- `((clear-rectangle ventana) posicion altura anchura)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

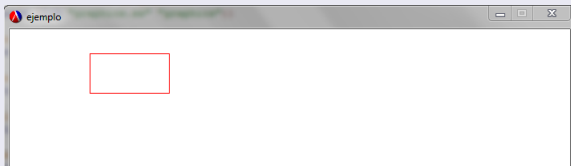
;;Dibujamos un rectángulo

((draw-rectangle ventana)(make-posn 100.0 30) 100 50 "red")

Comandos

Dibujos

Ejemplo



Rectángulo en Scheme

Comandos

Dibujos

Dibujando rectángulos sólidos:

- `((draw-solid-rectangle ventana) posicion altura anchura [color])`
- + *ventana*: ventana creada con *open-viewport*.
- + *posicion*: objeto de tipo posición creado con *make-posn*.
- + *altura*: dato de tipo numérico que representa la altura.
- + *anchura*: dato de tipo numérico que representa la anchura.
- + [*color*]: parámetro opcional que representa el color del rectángulo, por defecto es negro.
- `((clear-solid-rectangle ventana) posicion altura anchura)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

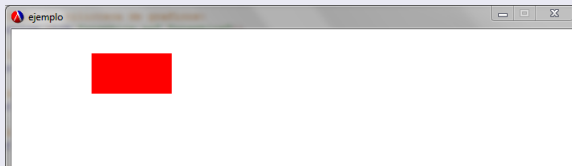
;;Dibujamos un rectángulo sólido

```
((draw-solid-rectangle ventana)(make-posn 100.0 30) 100 50  
"red")
```

Comandos

Dibujos

Ejemplo



Rectángulo sólido en Scheme

Comandos

Dibujos

Dibujando elipses:

- `((draw-ellipse ventana) posicion altura anchura [color])`
- + *ventana*: ventana creada con *open-viewport*.
- + *posicion*: objeto de tipo posición creado con *make-posn*.
- + *altura*: dato de tipo numérico que representa la altura.
- + *anchura*: dato de tipo numérico que representa la anchura.
- + [*color*]: parámetro opcional que representa el color de la elipse, por defecto es negro.
- `((clear-ellipse ventana) posicion altura anchura)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

;;Dibujamos una elipse

((draw-ellipse ventana)(make-posn 100.0 30) 50 100 "red")

Comandos

Dibujos

Ejemplo



Elipse en Scheme

Comandos

Dibujos

Dibujando elipses sólidas:

- `((draw-solid-ellipse ventana) posicion altura anchura [color])`
 - + **ventana**: ventana creada con *open-viewport*.
 - + **posicion**: objeto de tipo posición creado con *make-posn*.
 - + **altura**: dato de tipo numérico que representa la altura.
 - + **anchura**: dato de tipo numérico que representa la anchura.
 - + **[color]**: parámetro opcional que representa el color de la elipse, por defecto es negro.
- `((clear-solid-ellipse ventana) posicion altura anchura)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

;;Dibujamos una elipse sólida

((draw-solid-ellipse ventana)(make-posn 100.0 30) 50 100 "green")

Comandos

Dibujos

Ejemplo



Elipse sólida en Scheme

Comandos

Dibujos

Dibujando polígonos:

- `((draw-polygon ventana) x-posiciones posicion [color])`
 - + **ventana**: ventana creada con *open-viewport*.
 - + **x-posiciones**: Lista de objetos de tipo posición creados con *make-posn*.
 - + **posicion**: objeto de tipo posición creado con *make-posn*, considerado como desplazamiento.
 - + **[color]**: parámetro opcional que representa el color del polígono, por defecto es negro.
- `((clear-polygon ventana) x-posiciones posicion)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

;;Dibujamos un polígono

```
((draw-polygon ventana) (list (make-posn 10 20)
                               (make-posn 1 30)
                               (make-posn 3 40)
                               (make-posn 100 20))
 (make-posn 100.0 30) "red")
```

Comandos

Dibujos

Ejemplo



Polígono de cuatro vértices en Scheme

Comandos

Dibujos

Dibujando polígonos sólidos:

- `((draw-solid-polygon ventana) x-posiciones posicion [color])`
 - + **ventana**: ventana creada con *open-viewport*.
 - + **x-posiciones**: Lista de objetos de tipo posición creados con *make-posn*.
 - + **posicion**: objeto de tipo posición creado con *make-posn*, considerado como desplazamiento.
 - + **[color]**: parámetro opcional que representa el color del polígono, por defecto es negro.
- `((clear-solid-polygon ventana) x-posiciones posicion)`

Comandos

Dibujos

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

;;Dibujamos un polígono sólido

```
((draw-solid-polygon ventana) (list (make-posn 10 20)
                                     (make-posn 1 30)
                                     (make-posn 3 40)
                                     (make-posn 100 20))
 (make-posn 100.0 30) "red")
```

Comandos

Dibujos

Ejemplo



Polígono sólido de cuatro vértices en Scheme

Comandos

Texto

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- Posiciones
- Colores
- Dibujos
- **Texto**
- Pixmap

Comandos

Texto

Escribiendo cadenas de texto:

- `((draw-string ventana) posicion cadena [color])`
 - + **ventana**: ventana creada con *open-viewport*.
 - + **posicion**: objeto de tipo posición creado con *make-posn*, considerado como el vértice inferior izquierdo donde empezará la cadena.
 - + **cadena**: cadena de caracteres que se dibujará en la ventana.
 - + **[color]**: parámetro opcional que representa el color de la cadena, por defecto es negro.
- `((clear-string ventana) posicion cadena)`

Comandos

Texto

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

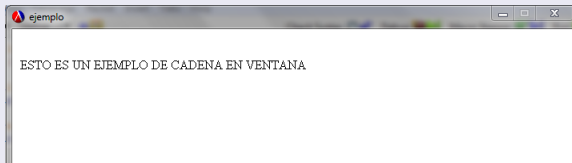
;;Dibujamos una cadena de texto

*((draw-string ventana) (make-posn 10 50) "ESTO ES UN EJEMPLO DE
CADENA EN VENTANA")*

Comandos

Texto

Ejemplo



Cadena de texto en ventana

Comandos

Texto

Tamaño de cadenas de texto:

- `((get-string-size ventana) cadena)`
 - + **ventana**: ventana creada con *open-viewport*.
 - + **cadena**: cadena de caracteres que se dibujará en la ventana.
- Este método devuelve una lista con la altura y anchura que ocupa la cadena.

Comandos

Texto

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

;;Obtenemos el tamaño de una cadena en la ventana

((get-string-size ventana) "ESTO ES UN EJEMPLO DE CADENA EN VENTANA")

;;devuelve (357.0 19.0)

Comandos

Pixmap

2 Comandos

- Apertura y de gráficos
- Creación y cierre de ventanas
- Posiciones
- Colores
- Dibujos
- Texto
- Pixmap

Comandos

Pixmap

Apertura de ventanas pixmap:

- *(open-pixmap nombreVentana ancho alto)*
 - + **nombreVentana**: es el nombre que le daremos a la ventana de pixmap.
 - + **ancho**: especifica el ancho en píxeles de la ventana de pixmap.
 - + **alto**: especifica el alto en píxeles de la ventana de pixmap.

Comandos

Pixmap

Cierre:

- *(close-pixmap nombreVentana)*
+ **nombreVentana**: es el nombre de la ventana de pixmap que queremos cerrar.

Nota

Con este procedimiento cerramos solo la ventana pixmap que indicamos, es decir, las demás ventanas no sufren cambio alguno.

Comandos

Pixmap

Dibujando en ventanas Pixmap:

- `((draw-pixmap-posn nombreFichero [tipo] ventana)
posicion [color])`
 - + **nombreFichero**: nombre del fichero que queremos pintar en la ventana pixmap.
 - + **[tipo]**: tipo de formato del fichero. Los tipos soportados son:
 - * `'gif`.
 - * `'xbm`.
 - * `'xpm`.
 - * `'bmp`.
 - * `'pict`, no está disponible su uso en sistemas operativos MAC.
 - * `'unknown`, que es el tipo por defecto.

Comandos

Pixmap

gg easy

Dibujando en ventanas Pixmap[continuación]:

- `((draw-pixmap-posn nombreFichero [tipo]) ventana)`
`posicion [color])`
 - + **ventana**: ventana pixmap creada con (*open-pixmap*).
 - + **posicion**: objeto de tipo posición creado con *make-posn*.
 - + **[color]**: se utiliza solo cuando la ventana pixmap es blanca o negra.
- `((draw-pixmap ventana) nombreFichero posicion [color])`
 - + Esta función es equivalente a la anterior, lo único que tiene especial, es que muestra una sintaxis más reducida.

Comandos

Pixmap

Aclaraciones:

- Las ventanas pixmap no se muestran por pantalla, con el uso de la siguiente función podemos mostrar en una ventana creada con *open-viewport* el resultado:
 - + (*copy-viewport* *ventana2* *ventana*)
 - + *ventana*: ventana creada con (*open-pixmap*) o ventana pixmap creada con (*open-pixmap*).
 - + *ventana2*: ventana creada con (*open-pixmap*) o ventana pixmap creada con (*open-pixmap*).
 - + El objetivo de esta función es copiar el contenido de *ventana*, a *ventana2*.

Comandos

Pixmap

Ejemplo

;;Carga y definición de la ventana como antes

;;....

;;

;; Se crea una ventana pixmap

(define ventana2 (open-pixmap "ejemplo2" horizontal vertical))

;;dibujamos la imagen a.png

((draw-pixmap ventana2) "a.png" (make-posn 0.0 0.0) "black")

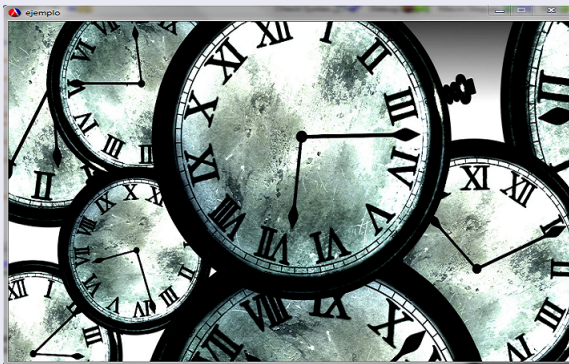
;;copiamos el contenido de una ventana a otra

(copy-viewport ventana2 ventana)

Comandos

Pixmap

Ejemplo



Ejemplo imagen cargada como Pixmap

Otros comandos

- 1 Introducción
- 2 Comandos
- 3 Otros comandos**
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Otros comandos

Procedimientos con el ratón

- 3 Otros comandos
 - Procedimientos con el ratón
 - Procedimientos con el teclado

Otros comandos

Procedimientos con el ratón

Obtención de clic's:

- *(get-mouse-click ventana)* y *(ready-mouse-click ventana)*
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un descriptor en el caso de que se haga clic.
 - + Devuelve *#f* en caso contrario.

Nota

La diferencia es que el primero espera a que se haga clic y el segundo devuelve directamente el valor se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's:

- *(get-mouse-click ventana)* y *(ready-mouse-click ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un descriptor en el caso de que se haga clic.
 - + Devuelve *#f* en caso contrario.

Nota

La diferencia es que el primero espera a que se haga clic y el segundo devuelve directamente el valor se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's:

- *(get-mouse-click ventana)* y *(ready-mouse-click ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un descriptor en el caso de que se haga clic.
 - + Devuelve *#f* en caso contrario.

