`array` — Arreglos eficientes de valores numéricos

Este modulo define un tipo de objeto que representa un arreglo de valores básicos: caracteres, números enteros y de punto flotante. Los arreglos son tipos de secuencias que se comportan de forma similar a las listas, a excepción que el tipo de objeto guardado es definido. El tipo es especificado al momento de crear el objeto mediante *type code*, que es un carácter simple. Se definen los siguientes tipos:

Código de tipo	Tipo C	Tipo Python	Tamaño mínimo en bytes	Notas
`'b'`	signed char	int	1	
`'B'`	unsigned char	int	1	
`'u'`	wchar_t	Carácter Unicode	2	(1)
`'h'`	signed short	int	2	
`'H'`	unsigned short	int	2	
`'i'`	signed int	int	2	
`'I'`	unsigned int	int	2	
`'l'`	signed long	int	4	
`'L'`	unsigned long	int	4	
`'q'`	signed long long	int	8	
`'Q'`	unsigned long long	int	8	
`'f'`	float	float	4	
`'d'`	double	float	8	

Notas:

1. Puede ser de 16 bits o 32 bits según la plataforma.

   Distinto en la versión 3.9: `array('u')` ahora usa `wchar_t` como tipo C en lugar de `Py_UNICODE` obsoleto. Este cambio no afecta su comportamiento porque `Py_UNICODE` es el alias de `wchar_t` desde Python 3.3.

   Deprecated since version 3.3, will be removed in version 4.0.

La representación real de los valores viene determinada por la arquitectura de la maquina (estrictamente hablando, por la implementación de C). El tamaño actual se puede obtener mediante el atributo `itemsize`.

El módulo define los siguientes tipos:

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

Un nuevo arreglo cuyos elementos son restringidos por *typecode*, e inicializados con el valor opcional *initializer*, el cual debe ser una lista, un bytes-like object, o un iterable sobre los elementos del tipo apropiado.

Si dada una lista o un string, el inicializador es pasado a los nuevos métodos `fromlist()`, `frombytes()`, `fromunicode()` del arreglo (ver abajo) para añadir nuevos elementos al arreglo. De forma contraria, el iterable inicializador se pasa al método `extend()`.

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

`array.`**`typecodes`**

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

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.

Los siguientes tipos de datos y métodos también son soportados:

`array.`**`typecode`**

El carácter typecode utilizado para crear el arreglo.

`array.`**`itemsize`**

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

`array.`**`append`**(*x*)

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

`array.`**`buffer_info`**()

Retorna una tupla `(address, length)` con la dirección de memoria actual y la longitud de los elementos en el buffer utilizado para almacenar temporalmente los elementos del arreglo. El tamaño del buffer de memoria es calculado como `array.buffer_info()[1] * array.itemsize`. Ocasionalmente es practico cuando trabajamos en interfaces E/S de bajo nivel (de manera inherentemente insegura) que requieren direcciones de memoria, por ejemplo ciertas operaciones `ioctl()`. Los números retornados son válidos mientras el arreglo exista y no se cambie la longitud del mismo.

> **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.

`array.`**`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.

`array.`**`count`**(*x*)

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

`array.`**`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.

`array.`**`frombytes`**(*s*)

Añade los elementos de la cadena de texto, interpretando la cadena de texto como un arreglo de valores máquina (como si se leyera de un fichero utilizando el método `fromfile()`).

Nuevo en la versión 3.2: `fromstring()` se renombra como `frombytes()` por claridad.

`array.`**`fromfile`**(*f*, *n*)

Read *n* items (as machine values) from the file object *f* and append them to the end of the array. If less than *n* items are available, `EOFError` is raised, but the items that were available are still inserted into the array.

`array.`**`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.

`array.`**`fromunicode`**(*s*)

Extiende este arreglo con datos de la cadena de texto unicode. El arreglo debe ser un arreglo tipo `'u'`; de forma contraria se lanza `ValueError`. Utiliza `array.frombytes(unicodestring.encode(enc))` para añadir datos Unicode a un arreglo de algún otro tipo.

`array.`**`index`**(*x*[, *start*[, *stop*]])

Return the smallest *i* such that *i* is the index of the first occurrence of *x* in the array. The optional arguments *start* and *stop* can be specified to search for *x* within a subsection of the array.
Raise `ValueError` if *x* is not found.

Distinto en la versión 3.10: Added optional *start* and *stop* parameters.

`array.`**`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.

`array.`**`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.

`array.`**`remove`**(*x*)

Elimina la primera ocurrencia de *x* del arreglo.

`array.`**`reverse`**()

Invierte el orden de los elementos en el arreglo.

`array.`**`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()` se renombra como `tobytes()` para claridad.

`array.`**`tofile`**(*f*)

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

`array.`**`tolist`**()

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

`array.`**`tounicode`**()

Convierte el arreglo a una cadena de texto unicode. El arreglo debe ser un arreglo tipo `'u'` ; en caso contrario se lanza `ValueError`. Utiliza `array.tobytes().decode(enc)` para obtener una cadena de texto unicode de un arreglo de algún otro tipo.

Cuando un objeto se imprime o se convierte a una cadena de texto, este se representa como `array(typecode, initializer)`. El *initializer* se omite cuando el arreglo está vacío, de forma contraria es una cadena de caracteres si su *typecode* es `'u'`, de lo contrario es una lista de números. La cadena de caracteres garantiza que es capaz de ser convertida de nuevo a un arreglo con el mismo tipo y valor utilizando `eval()`, hasta que la clase `array` ha sido importada utilizando `from array import array`. Ejemplos:

```
array('l')
array('u', 'hello \u2641')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.14])
```

Ver también

Módulo `struct`
: Empaquetado y desempaquetado de datos binarios heterogéneos.

Módulo `xdrlib`
: Empaquetado y desempaquetado de datos de Representación de datos externos (XDR) como los utilizados en algunos sistemas de llamadas de procedimientos remotos.

NumPy
: The NumPy package defines another array type.