

El lenguaje de programación Scheme

En cualquier lenguaje tenemos palabras que se combinan siguiendo ciertas reglas para formar frases con significado. A su vez, estas palabras se forman uniendo las letras de un abecedario. *Scheme*, como lenguaje de programación, utiliza de manera análoga a las palabras los denominados símbolos y estos se forman uniendo las letras del alfabeto (sin distinguir mayúsculas de minúsculas), los dígitos del 0 al 9 y cualquier otro carácter que aparezca en el teclado salvo:

() [] { } ; , " ' \ # \

ya que tienen un significado especial, similar al que tienen los signos de puntuación.

Los caracteres:

+ - .

también son especiales y no deben aparecer en primer lugar en un símbolo. Los números no se consideran símbolos en *Scheme*.

Un símbolo que es usado para representar un valor se denomina variable. El intérprete determinará el significado de cada variable; los números tienen su valor usual.

Siguiendo la análoga con los lenguajes el equivalente en Scheme a las frases son las expresiones, que pueden consistir en un símbolo, un número o una lista, es decir, un paréntesis izquierdo, seguido de expresiones separadas por espacios en blanco, y para terminar un paréntesis derecho. La primera de dichas expresiones debe evaluar a un procedimiento, evaluándose las restantes como los argumentos de este.

Notación

En lo que sigue utilizaremos la siguiente notación al escribir las expresiones:

(procedimiento *expresion*₁ ... *expresion*_k)

Es decir, el nombre del procedimiento aparecerá en negrita y los argumentos en itálica. Además, tendremos en cuenta que si el nombre de un argumento es el nombre de un tipo (ver el apartado PREDICADOS DE TIPO), entonces el argumento debe ser del tipo nombrado . Usaremos el siguiente convenio:

| | | | |
|---|-------------------|--------------------|-------------------------|
| <i>z</i> | número complejo | <i>l</i> | lista |
| <i>x</i> | número real | <i>cter</i> | carácter |
| <i>n</i> | número entero | <i>cad</i> | cadena |
| <i>k</i> | número natural | <i>proc</i> | procedimiento |
| <i>obj</i> , <i>expresión</i> , <i>ex</i> | de cualquier tipo | <i>expresiones</i> | sucesión de expresiones |

Utilizaremos los corchetes para denotar expresiones opcionales y los puntos suspensivos para denotar varias ocurrencias.

(procedimiento *obj*₁ ... *obj*_k) Indica que procedimiento es de aridad variable.

(procedimiento obj₁ [obj₂]) Indica que procedimiento tiene dos argumentos y el segundo es opcional.

A continuación, enumeraremos las expresiones más usuales seguidas de su valor.

Predicados de tipo

| | |
|--------------------------|--|
| (symbol? obj) | Si obj es de tipo símbolo entonces #t; en otro caso #f. |
| (procedure? obj) | Si obj es de tipo procedimiento entonces #t; en otro caso #f. |
| (number? obj) | Si obj es de tipo número entonces #t; en otro caso #f. |
| (pair? obj) | Si obj es de tipo par punteado entonces #t; en otro caso #f. |
| (null? obj) | Si obj es la lista vacía entonces #t; en otro caso #f. |
| (boolean? obj) | Si obj es uno de los valores de verdad (o booleanos), #t o #f, entonces #t; en otro caso #f. |
| (vector? obj) | Si obj es de tipo vector entonces #t; en otro caso #f. |
| (char? obj) | Si obj es de tipo carácter entonces #t; en otro caso #f. |
| (string? obj) | Si obj es de tipo cadena entonces #t; en otro caso #f. |

Ningún objeto verifica más de uno de los predicados anteriores; otro predicado útil es:

| | |
|----------------------|--|
| (list? obj) | Si <i>obj</i> es de tipo 'lista' entonces #t; en otro caso #f. |
|----------------------|--|

Predicados de igualdad

| | |
|--|--|
| (= z ₁ [z ₂ ...z _k]) | Igualdad numérica entre los argumentos |
| (eq? obj ₁ obj ₂) | Igualdad simbólica. |
| (eqv? obj ₁ obj ₂) | Igualdad numérico - simbólica. |
| (equal? obj ₁ obj ₂) | Igualdad de valores. |

Variables y literales

| | |
|-------------------------------------|--|
| <i>expresion</i> | Valor de <i>expresion</i> . |
| (define simbolo obj) | Le asigna a símbolo el valor de <i>obj</i> . |
| (quote obj) ≡ ' <i>obj</i> | <i>obj</i> . |

Los números, caracteres, cadenas y valores de verdad (o booleanos), evalúan a sí mismos por lo que no necesitan **quote**

| | |
|--|---|
| (let | (Todos los simbolo_i deben ser distintos). Evalúa cada obj_j , en un entorno local, asigna a cada simbolo_s el valor de obj_s y a continuación evalúa de forma consecutiva $\text{ex}_1 \dots \text{ex}_r$, devolviendo el valor de la última. |
| $([(\text{simbolo}_1 \text{obj}_1)$ | |
| ... | |
| $(\text{simbolo}_k \text{obj}_k)])$ | |
| $\text{ex}_1 [\text{ex}_2 \dots \text{ex}_r])$ | |
| (let* | En un entorno local, asigna, de manera secuencial, a cada simbolo_s el valor de obj_s y a continuación evalúa de forma consecutiva $\text{ex}_1 \dots \text{ex}_r$, devolviendo el valor de la última. |
| $([(\text{simbolo}_1 \text{obj}_1)$ | |
| ... | |
| $(\text{simbolo}_k \text{obj}_k)])$ | |
| $\text{ex}_1 [\text{ex}_2$ | |
| $\dots \text{ex}_r])$ | |
| (letrec <i>variables</i> | Similar a let , pero permite hacer definiciones de procedimientos recursivos. |
| <i>cuerpo</i>) | |
| (set! <i>simbolo</i> | Asigna el valor de expresión a símbolo, que ya debe tener asignado algún valor. Devuelve un valor no específico. |
| <i>expresion</i>) | |

Expresiones lambda

| | |
|--|--|
| (lambda <i>argumentos</i> | Devuelve un procedimiento |
| <i>cuerpo</i>) | |
| <i>argumentos</i> puede ser: | |
| $(\text{simbolo}_1$ | Lista de símbolos (todos distintos) que representan a cada argumento de la función, la cual será, por tanto, de aridad la longitud de dicha lista. |
| $\dots \text{simbolo}_k)$ | |
| <i>variable</i> | Símbolo que representa a una lista con los argumentos. Por tanto, será de aridad variable |
| $(\text{simb}_1 \dots \text{simb}_k .$ | Los primeros k argumentos se guardan en las variables simb_1 a simb_k . El resto de los argumentos se guardan en una lista en <i>variable</i> . El procedimiento es, por tanto, de aridad al menos k |
| $\text{variable})$ | |

cuerpo: Sucesión de expresiones que describen la función.

| | |
|---|---|
| (define $(\text{simb} \text{simb}_1$ | Equivalente a |
| $\dots \text{simb}_k)$ | $(\text{define } \text{simb}$ |
| <i>cuerpo</i>) | $(\text{lambda } (\text{simb}_1 \dots \text{simb}_k)$ |
| | <i>cuerpo</i>)) |
| (define $(\text{simbolo} .$ | Equivalente a |
| <i>variable</i>) | $(\text{define } \text{simbolo}$ |
| <i>cuerpo</i>) | $(\text{lambda } (\text{variable}$ |
| | <i>cuerpo</i>)) |

| | |
|---|--|
| <code>(define (simb s1 ...sk . variable) cuerpo)</code> | Equivalente a <code>(define simb (lambda (s1...s . var) cuerpo))</code> |
|---|--|

Abstracción de procedimientos

| | |
|--|--|
| <code>(apply proc ex1 [ex2 ...exk])</code> | (ex_k) una lista) aplica <i>proc</i> con argumentos $ex_1 \dots ex_{k-1}$ y los elementos de ex_k . |
| <code>(map proc l1 [l2 ...lk])</code> | Aplica <i>proc</i> a cada elemento de l_1 ; si existe más de una lista todas han de tener la misma longitud y aplica <i>proc</i> tomando como argumentos un elemento de cada lista, devuelve la lista de los resultados. |
| <code>(for-each proc l1 [l2 ...lk])</code> | Aplica <i>proc</i> a cada elemento de l_1 ; si existe más de una lista todas han de tener la misma longitud y aplica <i>proc</i> tomando como argumentos un elemento de cada lista, devuelve un valor no específico |

Procedimientos numéricos

| | |
|----------------------------------|---|
| <code>(+ [z1 ...zk])</code> | Suma de los argumentos; sin argumentos, 0. |
| <code>(- z1 [z2 ...zk])</code> | Resta de los argumentos, asociando por la izquierda; con un solo argumento, $-z_1$. |
| <code>(* [z1 ...zk])</code> | Producto de los argumentos; sin argumentos, 1. |
| <code>(/ z1 [z2 ...zk])</code> | División de los argumentos asociando por la izquierda; con un solo argumento, $1/z$. |
| <code>(sqrt z)</code> | Raíz cuadrada principal de z (si z es real, la raíz cuadrada positiva). |
| <code>(abs x)</code> | Valor absoluto de x . |
| <code>(sin z)</code> | Seno de z . |
| <code>(cos z)</code> | Coseno de z . |
| <code>(tan z)</code> | Tangente de z . |
| <code>(asin z)</code> | Arcoseno de z . |
| <code>(acos z)</code> | Arcocoseno de z . |
| <code>(atan z)</code> | Arcotangente de z . |
| <code>(max x1 [x2 ...xk])</code> | Máximo entre los argumentos. |
| <code>(min x1 [x2 ...xk])</code> | Mínimo entre los argumentos. |
| <code>(quotient n1 n2)</code> | (n_2 distinto de cero), cociente de n_1 entre n_2 . |
| <code>(remainder n1 n2)</code> | (n_2 distinto de cero), resto de n_1 entre n_2 . |
| <code>(expt z1 z2)</code> | La potencia $z_1 z_2$ (con $0^0 = 1$). |
| <code>(exp z)</code> | La potencia e^z . |

| | |
|---|---|
| (log <i>z</i>) | Logaritmo en base e de <i>z</i> . |
| (gcd [<i>n</i> ₁ ... <i>n</i> _{<i>k</i>}]) | Máximo común divisor entre los argumentos; sin argumentos, 0. |
| (lcm [<i>n</i> ₁ ... <i>n</i> _{<i>k</i>}]) | Mínimo común múltiplo entre los argumentos; sin argumentos, 1. |
| (floor <i>x</i>) | Mayor entero menor o igual que <i>x</i> . |
| (ceiling <i>x</i>) | Menor entero mayor o igual que <i>x</i> . |
| (truncate <i>x</i>) | Parte entera de <i>x</i> . |
| (round <i>x</i>) | Entero más cercano a <i>x</i> , en caso de equidistancia número entero par más cercano. |
| (exact->inexact <i>z</i>) | El número inexacto numéricamente más cercano a <i>z</i> . |
| (inexact->exact <i>z</i>) | El número exacto numéricamente más cercano a <i>z</i> . |

Predicados numéricos

| | |
|--------------------------------|--|
| (complex? <i>obj</i>) | Si <i>obj</i> es un número complejo entonces #t; en otro caso #f. |
| (real? <i>obj</i>) | Si <i>obj</i> es un número real entonces #t; en otro caso #f. |
| (rational? <i>obj</i>) | Si <i>obj</i> es un número racional entonces #t; en otro caso #f. |
| (exact? <i>z</i>) | Si <i>z</i> es exacto, entonces #t; en otro caso #f. |
| (inexact? <i>z</i>) | Si <i>z</i> es inexacto, entonces #t; en otro caso #f. |
| (integer? <i>obj</i>) | Si <i>obj</i> es un número entero entonces #t; en otro caso #f. |
| (even? <i>n</i>) | Si <i>n</i> es par entonces #t; en otro caso #f. |
| (odd? <i>n</i>) | Si <i>n</i> es impar entonces #t; en otro caso #f. |
| (zero? <i>z</i>) | Si <i>z</i> es el cero entonces #t; en otro caso #f. |
| (positive? <i>x</i>) | Si <i>x</i> es mayor estricto que cero entonces #t; en otro caso #f. |
| (negative? <i>x</i>) | Si <i>x</i> es menor estricto que cero entonces #t; en otro caso #f. |

Relaciones numéricas

| | |
|---|---|
| (> <i>x</i> ₁ [<i>x</i> ₂ ... <i>x</i> _{<i>k</i>}]) | Los argumentos están en orden decreciente. |
| (< <i>x</i> ₁ [<i>x</i> ₂ ... <i>x</i> _{<i>k</i>}]) | Los argumentos están en orden creciente. |
| (>= <i>x</i> ₁ [<i>x</i> ₂ ... <i>x</i> _{<i>k</i>}]) | Los argumentos están en orden no creciente. |
| (<= <i>x</i> ₁ [<i>x</i> ₂ ... <i>x</i> _{<i>k</i>}]) | Los argumentos están en orden no decreciente. |

Pares

| | |
|---|--|
| <code>(cons obj₁ obj₂)</code> | El par cuyo <i>car</i> es <i>obj₁</i> y cuyo <i>cdr</i> es <i>obj₂</i> |
| <code>(car par)</code> | Primer elemento de <i>par</i> . |
| <code>(cdr par)</code> | Segundo elemento de <i>par</i> . |

Procedimientos sobre listas

| | |
|--|--|
| <code>(cons obj lista)</code> | Lista que resulta al incluir <i>obj</i> como primer elemento de lista. |
| <code>(list [obj₁ ...obj_k])</code> | La lista de los argumentos; sin argumentos la lista vacía. |
| <code>(car lista)</code> | Primer elemento de <i>lista</i> . |
| <code>(cdr lista)</code> | Lista que resulta al quitarle el primer elemento a lista. |
| <code>(caar lista)</code> | Composiciones de <i>car</i> y <i>cdr</i> . |
| <code>(cadr lista)</code> | |
| <code>...</code> | |
| <code>(cdddar lista)</code> | Composiciones de <i>car</i> y <i>cdr</i> . |
| <code>(cddddr lista)</code> | |
| <code>(append [lista₁ ...lista_k])</code> | |
| <code>(reverse lista)</code> | Una lista con los mismos elementos que <i>lista</i> , pero dispuestos en orden inverso. |
| <code>(length lista)</code> | Longitud de lista. |
| <code>(list-ref lista k)</code> | Elemento de lista que ocupa la <i>k</i> -ésima posición. |
| <code>(list-tail lista k)</code> | Sub-lista de <i>lista</i> obtenida eliminando los <i>k</i> primeros elementos. |
| <code>(set-car! lista obj)</code> | Almacena <i>obj</i> como el <i>car</i> de <i>lista</i> y devuelve un valor no específico |
| <code>(set-cdr! lista obj)</code> | Hace que el <i>cdr</i> de lista apunte a <i>obj</i> y devuelve un valor no específico. |

Predicados de pertenencia

Teniendo en cuenta que una sublista de una lista se obtiene por aplicaciones sucesivas de `cdr`

| | |
|------------------------------------|---|
| <code>(memq obj lista)</code> | Primera sublista de <i>lista</i> cuyo primer elemento es igual que <i>obj</i> , comparando con <code>eq?</code> ; en otro caso <code>#f</code> . |
| <code>(memv obj lista)</code> | Primera sublista de <i>lista</i> cuyo primer elemento es igual que <i>obj</i> , comparando con <code>eqv?</code> ; en otro caso <code>#f</code> . |
| <code>(member obj lista)</code> | Primera sublista de <i>lista</i> cuyo primer elemento es igual que <i>obj</i> comparando con <code>equal?</code> ; en otro caso <code>#f</code> . |
| <code>(assq obj lista-par)</code> | Primer elemento de <i>lista-par</i> (una lista de pares puntuados) cuyo primer elemento es igual que <i>obj</i> , comparando con <code>eq?</code> ; en otro caso <code>#f</code> . |
| <code>(assv obj lista-par)</code> | Primer elemento de <i>lista-par</i> (una lista de pares puntuados) cuyo primer elemento es igual que <i>obj</i> , comparando con <code>eqv?</code> ; en otro caso <code>#f</code> . |
| <code>(assoc obj lista-par)</code> | Primer elemento de <i>lista-par</i> (una lista de pares puntuados) cuyo primer elemento es igual que <i>obj</i> , comparando con <code>equal?</code> ; en otro caso <code>#f</code> . |

Expresiones condicionales

| | |
|--|---|
| <code>(if test consecuencia [alternativa])</code> | Si <i>test</i> tiene como valor <code>#f</code> entonces <i>alternativa</i> (si no existe <i>alternativa</i> entonces un valor no específico), en otro caso <i>consecuencia</i> . |
| <code>(cond (test₁ [expresiones₁]) ... (test_k [expresiones_k]) (else [expresiones_{k+1}]))</code> | Evalúa <i>test</i> ₁ ... <i>test</i> _k <u>sucesivamente</u> hasta encontrar el primer <i>test</i> _i que no tenga como valor <code>#f</code> , en cuyo caso evalúa en orden las expresiones _i devolviendo el valor de la última (si no existen devuelve el valor de dicho <i>test</i> _i). Si todo <i>test</i> _i tiene como valor <code>#f</code> y existe la cláusula <i>else</i> , evalúa en orden expresiones _{k+1} devolviendo el valor de la última; si no existen o no existe cláusula <i>else</i> , devuelve un valor no específico. |
| <code>(case clave ((datos₁) expresiones₁) ... ((datos_k) expresiones_k) (else expresiones_{k+1}))</code> | Evalúa <i>clave</i> y compara el resultado obtenido con cada uno de los <i>datos</i> _i , <u>sucesivamente</u> . Si encuentra alguno que es igual (comparando con <code>eqv?</code>) evalúa en orden expresiones devolviendo el valor de la última. Si el valor de <i>clave</i> es distinto a todos los <i>datos</i> _i y existe la cláusula <i>else</i> , evalúa en orden expresiones _{k+1} devolviendo el valor de la última; si no existe cláusula <i>else</i> , devuelve un valor no específico. |

Operadores lógicos

| | |
|--|--|
| <code>(not obj)</code> | Si <i>obj</i> tiene como valor #f entonces #t; en otro caso #f. |
| <code>(or [obj₁ ...obj_k])</code> | Evalúa <i>obj₁ ...obj_k</i> sucesivamente hasta el primero que no tenga como valor #f y devuelve su valor, en otro caso #f; sin argumentos, #f. |
| <code>and [obj₁ ...obj_k])</code> | Evalúa <i>obj₁ ...obj_k</i> sucesivamente hasta el primero que tenga como valor #f, en otro caso el valor de <i>obj_k</i> ; sin argumentos, #t. |

Iteraciones

| | |
|---|---|
| <pre>(do ((simbolo₁ obj₁ [paso₁]) ... (simbolo_k obj_k [paso_k])) (test [ex₁ ...ex_r]) [expresion₁ ... expresion_s])</pre> | <p>Cada <i>simbolo_i</i> recibe el valor de <i>obj_i</i>. En cada iteración del bucle se evalúa <i>test</i>:</p> <p>sí es #f se evalúan sucesivamente y se actualizan los valores de cada <i>simbolo_i</i> según <i>paso_i</i> (cuando existen). Comienza una nueva iteración</p> <p>si no, evalúan <i>ex₁ ...ex_r</i> sucesivamente devolviendo el valor de la <i>última</i>. Termina el bucle.</p> |
|---|---|

Procedimientos sobre vectores

| | |
|--|--|
| <code>(make-vector k [obj])</code> | Construye un vector con k elementos iguales a <i>obj</i> , si no existe <i>obj</i> el contenido del vector es indeterminado. |
| <code>(vector [obj₁ ...obj_k])</code> | Construye un vector con <i>k</i> elementos, cada uno de los cuales es <i>obj_i</i> . |
| <code>(vector-ref vector k)</code> | Elemento de <i>vector</i> que ocupa la posición k-exima. |
| <code>(vector-length vector)</code> | Número de elementos de <i>vector</i> . |
| <code>(vector->list vector)</code> | Lista con los elementos de <i>vector</i> . |
| <code>(list->vector lista)</code> | <i>Vector</i> con los elementos de <i>lista</i> . |
| <code>(vector-set! vector k obj)</code> | Almacena <i>obj</i> en el k-eximo elemento de <i>vector</i> , devuelve un valor no específico. |
| <code>(vector-fill! vector obj)</code> | Cambia cada elemento de <i>vector</i> por <i>obj</i> . Devuelve un valor no específico. |

Procedimientos sobre caracteres

| | |
|--|--|
| <code>(char->integer caracter)</code> | Código ASCII de carácter. |
| <code>(integer->char n)</code> | Carácter con código ASCII n. |
| <code>(char<? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con <. |

| | |
|--|--|
| <code>(char<=? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con <=. |
| <code>(char>? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con >. |
| <code>(char>=? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con >=. |
| <code>(char=? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con =. |
| <code>(char-ci<? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con <, sin distinguir mayúsculas de minúsculas. |
| <code>(char-ci<=? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con <=, sin distinguir mayúsculas de minúsculas. |
| <code>(char-ci>? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con >, sin distinguir mayúsculas de minúsculas. |
| <code>(char-ci>=? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con >=, sin distinguir mayúsculas de minúsculas. |
| <code>(char-ci=? cter₁ [cter₂ ...cter_k])</code> | Comparación de los valores numéricos de cter _i con =, sin distinguir mayúsculas de minúsculas. |
| <code>(char-upcase character)</code> | Si carácter es minúscula devuelve el mismo en mayúscula; devuelve carácter en otro caso |
| <code>(char-downcase character)</code> | Si carácter es mayúscula devuelve el mismo en minúscula; devuelve carácter en otro caso |
| <code>(char-upper-case? obj)</code> | Si obj es de tipo carácter mayúscula , entonces #t; en otro caso #f. |
| <code>(char-lower-case? obj)</code> | Si obj es de tipo carácter minúscula , entonces #t; en otro caso #f. |
| <code>(char-alphabetic? obj)</code> | Si obj es de tipo carácter alfabético , entonces #t; en otro caso #f. |
| <code>(char-numeric? obj)</code> | Si obj es de tipo carácter numérico , entonces #t; en otro caso #f. |
| <code>(char-whitespace? obj)</code> | Si obj es de tipo carácter de espacio en blanco (espacio, tabulación, salto de línea, retorno de carro), entonces #t; en otro caso #f. |

Procedimientos sobre cadenas

| | |
|---|---|
| <code>(string-length cadena)</code> | Número de caracteres de cadena |
| <code>(make-string k [character])</code> | Construye una cadena con k caracteres iguales a carácter, si no existe carácter el contenido de la cadena es indeterminado. |
| <code>(string [cter1 ...cter_k])</code> | Construye una cadena de k caracteres, cada uno de los cuales es cter _i . |
| <code>(string-ref cadena k)</code> | Carácter de cadena que ocupa la posición k-exima. |
| <code>(string-set! cadena k cter)</code> | Almacena cter como elemento k-eximo de cadena y devuelve un valor no específico. |
| <code>(substring cadena k1 k2)</code> | Trozo de cadena entre las posiciones k1 y k2, la primera incluida. |
| <code>(string-append [cad1 ...cad_k])</code> | Concatenación de las cadenas cad _i , sin argumentos la cadena vacía. |
| <code>(string->list cadena)</code> | Lista con los caracteres de cadena. |
| <code>(list->string lista)</code> | Cadena con los caracteres de lista (todos sus elementos han de ser caracteres). |
| <code>(string-copy cadena)</code> | Una nueva cadena, copia exacta de cadena. |
| <code>(string-fill! cadena cter)</code> | Cambia cada elemento de cadena por el carácter cter. Devuelve un valor no específico. |
| <code>(string<? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char<. |
| <code>(string<=? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char<=. |
| <code>(string>? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char>. |
| <code>(string>=? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char>=. |
| <code>(string=? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char=. |
| <code>(string-ci<? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char-ci<. |
| <code>(string-ci<=? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char-ci<=. |
| <code>(string-ci>? cad1 [cad2 ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char-ci>. |

| | |
|---|--|
| <code>(string-ci>=? cad₁ [cad₂ ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char-ci>=. |
| <code>(string-ci=? cad₁ [cad₂ ...cad_k])</code> | Comparación de cad _i con la extensión lexicográfica de char-ci=. |

Procedimientos de entrada y salida

| | |
|---|---|
| <code>(input-port? obj)</code> <code>(output-port? obj)</code> | Devuelve #t si <i>obj</i> es un puerto de entrada o salida, respectivamente; en caso contrario devuelve #f. |
| <code>(current-input-port)</code> <code>(current-output-port)</code> | Devuelve el puerto de entrada o salida por defecto (inicialmente el teclado y el monitor, respectivamente). |
| <code>(open-input-file fichero)</code> | Devuelve un puerto capaz de proporcionar caracteres del fichero, que debe existir. |
| <code>(open-output-file fichero)</code> | Devuelve un puerto capaz de escribir caracteres en fichero, que es creado al evaluar la expresión y, por tanto, inicialmente no debe existir |
| <code>(call-with-input-file fichero proc)</code> <code>(call-with-output-file fichero proc)</code> | Evalúan el procedimiento <i>proc</i> con un argumento: el puerto obtenido al abrir <i>fichero</i> para entrada o salida, respectivamente |
| <code>(with-input-from-file fichero proc)</code> <code>(with-output-to-file fichero proc)</code> | Se crea un puerto de entrada o salida conectado a chero, convirtiéndose dicho puerto en el puerto por defecto. A continuación, se evalúa <i>proc</i> sin argumentos. |
| <code>(eof-object? obj)</code> | Devuelve #t si <i>obj</i> es un dato de tipo <i>final</i> de chero y #f en caso contrario. |
| <code>(read [puerto])</code> | Devuelve el siguiente objeto de Scheme que se puede obtener del puerto de entrada dado (o el puerto de entrada por defecto, si se omite). Al llegar al <i>final</i> del fichero conectado a puerto, devuelve un objeto de tipo <i>final de fichero</i> . |
| <code>(read-char [puerto])</code> | Devuelve el siguiente carácter que se puede obtener del puerto de entrada dado (o el puerto de entrada por defecto, si se omite). Al llegar al final del chero conectado a puerto, devuelve un objeto de tipo final de fichero. |
| <code>(peek-char [puerto])</code> | Devuelve el siguiente objeto de Scheme que se puede obtener del puerto de entrada dado (o el puerto de entrada por defecto, si se omite), sin actualizar dicho puerto para que apunte al siguiente carácter. Si no hay caracteres disponibles, devuelve un objeto de tipo final de fichero. |
| <code>(newline [puerto])</code> | Escribe un objeto de tipo final de línea en puerto (o el puerto de entrada por defecto, si Este se omite). Devuelve un valor no específico. |

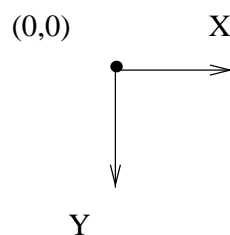
| | |
|---|--|
| (write <i>obj</i> [<i>puerto</i>]) | Escribe una representación escrita de <i>obj</i> en puerto (o el puerto de entrada por defecto, si este se omite). Las cadenas se escriben delimitadas por comillas y los caracteres con la notación #\ . Devuelve un valor no específico. |
| (display <i>obj</i> [<i>puerto</i>]) | Escribe una representación de <i>obj</i> en puerto (o el puerto de entrada por defecto, si este se omite). Las cadenas se escriben sin estar delimitadas por comillas y los caracteres sin la notación #\ . Devuelve un valor no específico. |
| Nota: Write se usa para producir salida legible por el ordenador, mientras que display se usa para producir salida legible por una persona. | |
| (write-char <i>cter</i> [<i>puerto</i>]) | Escribe el carácter <i>cter</i> (sin usar la notación #\) en <i>puerto</i> (o en el puerto de entrada por defecto, si este se omite). Devuelve un valor no especo. |

Procedimientos de la librería gráfica

Para utilizar la librería gráfica que se describe a continuación, es necesario cargarla mediante la siguiente orden: (require-library graphics.ss graphics).

| | |
|--|--|
| (open-graphics) | Inicializa las rutinas de la librería gráfica. Debe ser utilizada antes que ningún otro procedimiento de la misma. |
| (close-graphics) | Cierra todas las ventanas. Hasta que no vuelva a utilizarse open-graphics no funcionara ninguna rutina grafica |
| (open-viewport <i>cad</i> <i>n1</i> <i>n2</i>) | Crea una ventana nueva, con nombre <i>cad</i> , de <i>n1</i> pixels de anchura y <i>n2</i> pixels de altura. Devuelve un objeto de tipo ventana. |
| (close-viewport <i>ventana</i>) | Borra la ventana indicada de la pantalla, impidiendo su uso posterior. |

Un objeto de tipo posición es la localización de un pixel en un objeto de tipo ventana. El pixel situado en la esquina superior izquierda es el de coordenadas (0,0) y la orientación del eje de coordenadas viene representada en el siguiente dibujo:



| | |
|---|--|
| (make-posn <i>n1</i> <i>n2</i>) | Devuelve el objeto de tipo posición de coordenadas <i>n1</i> y <i>n2</i> . |
| (posn-x <i>posicion</i>) | Devuelve las coordenadas X e Y, respectivamente, de posición. |
| (posn-y <i>posicion</i>) | |

(posn? obj) Devuelve #t si obj es un objeto de tipo posición; en caso contrario devuelve #f.

Un color puede representarse de dos formas distintas: como una cadena (con el nombre de este, p.e. red) o como un objeto de tipo rgb. Cualquiera de los procedimientos que toma color como argumento acepta cualquiera de las representaciones

(make-rgb rojo verde azul) Dados tres números reales de 0 (oscuro) a 1 (claro) devuelve un objeto de tipo rgb.

(rgb-red rgb) Devuelve los valores de rojo, verde y azul, respectivamente, de rgb.

(rgb-green rgb)

(rgb-blue rgb)

(rgb? obj) Devuelve #t si obj es un objeto de tipo rgb; en caso contrario devuelve #f.

((draw-viewport ventana) [color]) Dada una *ventana* devuelve un procedimiento que colorea (resp. borra) el contenido completo de la misma utilizando *color* o negro, si se omite éste.

((clear-viewport ventana))

((draw-pixel ventana) posicion [color])

Dada una ventana devuelve un procedimiento que dibuja (resp. borra) un pixel en la misma, en la posición especi cada utilizando color; o negro, si se omite este.

((clear-pixel ventana) posicion)

((draw-line ventana) posicion1 posicion2 [color])

Dada una ventana devuelve un procedimiento que dibuja (resp. borra) una l nea en la misma, conectando las posiciones especificadas utilizando *color*; o negro, si se omite este.

((clear-line ventana) posicion1 posicion2)

((draw-rectangle ventana) posicion altura anchura [color])

Dada una ventana devuelve un procedimiento que dibuja (resp. borra) el borde de un rectángulo en la misma, de *altura* y *anchura* dadas, siendo el vértice superior izquierdo el que ocupa la posición especificada, utilizando *color*; o negro, si se omite éste.

((clear-rectangle ventana) posicion altura anchura)

((draw-solid-rectangle ventana) posicion altura anchura [color]) Como el anterior, pero colorea toda la figura utilizando *color*; o negro, si se omite éste.

((clear-solid-rectangle ventana) posicion altura anchura)

| | |
|---|---|
| ((draw-ellipse <i>ventana</i>) <i>posición altura anchura</i> [<i>color</i>]) | Dada una ventana devuelve un procedimiento que dibuja (resp. borra) el borde de la elipse inscrita en el rectángulo de altura y anchura dadas, siendo el vértice superior izquierdo el que ocupa la posición especificada, utilizando color; o negro, si se omite este. |
| ((clear-ellipse <i>ventana</i>) <i>posición altura anchura</i>) | |
| ((draw-solid-ellipse <i>ventana</i>) <i>posición altura anchura</i> [<i>color</i>]) | Como el anterior, pero colorea toda la gura utilizando color; o negro, si se omite este. |
| ((clear-solid-ellipse <i>ventana</i>) <i>posición altura anchura</i>) | |
| ((draw-polygon <i>ventana</i>) <i>l-</i> <i>posiciones posición</i> [<i>color</i>]) | Dada una ventana devuelve un procedimiento que dibuja (resp. borra) el borde de un polígono en la misma, siendo los vértices los elementos de <i>l-</i> posiciones y considerando posición como el desplazamiento de este. Utilizando color; o negro, si se omite Este. |
| ((clear-polygon <i>ventana</i>) <i>l-</i> <i>posiciones posición</i>) | |
| ((draw-solid-polygon <i>ventana</i>) <i>l-posiciones posición</i> [<i>color</i>]) | Como el anterior, pero colorea toda la gura utilizando color; o negro, si se omite este. |
| ((clear-solid-polygon <i>ventana</i>) <i>l-posiciones posición</i>) | |
| ((draw-string <i>ventana</i>) <i>posiciones posición</i> [<i>color</i>]) | Dada una <i>ventana</i> devuelve un procedimiento posición <i>cadena</i> [<i>color</i>]) que dibuja (resp. borra) una cadena en la misma, siendo el vértice inferior izquierdo el que posición <i>cadena</i>) ocupa la posición especificada, utilizando color; o negro, si se omite este. |
| ((clear-string <i>ventana</i>) | |
| ((get-string-size <i>ventana</i>) | Dada una ventana devuelve un procedimiento <i>cadena</i>) que devuelve el tamaño de cadena como una lista de dos números, la anchura y la altura. |

Otras funciones de interés

| | |
|---|--|
| (load <i>nombre-fichero</i>) | Evalúa el contenido de nombre-fichero. |
| (trace [<i>proc1...prock</i>]) | Rede ne <i>proc1 ...prock</i> . Estos nuevos procedimientos escriben los argumentos y salidas de cada llamada a los mismos. Devuelve la lista (<i>proc1 ...prock</i>). |
| (untrace [<i>proc1...procr</i>]) | Anula la anterior para cada <i>proc_i</i> . Devuelve la lista (<i>proc1 ...procr</i>). |
| (error <i>cadena</i> [<i>obj1...objk</i>]) | Interrumpe al intérprete y devuelve un mensaje compuesto por <i>cadena</i> y una versión en forma de cadena de <i>obj1 ...objk</i> . |

| | |
|--------------------------------------|---|
| (begin [$ex_1 \dots ex_k$]) | Evalúa de manera sucesiva $ex_1 \dots ex_k$ y devuelve el valor de ex_k ; sin argumentos devuelve un valor no específico. |
| (random k) | Devuelve un número entero aleatorio entre 0 y $k-1$. |
| (exit) | Cierra una sesión con el intérprete de Scheme. |