In [1]:
```
# COMPUTACIÓN BLANDA - Sistemas y Computación
# ------------------------------------------
# Introducción a numpy
# ------------------------------------------
# Lección 02
#
# ** Técnicas de apilamiento
# ** División de arrays
# ** Propiedades de arrays
#
# ------------------------------------------
```

In [29]:
```
# 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(12).reshape(4,3)
print('a =\n', a, '\n')
# Matriz b, creada a partir de la matriz a
b = a*2
print('b =\n', b, '\n')
c = a*3
print('c =\n', c)
# Utilizaremos estas dos matrices para mostrar los mecanismos
# de apilamiento disponibles
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

c =
 [[ 0  3  6]
 [ 9 12 15]
 [18 21 24]
 [27 30 33]]
```

In [30]:
```
# APILAMIENTO HORIZONTAL
# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')
print('c =\n', c, '\n')
# Apilamiento horizontal
print('Apilamiento horizontal =\n', np.hstack((a,b,c)) )
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

c =
 [[ 0  3  6]
 [ 9 12 15]
 [18 21 24]
 [27 30 33]]

Apilamiento horizontal =
 [[ 0  1  2  0  2  4  0  3  6]
 [ 3  4  5  6  8 10  9 12 15]
 [ 6  7  8 12 14 16 18 21 24]
 [ 9 10 11 18 20 22 27 30 33]]
```

In [35]:
```
# 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=0) )
# Si axis=1, el apilamiento es horizontal
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

Apilamiento horizontal con concatenate =
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]
```

In [36]:
```
# APILAMIENTO VERTICAL
# Matrices origen
print('a =\n', a, '\n')
print('b =\n', b, '\n')
print('c =\n', c, '\n')
# Apilamiento vertical
print( 'Apilamiento vertical =\n', np.vstack((a,b,c)) )
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

c =
 [[ 0  3  6]
 [ 9 12 15]
 [18 21 24]
 [27 30 33]]

Apilamiento vertical =
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]
 [ 0  3  6]
 [ 9 12 15]
 [18 21 24]
 [27 30 33]]
```

In [37]:
```
# 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=1) )
# Si axis=0, el apilamiento es vertical
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

Apilamiento vertical con concatenate =
 [[ 0  1  2  0  2  4]
 [ 3  4  5  6  8 10]
 [ 6  7  8 12 14 16]
 [ 9 10 11 18 20 22]]
```

In [38]:
```
# 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')
print('c =\n', c, '\n')
# Apilamiento en profundidad
print( 'Apilamiento en profundidad =\n', np.dstack((a,b,c)) )
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

c =
 [[ 0  3  6]
 [ 9 12 15]
 [18 21 24]
 [27 30 33]]

Apilamiento en profundidad =
 [[[ 0  0  0]
  [ 1  2  3]
  [ 2  4  6]]

 [[ 3  6  9]
  [ 4  8 12]
  [ 5 10 15]]

 [[ 6 12 18]
  [ 7 14 21]
  [ 8 16 24]]

 [[ 9 18 27]
  [10 20 30]
  [11 22 33]]]
```

In [41]:
```
# 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')
print('c =\n', c, '\n')

d = b*2
print('d =\n', d, '\n')

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

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

c =
 [[ 0  3  6]
 [ 9 12 15]
 [18 21 24]
 [27 30 33]]

d =
 [[ 0  4  8]
 [12 16 20]
 [24 28 32]
 [36 40 44]]

Apilamiento por columnas =
 [[ 0  1  2  0  2  4  0  3  6  0  4  8]
 [ 3  4  5  6  8 10  9 12 15 12 16 20]
 [ 6  7  8 12 14 16 18 21 24 24 28 32]
 [ 9 10 11 18 20 22 27 30 33 36 40 44]]
```

In [42]:
```
# 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')
print('d =\n', d, '\n')
# Apilamiento vertical
print( 'Apilamiento por filas =\n',
np.row_stack((a,b,d)) )
```

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

b =
 [[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

d =
 [[ 0  4  8]
 [12 16 20]
 [24 28 32]
 [36 40 44]]

Apilamiento por filas =
 [[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]
 [ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]
 [ 0  4  8]
 [12 16 20]
 [24 28 32]
 [36 40 44]]
```

In [10]:
```
# 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
```

In [51]:
```
# DIVISIÓN HORIZONTAL
print(a, '\n')
print(b, '\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, 3), '\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, 3, axis=1))
print('Array con división horizontal, uso de split() =\n',
np.split(b, 3, axis=1))
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

[[ 0  2  4]
 [ 6  8 10]
 [12 14 16]
 [18 20 22]]

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

Array con división horizontal, uso de split() =
 [array([[0],
       [3],
       [6],
       [9]]), array([[ 1],
       [ 4],
       [ 7],
       [10]]), array([[ 2],
       [ 5],
       [ 8],
       [11]])]
Array con división horizontal, uso de split() =
 [array([[ 0],
       [ 6],
       [12],
       [18]]), array([[ 2],
       [ 8],
       [14],
       [20]]), array([[ 4],
       [10],
       [16],
       [22]])]
```

In [54]:
```
# 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, 3, axis=1))
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]

División Vertical =
 [array([[0, 1, 2]]), array([[3, 4, 5]]), array([[6, 7, 8]]), array([[ 9, 10,
11]])]

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

In [61]:
```
# 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(125).reshape(5, 5, 5)
print(c, '\n')
# Se realiza la división
print('División en profundidad =\n', np.dsplit(c,5), '\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  27  28  29]
  [ 30  31  32  33  34]
  [ 35  36  37  38  39]
  [ 40  41  42  43  44]
  [ 45  46  47  48  49]]

 [[ 50  51  52  53  54]
  [ 55  56  57  58  59]
  [ 60  61  62  63  64]
  [ 65  66  67  68  69]
  [ 70  71  72  73  74]]

 [[ 75  76  77  78  79]
  [ 80  81  82  83  84]
  [ 85  86  87  88  89]
  [ 90  91  92  93  94]
  [ 95  96  97  98  99]]

 [[100 101 102 103 104]
  [105 106 107 108 109]
  [110 111 112 113 114]
  [115 116 117 118 119]
  [120 121 122 123 124]]]

División en profundidad =
 [array([[[  0],
         [  5],
         [ 10],
         [ 15],
         [ 20]],

        [[ 25],
         [ 30],
         [ 35],
         [ 40],
         [ 45]],

        [[ 50],
         [ 55],
         [ 60],
         [ 65],
         [ 70]],

        [[ 75],
         [ 80],
         [ 85],
         [ 90],
         [ 95]],

        [[100],
         [105],
```

```
 [110],
 [115],
 [120]]]), array([[[  1],
 [  6],
 [ 11],
 [ 16],
 [ 21]],

[[ 26],
 [ 31],
 [ 36],
 [ 41],
 [ 46]],

[[ 51],
 [ 56],
 [ 61],
 [ 66],
 [ 71]],

[[ 76],
 [ 81],
 [ 86],
 [ 91],
 [ 96]],

[[101],
 [106],
 [111],
 [116],
 [121]]]), array([[[  2],
 [  7],
 [ 12],
 [ 17],
 [ 22]],

[[ 27],
 [ 32],
 [ 37],
 [ 42],
 [ 47]],

[[ 52],
 [ 57],
 [ 62],
 [ 67],
 [ 72]],

[[ 77],
 [ 82],
 [ 87],
 [ 92],
 [ 97]],

[[102],
 [107],
 [112],
```

```
 [117],
 [122]]]), array([[[  3],
 [  8],
 [ 13],
 [ 18],
 [ 23]],

[[ 28],
 [ 33],
 [ 38],
 [ 43],
 [ 48]],

[[ 53],
 [ 58],
 [ 63],
 [ 68],
 [ 73]],

[[ 78],
 [ 83],
 [ 88],
 [ 93],
 [ 98]],

[[103],
 [108],
 [113],
 [118],
 [123]]]), array([[[  4],
 [  9],
 [ 14],
 [ 19],
 [ 24]],

[[ 29],
 [ 34],
 [ 39],
 [ 44],
 [ 49]],

[[ 54],
 [ 59],
 [ 64],
 [ 69],
 [ 74]],

[[ 79],
 [ 84],
 [ 89],
 [ 94],
 [ 99]],

[[104],
 [109],
 [114],
 [119],
```

```
[124]]])]
```

In [14]:
```
# PROPIEDADES DE LOS ARRAYS
```

In [15]:
```
# El atributo ndim calcula el número de dimensiones

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

```
[[ 0  2  4]
 [ 6  8 10]
 [12 14 16]]

ndim:  2
```

In [16]:
```
# El atributo size calcula el número de elementos

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

```
[[ 0  2  4]
 [ 6  8 10]
 [12 14 16]]

size:  9
```

In [17]:
```
# El atributo itemsize obtiene el número de bytes por cada

# elemento en el array
print('itemsize: ', b.itemsize)
```

```
itemsize:  4
```

In [18]:
```
# 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  2  4]
 [ 6  8 10]
 [12 14 16]]

nbytes:  36

nbytes equivalente:  36
```

In [19]:
```
# El atributo T tiene el mismo efecto que la transpuesta de la matriz

b.resize(6,4)
print(b, '\n')
print('Transpuesta: ', b.T)
```

```
[[ 0  2  4  6]
 [ 8 10 12 14]
 [16  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0  0]
 [ 0  0  0  0]]

Transpuesta:  [[ 0  8 16  0  0  0]
 [ 2 10  0  0  0  0]
 [ 4 12  0  0  0  0]
 [ 6 14  0  0  0  0]]
```

In [20]:
```
# Los números complejos en numpy se representan con j

b = np.array([1.j + 1, 2.j + 3])
print('Complejo: \n', b)
```

```
Complejo:
 [1.+1.j 3.+2.j]
```

In [21]:
```
# El atributo real nos da la parte real del array,
# o el array en sí mismo si solo contiene números reales
print('real: ', b.real, '\n')
# El atributo imag contiene la parte imaginaria del array
print('imaginario: ', b.imag)
```

```
real:  [1. 3.]

imaginario:  [1. 2.]
```

In [22]:
```
# Si el array contiene números complejos, entonces el tipo de datos

# se convierte automáticamente a complejo
print(b.dtype)
```

```
complex128
```

In [23]:
```
# El atributo flat devuelve un objeto numpy.flatiter.
# Esta es la única forma de adquirir un flatiter:
# no tenemos acceso a un constructor de flatiter.
# El apartamento El iterador nos permite recorrer una matriz
# como si fuera una matriz plana, como se muestra a continuación:
# En el siguiente ejemplo se clarifica este concepto
b = np.arange(4).reshape(2,2)
print(b, '\n')
f = b.flat
print(f, '\n')
# Ciclo que itera a lo largo de f
for item in f: print (item)
# Selección de un elemento
print('\n')
print('Elemento 2: ', b.flat[2])
# Operaciones directas con flat
b.flat = 7
print(b, '\n')
b.flat[[1,3]] = 1
print(b, '\n')
```

```
[[0 1]
 [2 3]]

<numpy.flatiter object at 0x02E58C70>

0
1
2
3


Elemento 2:  2
[[7 7]
 [7 7]]

[[7 1]
 [7 1]]
```

In [ ]:
```
```