Nota

La diferencia es que el primero espera a que se haga clic y el segundo devuelve directamente el valor se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's:

- *(get-mouse-click ventana)* y *(ready-mouse-click ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un descriptor en el caso de que se haga clic.
 - + Devuelve **#f** en caso contrario.

Nota

La diferencia es que el primero espera a que se haga clic y el segundo devuelve directamente el valor se haga o no clic.

Otros comandos

Prodecimientos con el ratón

Obtención de clic's:

- *(get-mouse-click ventana)* y *(ready-mouse-click ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un descriptor en el caso de que se haga clic.
 - + Devuelve **#f** en caso contrario.

Nota

La diferencia es que el primero espera a que se haga clic y el segundo devuelve directamente el valor se haga o no clic.

Otros comandos

Prodecimientos con el ratón

Obtención de clic's [continuación]:

- *(ready-mouse-release ventana)*
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un evento asociado al clic, como dejar de hacer clic (*button-down*), o hacer clic (*button-up*).
 - + Devuelve *#f* en caso contrario.

Nota

Al igual que con (ready-mouse-click), se comprueba directamente se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(ready-mouse-release ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un evento asociado al clic, como dejar de hacer clic (*button-down*), o hacer clic (*button-up*).
 - + Devuelve *#f* en caso contrario.

Nota

Al igual que con (ready-mouse-click), se comprueba directamente se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(ready-mouse-release ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un evento asociado al clic, como dejar de hacer clic (*button-down*), o hacer clic (*button-up*).
 - + Devuelve *#f* en caso contrario.

Nota

Al igual que con (ready-mouse-click), se comprueba directamente se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(ready-mouse-release ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un evento asociado al clic, como dejar de hacer clic (*button-down*), o hacer clic (*button-up*).
 - + Devuelve **#f** en caso contrario.

Nota

Al igual que con (ready-mouse-click), se comprueba directamente se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(ready-mouse-release ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un evento asociado al clic, como dejar de hacer clic (*button-down*), o hacer clic (*button-up*).
 - + Devuelve **#f** en caso contrario.

Nota

Al igual que con (ready-mouse-click), se comprueba directamente se haga o no clic.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(query-mouse-posn ventana)*
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- *(left-mouse-click? mouse-click)*, *(middle-mouse-click? mouse-click)*, *(right-mouse-click? mouse-click)*
 - + *mouse-click*: descriptor devuelto por *(get-mouse-click)* o *(ready-mouse-click)*.
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(query-mouse-posn ventana)*
 - + **ventana**: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve **#f** en caso contrario.
- *(left-mouse-click? mouse-click)*, *(middle-mouse-click? mouse-click)*, *(right-mouse-click? mouse-click)*
 - + **mouse-click**: descriptor devuelto por *(get-mouse-click)* o *(ready-mouse-click)*.
 - + Devuelve **#t** si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve **#f** en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- (*query-mouse-posn* *ventana*)
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- (*left-mouse-click?* *mouse-click*), (*middle-mouse-click?* *mouse-click*), (*right-mouse-click?* *mouse-click*)
 - + *mouse-click*: descriptor devuelto por (*get-mouse-click*) o (*ready-mouse-click*).
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- (*query-mouse-posn* *ventana*)
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- (*left-mouse-click?* *mouse-click*), (*middle-mouse-click?* *mouse-click*), (*right-mouse-click?* *mouse-click*)
 - + *mouse-click*: descriptor devuelto por (*get-mouse-click*) o (*ready-mouse-click*).
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(query-mouse-posn ventana)*
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- *(left-mouse-click? mouse-click)*, *(middle-mouse-click? mouse-click)*, *(right-mouse-click? mouse-click)*
 - + *mouse-click*: descriptor devuelto por *(get-mouse-click)* o *(ready-mouse-click)*.
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- (*query-mouse-posn* *ventana*)
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- (*left-mouse-click?* *mouse-click*), (*middle-mouse-click?* *mouse-click*), (*right-mouse-click?* *mouse-click*)
 - + *mouse-click*: descriptor devuelto por (*get-mouse-click*) o (*ready-mouse-click*).
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(query-mouse-posn ventana)*
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- *(left-mouse-click? mouse-click)*, *(middle-mouse-click? mouse-click)*, *(right-mouse-click? mouse-click)*
 - + *mouse-click*: descriptor devuelto por *(get-mouse-click)* o *(ready-mouse-click)*.
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el ratón

Obtención de clic's [continuación]:

- *(query-mouse-posn ventana)*
 - + *ventana*: ventana creada con *open-viewport*.
 - + Devuelve un objeto de tipo posición como ya vimos en su momento, con las coordenadas donde se ha hecho clic
 - + Devuelve *#f* en caso contrario.
- *(left-mouse-click? mouse-click)*, *(middle-mouse-click? mouse-click)*, *(right-mouse-click? mouse-click)*
 - + *mouse-click*: descriptor devuelto por *(get-mouse-click)* o *(ready-mouse-click)*.
 - + Devuelve *#t* si el clic se ha hecho con el botón derecho, medio, izquierdo respectivamente
 - + Devuelve *#f* en caso contrario.

Otros comandos

Procedimientos con el teclado

- 3 Otros comandos
 - Procedimientos con el ratón
 - Procedimientos con el teclado

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado:

- `(get-key-press ventana)` y `(ready-key-press ventana)`
 - + `ventana`: ventana creada con `open-viewport`.
 - + Devuelve un descriptor en el caso de que se pulse una tecla.
 - + Devuelve `#f` en caso contrario.

Nota

La diferencia es que el primero espera a que se pulse una tecla y el segundo, devuelve directamente el valor se pulse dicha tecla o no.

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado:

- `(get-key-press ventana)` y `(ready-key-press ventana)`
 - + **ventana**: ventana creada con `open-viewport`.
 - + Devuelve un descriptor en el caso de que se pulse una tecla.
 - + Devuelve `#f` en caso contrario.

Nota

La diferencia es que el primero espera a que se pulse una tecla y el segundo, devuelve directamente el valor se pulse dicha tecla o no.

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado:

- `(get-key-press ventana)` y `(ready-key-press ventana)`
 - + **ventana**: ventana creada con `open-viewport`.
 - + Devuelve un descriptor en el caso de que se pulse una tecla.
 - + Devuelve `#f` en caso contrario.

Nota

La diferencia es que el primero espera a que se pulse una tecla y el segundo, devuelve directamente el valor se pulse dicha tecla o no.

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado:

- `(get-key-press ventana)` y `(ready-key-press ventana)`
 - + **ventana**: ventana creada con `open-viewport`.
 - + Devuelve un descriptor en el caso de que se pulse una tecla.
 - + Devuelve `#f` en caso contrario.

Nota

La diferencia es que el primero espera a que se pulse una tecla y el segundo, devuelve directamente el valor se pulse dicha tecla o no.

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado:

- `(get-key-press ventana)` y `(ready-key-press ventana)`
 - + `ventana`: ventana creada con `open-viewport`.
 - + Devuelve un descriptor en el caso de que se pulse una tecla.
 - + Devuelve `#f` en caso contrario.

Nota

La diferencia es que el primero espera a que se pulse una tecla y el segundo, devuelve directamente el valor se pulse dicha tecla o no.

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado [continuación]:

- *(key-value TeclaPulsada)*
 - + *TeclaPulsada*: descriptor devuelto por *(get-key-press ventana)* o *(ready-key-press ventana)*.
 - + Devuelve la tecla pulsada.
 - + Devuelve *#f* en caso contrario.

Nota

Para más información sobre los posibles valores devueltos mirar [aquí](#).

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado [continuación]:

- *(key-value TeclaPulsada)*
 - + **TeclaPulsada**: descriptor devuelto por *(get-key-press ventana)* o *(ready-key-press ventana)*.
 - + Devuelve la tecla pulsada.
 - + Devuelve **#f** en caso contrario.

Nota

Para más información sobre los posibles valores devueltos mirar [aquí](#).

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado [continuación]:

- *(key-value TeclaPulsada)*
 - + **TeclaPulsada**: descriptor devuelto por *(get-key-press ventana)* o *(ready-key-press ventana)*.
 - + Devuelve la tecla pulsada.
 - + Devuelve **#f** en caso contrario.

Nota

Para más información sobre los posibles valores devueltos mirar [aquí](#).

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado [continuación]:

- *(key-value TeclaPulsada)*
 - + **TeclaPulsada**: descriptor devuelto por *(get-key-press ventana)* o *(ready-key-press ventana)*.
 - + Devuelve la tecla pulsada.
 - + Devuelve **#f** en caso contrario.

Nota

Para más información sobre los posibles valores devueltos mirar [aquí](#).

Otros comandos

Procedimientos con el teclado

Obtención de pulsaciones en teclado [continuación]:

- *(key-value TeclaPulsada)*
 - + **TeclaPulsada**: descriptor devuelto por *(get-key-press ventana)* o *(ready-key-press ventana)*.
 - + Devuelve la tecla pulsada.
 - + Devuelve **#f** en caso contrario.

Nota

Para más información sobre los posibles valores devueltos mirar [aquí](#).

Crear ejecutables

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables**
- 5 Ejemplo: Torres de Hanoi
- 6 Referencias

Crear ejecutables

Introducción

- 4 Crear ejecutables
 - Introducción
 - ¿Qué es necesario?
 - Pasos a seguir

Crear ejecutables

Introducción

¿Cómo crear?

- Dos formas de crear ejecutables:
 - + Utilizando el comando `raco`.
 - + Con ayuda del `intérprete`.

Nota

Sólo se explicará la creación por medio del intérprete.

Crear ejecutables

¿Qué es necesario?

- 4 Crear ejecutables
 - Introducción
 - ¿Qué es necesario?
 - Pasos a seguir

Crear ejecutables

¿Qué es necesario?

Necesitamos

- Incluir al inicio de nuestro fichero la siguiente línea:
 - + `#lang racket`
- Seleccionar el siguiente lenguaje:
 - + `Determine language from source`

Crear ejecutables

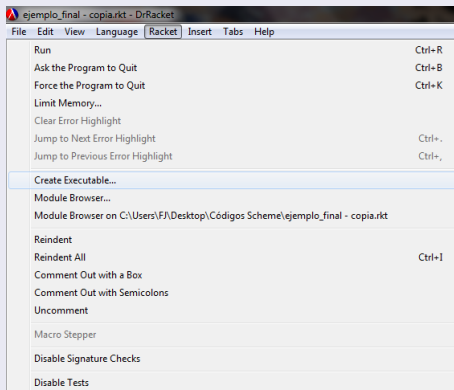
Pasos a seguir

- 4 Crear ejecutables
 - Introducción
 - ¿Qué es necesario?
 - Pasos a seguir

Crear ejecutables

Pasos a seguir

Paso 1:

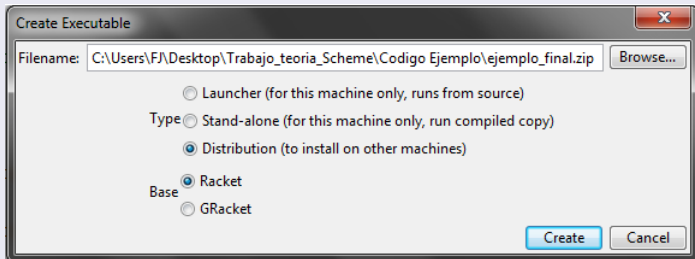


Crear ejecutables

Pasos a seguir

Paso 2:

