In [1]:
```
# Se importa la librería numpy

import numpy as np

# APILAMIENTO
# -----------
# Apilado
# Las matrices se pueden apilar horizontalmente, en profundidad o
# verticalmente. Podemos utilizar, para ese propósito,
# las funciones vstack, dstack, hstack, column_stack, row_stack y concatenate.

# Para empezar, vamos a crear dos arrays

# Matriz a
a = np.arange(16).reshape(4,4)
print('a =\n', a, '\n')

# Matriz b, creada a partir de la matriz a
b = a*4
print('b =\n', b)

# Utilizaremos estas dos matrices para mostrar los mecanismos
# de apilamiento disponibles
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]
```

In [2]:
```
# APILAMIENTO HORIZONTAL

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento horizontal
print('Apilamiento horizontal =\n', np.hstack((a,b)) )
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento horizontal =
 [[ 0  1  2  3  0  4  8 12]
 [ 4  5  6  7 16 20 24 28]
 [ 8  9 10 11 32 36 40 44]
 [12 13 14 15 48 52 56 60]]
```

In [3]:
```
# APILAMIENTO HORIZONTAL - Variante

# Utilización de la función: concatenate()

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento horizontal
print( 'Apilamiento horizontal con concatenate = \n',
np.concatenate((a,b), axis=1) )

# Si axis=1, el apilamiento es horizontal
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento horizontal con concatenate =
 [[ 0  1  2  3  0  4  8 12]
 [ 4  5  6  7 16 20 24 28]
 [ 8  9 10 11 32 36 40 44]
 [12 13 14 15 48 52 56 60]]
```

In [4]:
```
# APILAMIENTO VERTICAL

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento vertical
print( 'Apilamiento vertical =\n', np.vstack((a,b)) )
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento vertical =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]
```

In [5]:
```
# APILAMIENTO VERTICAL - Variante

# Utilización de la función: concatenate()

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento vertical
print( 'Apilamiento vertical con concatenate =\n',
np.concatenate((a,b), axis=0) )

# Si axis=0, el apilamiento es vertical
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento vertical con concatenate =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]
```

In [6]:
```
# APILAMIENTO EN PROFUNDIDAD

# En el apilamiento en profundidad, se crean bloques utilizando
# parejas de datos tomados de las dos matrices

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento en profundidad
print( 'Apilamiento en profundidad =\n', np.dstack((a,b)) )
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento en profundidad =
 [[[ 0  0]
  [ 1  4]
  [ 2  8]
  [ 3 12]]

 [[ 4 16]
  [ 5 20]
  [ 6 24]
  [ 7 28]]

 [[ 8 32]
  [ 9 36]
  [10 40]
  [11 44]]

 [[12 48]
  [13 52]
  [14 56]
  [15 60]]]
```

In [7]:
```
# APILAMIENTO POR COLUMNAS

# El apilamiento por columnas es similar a hstack()

# Se apilan las columnas, de izquierda a derecha, y tomándolas
# de los bloques definidos en la matriz

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento vertical
print( 'Apilamiento por columnas =\n',
np.column_stack((a,b)) )
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento por columnas =
 [[ 0  1  2  3  0  4  8 12]
 [ 4  5  6  7 16 20 24 28]
 [ 8  9 10 11 32 36 40 44]
 [12 13 14 15 48 52 56 60]]
```

In [8]:
```
# APILAMIENTO POR FILAS

# El apilamiento por fila es similar a vstack()

# Se apilan las filas, de arriba hacia abajo, y tomándolas
# de los bloques definidos en la matriz

# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')

# Apilamiento vertical
print( 'Apilamiento por filas =\n',
np.row_stack((a,b)) )
```

```
a =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

b =
 [[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

Apilamiento por filas =
 [[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]
```

