VECTORES

Los vectores o arrays nos permiten almacenar varios valores de un mismo tipo en una sola variable, y nos evitan tener que usar muchas variables para múltiples datos de un mismo tipo. Un vector crea una estructura de datos con n posiciones. Como en las cadenas, es posible acceder a los datos que hay en un vector en Scheme, iniciando en la posición 0. Un vector en Scheme puede contener diferentes tipos de datos en su interior pero en otros lenguajes sólo contiene el mismo tipo de datos. Es muy útil cuando se quiere trabajar con tablas o para ordenar datos. *Por ejemplo,* si quisiéramos guardar las notas que tenemos de una materia, usamos un vector que nos guarde todas las notas en vez de utilizar varios parámetros para hacer esto.

SINTAXIS DE VECTORES

(make-vector N) *Crea un vector de cantidad n que se llena con ceros.*
Ej.:
(make-vector 6) *; -> #(0 0 0 0 0 0) Se crea un vector de 6 posiciones.*

(make-vector N DATO) *Crea un vector de cantidad n que se llena con el dato en todas las posiciones.*
Ej.:
(make-vector 5 #\q) *; -> #(#\q #\q #\q #\q #\q) Se llena un vector con el caracter q.*

(vector DATO1 DATO2 DATO3 ....) *Crea un vector y lo llena con los datos ingresados.*
Ej.:
(vector 0 "1" "Hola" "casa") *; -> #(0 "1" "Hola" "casa") Se crea un vector con los datos ingresados*

(vector-length vector); *Devuelve el numero de elementos del vector*
Ej.:
```
define (TamVector v)
  (vector-length v)
)

(TamVector (vector 0 "1" "Hola" "casa"))  ;-> 4
```

(vector-ref NOMBRE_VECTOR POSICION) *Lee el valor de un dato del vector en la posición dada.*
Ej.:
```
(define (Muestra2DatosDelVector vector1)
  (display(vector-ref vector1 0))  ; Muestra el dato que se encuentra en la primera posición del vector vector1, →2
  (newline)
  (display(vector-ref vector1 3))  ; Muestra el dato que se encuentra en  la cuarta ubicación del vector vector1.-->amigos
 )

(Muestra2DatosDelVector (vector1 2 "1" "Hola" "amigos"))
```

Ejemplo usando la función string-length del tema de cadenas:
```
(define (MuestraTamCadena vector1)
  (string-length (vector-ref vector1 3)) ; -> 6  ; Muestra el tamaño de la cadena "amigos". Se puede usar porque el dato es una cadena.
 )

(MuestraTamCadena (vector 12 "1" "Hola" "amigos"))
```

Ejemplo con números:

```
(define (DevuelveValor1DelVector vector1)
  (vector-ref vector1 0)   ; Lee el valor que se encuentra en la posicion 0.
 )

(+ 10(DevuelveValor1DelVector (vector 12 "1" "Hola" "amigos") )) ; -> 22  ; me devuelve la suma del dato de la posición 0 mas 10, es decir: 10+12=22
```

(vector-set! Vector posición nuevo-dato) *Se ingresa el nuevo dato en el vector en la posición dada.*

Ej.:

```
(define (IngresarDatoAlVector v)
  (vector-set! v 0 876) ; se ingresa 876 en la posicion 0 del vector "v".
   v    ; Devuelve el vector con el dato modificado, cambia 12 por 876
 )

(IngresarDatoAlVector (vector 12 "1" "Hola" "amigos"))
```

Ej. Usando random:

```
(define (CambiarAleatoriamenteDato v) ; Ejecutar varias veces para ver que pasa.
  (vector-set! v 2 (+ (random 100) 1 ) ); En la posición 2 de v se ingresa un número aleatorio entre 1 y 100
   v; Devuelve el vector con el dato modificado.
 )

(CambiarAleatoriamenteDato (vector 12 "1" "Hola" "amigos"))
```

(vector-fill! vector dato) *Ingresa un dato dado en todas las posiciones del vector.*

Ej.:

```
(define (LlenarVectorConUnDato v)
   (vector-fill! v "hola");  -> #("hola" "hola" "hola" "hola")
   (display v) (newline)
   (vector-fill! v 56) ; -> #(56 56 56 56)
   (display v)

 )

(LlenarVectorConUnDato (vector12 "1" "Hola" "amigos"))
```