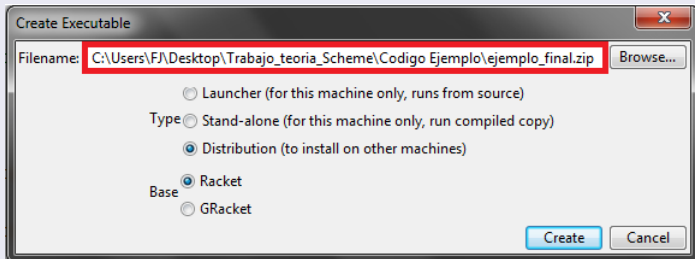
- Al terminar el paso 1, nos saldrá una ventana como la siguiente:



Crear ejecutables

Pasos a seguir

Paso 2 [parte 1/3]:

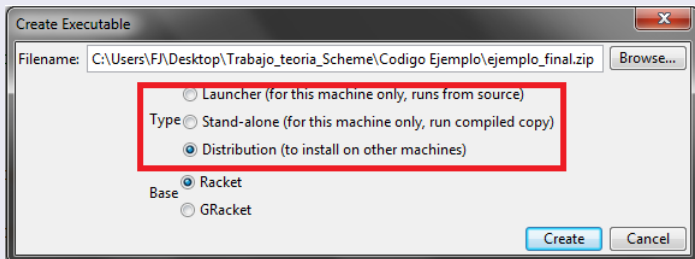


- Seleccionamos dónde queremos guardar el fichero **.zip** con el **ejecutable**

Crear ejecutables

Pasos a seguir

Paso 2 [parte 2/3]:

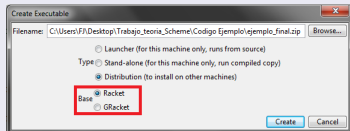


- Seleccionamos la portabilidad del [ejecutable](#)

Crear ejecutables

Pasos a seguir

Paso 2 [parte 3/3]:



- Seleccionamos el **lenguaje base** de compilación

Crear ejecutables

Pasos a seguir

Nota

*En el caso de que el ejecutable tenga referencias a **elementos externos** como por ejemplo imágenes, necesitamos incluirlos en el archivo **.zip** a mano, ya que este proceso sólo compila el código dejando atrás **los elementos externos**.*

Ejemplo: Torres de Hanoi

- 1 Introducción
- 2 Comandos
- 3 Otros comandos
- 4 Crear ejecutables
- 5 Ejemplo: Torres de Hanoi**
- 6 Referencias

Ejemplo: Torres de Hanoi

Historia

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - Complejidad
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Historia

¿Qué es y quién lo inventó?

- Las torres de Hanoi es un problema matemático.
- Inventado en 1883 por el matemático francés Édouard Lucas.



Édouard Lucas

Ejemplo: Torres de Hanoi

Reglas

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - Complejidad
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Reglas

Problema clásico

- El objetivo es mover una serie de discos que tienen diferentes tamaños de una varilla a otra final.

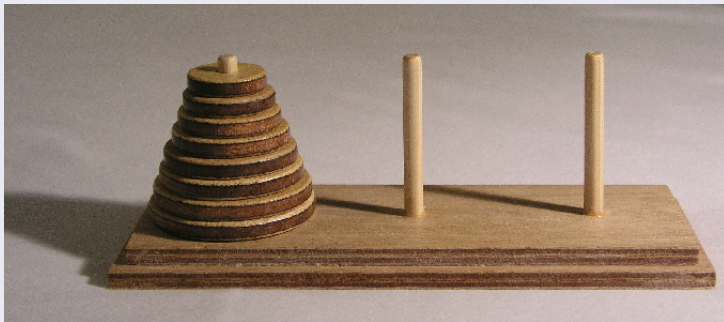
Limitaciones

- Sólo se tienen tres varillas disponibles
- Un disco de mayor tamaño no puede colocarse sobre uno más pequeño.
- Sólo se puede mover el disco que se encuentre arriba en cada varilla.
- En cada paso sólo se permite el movimiento de un disco, es decir, se prohíbe mover varios discos a la vez.

Ejemplo: Torres de Hanoi

Reglas

Ejemplo



Juego real de madera

Ejemplo: Torres de Hanoi

Formas de Implementar

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - Complejidad
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Formas de Implementar

Hay dos implementaciones:

- Implementación *Recursiva*.
- Implementación *Iterativa*.

Nota

*Solo se verá la versión *Recursiva*, ya que el ejemplo final está preparado con esta implementación.*

Ejemplo: Torres de Hanoi

Formas de Implementar

Hay dos implementaciones:

- Implementación *Recursiva*.
- Implementación *Iterativa*.

Nota

*Solo se verá la versión *Recursiva*, ya que el ejemplo final está preparado con esta implementación.*

Ejemplo: Torres de Hanoi

Formas de Implementar

Hay dos implementaciones:

- Implementación *Recursiva*.
- Implementación *Iterativa*.

Nota

*Solo se verá la versión *Recursiva*, ya que el ejemplo final está preparado con esta implementación.*

Ejemplo: Torres de Hanoi

Pseudocódigo

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - Complejidad
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Pseudocódigo

Entrada: Tres pilas de números *origen*, *auxiliar*, *destino*, con la pila *origen* ordenada

Salida: La pila *destino*

1. **si** *origen* == {1} **entonces**

1. **mover** el disco 1 de pila *origen* a la pila *destino* (insertarlo arriba de la pila *destino*)
2. **terminar**

2. **si no**

//mover todas las fichas menos la más grande (*n*) a la varilla auxiliar

1. *hanoi*({1,...,*n*-1}, *destino*, *auxiliar*)

//mover la ficha grande hasta la varilla final

3. **mover** disco *n* a *destino*

//mover todas las fichas restantes, 1...*n*-1, encima de la ficha grande (*n*)

4. *hanoi* (*auxiliar*, *origen*, *destino*)

5. **terminar**

Ejemplo: Torres de Hanoi

Complejidad

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - **Complejidad**
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Complejidad

¿Es sencillo?

- Aunque no lo parezca es un problema complejo, en el que el número de movimientos crece de forma exponencial.

Discos	Movimientos
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

Movimientos para varios discos

Ejemplo: Torres de Hanoi

Implementación en Scheme

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - Complejidad
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (1/15)

```
;; Carga la biblioteca de graficos:  
(require (lib "graphics.ss" "graphics"))  
  
;; Inicializar los graficos:  
(open-graphics)  
  
;; Anchura de la ventana de graficos  
(define horizontal 700)  
  
;; Altura de la ventana de graficos  
(define vertical 600)  
  
;; Se crea una ventana grafica  
(define v1 (open-viewport "TORRES HANOI RECURSIVO" horizontal vertical))  
  
;; Se crea una ventana pixmap  
(define v2 (open-pixmap "TORRES HANOI RECURSIVO" horizontal vertical))
```


Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (2/15)

```
;;VARIABLE GLOBAL PARA DEFINIR EL NUMERO DE DISCOS DEL PROBLEMA
(define NUM_TOTAL_DISCOS 0)

;;VARIABLE GLOBAL PARA CONTROLAR EL CORRECTO FUNCIONAMIENTO DEL BOTN DE CERRAR
(define flag-close #f)

;;VARIABLE GLOBAL PARA CONTROLAR EL NUMERO DE PASOS
(define numPasos 0)

;;definir posición donde se dibujaran los palos
(define origen_coordenadas_y (+ (/ vertical 2) 30))
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (3/15)

```
;;FUNCIÓN DIBUJAR ELIPSES DADO SU CENTRO, LARGO, ANCHO Y COLOR
(define (ellipse-centro viewport centro largo ancho color)
  ((draw-solid-ellipse viewport) (make-posn (- (posn-x centro) (/ largo 2))
                                              (- (posn-y centro) (/ ancho 2)))
    largo ancho color))

;;FUNCIÓN BORRAR ELIPSES DADO SU CENTRO, LARGO, ANCHO
(define (eliminar-ellipse-centro viewport centro largo ancho)
  ((clear-solid-ellipse viewport) (make-posn (- (posn-x centro) (/ largo 2))
                                              (- (posn-y centro) (/ ancho 2)))
    largo ancho))
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (4/15)

```
;;CODIGO PARA REALIZAR EL PAUSE
(define (pause-or-exit click)
  (cond
    ;;botón de salir
    ((and (<= 560 (posn-x (mouse-click-posn click)) 650)
          (<= 480 (posn-y (mouse-click-posn click)) 570)) (close-graphics) (set! flag-close #t))

    ;;botón de continuar
    ((and (<= 200 (posn-x (mouse-click-posn click)) 500)
          (<= 499 (posn-y (mouse-click-posn click)) 546)))

    ;;no continuamos hasta que pulsamos una de las dos opciones
    (else (pause-or-exit (get-mouse-click vl)))
  )
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (5/15)

```
;;CODIGO PARA REALIZAR EL PAUSE
(define (display-pasos numPasos)
  ;Borramos el numero anterior
  ((draw-solid-rectangle v1) (make-posn 65 508) 50 30 "white")

  ;escribimos el nuevo número
  ((draw-string v1) (make-posn 85 525) (number->string numPasos))
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (6/15)

```
;;DIBUJA LOS EJES DE CADA TORRE
(define (pintar-ejes viewport)
  ((draw-line vl)
   (make-posn 150.0 200.0)
   (make-posn 150.0 (- origen_coordenadas_y (* (- 6 NUM_TOTAL_DISCOS) 20)))
   (make-rgb 0.0 0.0 0.0)
  )

  ((draw-line vl)
   (make-posn 350.0 200.0)
   (make-posn 350.0 (- origen_coordenadas_y (* (- 6 NUM_TOTAL_DISCOS) 20)))
   (make-rgb 0.0 0.0 0.0)
  )

  ((draw-line vl)
   (make-posn 550.0 200.0)
   (make-posn 550.0 (- origen_coordenadas_y (* (- 6 NUM_TOTAL_DISCOS) 20)))
   (make-rgb 0.0 0.0 0.0)
  )

  ;; Se dibujan los palos en el eje horizontal
  ((draw-solid-rectangle vl) (make-posn 100.0 (- origen_coordenadas_y (* (- 6 NUM_TOTAL_DISCOS) 20))) 100 10)
  ((draw-solid-rectangle vl) (make-posn 300.0 (- origen_coordenadas_y (* (- 6 NUM_TOTAL_DISCOS) 20))) 100 10)
  ((draw-solid-rectangle vl) (make-posn 500.0 (- origen_coordenadas_y (* (- 6 NUM_TOTAL_DISCOS) 20))) 100 10)
  )
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (7/15)

```
;;DIBUJAR INTERFAZ DEL PROGRAMA
(define (dibujar-Interfaz v1)
  ;;se dibujan los botones para controlar el programa.
  ((draw-rectangle v1) (make-posn 100 500) 30 30 "red");cuadro donde se escribe "1"
  ((draw-rectangle v1) (make-posn 132 500) 30 30 "red");cuadro donde se escribe "2"
  ((draw-rectangle v1) (make-posn 164 500) 30 30 "red");cuadro donde se escribe "3"
  ((draw-rectangle v1) (make-posn 100 531) 30 30 "red");cuadro donde se escribe "4"
  ((draw-rectangle v1) (make-posn 132 531) 30 30 "red");cuadro donde se escribe "5"
  ((draw-rectangle v1) (make-posn 164 531) 30 30 "red");cuadro donde se escribe "6"

  ;;se dibujan los botones
  ((draw-string v1) (make-posn 100 490) "DISCOS A PONER: ")
  ((draw-string v1) (make-posn 110 520) "1")
  ((draw-string v1) (make-posn 142 520) "2")
  ((draw-string v1) (make-posn 174 520) "3")
  ((draw-string v1) (make-posn 110 552) "4")
  ((draw-string v1) (make-posn 142 552) "5")
  ((draw-string v1) (make-posn 174 552) "6")

  ;;se muestra el texto para salir
  ((draw-string v1) (make-posn 590 490) "SALIR: ")

  ;;se comprueba donde hacemos clic
  (define click (get-mouse-click v1))
  (Comprobar-Click v1 click)
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (8/15)

```
;;DIBUJAR INTERFAZ DEL PROGRAMA UNA VEZ SELECCIONADOS EL NUMERO DE DISCOS
(define (dibujar-Interfaz-pasos)
  ;;dibujamos de nuevo otra imagen de fondo de la interfaz
  (define v3 (open-pixmap "espacio" horizontal vertical))
  ((draw-pixmap v3) "img3.png" (make-posn 0.0 0.0) "black")
  (copy-viewport v3 v1)

  ;;pintamos los ejes
  (pintar-ejes v1)

  ((draw-string v1) (make-posn 45 490) "N° DE PASOS: ")
  ((draw-string v1) (make-posn 236 524) "PARA CONTINUAR PULSE AQUÍ")
  ((draw-string v1) (make-posn 590 490) "SALIR: ")
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (9/15)

```

;;COMPROBAR CILCS EN LA v1
(define (Comprobar-Click v1 click)
  (cond
    ((and (<= 100 (posn-x (mouse-click-posn click)) 130)
          (<= 500 (posn-y (mouse-click-posn click)) 530)) ((draw-rectangle v1) (make-posn 100 500) 30 30 "blue")
              (set! NUM_TOTAL_DISCOS 1) (dibujar-Interfaz-pasos))

    ((and (<= 132 (posn-x (mouse-click-posn click)) 162)
          (<= 500 (posn-y (mouse-click-posn click)) 530)) ((draw-rectangle v1) (make-posn 132 500) 30 30 "blue")
              (set! NUM_TOTAL_DISCOS 2) (dibujar-Interfaz-pasos))

    ((and (<= 164 (posn-x (mouse-click-posn click)) 196)
          (<= 500 (posn-y (mouse-click-posn click)) 530)) ((draw-rectangle v1) (make-posn 164 500) 30 30 "blue")
              (set! NUM_TOTAL_DISCOS 3) (dibujar-Interfaz-pasos))

    ((and (<= 100 (posn-x (mouse-click-posn click)) 130)
          (<= 531 (posn-y (mouse-click-posn click)) 561)) ((draw-rectangle v1) (make-posn 100 531) 30 30 "blue")
              (set! NUM_TOTAL_DISCOS 4) (dibujar-Interfaz-pasos))

    ((and (<= 132 (posn-x (mouse-click-posn click)) 162)
          (<= 531 (posn-y (mouse-click-posn click)) 561)) ((draw-rectangle v1) (make-posn 132 531) 30 30 "blue")
              (set! NUM_TOTAL_DISCOS 5) (dibujar-Interfaz-pasos))

    ((and (<= 164 (posn-x (mouse-click-posn click)) 196)
          (<= 531 (posn-y (mouse-click-posn click)) 561)) ((draw-rectangle v1) (make-posn 164 531) 30 30 "blue")
              (set! NUM_TOTAL_DISCOS 6) (dibujar-Interfaz-pasos))

    ;;comprobación del botón de salir
    ((and (<= 560 (posn-x (mouse-click-posn click)) 650)
          (<= 480 (posn-y (mouse-click-posn click)) 570)) (close-graphics) (set! flag-close #t))

    (else (Comprobar-Click v1 (get-mouse-click v1)))

  )
)

```


Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (10/15)

```
;;COMPROBACION DEL BOTON DE SALIR
(define (salir click)
  (cond
    ((and (<= 560 (posn-x (mouse-click-posn click)) 650)
          (<= 480 (posn-y (mouse-click-posn click)) 570)) (close-graphics))
    (else (salir (get-mouse-click v1))))
  )
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (11/15)

```
;;INICIALIZAR EL ESTADO DE LAS TORRES
(define (inicializar-torres torres n)
  (do
    (
      (iterador 0 (+ iterador 1))
    )
    ((>= iterador n))
    ;;cuerpo del bucle
    (do
      (
        (iterador2 0 (+ iterador2 1))
      )
      ((>= iterador2 (vector-length (vector-ref (vector-ref torres iterador) 1))))
      (vector-set! (vector-ref (vector-ref torres iterador) 1) iterador2
        (make-posn (+ 150 (* iterador 200)) (+ 200 (* (+ iterador2 1) 20))))
    )
  )
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (12/15)

```
;;FUNCION PARA MOVER DISCOS DE UNA TORRE A OTRA
(define (disco-a-palo torres discos disco paloOrigen paloDestino)
  ;;detenemos la ejecución hasta que se pulse salir o continuar
  (cond
    ((equal? flag-close #t))
    (else (pause-or-exit (get-mouse-click v1))))
  )

;;movemos un disco
(define posiciónDestino (- (- NUM_TOTAL_DISCOS 1) (vector-ref (vector-ref torres paloDestino) 0)))
(ellipse-centro v1 (vector-ref (vector-ref (vector-ref torres paloDestino) 1) posiciónDestino)
  (* (- NUM_TOTAL_DISCOS disco) 30) 20 (vector-ref discos disco))

(vector-set! (vector-ref torres paloDestino) 0 (+ (vector-ref (vector-ref torres paloDestino) 0) 1))

;;borramos el disco de donde estaba
(define posiciónALiberar (vector-ref (vector-ref torres paloOrigen) 0))
(eliminar-ellipse-centro v1 (vector-ref (vector-ref (vector-ref torres paloOrigen) 1)
  (- NUM_TOTAL_DISCOS posiciónALiberar)) (* (- NUM_TOTAL_DISCOS disco) 30) 20)

(vector-set! (vector-ref torres paloOrigen) 0 (- (vector-ref (vector-ref torres paloOrigen) 0) 1))

;;pintamo de nuevo los ejes para que al borrar no se queden los ejes mal pintados
(pintar-ejes v1)

;;aumentamos el numero de movimientos
(set! numPasos (+ numPasos 1))
(display-pasos numPasos)
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (13/15)

```
;;INICIAR EL ESTADO INICIAL DE LA PRIMERA TORRE
(define (inicializar-estado-inicial-hanoi torres discos numDiscos)
  (do
    (
      (iterador 0 (+ iterador 1))
    )
    ((>= iterador numDiscos))
    ;;cuerpo del bucle
    (ellipse-centro v1 (vector-ref (vector-ref (vector-ref torres 0) 1) (- (- numDiscos 1) iterador))
      (* (+ (- (- numDiscos 1) iterador) 1) 30) 20 (vector-ref discos iterador))
    (vector-set! (vector-ref torres 0) 0 (+ (vector-ref (vector-ref torres 0) 0) 1))
  )
)
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (14/15)

```
;;PROGRAMA PRINCIPAL
(define (programa)
  ;;IMGAEN DE FONDO EN LA v1 A TRAVES DE PIXMAP
  ((draw-pixmap v2) "img2.png" (make-posn 0.0 0.0) "black")
  (copy-viewport v2 v1)

  ;;DIBUJAR EJES CON COLOR
  (dibujar-Interfaz v1)

  ;;Torres definida de forma global
  (define torres (vector (vector 0 (make-vector NUM_TOTAL_DISCOS))
                        (vector 0 (make-vector NUM_TOTAL_DISCOS))
                        (vector 0 (make-vector NUM_TOTAL_DISCOS))))

  ;;COMO HAY 3 TORRES SE INICIALIZAN
  (inicializar-torres torres 3)

  ;;Discos predefinidos de forma local
  (define discos (vector (make-rgb 1 0 0) (make-rgb 0 1 0) (make-rgb 0 0 1)
                        (make-rgb 0.5 0 0) (make-rgb 0 0.5 0) (make-rgb 0 0 0.5)))

  ;;Iniciamos la torre inicial con el numero de discos deseados
  (inicializar-estado-inicial-hanoi torres discos NUM_TOTAL_DISCOS))
```

Ejemplo: Torres de Hanoi

Implementación en Scheme: Código parte (15/15)

```
;;ALGORITMO DE HANOI
(define (hanoi NumDiscos palo1 palo2 palo3)
  (cond
    ((<= NumDiscos 0) (display "EL PROGRAMA HA TERMINADO"))
    ((= NumDiscos 1) (disco-a-palo torres discos (- NUM_TOTAL_DISCOS NumDiscos) palo1 palo3))
    ((> NumDiscos 1) (hanoi (- NumDiscos 1) palo1 palo3 palo2)
                      (disco-a-palo torres discos (- NUM_TOTAL_DISCOS NumDiscos) palo1 palo3)
                      (hanoi (- NumDiscos 1) palo2 palo1 palo3))
  )
)

;;llamada al algoritmo de HANOI
(hanoi NUM_TOTAL_DISCOS 0 1 2)

;;COMPROBACION DE BOTON DE SALIR
(cond
  ((equal? flag-close #t))
  ((equal? flag-close #f) (salir (get-mouse-click v1)))
)

;;LLAMADA AL PROGRAMA PRINCIPAL
(programa)
```

Ejemplo: Torres de Hanoi

Resultados

- 5 Ejemplo: Torres de Hanoi
 - Historia
 - Reglas
 - Formas de Implementar
 - Pseudocódigo
 - Complejidad
 - Implementación en Scheme
 - Resultados

Ejemplo: Torres de Hanoi

Resultados

Ejemplo

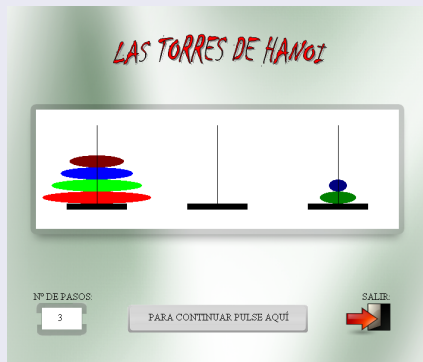


Ventana Inicial

Ejemplo: Torres de Hanoi

Resultados

Ejemplo



Ejecución

Referencias

1 Introducción

2 Comandos

3 Otros comandos

4 Crear ejecutables

5 Ejemplo: Torres de Hanoi

6 Referencias

Referencias

Sitios de interés:



Fernández García, N. L. (2013)

[Introducción al Lenguaje Scheme](#)

Consultado el 27 de octubre de 2013



[Scheme](#) (s.f.)

Obtenido de enciclopedia libre Wikipedia

Consultado el 27 de octubre de 2013



Departamento de Ciencias de la computación e inteligencia artificial de la Universidad de Sevilla

[Manual de referencia de Scheme](#)

Consultado el 27 de octubre de 2013

Referencias

Sitios de interés:



Navas, E. (2010)

Programando con Racket 5

Consultado el 27 de octubre de 2013



The Racket Reference (Ver.5.3.6.)

Graphics: Legacy Library.

Consultado el 27 de octubre de 2013



Torres de Hanói (s.f.)

Obtenido de enciclopedia libre Wikipedia

Consultado el 27 de octubre de 2013

Referencias

Sitios de interés:



Ajay (2009)

Excel Color Palette and Color Index change using VBA

Consultado el 27 de octubre de 2013



The Racket Reference (Ver.5.3.6.)

key-event objects

Consultado el 27 de octubre de 2013



The Racket Reference (Ver.5.3.6.)

raco exe: Creating Stand-Alone Executables

Consultado el 27 de octubre de 2013

Gracias por su atención