In [9]:
```
# DIVISIÓN DE ARRAYS

# Las matrices se pueden dividir vertical, horizontalmente o en profundidad.
# Las funciones involucradas son hsplit, vsplit, dsplit y split.
# Podemos hacer divisiones de las matrices utilizando su estructura inicial
# o hacerlo indicando la posición después de la cual debe ocurrir la división

# DIVISIÓN HORIZONTAL

print(a, '\n')

# El código resultante divide una matriz a lo largo de su eje horizontal
# en tres piezas del mismo tamaño y forma:}
print('Array con división horizontal =\n', np.hsplit(a, 4), '\n')

# El mismo efecto se consigue con split() y utilizando una bandera a 1
print('Array con división horizontal, uso de split() =\n',
np.split(a, 4, axis=1))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]]

Array con división horizontal =
 [array([[ 0],
       [ 4],
       [ 8],
       [12]]), array([[ 1],
       [ 5],
       [ 9],
       [13]]), array([[ 2],
       [ 6],
       [10],
       [14]]), array([[ 3],
       [ 7],
       [11],
       [15]])]

Array con división horizontal, uso de split() =
 [array([[ 0],
       [ 4],
       [ 8],
       [12]]), array([[ 1],
       [ 5],
       [ 9],
       [13]]), array([[ 2],
       [ 6],
       [10],
       [14]]), array([[ 3],
       [ 7],
       [11],
       [15]])]
```

In [10]:
```
# DIVISIÓN VERTICAL

print(a, '\n')

# La función vsplit divide el array a lo largo del eje vertical:
print('División Vertical = \n', np.vsplit(a, 4), '\n')

# El mismo efecto se consigue con split() y utilizando una bandera a 0
print('Array con división vertical, uso de split() =\n',
np.split(a, 4, axis=0))
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]] 

División Vertical = 
 [array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]]), arra
y([[12, 13, 14, 15]])] 

Array con división vertical, uso de split() = 
 [array([[0, 1, 2, 3]]), array([[4, 5, 6, 7]]), array([[ 8,  9, 10, 11]]), arra
y([[12, 13, 14, 15]])]
```

In [11]:
```
# DIVISIÓN EN PROFUNDIDAD

# La función dsplit, como era de esperarse, realiza división
# en profundidad dentro del array

# Para ilustrar con un ejemplo, utilizaremos una matriz de rango tres
c = np.arange(27).reshape(3, 3, 3)

print(c, '\n')

# Se realiza la división
print('División en profundidad =\n', np.dsplit(c,3), '\n')
```

```
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]
  [24 25 26]]]

División en profundidad =
 [array([[[ 0],
         [ 3],
         [ 6]],

        [[ 9],
         [12],
         [15]],

        [[18],
         [21],
         [24]]]), array([[[ 1],
         [ 4],
         [ 7]],

        [[10],
         [13],
         [16]],

        [[19],
         [22],
         [25]]]), array([[[ 2],
         [ 5],
         [ 8]],

        [[11],
         [14],
         [17]],

        [[20],
         [23],
```

```
[26]]])]
```

In [12]:
```
# PROPIEDADES DE LOS ARRAYS

# El atributo ndim calcula el número de dimensiones

print(b, '\n')
print('ndim: ', b.ndim)
```

```
[[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

ndim:  2
```

In [13]:
```
# El atributo size calcula el número de elementos

print(b, '\n')
print('size: ', b.size)
```

```
[[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

size:  16
```

In [14]:
```
# El atributo itemsize obtiene el número de bytes por cada
# elemento en el array

print('itemsize: ', b.itemsize)
```

```
itemsize:  4
```

In [15]:
```
# El atributo nbytes calcula el número total de bytes del array

print(b, '\n')

print('nbytes: ', b.nbytes, '\n')

# Es equivalente a la siguiente operación
print('nbytes equivalente: ', b.size * b.itemsize)
```

```
[[ 0  4  8 12]
 [16 20 24 28]
 [32 36 40 44]
 [48 52 56 60]]

nbytes:  64

nbytes equivalente:  64
```

In [ ]: