COMANDOS PYTHON

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

`array.`**`typecodes`**

Una cadena de caracteres con todos los códigos de tipos disponible.

El módulo define los siguientes tipos:

`class` `array.`**`array`**`(`*`typecode`*`[, `*`initializer`*`])`

A new array whose items are restricted by *typecode*, and initialized from the optional *initializer* value, which must be a `bytes` or `bytearray` object, a Unicode string, or iterable over elements of the appropriate type.

If given a `bytes` or `bytearray` object, the initializer is passed to the new array's `frombytes()` method; if given a Unicode string, the initializer is passed to the `fromunicode()` method; otherwise, the initializer's iterator is passed to the `extend()` method to add initial items to the array.

Los objetos tipo arreglo soportan operaciones de secuencia ordinarias de indexación, segmentación, concatenación y multiplicación . Cuando se utiliza segmentación, el valor asignado debe ser un arreglo con el mismo código de tipo, en todos los otros casos se lanza `TypeError`. Los arreglos también implementan una interfaz de buffer, y puede ser utilizada en cualquier momento cuando los objetos bytes-like objects son soportados.

Lanza un evento de auditoría `array.__new__` con argumentos `typecode`, `initializer`.

`typecode`

El carácter typecode utilizado para crear el arreglo.

`itemsize`

La longitud en bytes de un elemento del arreglo en su representación interna.

`append``(`*`x`*`)`

Añade un nuevo elemento con valor *x* al final del arreglo.

`buffer_info``()`

Return a tuple `(address, length)` giving the current memory address and the length in elements of the buffer used to hold array's contents. The size of the memory buffer in bytes can be computed as `array.buffer_info()[1] * array.itemsize`. This is occasionally useful when working with low-level (and inherently unsafe) I/O interfaces that require memory addresses, such as certain `ioctl()` operations. The returned numbers are valid as long as the array exists and no length-changing operations are applied to it.

Nota Cuando utilizamos objetos tipo arreglo escritos en C o C++ (la única manera de utilizar esta información de forma más efectiva), tiene más sentido utilizar interfaces buffer que soporten objetos del tipo arreglo. Este método es mantenido con retro compatibilidad y tiene que ser evitado en el nuevo código. Las interfaces de buffer son documentadas en Protocolo búfer.

byteswap()

«Byteswap» todos los elementos del arreglo. Solo es soportado para valores de tamaño 1,2,3,4 o 8 bytes; para otros valores se lanza `RuntimeError`. Es útil cuando leemos información de un fichero en una máquina con diferente orden de bytes.

count(*x*)

Retorna el número de ocurrencias de *x* en el arreglo.

extend(*iterable*)

Añade los elementos del *iterable* al final del arreglo. Si el *iterable* es de otro arreglo, este debe ser *exactamente* del mismo tipo; si no, se lanza `TypeError`. Si el *iterable* no es un arreglo, este debe de ser un iterable y sus elementos deben ser del tipo correcto para ser añadidos al arreglo.

frombytes(*buffer*)

Appends items from the bytes-like object, interpreting its content as an array of machine values (as if it had been read from a file using the `fromfile()` method).

Nuevo en la versión 3.2: `fromstring()` is renamed to `frombytes()` for clarity.

fromfile(*f*, *n*)

Lee *n* elementos (como valores de máquina) del file object *f* y los añade al final del arreglo. Si hay menos de *n* elementos disponibles, se lanza `EOFError`, pero los elementos que estaban disponibles todavía se insertan en el arreglo.

fromlist(*list*)

Añade los elementos de la lista. Es equivalente a `for x in list: a.append(x)` excepto que si hay un error de tipo, el arreglo no se modifica.

fromunicode(*s*)

Extends this array with data from the given Unicode string. The array must have type code `'u'`; otherwise a `ValueError` is raised. Use `array.frombytes(unicodestring.encode(enc))` to append Unicode data to an array of some other type.

index(*x*[, *start*[, *stop*]])

Retorna el *i* más pequeño de modo que *i* es el índice de la primera aparición de *x* en el arreglo. Los argumentos opcionales *start* y *stop* pueden ser especificados para buscar *x* dentro de una subsección del arreglo. Lanza `ValueError` si no se encuentra *x*.

Distinto en la versión 3.10: Se agregaron parámetros opcionales *start* y *stop*.

insert(*i*, *x*)

Inserta un nuevo elemento con valor *x* en el arreglo antes de la posición *i*. Si hay valores negativos son tratados como relativos a la posición final del arreglo.

pop([*i*])

Elimina el elemento con índice *i* del arreglo y lo retorna. El argumento opcional por defecto es `-1`, en caso de utilizar el argumento por defecto el ultimo elemento es eliminado y retornado.

remove(*x*)

Elimina la primera ocurrencia de *x* del arreglo.

reverse()

Invierte el orden de los elementos en el arreglo.

tobytes()

Convierte el arreglo en un arreglo de valores máquina y retorna una representación en formato de bytes (la misma secuencia de bytes que se deben escribir en un fichero por el método `tofile()`.)

Nuevo en la versión 3.2: `tostring()` is renamed to `tobytes()` for clarity.

tofile(*f*)

Escribe todos los elementos (incluido elementos máquina) a el file object *f*.

tolist()

Convierte el arreglo a una lista ordinaria con los mismos elementos.

`tounicode``()`

Convert the array to a Unicode string. The array must have a type `'u'`; otherwise a `ValueError` is raised. Use `array.tobytes().decode(enc)` to obtain a Unicode string from an array of some other type.

The string representation of array objects has the form `array(typecode, initializer)`. The *initializer* is omitted if the array is empty, otherwise it is a Unicode string if the *typecode* is `'u'`, otherwise it is a list of numbers. The string representation is guaranteed to be able to be converted back to an array with the same type and value using `eval()`, so long as the `array` class has been imported using `from array import array`. Variables `inf` and `nan` must also be defined if it contains corresponding floating point values. Examples: